soapui

soapUI v.1.7

Project Documentation

eviware **10 April 2007**

Table of Contents

1 **User Guide** . . . . .
1.1 Overview 1
1.1.1 Tabbed Desktop 1
1.1.2 License Management 3
1.1.3 XPath Selection 5
1.1.4 Preferences 7
1.2 Workspaces and Projects 16
1.3 Interfaces 20
1.3.1 Interface Viewer 26
1.4 Operations 28
1.5 Working with Requests 30
1.5.1 Form Editor 38
1.5.2 Outline Editor 42
1.5.3 Message Inspectors 44
1.5.4 Attachments/Inline Files 49
1.6 Functional Testing 55
1.6.1 Getting Started 56
1.6.2 TestSuites 59
1.6.3 TestCases 61
1.6.3.1 Test Requests 66
1.6.3.2 Property Transfers 70
1.6.3.3 Conditional Gotos 75
1.6.3.4 Groovy Scripts 78
1.6.3.5 Properties Step 83
1.6.3.6 Delay Step 85
1.6.3.7 Mock Response 86
1.6.3.8 DataSource 89
1.6.3.9 DataSource Loop 95

1.6.4 Property Expansion 96
1.7 Load Testing 98
1.7.1 Getting Started 103
1.7.2 Limit & Strategies 108
1.7.3 Execution 111
1.7.4 Assertions 115
1.7.5 Diagrams 120
1.7.6 JMeter comparison 122
1.8 Mocking 127
1.8.1 Getting Started 128
1.8.2 Mock Services 134
1.8.3 Mock Operations 139
1.8.4 Mock Responses 143
1.9 Usage Scenarios 147
1.9.1 Data-Driven Testing 148
1.9.2 Template-Driven Testing 153
1.9.3 Interactive Testing 158
1.9.4 Surveillance Testing 164
1.10 Tool Integrations 172
1.10.1 Code Generation 174
1.10.2 WSDL Generation 179
1.10.3 WS-I Tools 181
1.10.4 Apache Tcp-Mon 185
1.11 CommandLine Tools 1
1.11.1 TestCaseRunner 1
1.11.2 LoadTestRunner 6
1.11.3 MockServiceRunner 8
1.11.4 ToolRunner 10
1.12 IDE/Tool Plugins 12
1.12.1 Maven Plugins 13
1.12.1.1 Maven 1.X Plugin 14
1.12.1.2 Maven 2.X Plugin 19
1.12.2 NetBeans Plugin 21

1.12.2.1 Installation 22
1.12.3 IntelliJ Plugin 26
1.12.4 Eclipse Plugin 27
1.12.4.1 soapUI Nature 30
1.12.5 JBossWS Plugin 34
1.12.5.1 New in 2.0.0 beta2 36
1.12.5.2 Getting Started 42
1.12.5.3 Publishing Web Services 45
1.12.5.4 Consuming a Web Service 50
1.12.5.5 Implementing a Web Service 54
1.12.5.6 Web Service Annotations 56
1.13 Keyboard shortcuts 59

1.1 Overview

soapUI Overview

Prerequisites

Although soapUI should be easy to get started with, you will need to have a good grasp of the following concepts/technologies to get the most out of soapUI's functionality.

- WSDL - basic concepts (services, ports, bindings, porttypes), relation to xml-schema
- SOAP - basic concepts, relation to WSDL (bindings, etc), different encoding types (soap-encoded/literal) and message styles (document/rpc)
- XML and related technologies - XPath, XML Schema, namespaces, etc

You can find most specifications at W3C and tutorials at several sites, for example W3Schools and zvon.org .

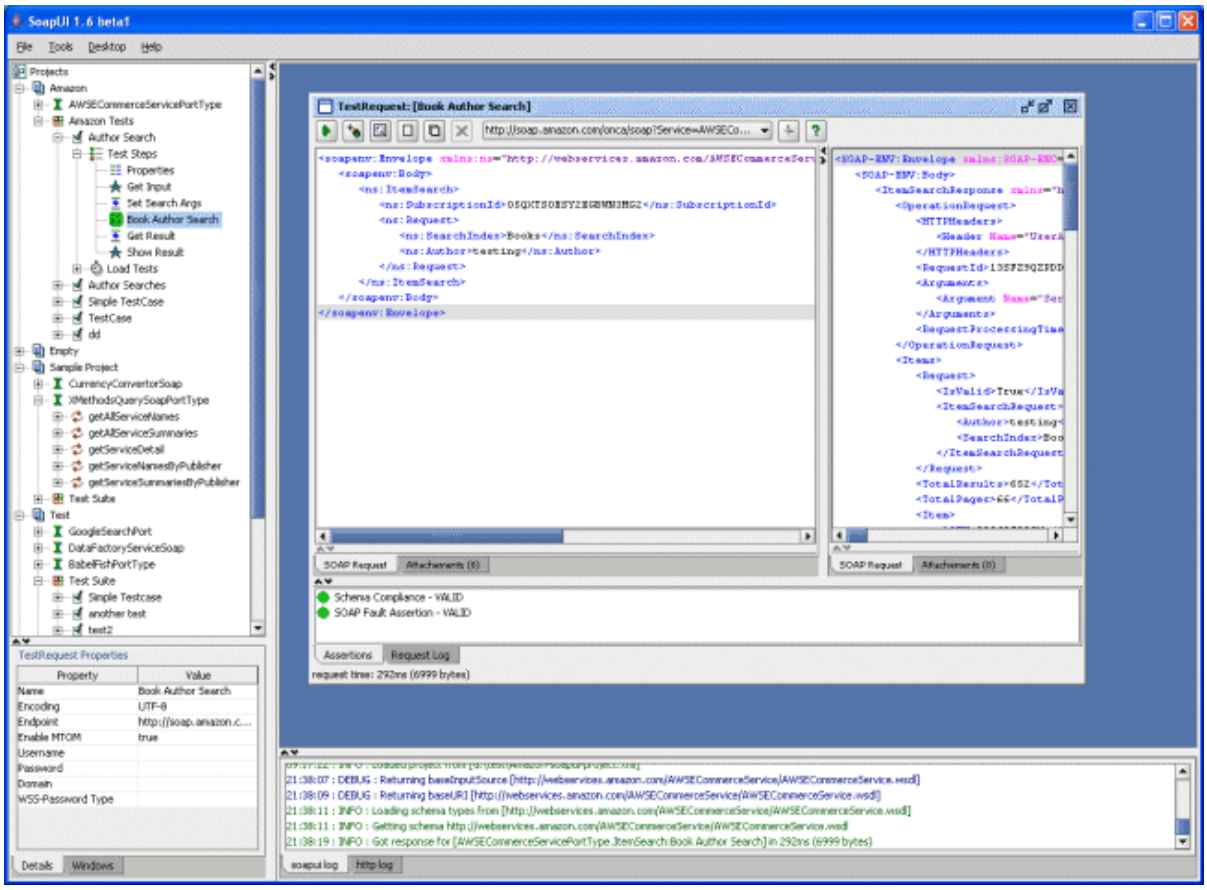
The soapUI interface

soapUI is a standard desktop application adhering to established ui-concepts (as available in for example eclipse, IDEA, etc). Most actions have keyboard shortcuts and tooltips, you should have no problems finding your way around...

The soapUI window is divided into the following views:

bottom-left
a tabbed pane containing 2 tabs:
- The **Windows** tab show a list of all the windows currently open desktop panes. Double-clicking an item will bring it into focus
- The **Details** tab will show information on the item currently selected in the navigator

bottom right
A color-coded log view showing log messages from soapUI. Three tabs will be visible:
- soapUI log : shows general soapUI messages
- http log : shows the actual http messages sent (disabled during loadtests)
- groovy log : shows log output from Groovy Test Steps (optionally disabled during loadtests, see UI Settings)
- jetty log : shows log output related to MockService functionality from the integrated http server



(One can set a centered background image for the default desktop by placing a soapui-backgroung.gif/jpg/png file in the soapUI bin folder)

Navigator Tree Model

The following objects are currently shown in the navigator tree:

- **Projects** node : the soapUI workspace
  - Project node(s) : for each project in the workspace
    - Interface node(s) : for each interface in the project
      - Operation node(s) : for each operation in the interface
        - Request node(s) : for each request created for the operation
    - TestSuite node(s) : for each testsuite in the project
      - TestCase node(s) : for each testcase in the testsuite
        - **TestSteps** node, contains the TestCases'
          - TestStep node(s) : for each testcase-step, together with a colored icon indicating the status of that step.

- **LoadTests** node, contains the TestCases'
  - LoadTest node(s) : for each loadtest in the containing testcase

- MockService node(s) : for each MockService in the project
  - MockOperation node(s) : for each MockOperation in the MockService
    - MockResponse node(s) : for each MockResponse in the containing MockOperation

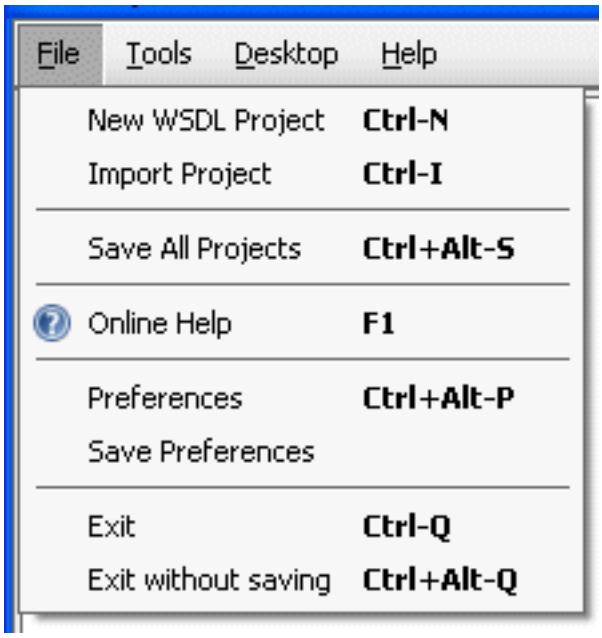
The tree can be navigated using the standard keyboard actions. An items' associated desktop panel can be opened by double-clicking it or selecting it and pressing Enter.

Main Menu

Most actions in soapUI are performed through toolbar buttons or context-sensitive right-button menus. The following actions are available from the main menu:

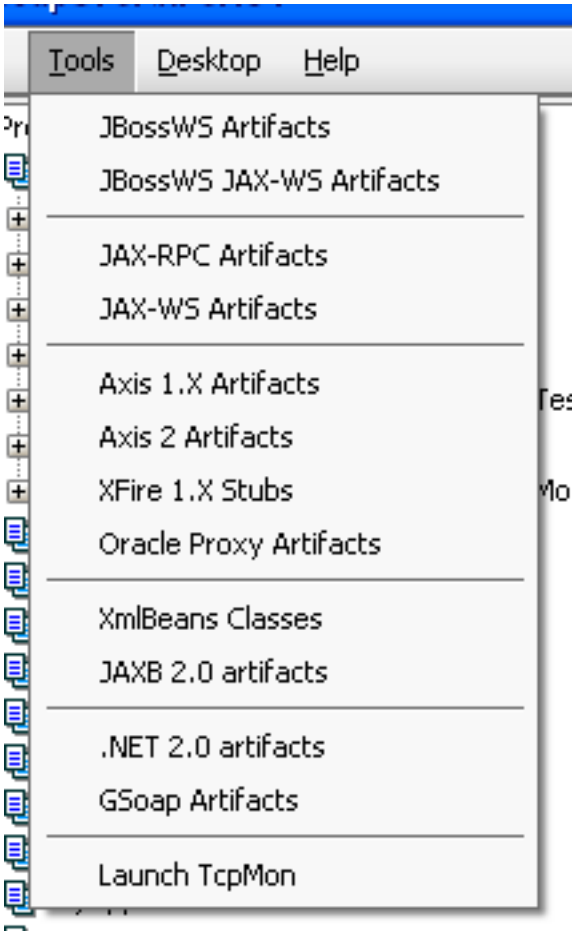
File Menu

- **New WSDL Project** - adds a new WSDL-based project to the current workspace
- **Import Project** - imports an existing project file into the current workspace
- **Save all projects** - Saves the current state of all projects
- **Online Help** - displays this page in an external browser
- **Preferences** - Sets global soapUI Preferences
- **Save preferences** - Saves the current global settings
- **Exit** - prompts to exit soapUI
- **Exit without saving** - prompts to exit soapUI without saving anything



Tools Menu

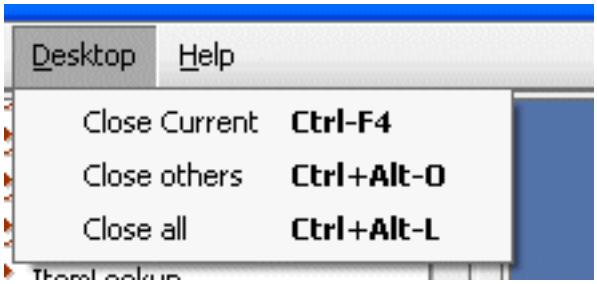
- Contains actions for invoking integrated tools as described in the Tool Integrations section.



Desktop Menu - Shows actions related to the currently selected Desktop.

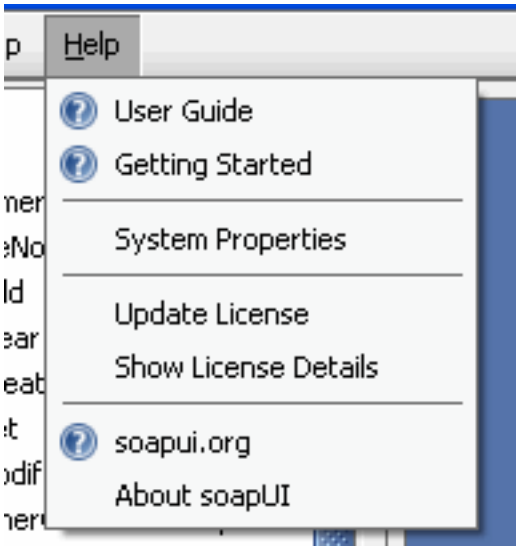
- **Close Current** - closes the currently active desktop pane

- **Close All** - closes all desktop panes
- **Closes Others** - closes all desktop panes except the currently active one



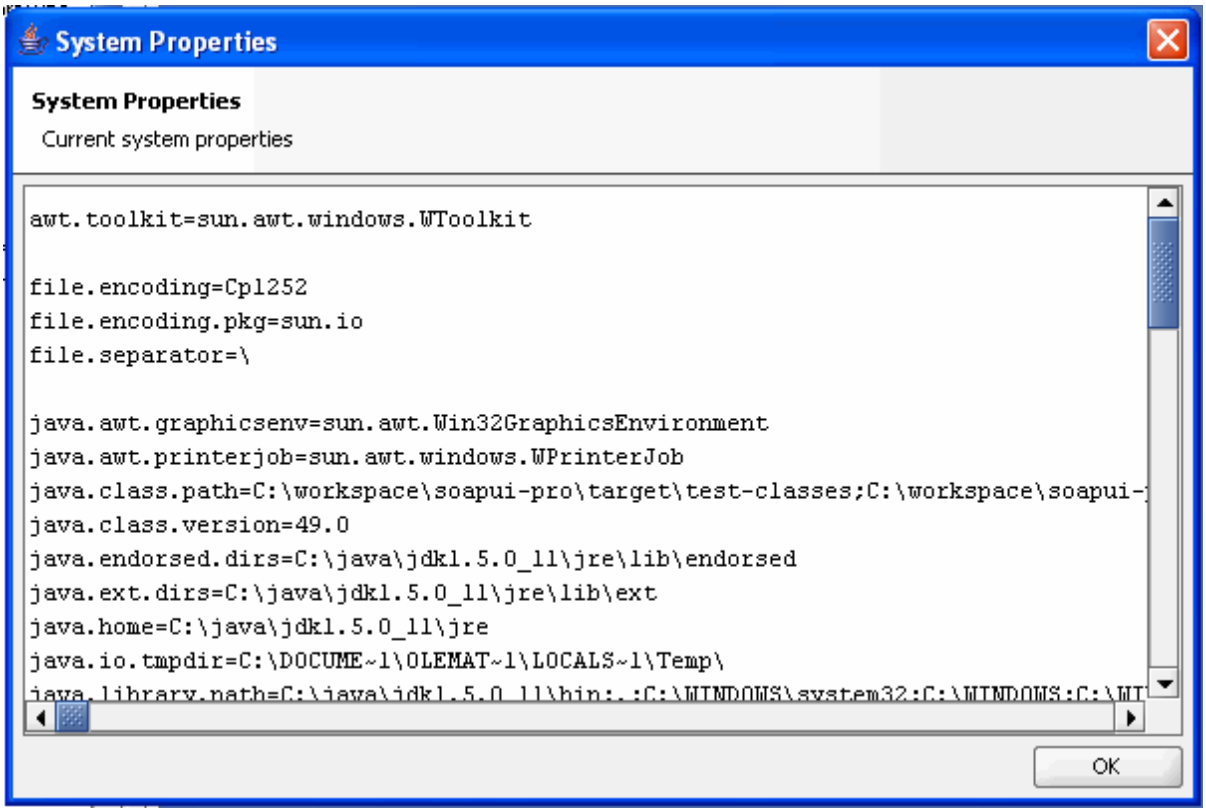
Help Menu - general information and soapUI Pro specific actions

- **User Guide** - Opens the soapUI User Guide
- **Getting Started** - Opens the soapUI Getting Started documents
- **System Properties** - Opens a list of all currently defined sysem properties (see below)
- **Update License** (soapUI Pro only) - see License Management
- **Show License Details** (soapUI Pro only) - see License Management
- **soapUI.org** - Opens http://www.soapui.org
- **About soapUI** - Shows a version information dialog



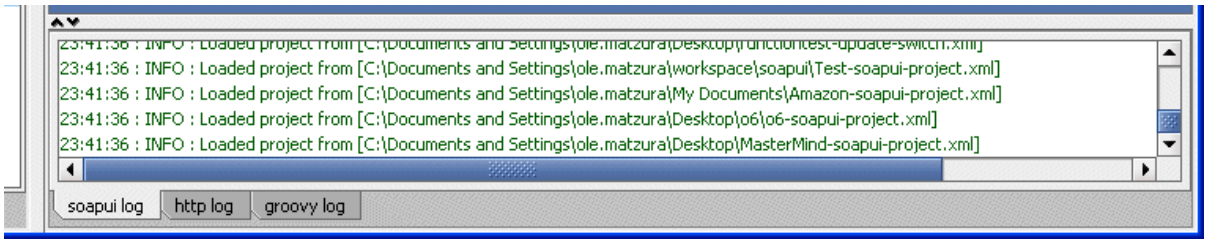
System Properties

The System Properties option from the help menu opens a simple dialog showing all current defined system properties, which can be usefull for debugging system/security settings, etc..



Log Tabs

The bottom right of the soapUI workspace contains a number of log windows, each displaying corresponding output;



Right clicking on a log will show a popup menu with options to clear and enable/disable the log and to copy selected rows to the clipboard. Also, log-entries can be selected/copied to the clipboard for transfer to external tools. An option to set the maximum number of rows is available (default is 1000), when exceeding this limit, old rows are removed from the log (ie fifo).

soapUI uses log4j for logging, you can customize the log4j configuration by putting a log4j xml configuration file named "soapui-log4j.xml" in the soapUI bin directory (or the current working directory when not running from there).

Next: soapUI Preferences

1.1.1 Tabbed Desktop

soapUI Tabbed Desktop

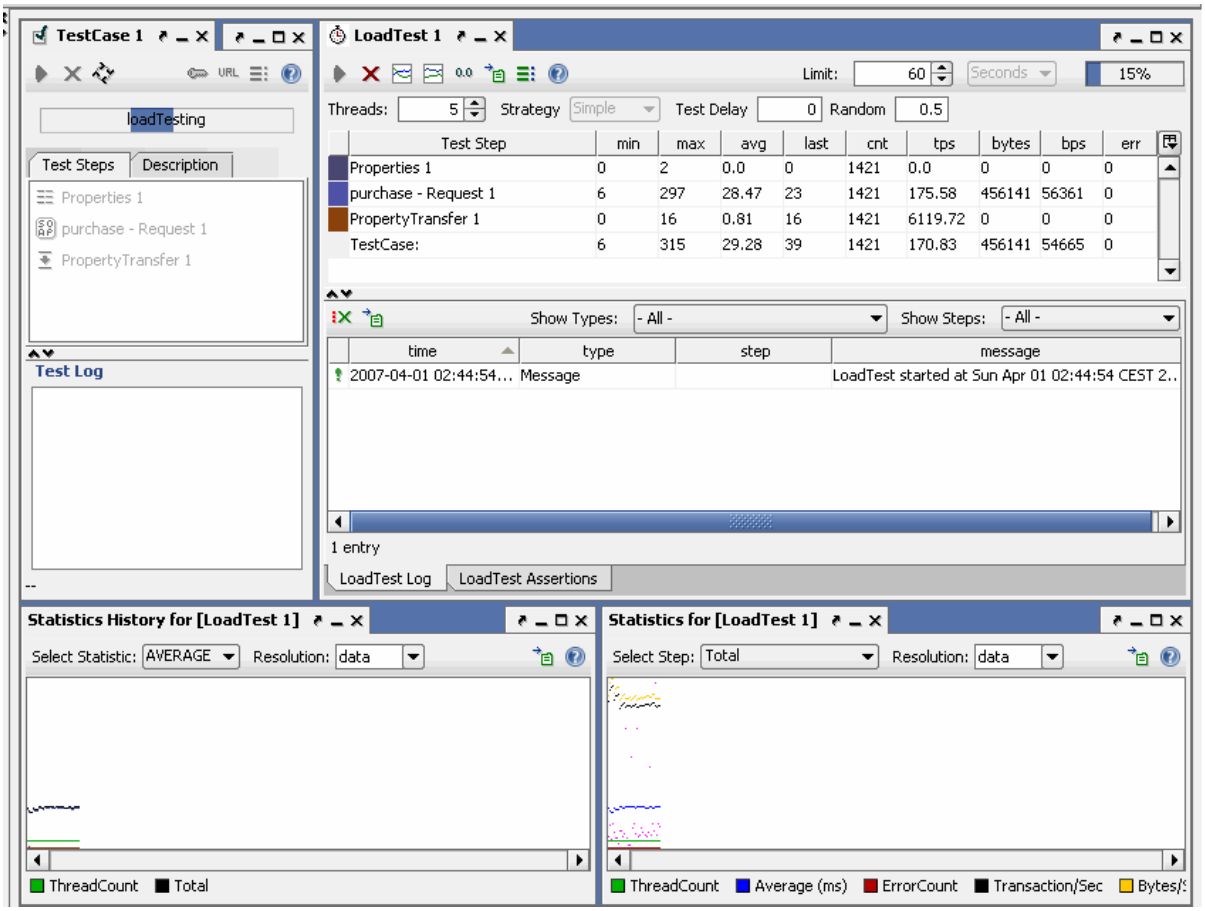
soapUI Pro adds a tabbed desktop as commonly available in other tools/IDE's. The layout is activated from the UI Settings Tab in the soapUI global preferences dialog; select "Tabbed" from the Desktop Type combo box.

Opened windows are displayed in tabs that can easily be

- dragged/arranged into multiple tab groups. Each tab-group has a top-right menu for tab-group related actions.
- docked/undocked from the desktop into seperate windows (handy with multiple monitors) - from the tabs right-click menu. Tab-Groups can like wise be docked/undocked from the Tab-Groups top-right menu.
- "glued" and/or minimized to any side of the desktop, which can come in handy for windows that need to be open but are not always in focus, for example MockService windows, TestCase runners, etc.

The tab right-click menu has a number of options for performing some of the above tasks and for changing tab orientation/directions.

For example the following screenshot shows a layout where the TestCase window has been dragged to the left and an associated LoadTest to the right. The 2 load-test diagrams have been dragged to the bottom.

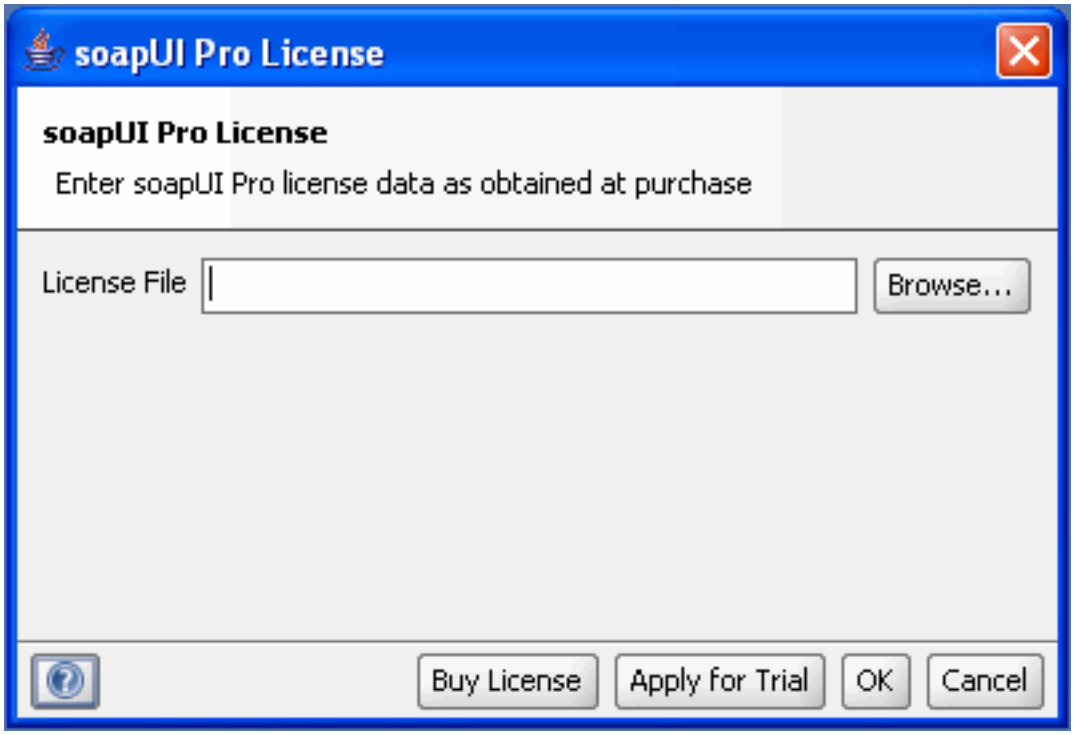


Next:

1.1.2 License Management

License Management

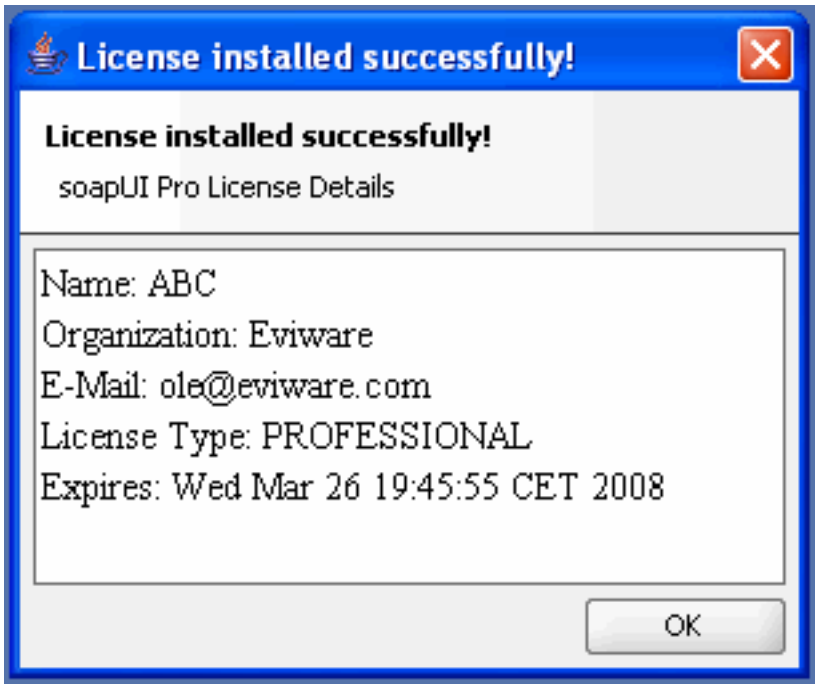
soapUI Pro license-management is straight forward; when starting soapUI the first time (or when the license has expired), the following dialog will prompt for a new License:



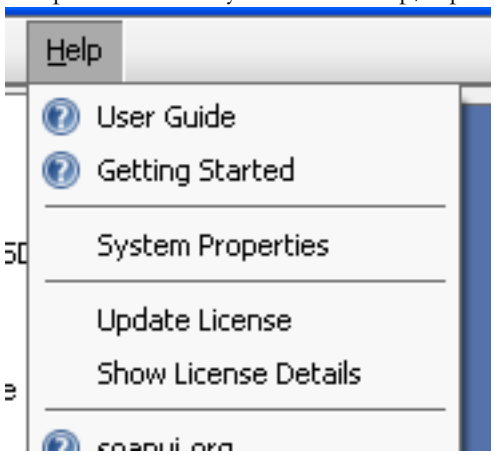
Selecting either the "Apply for Trial" or "Buy License" buttons will open corresponding pages in an external browser.

Choosing to cancel the dialog without already having a valid license installed will prompt to run soapUI in "Guest"-mode, which has all Pro-related features disabled but can still be fully used as a soapUI standalone installation.

After specifying a valid file, you will be prompted to accept the soapUI license agreement and a confirmation dialog will be shown:



This dialog can always be opened from the "Help/Show License Details" main menu option. A license can also be updated manually with the "Help/Update License" option which will restart the above process.



Command-Line, Plugins and Licenses

None of the command-line runners will require a license to run. The associated Tool/IDE plugins will be released in pro-enabled versions shortly after the standalone soapUI Pro release.

Next: XPath Selection

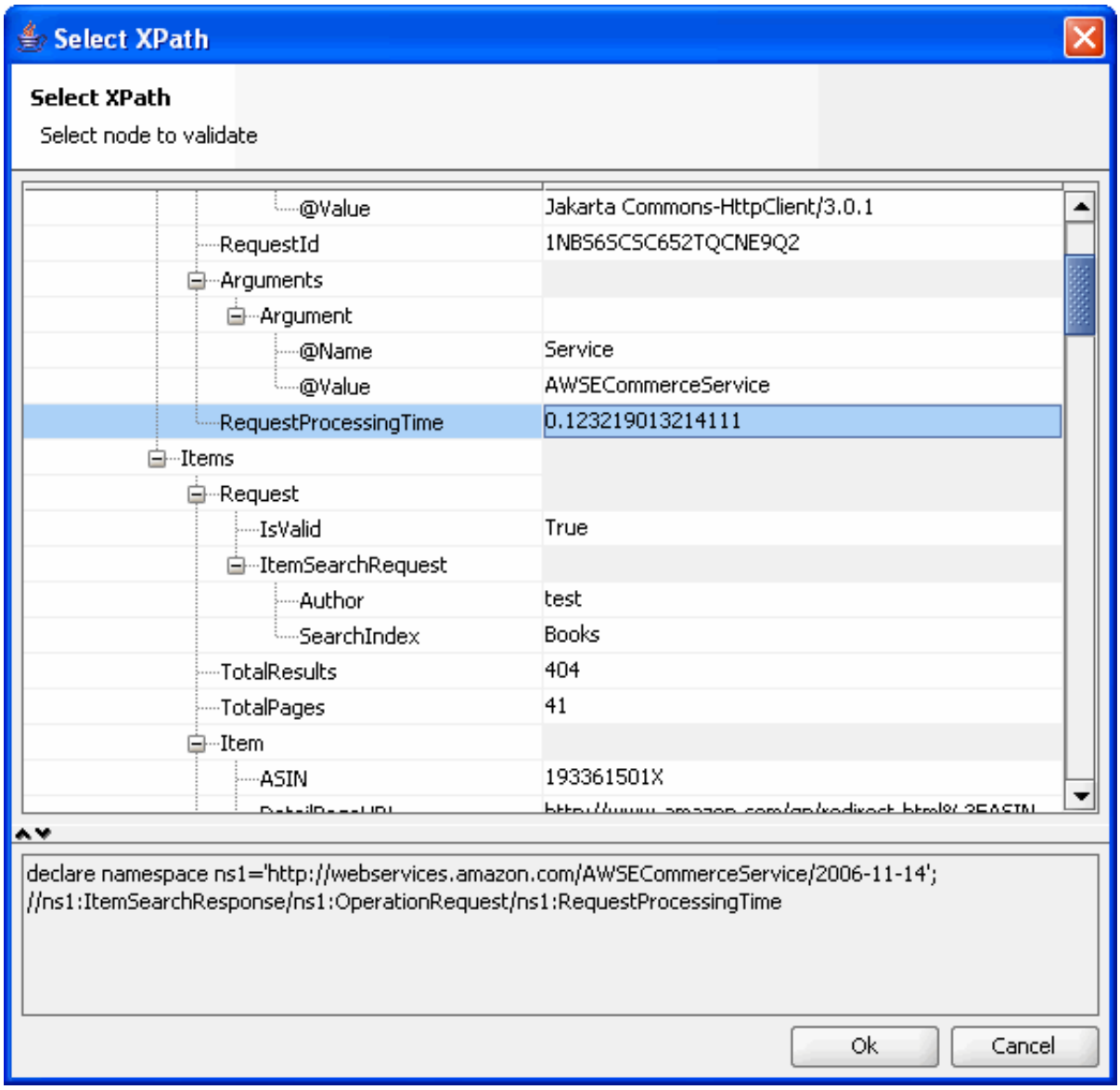
1.1.3 XPath Selection

XPath Selection Dialog

soapUI Pro includes a XPath Wizard/Selection dialog that is available whereever an XPath expression is required:

- The XPath Assertion editor
- The Property-Transfer editor
- The MockOperation editor
- The ConditionalGoto editor
- etc..

The selector displays general instructions and a tree-view of the message/xml to be selected from. Navigating the tree displays the corresponding XPath expression for the current node below. Selecting the desired node and pressing OK will close the dialog and return the selected XPath expression to the calling functionality.



Next:

1.1.4 Preferences

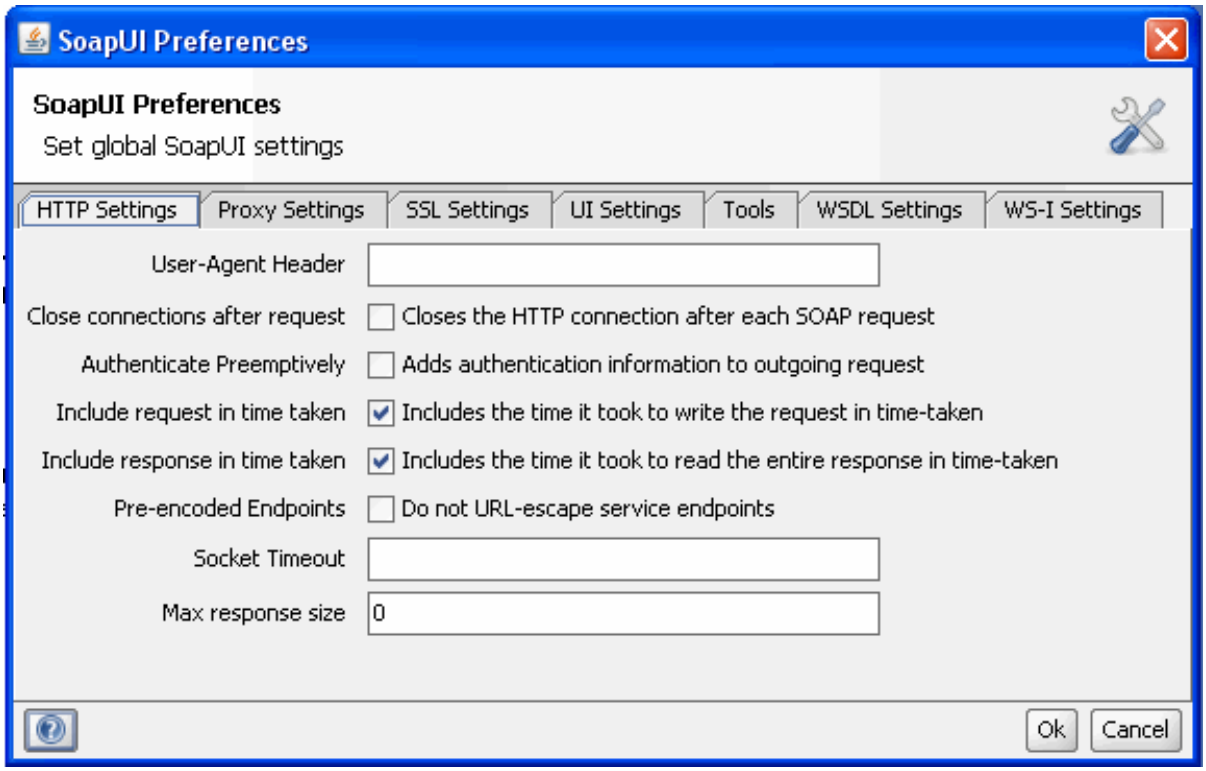
soapUI Preferences

The preferences dialog opened when selecting the "Preferences" option from the file menu contains the following tabs:

Tab	Description
HTTP Settings	Sets various HTTP-related options
Proxy Settings	Sets HTTP Proxy address and authentication
SSL Settings	Sets SSL-related options
UI Settings	Sets UI-related options
Integrated Tools	Sets paths to integrated tools
WSDL Settings	Sets WSDL-related options
WSI Settings	Sets WS-I Basic Profile validation options

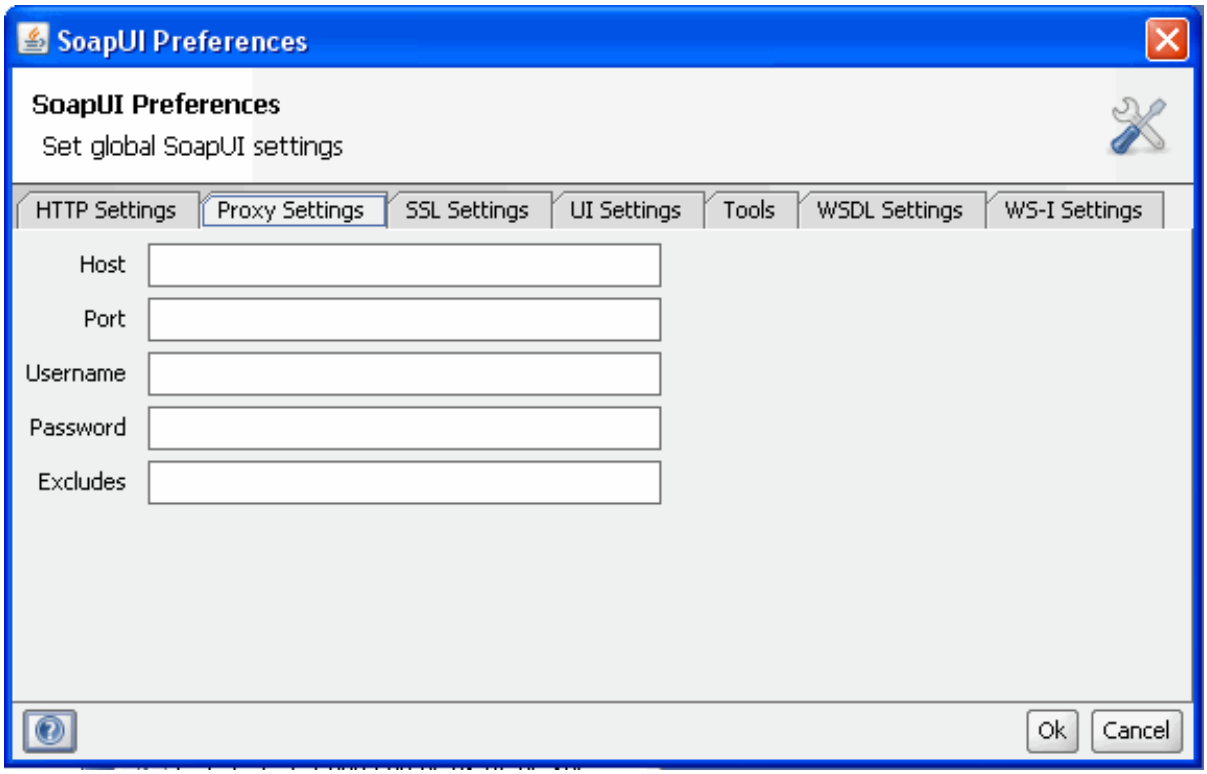
HTTP Settings

Setting	Description
User-Agent Header	Sets the HTTP User-Agent Header. If none is specified the default HttpClient header is used
Close connections after request	Disables HTTP Keep-Alives by requesting to close the HTTP connection after each request. This will have a negative impact on performance but may give more "realistic" values during loadtesting
Authenticate Preemptively	Send Authentication headers with each request without first recieving an authentication challenge. This is a potential security hazard but will improve performance since only one request will be required for authenticated endpoints instead of two
Include request in time taken	Includes the time it took to write the request in time-taken
Include response in time taken	Includes the time it took to read the response body in time-taken
Preencoded Endpoints	Do not URL-encode endpoints, set this if your endpoint URLs are already URL-encoded (containing for example %20 or %3A)
Socket Timeout	The socket timeout for HTTP requests in milliseconds
Max Response Size	The maximum number of bytes to read from a response (0 = unlimited)



Proxy Settings

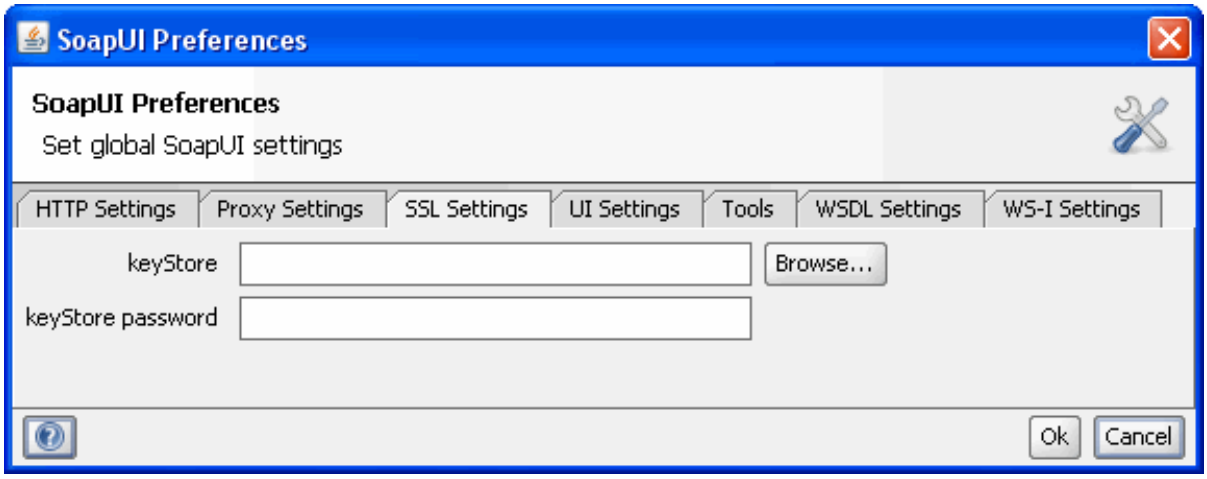
Setting	Description
Proxy Host	The HTTP Proxy host to use
Proxy Port	The HTTP Proxy port to use
Proxy Username	The username sent for proxy authentications
Proxy Password	The password sent for proxy authentications
Excludes	A comma-seperated list of hosts to exclude, for example "127.0.0.1:8080,myserver.com" will not use a proxy for 127.0.0.1 on port 8080 and myserver.com on any port.



SSL Settings

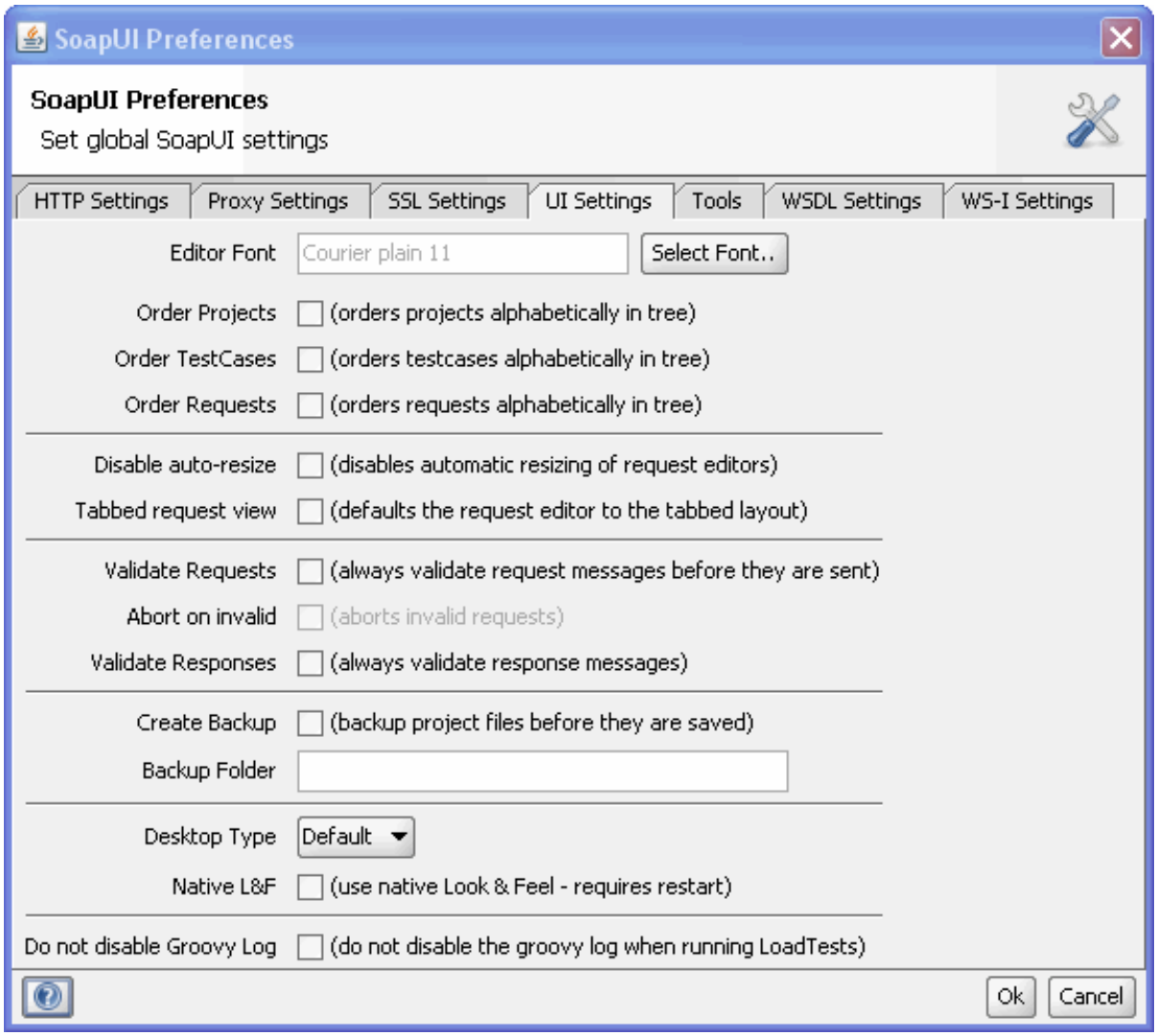
Setting	Description
keyStore	Path to the keyStore to use when locating client certificates
keyStore password	the keyStore password

Changing either of these settings should not require a restart of soapUI to get activated



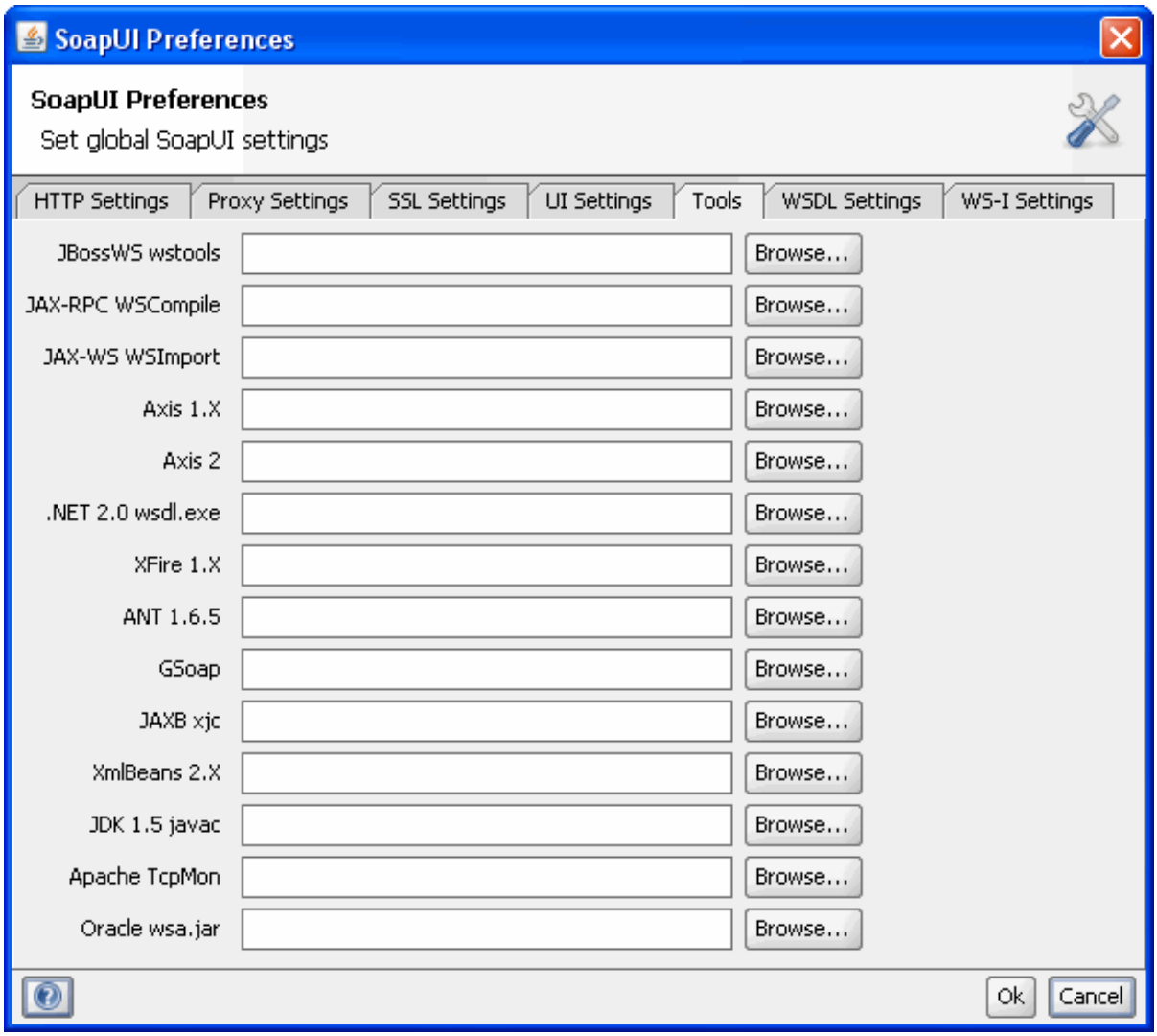
UI Settings

Setting	Description
Editor Font	The font to be used by all XML Editors. The "Select Font" button opens a dialog for selecting the desired font and size.
Order Projects	Sorts projects in alphabetical order in the navigator.
Order TestCases	Sorts testcases in alphabetical order in the navigator.
Order Requests	Sorts requests in alphabetical order in the navigator.
Disable auto-resize	Disables automatic resizing of request/response editors.
Tabbed Request view	Sets the tab-layout as the defauly layout for request/response editors (see Requests).
Validate Requests	Turns on automatic validation of requests before they are submitted from a request editor. The validation performed is the same as when pressing Alt-V in the editor.
Abort on Invalid	Enabled in conjunction with "Validate Requests", if selected any requests that fail validation will not be submitted.
Validate Responses	Turns on automatic validation of response messages when they are received in a response editor. The validation performed is the same as when pressing Alt-V in the editor.
Create Backup	Creates a backup copy of a proejcts project-file before saving.
Backup Folder	The folder where to save backups, if relative or empty the folder is relative to the project files folder
Desktop Type	Selects which desktop layou to use. soapUI Pro adds a Tabbed Desktop as an alternative to the default layout. Changing this setting will be applied when closing the preferences dialog.
Native L&F	Turns off the use of the default Look & Feel and uses the JRE default instead. Requires a restart.
Do not disable Groovy Log	Keeps the groovy log active during LoadTests, which can be usefull for debugging, etc.



Integrated Tools

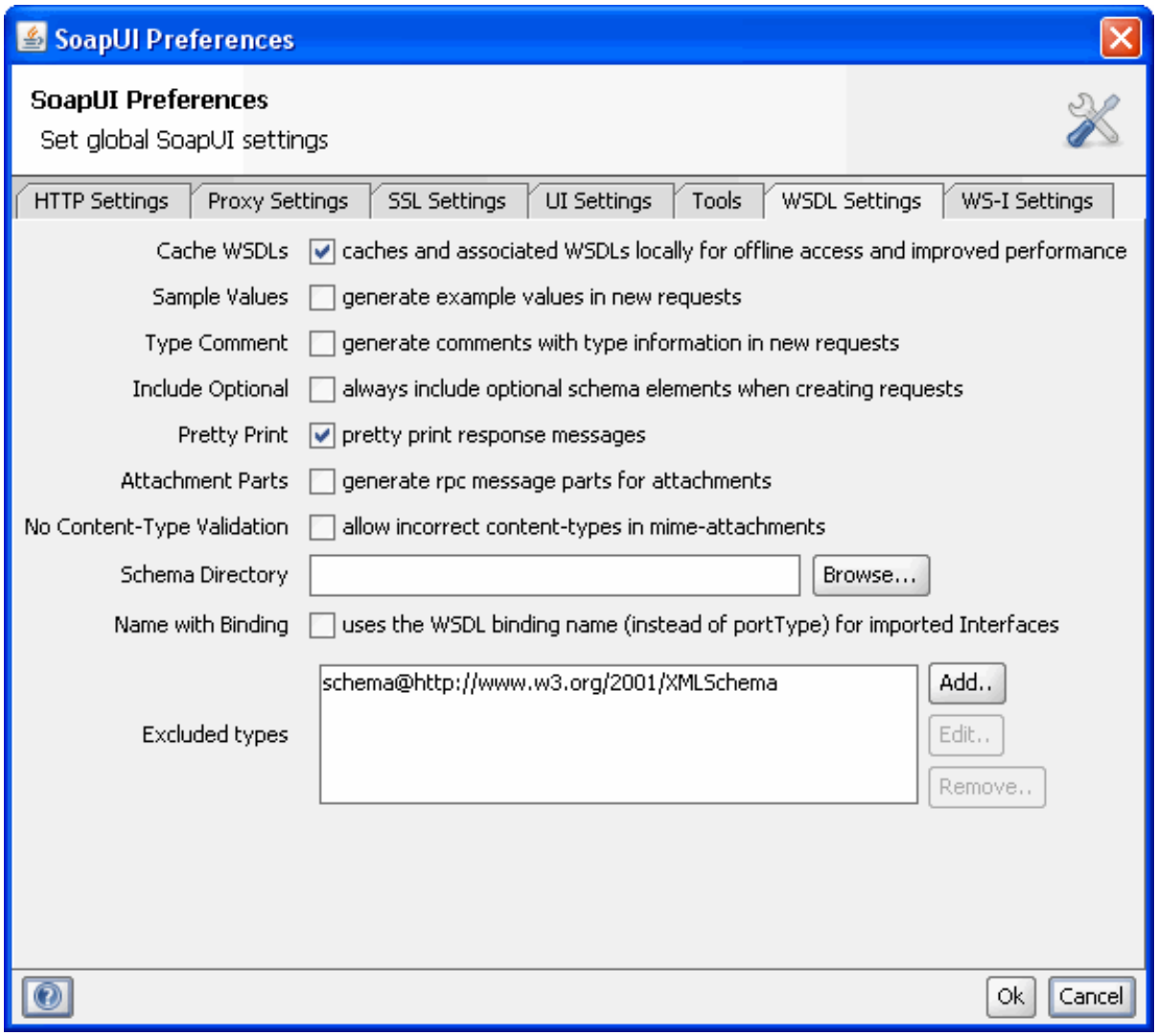
Paths to integrated Tools as described under Tool Integrations



WSDL Settings

Setting	Description
Cache WSDLs	Turns on/off caching of wsdls as described under Caching Definitions .
Sample Values	Generates example values in requests when creating from schema
Type Comment	Generates comments with type information in new requests
Include Optional	Always includes optional elements in generated requests
Pretty Print	Pretty prints response messages in response editor
Attachment Parts	Generates part-elements in request messagas for mime-attachments in RPC messages (required by some ws-stacks)
No Content-Type Validation	Does not validate the content-type of a mime-attachment against the type(s) specified in the SOAP-Binding

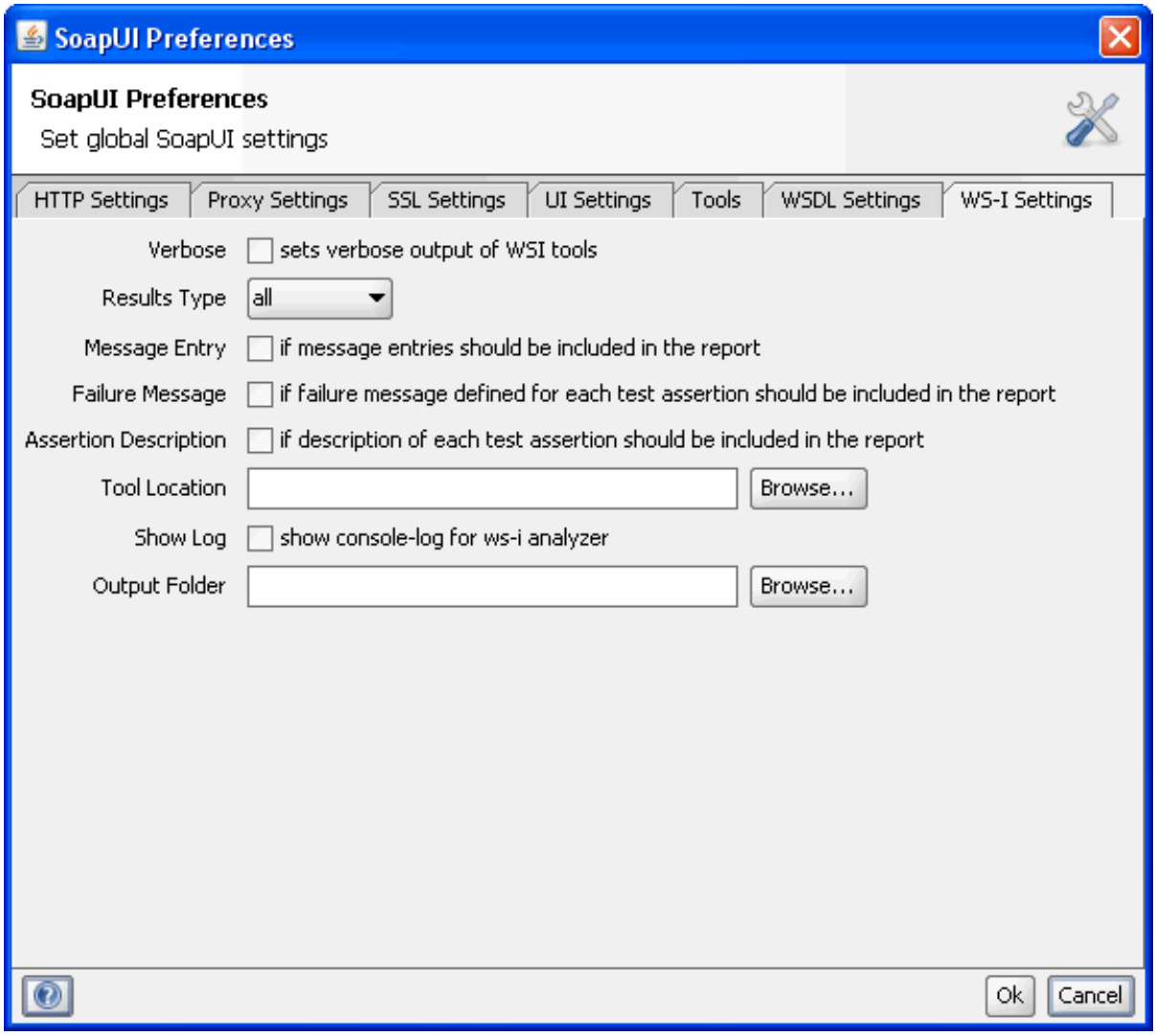
Setting	Description
Schema Directory	Specifies a directory containing schema (.xsd) files that should be automatically added when parsing/validating wsdl/requests. Changing the contents of this directory requires a restart.
Name with Binding	Tells soapUI to name imported interfaces with the name of their corresponding soap/http binding, and not with their portType (as described in Interfaces). This ensures that WSDL containing bindings for both SOAP 1.1 and SOAP 1.2 will get unique names during import. This setting defaults to true.
Excluded Types	A list of XML-Schema types/global-elements in the form of name@namespace which will be used when generating sample requests/responses and input forms in the soapUI-Pro Form editor. By default the XML-Schema root element is added since it is quite common in .NET services and generates a sample xml fragment of about 300 kb!.



WSI Settings

These settings are related to WS-I Validation functionality as described under WS-I Integrations .

Setting	Description
Verbose	Sets verbose output of WS-I tools
Results Type	Sets which results to show in the generated report
Message Entry	Shows message entries in report
Failure Message	Includes defined failure messages in report
Assertion Description	Includes description of each test assertion in report
Location	Local path to installed wsi test tools
Show Log	Shows Log window when running WS-I tools
Output Folder	If specified, generated HTML reports will automatically be exported to this folder, which is required when running ws-i validation from the command-line or one of the maven-plugins



Next: Workspaces and Projects

1.2 Workspaces and Projects

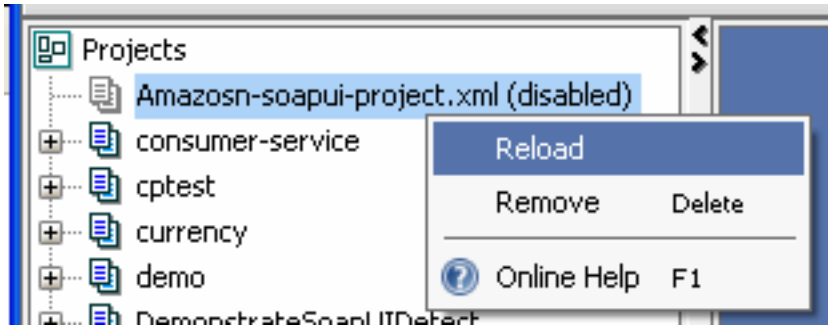
Workspaces / Projects

soapUI uses the same workspace/project metaphor as for example eclipse:

- Workspace information is maintined in the `${user.home}/default-soapui-workspace.xml` file. If you want to use multiple workspaces specify another filename (for example "my-soapui-workspace.xml") as the only command-line argument to soapUI, the corresponding file will be created/used instead.
- Any number of projects can be added to the workspace.

When starting, soapUI loads all project files contained in the current workspace and displays these as project nodes in the navigator.

If a project file is for some reason not available it will be grayed out and displayed as disabled, with right-click options to either remove it from the workspace or reload it (optionally from a new location).

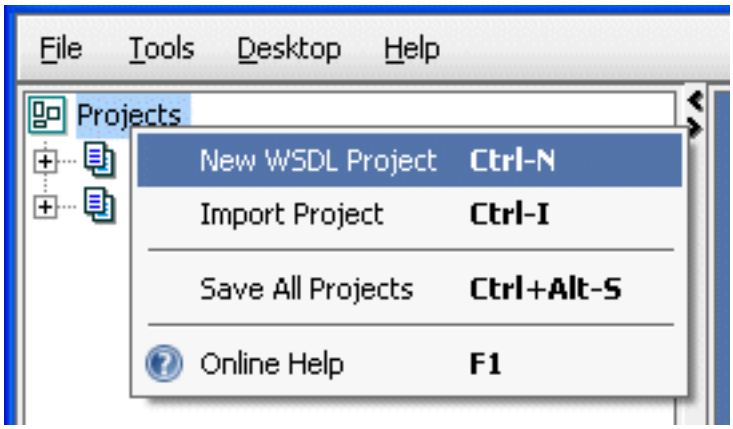


When exiting, soapUI will automatically save all project files in the workspace. If soapUI detects that a project file has been modified externally since it was last loaded (based on modification date), it will first prompt if to overwrite the file or leave it as it is.

Workspace Actions

The following actions are available from the workspace-nodes right-button menu (and from the main "File" menu):

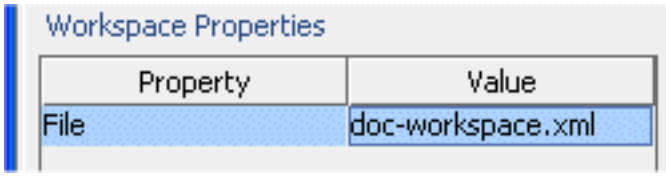
- **New WSDL Project** : Prompts to create a new WSDL project as described below.
- **Import Project** : Prompts for a filename for an existing soapui-project. The project will be added to the current workspace. If the project-file is read-only it can still be added but no changes will be saved and a warning will be shown in the soapUI log.
- **Save All Projects** - Saves all projects
- **Online Help** - Displays this page in an external browser



Workspace Details Tab

The "Details" tab shown in the bottom-left shows the following values when the workspace node is selected in the navigation tree

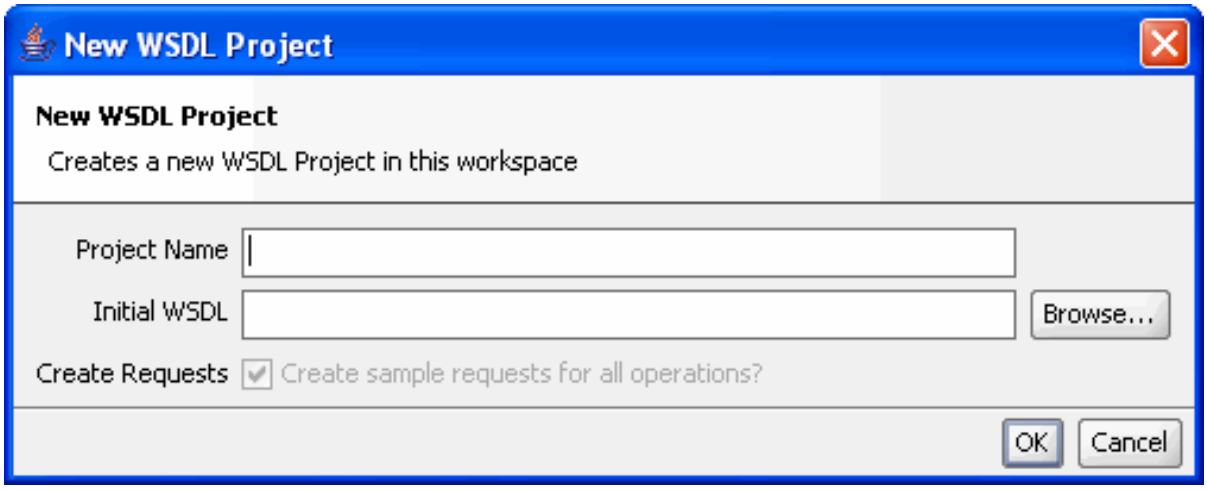
- **File** (read-only): the current workspace file used



Creating Projects

Selecting the New WSDL Project option opens this dialog which prompts for:

- A name for the project is required
- An optional inital WSDL to import into the project (either file or URL)
- An option to optionally create default requests when specifying an initial WSDL



Once specified, soapUI will prompt for a filename to which the project is to be saved (defaults to `<project-name>-soapui-project.xml`). Put this file anywhere you want, te file is a regular XML file and can be checked into SCM systems and edited with any XML editor.

soapUI WSDL Projects

Internally, soapUI abstracts the actual nature of projects and their contained interfaces, tests, etc... opening the possiblity of support for other service-definitions/protocols than WSDL/SOAP (check out the `com.eviware.soapui.model` package). Currently though, the only implementation for these interfaces is for WSDL 1.1 and the SOAP/HTTP binding (as required by Basic Profile 1.0). Therefore, all actions and functionality described will be in regard to this implementation.

A soapUI WSDL project contains the following items:

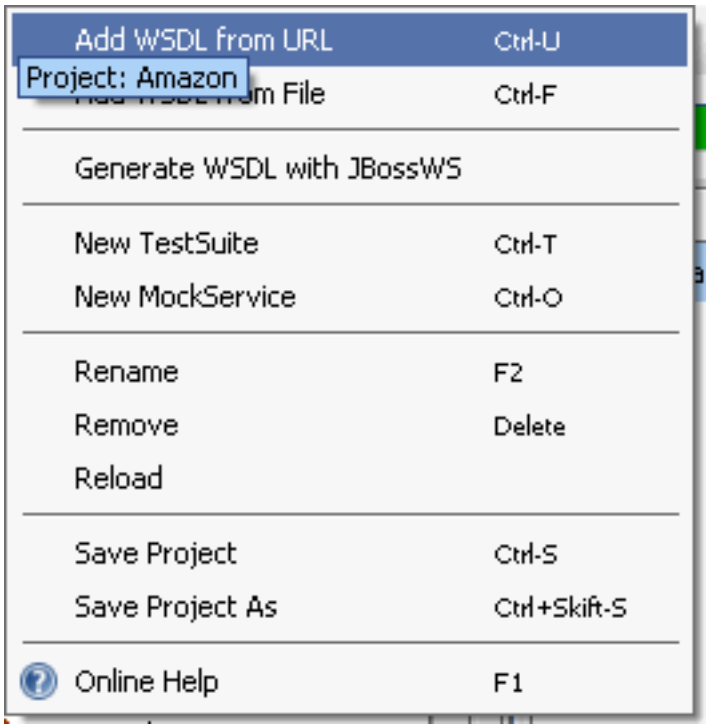
- A number of Interfaces each corresponding to a SOAP/HTTP Binding for a defined PortType
- A number of Test-Suites containing TestCases for these Interfaces' Operations
- A number of MockServices containing Mock implementations of these Interfaces' Operations

soapUI projects are saved in a self contained xml-file upon creation (as described above). This file can be safely moved around, checked into CVS, sent by email, etc. It only has file-system references if an interface/WSDL has been imported from a local file (not recommended due to this limitation) and/or if referenced attachments / inline files are used (see Attachments). When obtaining an existing soapUI project file, this can be added to the current workspace with the "Import Project" workspace action as described above

Project Actions

The following actions are available from the project nodes' right-button menu:

- **Add WSDL from URL** - Prompts for a WSDL URL that will be parsed and imported accordingly
- **Add WSDL from file** - Prompts for a local WSDL file that will be parsed and imported accordingly
- **Generate WSDL with JBossWS** - Invokes the integrated JBossWS tools for generating a WSDL from code as described under JBossWS Integration
- **New TestSuite** - Prompts to create a new TestSuite in the selected project
- **New MockService** - Prompts to create a new MockService in the selected project
- **Rename** - Prompts to rename the selected project. The new name will be shown in the navigation tree. The filename of the project will be unaffected
- **Remove** - Prompts to remove the selected project from the workspace. The project file will not be deleted and can be imported back into the workspace subsequently
- **Save Project** - Saves the state of the project and all its contained items to the underlying project file
- **Save Project As** - Saves the state of the project and all its contained items to a new project file
- **Online Help** - Displays this page in an external browser



Project Details Tab

The "Details" tab shown in the bottom-left shows the following values when the workspace node is selected in the navigation tree

- **File** (read-only): the selected projects project-file
- **Cache Definitions** : sets if definitions should be cached for this project (see Caching Definitions). This overrides the global "Cache WSDLs" setting under the WSDL Preferences settings tab

Next: Interfaces and Operations

1.3 Interfaces

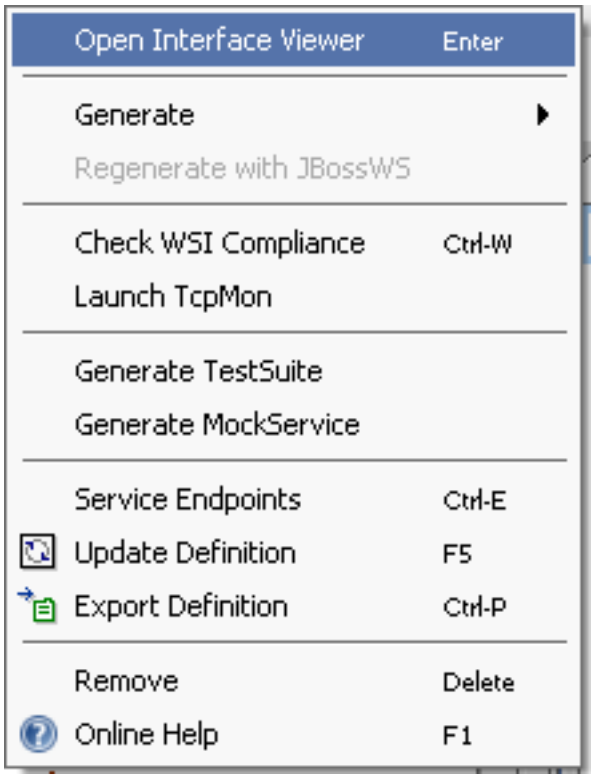
Interfaces

In the current implementation of the internal soapUI object model, Interfaces in soapUI correspond to SOAP/HTTP Bindings for a PortType. Once imported using one of the "Import WSDL from.." actions, the interfaces' operations and default requests will be visible in the navigator tree. The actual import process is below described in more detail .

Interface Actions

The following actions are available from the interface nodes' right-button menu:

- **Open Interface Viewer** : Opens the Interface Viewer window
- **Generate ->** : Invokes one of the integrated tools as described under Tool Integrations
- **Regenerate with JBossWS** : Enabled if this interface was generated using the Generate with JBossWS option in the project menu. Invoking this will regenerate the WSDL and update the definition accordingly
- **Check WS-I Compliance** : Invokes the WS-I Basic Profile validation tools for this interface
- **Launch TcpMon** : launches TcpMon for this interface as described in TcpMon Integration .
- **Generate TestSuite** : Generates a complete TestSuite for this interface (see below)
- **Generate MockService** : Generates a complete MockService for this interface (see below)
- **Service Endpoints** : Opens a dialog for managing the available service endpoints for this interface (see below)
- **Update Definition** : see below
- **Export Definition** : prompts for a local folder and saves the entire definition (including imported/included documents) to that folder. Import/Include statements in the exported files will be modified accordingly to maintain the definitions integrity
- **Remove** : prompts to remove the interface from its containing project. When removing an Interface, all test-requests associated with operations in that Interface will also be removed from their respective TestCases.
- **Online Help** - Displays this page in an external browser



Updating the Interface Definition

The "Update Definition" action prompts from where to reload the definition for this interface. Updating will synchronize any changes in the underlying Binding/PortType as best as possible:

- new operations are added as new Operations in the navigator
- operations not available any more can either be mapped to new ones (if they were renamed) or removed. When removing operations, all test-requests associated with them will also be removed from their respective TestCases.

This action is useful in the following situations:

- When the definition URL for a service/interface has changed (for example moved to another server)
- When the definition itself has changed (for example new operations have been added or the associated XML Schema has been modified

If there is no binding in the specified definition matching the previous binding for the selected interface (for example if the binding has been renamed) an error will be displayed.

(Be aware that this action does not update any requests to comply with altered XML Schema constructs, this must currently be performed manually)

Caching Definitions

By default (from version 1.6), soapUI caches the entire WSDL/XSD definition in the project file, mainly for the following reasons:

- **Change Independance** - validations, etc are not affected by external changes to the definition. This may be both good or bad
- **Work Offline** - one can work offline with request generation, validations, tests, etc.
- **Fast Access** - the soapUI attachments feature always requires access to the interface definition

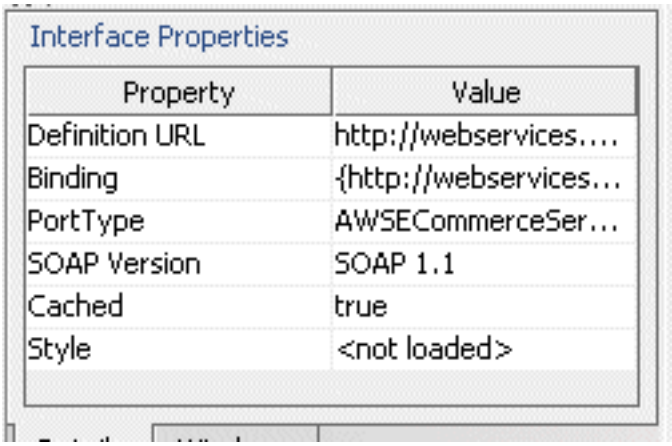
The major drawback is that the soapUI project-file will grow accordingly; if the definition is a complex one this may take considerable processing power. If you for this or some other reason would like to disable definition caching this can be done either globally in the soapUI Preferences dialog, or on a per-project basis in the Projects Detail Tab .

Turning this option off will clear currently cached defintions first when soapUI restarts. If you want to force the reloading of a WSDL definition (whether it is cached or not) use the Update Definition action described above.

Interface Details Tab

The "Details" tab shown in the bottom-left shows the following values when an Interface node is selected in the navigation tree

- **Definition URL** (read-only): the URL where the Interface is defined (ie from where it was imported)
- **Binding** (read-only): The name of the WSDL Binding for the interface in `{namespace}name` format
- **PortType** (read-only): The WSDL PortType that the Binding exposes
- **SOAP Version** : The SOAP Version to use for this interface; SOAP 1.1 or SOAP 1.2. This will affect both generated requests (namespaces and content) and according HTTP Headers
- **Cached** (read-only): Shows if this definition is currently cached in memory
- **Style** (read-only): Shows if this is a Document or RPC definition

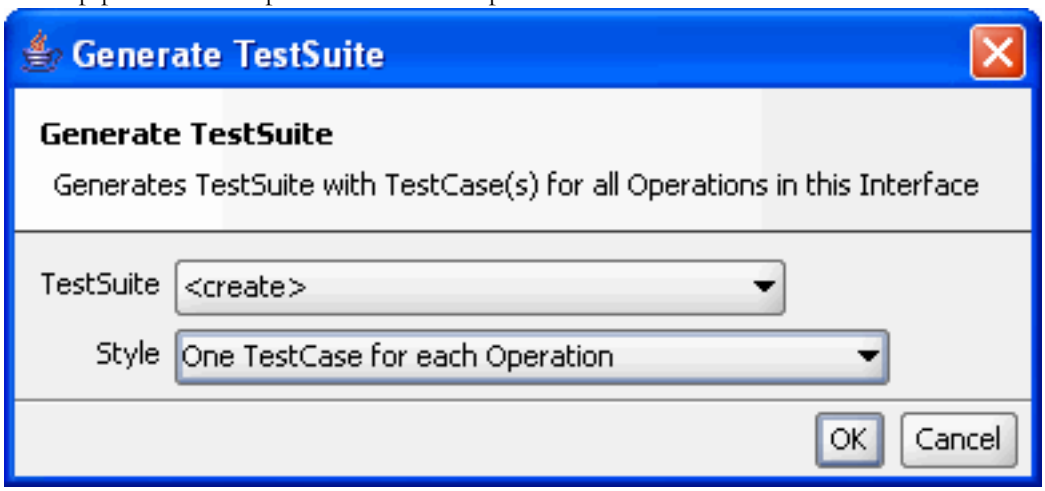


Generating TestSuites

Selecting the "Generate TestSuite" option from the Interface menu prompts to generate a complete TestSuite for the selected interface. There are two possible styles;

- One TestCase for each Operation - creates a TestSuite with one TestCase for each operation containing a default request for that operation.
- Single TestCase with one Request for each Operation - create just that!

Specifying "<create>" in the TestSuite Combo will prompt to create a new TestSuite, otherwise the generated TestCase(s) will be added to the selected one. After generating, either the TestSuite or TestCase desktop panel will be opened in the desktop.

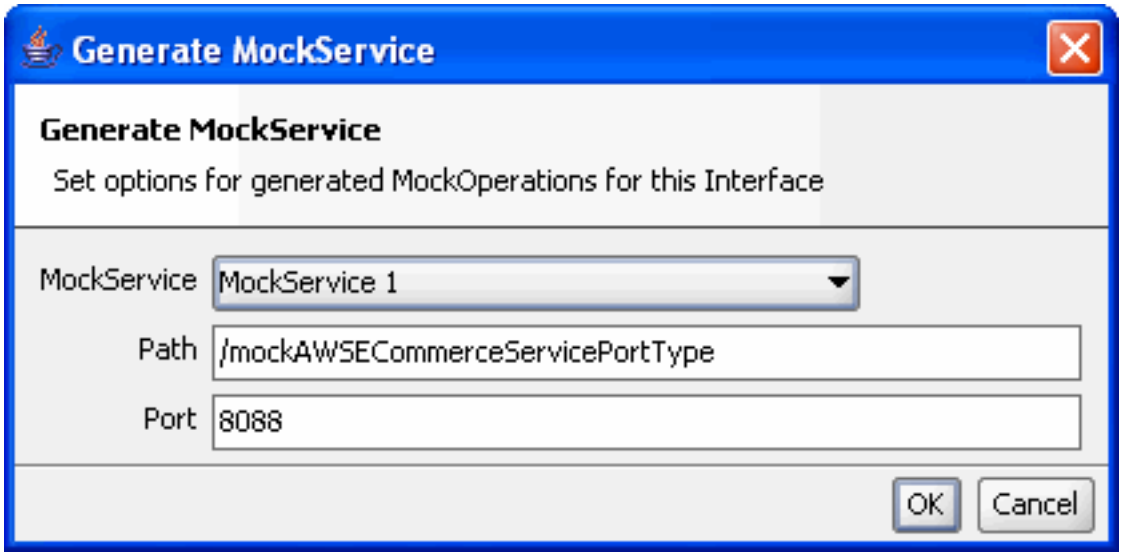


Generating MockServices

Selecting the "Generate MockService" option from the Interface menu prompts to generate a complete MockService for the selected interface.

After specifying the path/port and whether to create a new or add to an existing MockService, soapUI will generate a MockService containing a MockOperation for each Operation and in turn one default MockResponse for each MockOperation.

After generating, the MockService desktop panel is opened and ready to launch as described in Running a MockService



Service Endpoints

Selecting the "Service Endpoints" action from the interface context menu opens a dialog showing the available service endpoints for this interface. If the interface was imported from a WSDL definition, that

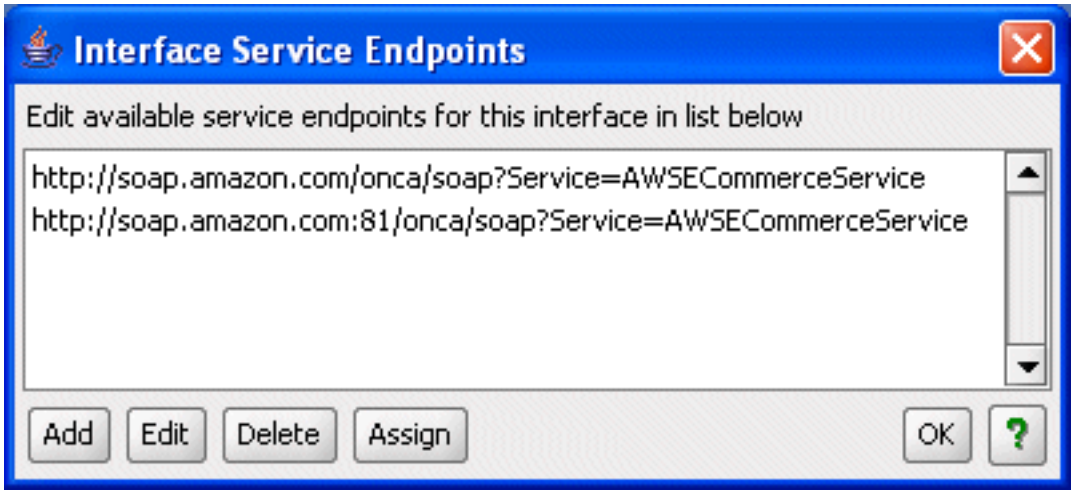
services ports URL will be available in this list, but you are free to add/edit/remove as many service endpoints as you want. If your endpoints are already URL-encoded, you should select the "Preencoded Endpoints" option in the Http Settings dialog to avoid that they get reencoded during requests. When deleting an endpoint, all requests that had that endpoint will have their endpoint set to null.

The **Assign** options prompts to assign the selected endpoint to requests, the following options are displayed:

- `- all requests -` : will assign the selected endpoint to all requests for operations in the current interface.
- `- all requests with no endpoint -` : will assign the selected endpoint to all requests for operations in the current interface that have no endpoint.
- `<endpoint>` : will assign the selected endpoint to all requests for operations in the current interface that have this endpoint.

Assigning endpoints with this option will **not** assign to TestRequests, use the "Set TestCase Endpoint" action in the TestCase Editor for this instead.

When closing the dialog with the "Ok" button, the available endpoints will be selectable in associated operations requests/test-requests editor panes.



WSDL Importing Details

When importing a WSDL using one of the "Import WSDL .." project actions, the import proceeds as follows;

- The specified WSDL is loaded and searched for defined WSDL Services. If the WSDL is secured soapUI will display an authentication dialog asking for relevant creadentials. Once the WSDL has been loaded, each of the included services' ports' binding is imported as a separate Interface into the soapUI project (provided that it is a SOAP 1.1/1.2 binding). If no services are found, the importer searches for standalone SOAP Bindings that are not bound to any service, these are imported as separate interfaces
- For each imported Binding, its PortTypes Operations are added as Operations and a default request is created (if specified) from the operations request schema.

- Either the name of the Binding or its PortType is used as the Interface name in soapUI, as controlled by the corresponding "Name with Binding" WSDL Setting .

The reasons for mapping Interfaces to WSDL Bindings and not WSDL services directly are:

- A WSDL Service can "implement" multiple bindings (through WSDL ports), each is imported as a separate Interface into soapUI
- A Binding can be "implemented" by several WSDL Services (for example on different servers or as a part of a more complex service interface). soapUI allows association of multiple service endpoints (see below) for a binding so it is easy to work with the same binding (ie Interface) without having to import each separately.
- Bindings can be defined without any referencing service, it should still be possible to work with these from soapUI

When importing the following XML-Schemas are added by default and need not be imported explicitely by the WSDL and/or its contained/imported/included types:

- http://www.w3.org/2004/08/xop/include : XOP includes used by MTOM
- http://www.w3.org/2004/11/xmlmime : xmime types used by MTOM
- http://www.w3.org/2005/05/xmlmime : xmime types used by MTOM
- http://www.w3.org/XML/1998/namespace : XML Namespaces
- http://ws-i.org/profiles/basic/1.1/xsd : swaref types used by SOAP with Attachments support
- http://www.w3.org/2001/XMLSchema : XML Schema itself

It is possible to specify a custom directory containing additional default schemas in the WSDL Settings preferences tab; "Schema Directory". Note that content-changes to an already specified directory will not be detected by soapUI and thus currently require a restart.

Next: Operations

1.3.1 Interface Viewer

Interface Viewer

The Interface viewer allows relatively easy navigation/inspection of the entire contract for an imported WSDL, including all imported/included wsdl/xsd files and their contained types, definitions, etc.

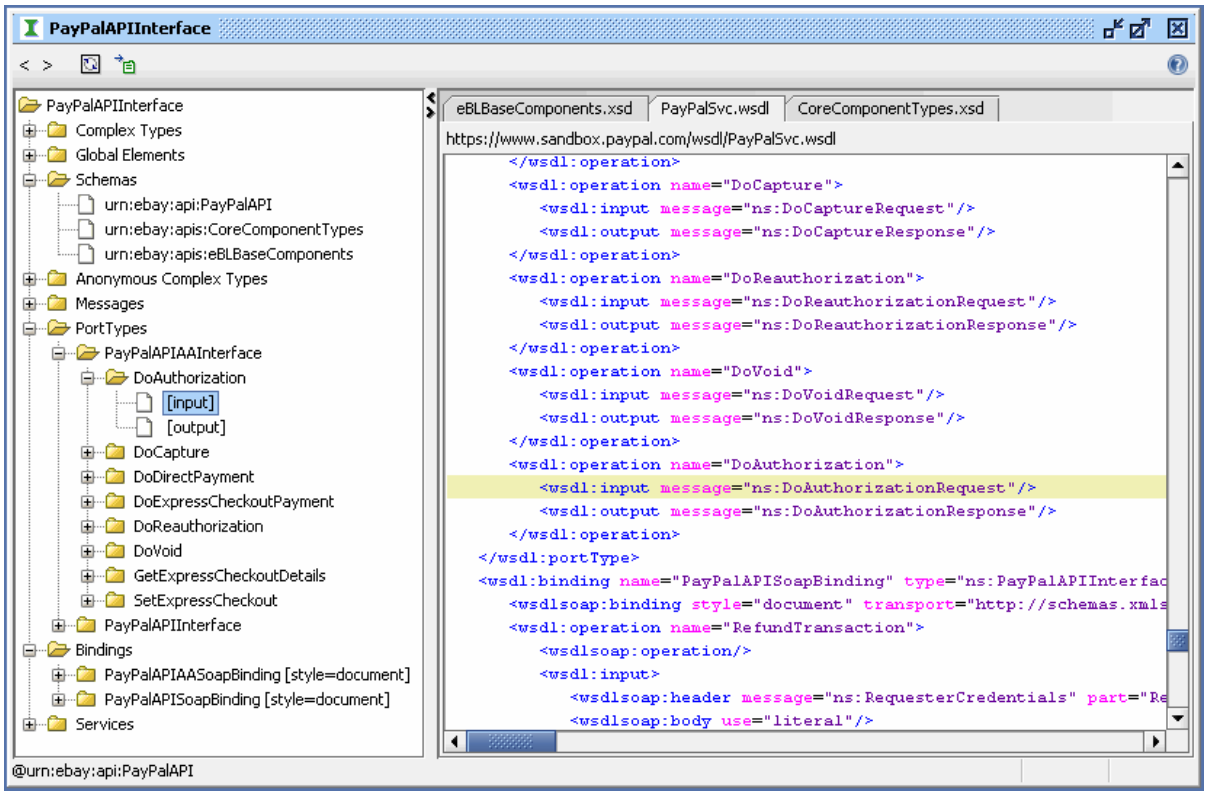
The navigation tree to the left shows contained elements in the contract, selecting a node will highlight that element in its containing file to the right, automatically switching tabs if neccessary. Elements referring to other elements in the contract are also double-clickable and will focus to the referenced element in the tree as follows;

Double-click on	will focus on (if available)
Global Element	its Complex Type
Message Part	its Complex Type or Global Element
Operation input/output	its Message
Binding Operation input/output	its corresponding Operation input/output
Port	its Binding Operation

The viewer toolbar has the following actions (left to right)

- Back - navigates to the previously selected node
- Forward - navigates to the next selected node (after navigating back..)
- Update Definition - updates the wsdl-contract as described under Updating the Interface Definition
- Export WSDL - prompts to export the entire wsdl and all imported/included files to a local folder. Import/Includes are replaced with valid relative ones.

The following screenshot shows the viewer for the paypal api;



Next:

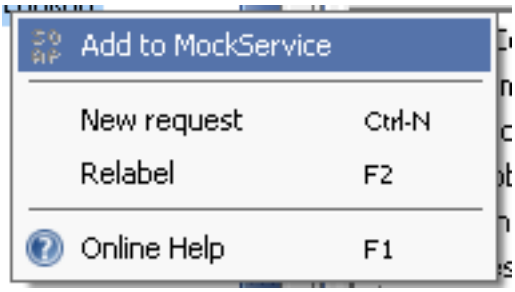
1.4 Operations

Operations

When adding an interface as described above, each of that interfaces operations is added as a node in the navigator named after the operation in the underlying PortType.

The following actions are available from the operation nodes' right-button menu:

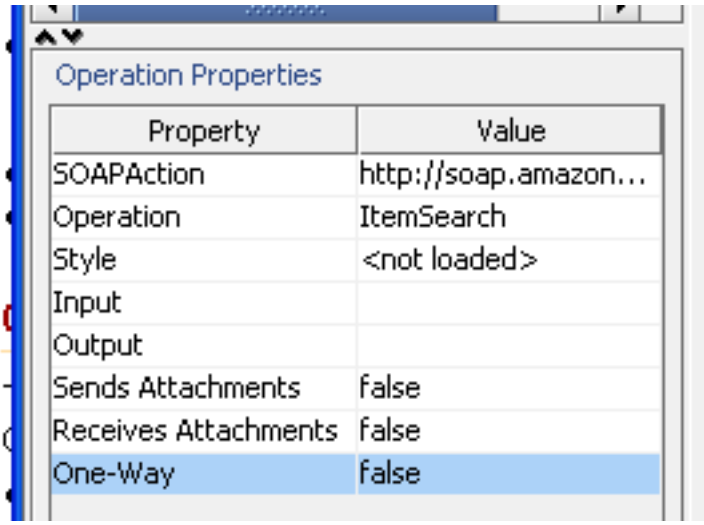
- **Add to MockService** - prompts to create a MockOperation for this operation in an existing or new MockService.
- **New Request** - creates a new Request for this operation from its wsdl/schema definition. soapUI will prompt for a name and ask if optional schema elements should be created.
- **Relabel** - changes the label of the operation in the navigator.
- **Online Help** - Displays this page in an external browser



Operation Details Tab

The "Details" tab shown in the bottom-left shows the following values when a Operation node is selected in the navigation tree

- **SOAPAction** (read-only): the SOAP Action specified for the operation in its SOAP/HTTP binding
- **Operation** (read-only): The name of the operation in the underlying PortType
- **Style** (read-only): Shows if this is a Document or RPC operation
- **Input** (read-only): The explicit name of the operations' input message (may be empty)
- **Output** (read-only): The explicit name of the operations' ouput message (may be empty)
- **Sends Attachments** (read-only): Signals that the operation is defined to send Mime Attachments
- **Receives Attachments** (read-only): Signals that the operation is defined to receive Mime Attachments
- **One-Way** (read-only): Shows that this is a one-way operation (ie request only)



Next: Working with Requests

1.5 Working with Requests

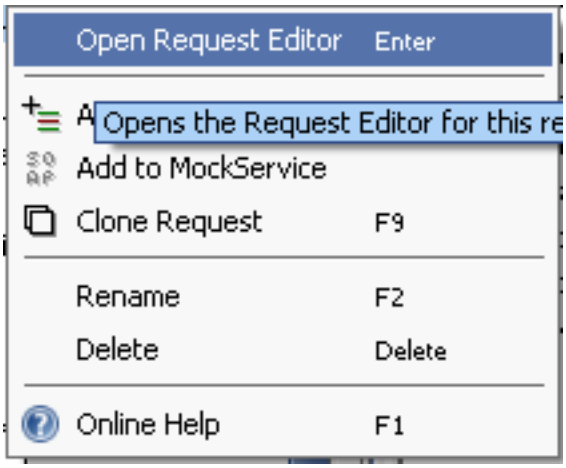
Working with Requests

An unlimited number of requests can be added to an operation in soapUI. Each request can have its own message content, endpoint, encoding, etc. Requests are mainly managed through the request-editor displayed when double clicking a request node in the navigator (described below)

Request Actions

The following actions are available from the request nodes' right-button menu:

- **Open Request Editor** - opens the Request Editor described below
- **Add to TestCase** - prompts to add the request to a TestCase. If no Testcase is available, soapUI will prompt to create both a TestSuite or TestCase if required
- **Add to MockService** - prompts to add the requests' operation to a MockService. If a response is available for the request, it will be used as the default MockResponse.
- **Clone Request** - prompts to clone this request, the cloned request will be added to the operation and its request-editor will be opened
- **Rename** - prompts to rename the request in the navigator
- **Delete** - prompts to remove the request from its operation
- **Online Help** - Displays this page in an external browser

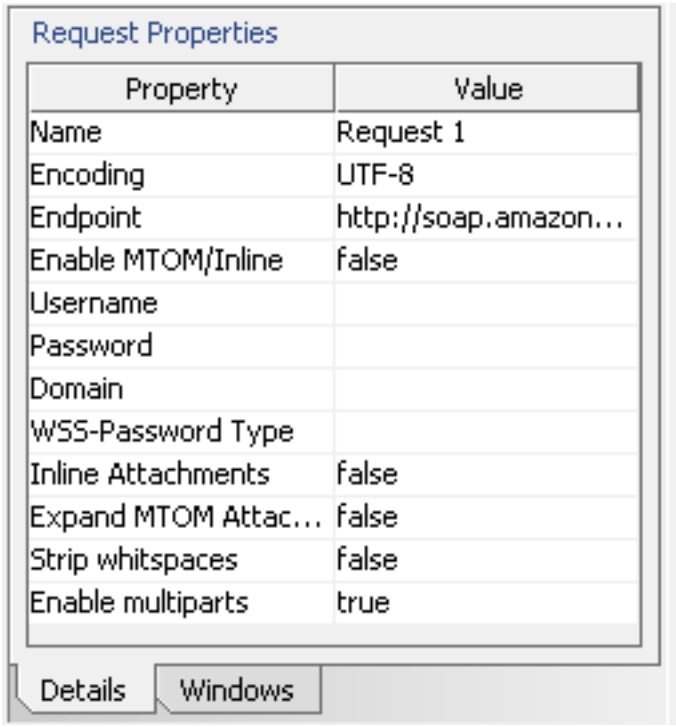


Request Details Tab

The "Details" tab shown in the bottom-left shows the following values when a Request node is selected in the navigation tree

- **Name** : the name of the request

- **Encoding** : The encoding used by the request
- **Endpoint** (read-only> : The endpoint for the request
- **Enable MTOM/Inline** : Enable MTOM-based transfer and inlining of binary data in the request (see Attachments)
- **Username** : The username to use if the request requires authentication
- **Password** : The password to use if the request requires authentication
- **Domain** : The domain to use if the request requires NTLM authentication)
- **WSS-Password Type** : Selects the type of WS-Security username/password token to dynamically add to each outgoing request. The above specified Username/Password values will be used fo the corresponding header values
- **Inline Response Attachments** : the entire HTTP response for a request (including MIME/XOP attachments, etc..) will be shown in the response editor instead of attachments being parsed and visible in the Response Attachments tab.
- **Expand MTOM Attachments** : received MTOM attachments will be inlined into the response message editor, allowing correct viewing and validation of the actual message.
- **Strip Whitespaces** : strips outgoing requests of any unneccesary whitespaces, required by some servers.
- **Enable Multipars** : bundles attachments assigned to the same attachment part into one multipart attachment instead of sending them as seperate attachments, see Multipart Attachments for more details.



The Request Editor

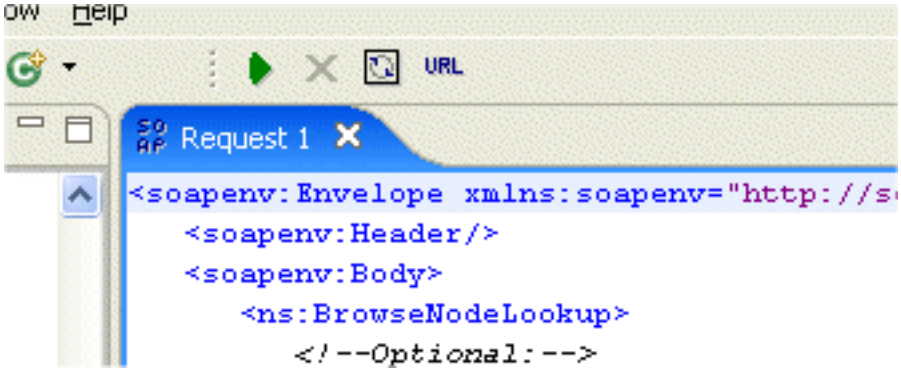
The request editor is opened by either double-clicking a request in the navigator or by selecting its **Open**

Request Editor popup menu item, and displays the current request/response messages together with a toolbar and a number of detail tabs as shown below in a configurable Editor Layout .

The following tabs are available at the bottom of each message editor;

- SOAP Request/Response : shows the message content editor
- Request/Response Attachments : show the associated request/response Attachments editors
- HTTP Headers : shows the request/response HTTP Header editor/viewer
- SSL Info (Response only) : shows detailed response SSL Details for the current response

Editor Toolbar



The request-editor toolbar has the following actions (from left to right)

- **Submit** - submits the request to the specified endpoint
- **Add to TestCase** - prompts to add the request to a TestCase, same as the request-menu action described above
- **Add to MockService** - prompts to add the requests operation to a MockService, same as the request-menu action described above
- **Recreate Request** - prompts to recreate the request xml from the operations input message definition. Also prompts if any values in the current request should be kept in the recreated request
- **Create Empty** - prompts to create an empty soap-request with no soap-body
- **Clone Request** - prompts to clone the request, same as the request-action described above. You will be prompted for a name for the new request and the new request will be placed after the cloned request.
- **Cancel Request** - cancels an ongoing request (disabled if no request is running)

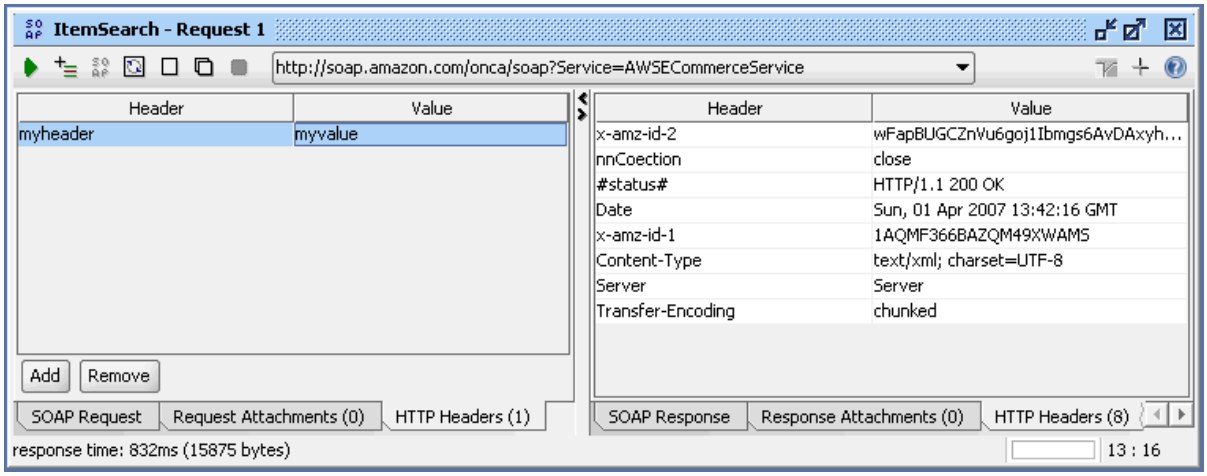
- **Endpoint Combo** - a combo-box listing the endpoints available for this requests operations' interface. The combobox also contains the following options (as shown in the image):
  - `[edit current...]` option for changing the current new endpoint directly
  - `[add new endpoint...]` option for adding a new endpoint directly
  - `[delete current...]` option for deleting the current endpoint directly
- **Tab Layout** - toggles between the tab and split request editor layouts as described below
- **Editor Orientation** - changes the orientation of the request/response editors between left/right and top/bottom
- **Online Help** - Displays this page in an external browser

Editor Layouts

The editor is available in 2 layout modes which can be toggled with the "Tab Layout" toolbar button (top right):

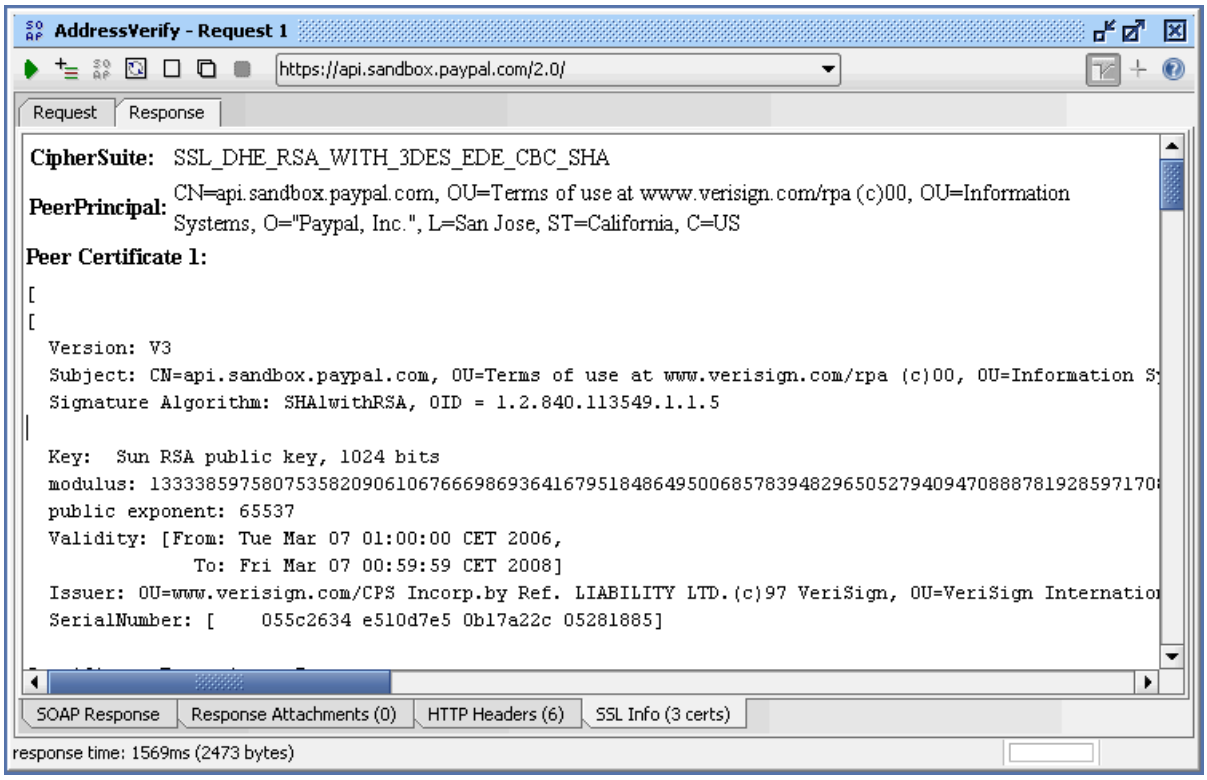
HTTP Headers

The HTTP Headers tab at the bottom of each message editor displays the configured/received HTTP headers for the corresponding request/response message. For request message headers, buttons for adding/removing HTTP headers to be sent with the request are available from a toolbar below the table.



SSL Details

The SSL Details tab at the bottom of the response editor displays a list of certificates that were involved in negotiating the current message exchange, the following screenshot shows this information for a default request to the PayPal API.



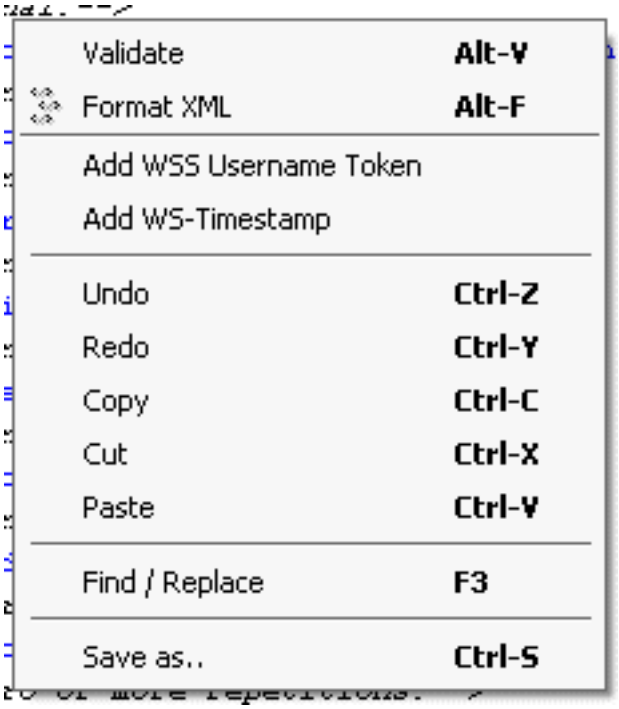
XML Source Editor

The request/response xml-editors support syntax-highlighting, undo/redo, copy/paste, etc. (soapUI Pro also adds a number of additional editor views to the default "XML"; "Outline" and "Form" , together with a number of Message Inspectors shown below the request/response message) The following actions are available from the request/response editors right-button menu:

- **Validate** - validates the editors' content against its message definition as described below
- **Format XML** - pretty-prints the xml in the editor
- **Add WSS-Username Token** - prompts to add a WSS-Username SOAP-Header to the request message using the username/password specified in the Request Details Tab
- **Add WS-Timestamp** - prompts to add a WS-Timestamp SOAP-Header to the request message
- **Undo/Redo/Copy/Cut/Paste** - standard editing functions
- **Find/Replace** - opens a standard File/Replace dialog
- **Save as** - prompts to save the editors content to a local file

Also, a number of custom keyboard actions are available to make your "everyday-soapui-editing" easier:

- Alt-Enter : submits the request to the specified endpoint
- Alt-X : cancels a running submit
- Alt-left/right arrow : moves between element values
- Shift-Tab : moves focus between request/response areas



Submitting requests

When submitting a request using either the submit toolbar button or the Alt-Enter editor action, the request message is sent to the selected endpoint using the specified encoding. Submittal is done in a background thread so any number of requests can be running simultaneously (an animated icon in the navigator shows that they are running).

During submittal, the request/response editors are made read-only and all actions except the cancel button in the toolbar are disabled. One can cancel an ongoing request either by the "Cancel" toolbar button or by pressing Alt-X in one of the editors. If you choose to close the editor window while a request is running, you will be prompted if you first want to cancel the ongoing request. Depending on the state of the request, its thread may actually continue in the background even though soapUI shows it as canceled, but from a usability point-of-view, you should never notice.

Once a response has been received, it will be displayed in the response area and a log message will be displayed in the soapUI log view and the request editors status bar showing the time taken and size of the received response. If you choose to close the request editor without canceling an ongoing request, the response will be shown in the response area when the request editor is made visible again.

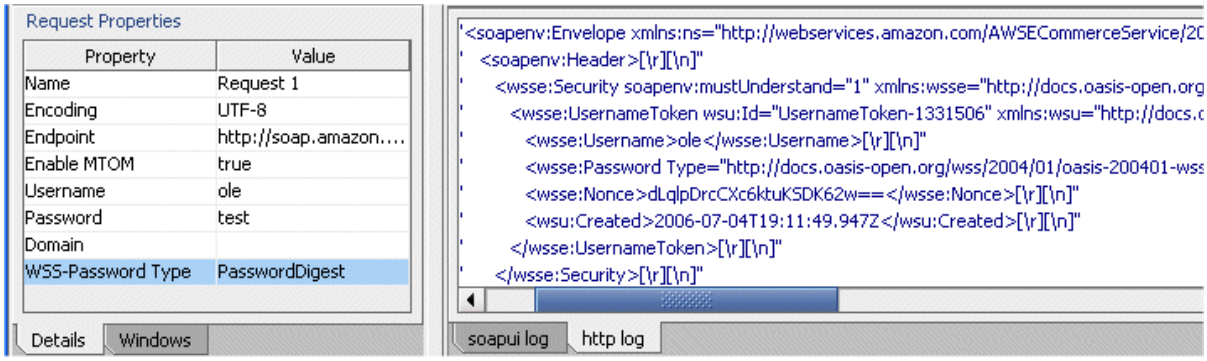
A HTTP Proxy can either be set by using the `http.proxyHost` and `http.proxyPort` system properties or through the soapUI Preferences dialog. If both are available, the system properties take precedence.

Authentication

If the service requires authentication, soapUI will attempt to authenticate using the specified username/password/domain values specified in the Request Details Tab. Currently supported authentication types are those supported by HttpClient, ie Basic, Digest and NTLM authentication. SSL

should work (=not tested!) if you use `https` urls and install JSSE accordingly.

If one of the WSS-Password Type options has been selected in the Request Details Tab, soapUI will automatically add the corresponding WS-Security headers to the outgoing message, as can be seen in the http log;



Message Validation

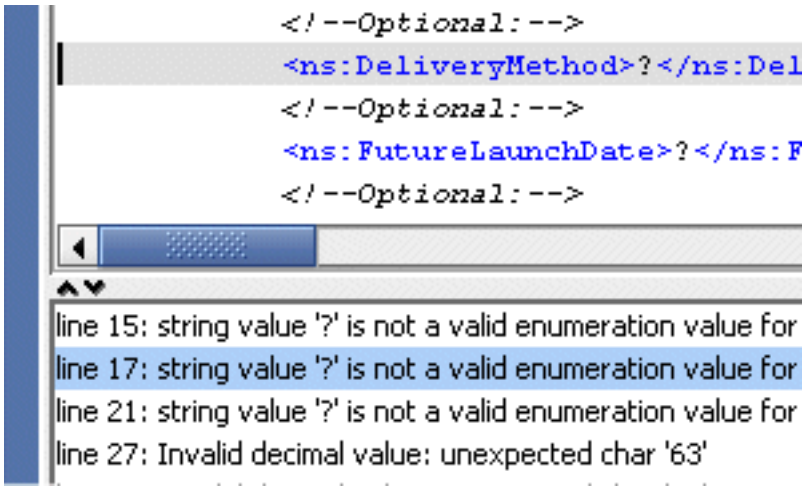
When pressing Alt-V in the request or the response area, the contained message is validated against the operations message definition;

1. The message body is validated against its schema definition
2. The SOAP envelope and structure is validated in accordance with the WSDL and SOAP schemas
3. If the message sends MIME attachments, their presence is validated in the Attachments Tab

Validation errors are shown in a list-box displayed under the editor. Double clicking an error will move the corresponding editor caret to the line of the error (if possible).

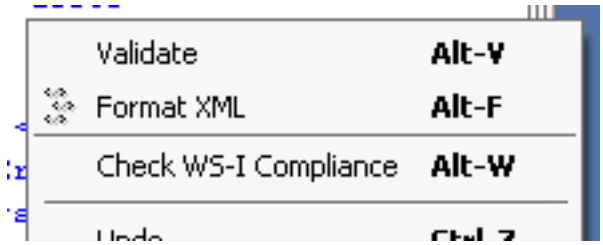
If the message is valid the error-list is hidden and an Ok-message is displayed.

Validation is currently only supported for literal-encoded messages (both rpc and document), SOAP-encoded messages are *not* supported.



The response message has a similar popup menu to the request-editor containing a "Check WS-I Compliance" option. Selecting this option (if the WS-I validation tools have been configured correctly) will validate the current request/response message as configured and display a WS-I Compliance report in a separate window as

described under WS-I Integrations .

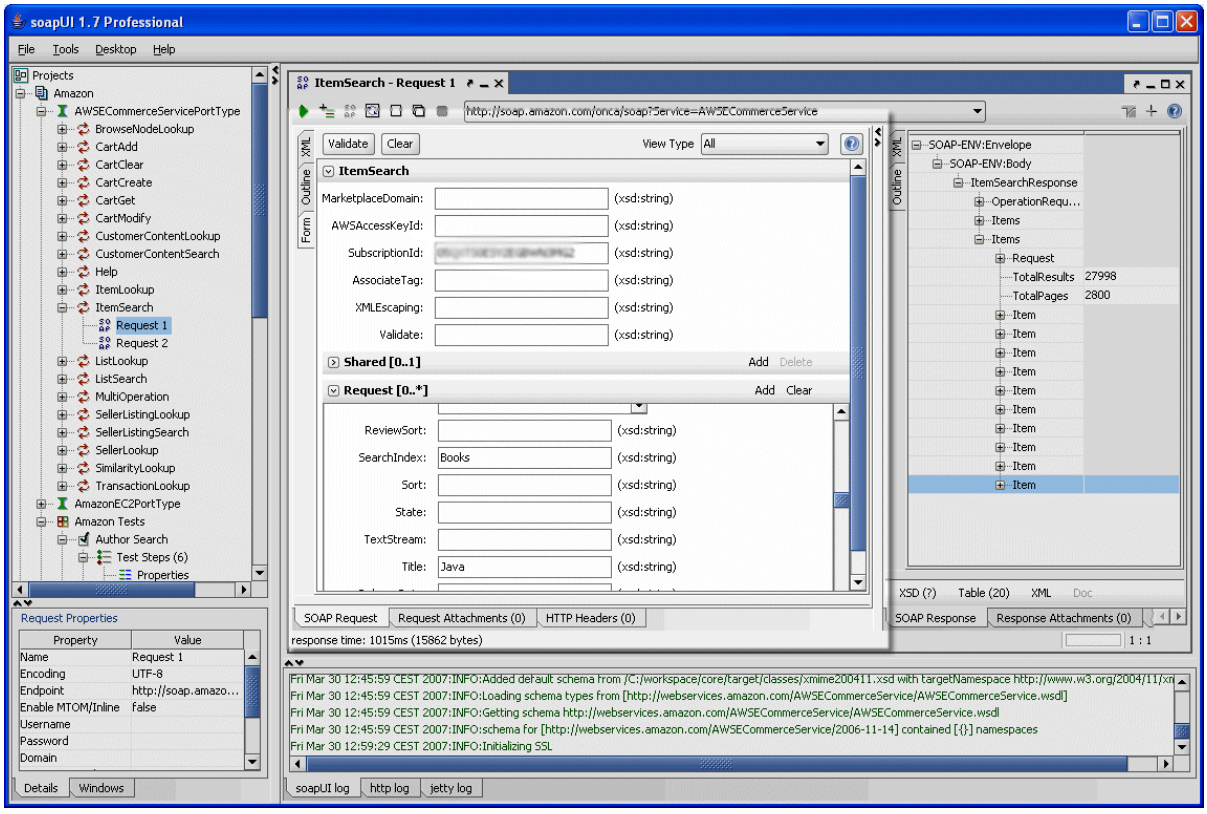


Next: Attachments and Inline Files

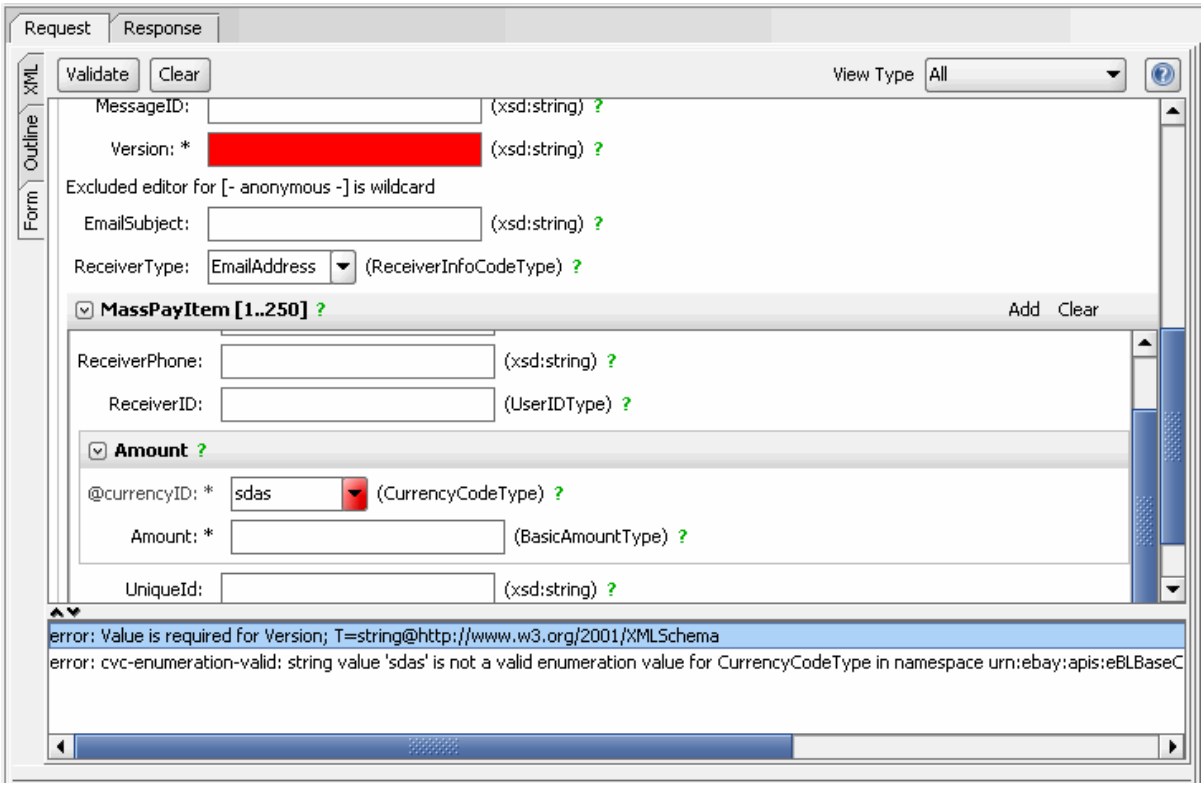
1.5.1 Form Editor

Form Editor

The form editor attempts to dynamically build a user-friendly input form from the underlying XML-Schema definition for the current request message. The following screenshot shows the form created for one of the paypal api calls;



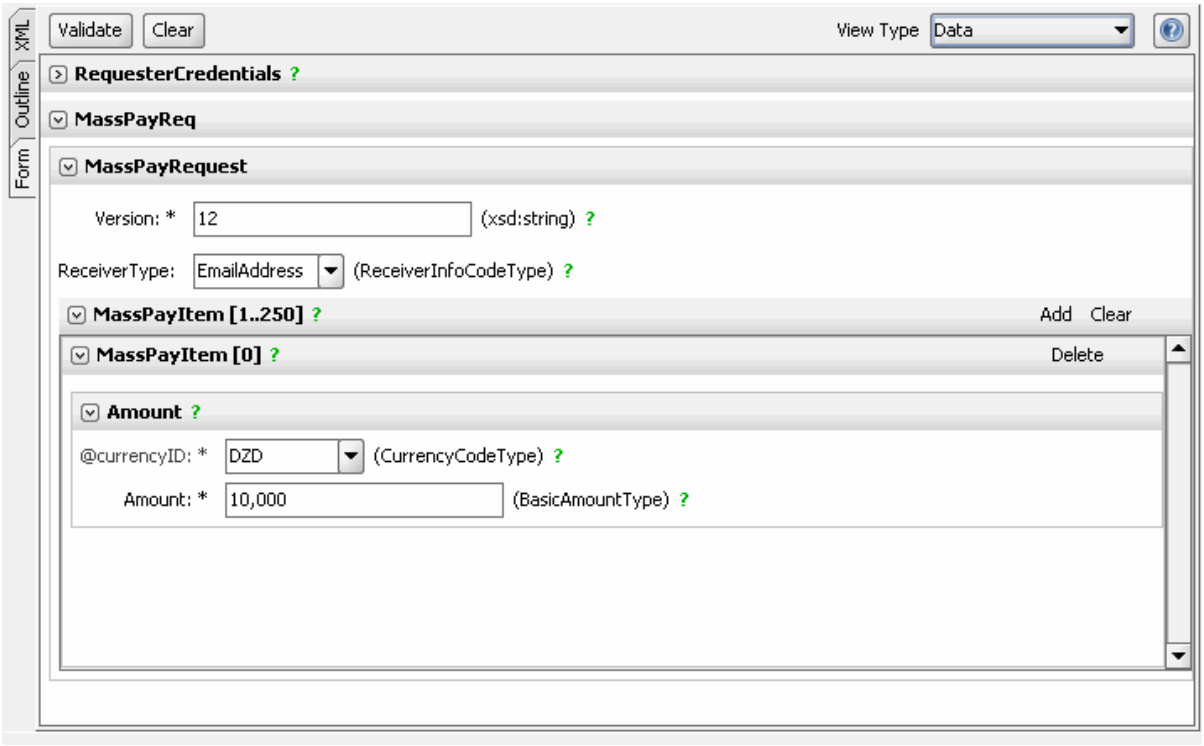
The toolbar contains buttons to Validate and Clear the current form. Clear will recreate an empty underlying request containg only required elements. Validating will show a clickable list of errors below the editor that can be used to move focus to the corresponding field that caused the error:



By default, the editor creates fields for all elements/attributes in the underlying schema for those structures that are available in the underlying message, which can both be usefull for inspection purposes but also annoying for testing purposes. Therefore, the toolbar contains a "View Type" combo which offers the following options:

- All - will display all fields (default)
- Mandatory - will display only mandatory fields
- Data - will display only fields that contain any data. This can come in handy during testing when preparing a request and wanting to hide remainin fields
- Mandatory and Data - will display all fields containing data and those that are mandatory

For example the follwing shows the above request with the "Data" value selected



This setting will be rememebered between editing sessions.

Schema Support

The Form Editor supports most commons XML-Schema constructs but may have trouble displaying/editing/validating more complex schemas. Therefore, it is advisable to initially double-check the XML generated by the editor in one of the other editors. Since it is our ambition to fully support XML Schema, please let us know if you have schema constructs that are giving you (or the editor) trouble..

The following constructs should currently work fine:

- Standard elements; if maxOccurs is more than 1 a listbox with add/edit/remove buttons will be shown
- Enumerations/Booleans will be displayed as combo-boxes
- Binary fields are linked to the attachment-support, a "Browse.." button will be displayed next to the input field allowing browsing and automatic associated of a file with a binary field
- Attributes are displayed with a @ before the field name
- Standard Sequence/Choice/All complex types - Choice will be displayed with a combo-box in the section header from which the desired choice should be selected.
- Multiple occurrences of an complex element will be displayed as a list with add/remove buttons in the top right. Each entry in the list will be displayed with a list index next to the section name, for example in the last screenshot above, the MassPayItem section can contain 1-250 MassPayItems, currently it contains only one (seen with an index of [0]).
- AbstractTypes will display a drop-down of global derived types from the schema, allowing selection

of which derived type to use

A '*' after a field name denotes that it is required. All fields can show a small green question-mark which will show documentation information from the associated type in a tooltip.

Next: Outline Editor

1.5.2 Outline Editor

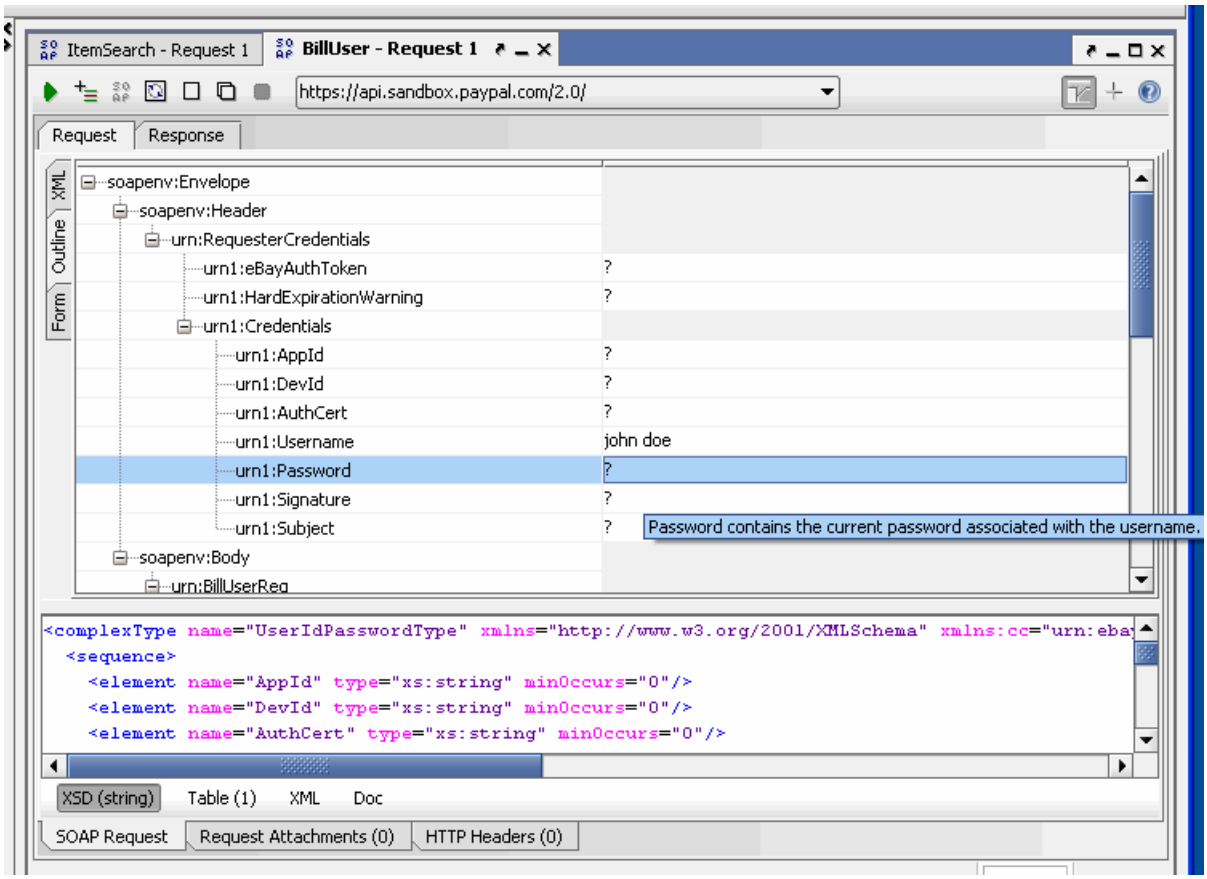
Outline Editor

The soapUI Pro Outline Editor is available for all request/response messages and shows a tree-view of the current message content. The left column of the editor shows the tree structure and the right column shows editable node values. Grayed node-values are not editable due to unavailability and/or schema restrictions. Node values are editable by selecting/editing as "usual".

Navigation with keyboard is as follows:

- DOWN - moves down one row and expands new row if possible
- ALT+DOWN - moves down to the next sibling node, skipping any child nodes
- UP - moves up one row
- ALT+UP - moves up one sibling, skipping any intermediate child nodes
- ALT+LEFT - moves to the current nodes parent node
- ALT+RIGHT - moves to and expands the current nodes first child node if available
- ALT+C - collapses current node if possible (ie hides all child nodes)
- ALT+E - expands current node if possible (ie shows all child nodes)
- ALT+SHIFT+C - collapses all child nodes if possible
- ALT+SHIFT+E - expands all child nodes if possible

If the underlying schema definition for a specific node has documentation annotations, these will be displayed in the tooltip as shown in the screen-shot below.



A number of right-button actions are available for creation of XPath assertions and Property-Transfers/Expansion when editing Mock Response and Test Request steps.

Next:

1.5.3 Message Inspectors

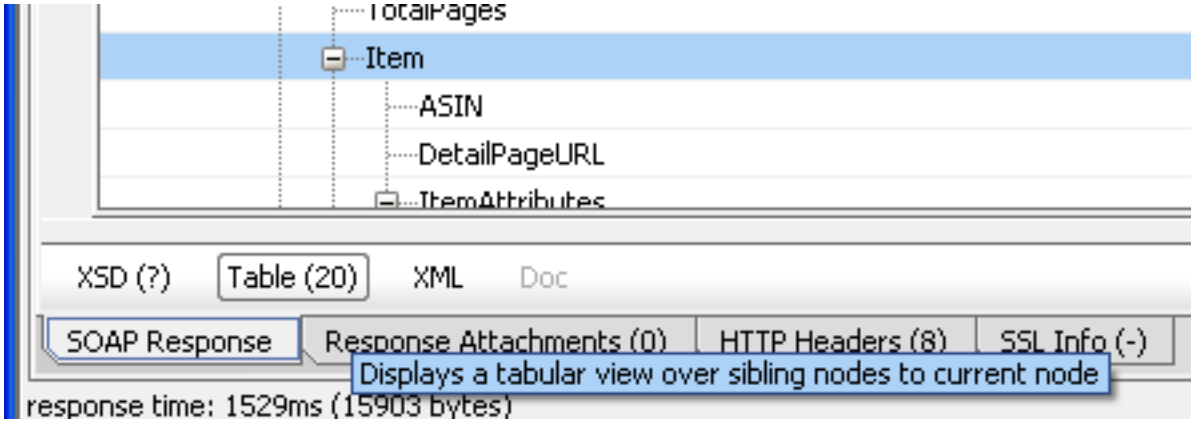
Message Inspectors

soapUI Pro introduces a number of context-sensitive message inspectors that are available for both the standard XML Source editor and the Outline Editor displaying context sensitive details for the currently selected node. The inspectors are available as a number of tabs below the message editor and can be shown/hidden one at a time by clicking their corresponding tabs.

The following inspectors are currently available:

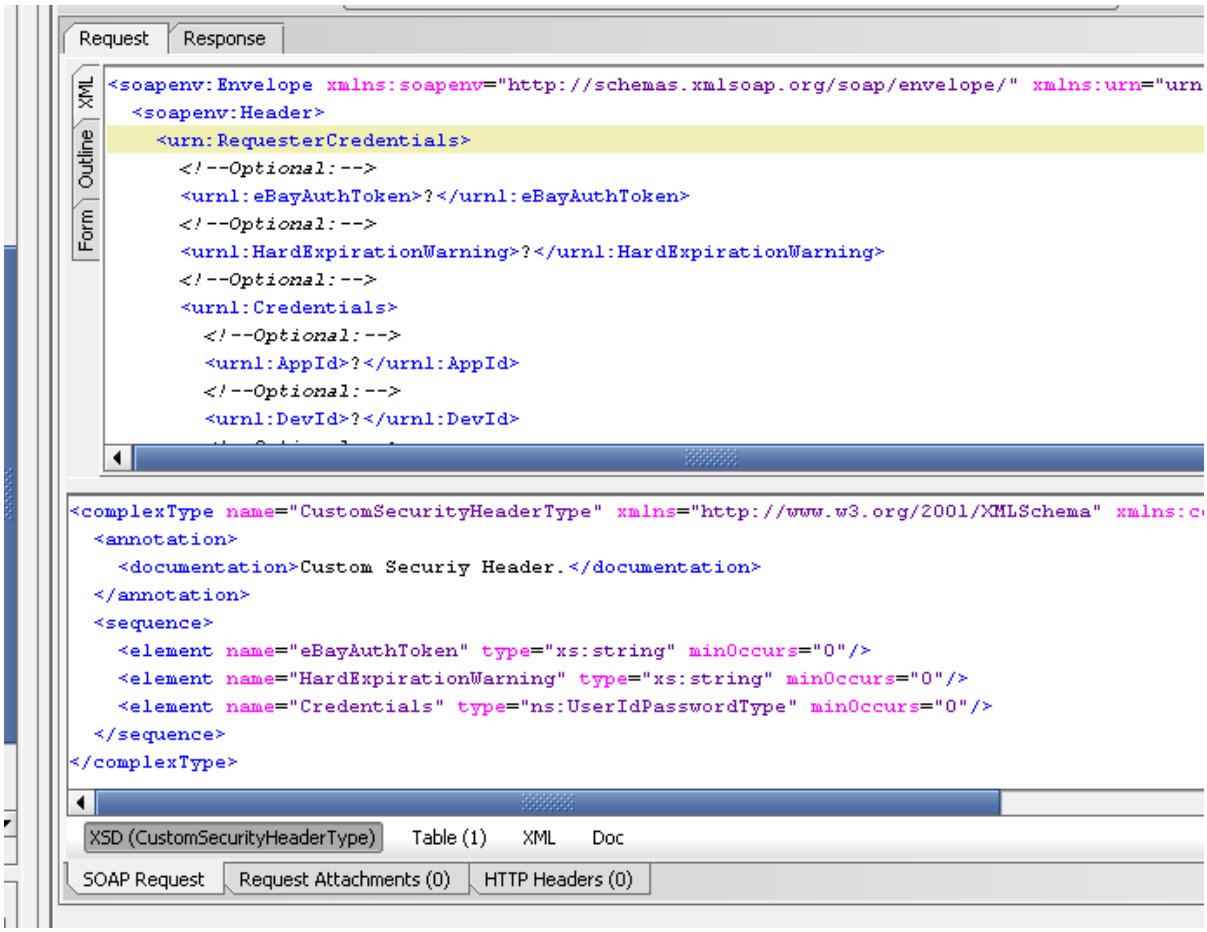
- XSD - displays XML Schema information for the selected node
- Table - displays a tabular view of the current nodes' siblings and children
- XML - displays XML fragment and XPath information for the current node
- Doc - displays schema documentation annotations if available

Each tab is enabled/disabled if corresponding content is available.



XSD/XML Schema Inspector

The XML Schema Inspector displays the current nodes corresponding XML Schema definition if possible. For simple types (for example in a sequence), the containing complex type is displayed instead. The tab title will contain the name of the current type if available. The following screenshot shows schema information for the selected RequesterCredentials SOAP Header in a request to the PayPal API



Note that the inspector requires the message to comply with the currently imported WSDL file, for example if a received response has an updated namespace, the inspector will not be able to display schema information.

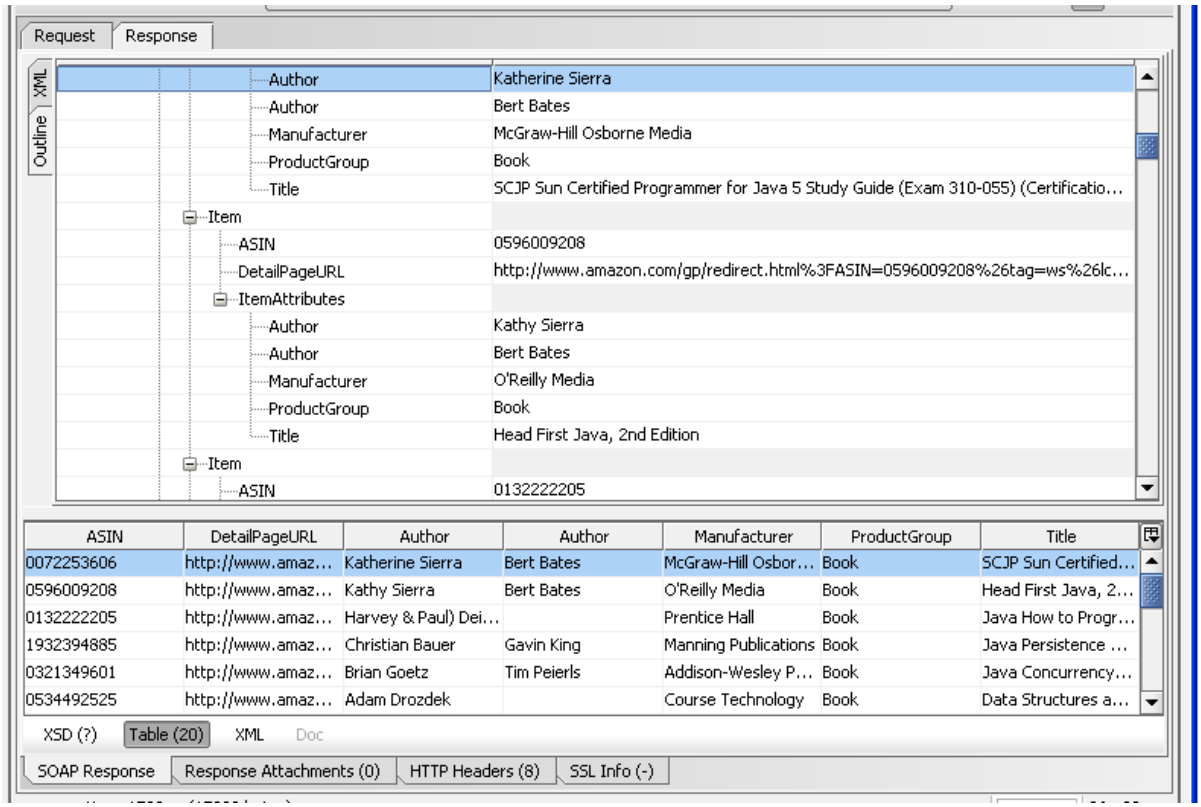
Table Layout Inspector

The Table Inspector builds a tabular view of the current selected node and its siblings and children;

- siblings correspond to rows in the table
- children correspond to columns. For each sibling the same children that are available for the selected node are displayed

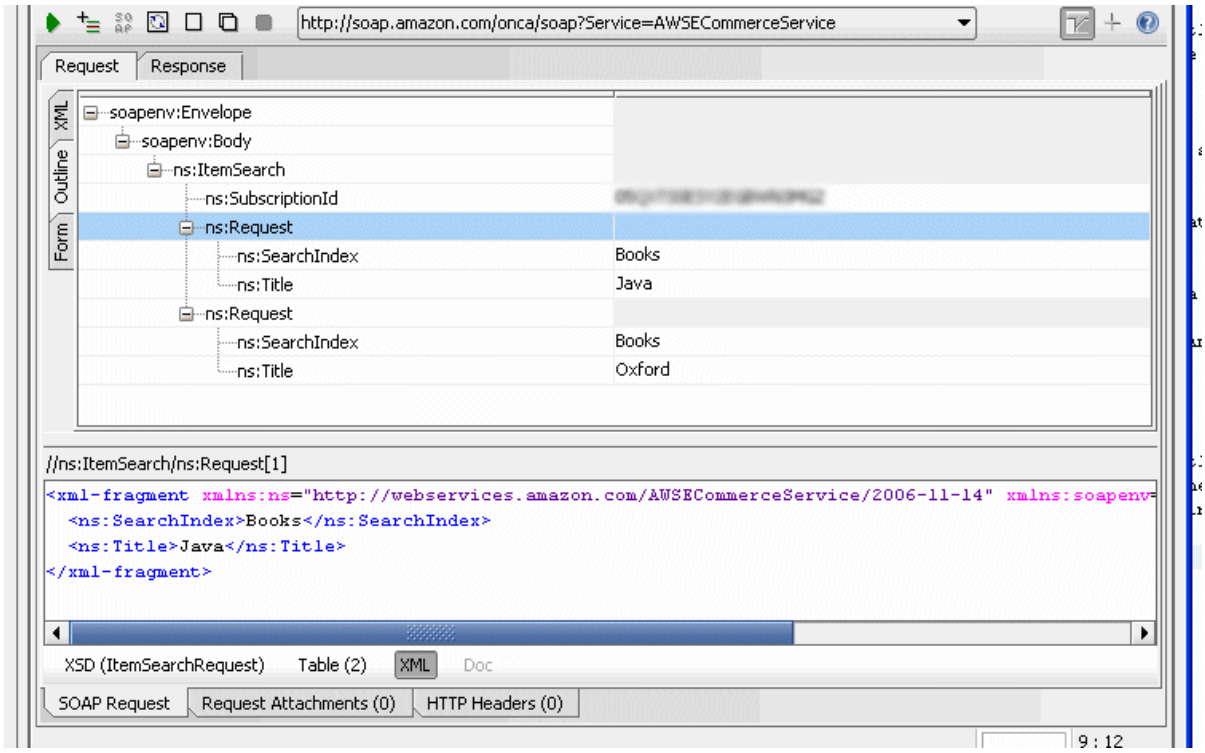
Once selected, it is possible to navigate "both ways", ie clicking in a new row in the table moves focus in the current editor to the correspondning node. The tab title displays the currently number of available rows.

The following screenshot shows a tabular view of the items found in a Amazon search result;



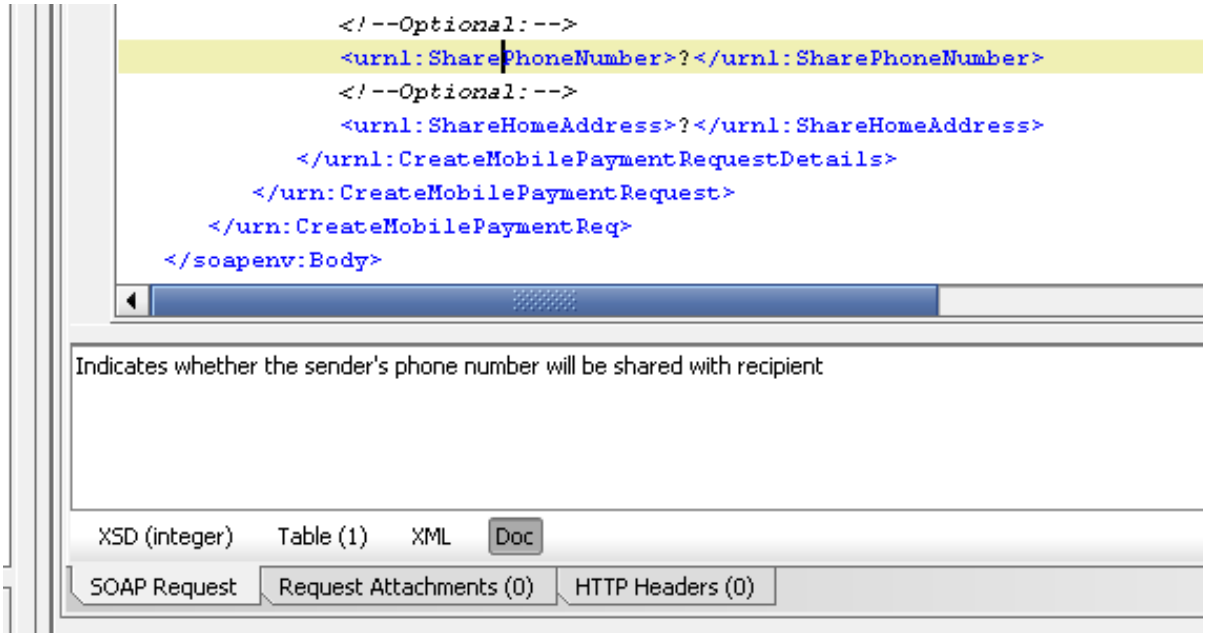
XML - XML Fragment/XPath Inspector

The XML Fragment/XPath inspector shows the XML fragment for the currently selected node, which can be usefull when using the Outline Editor to navigate a message. It also displays a the currently selected nodes XPath expression, which can be selected and copies to the clipboard with Ctrl-C. The following screenshot shows an XML fragment for a node in found in a Amazon search result;



Doc - XML Schema Documentation Inspector

The Doc inspector displays any schema annotations for the currently selected type. These are also available as tooltips in the Outline Editor but can be usefull in the Source Editor as shown in the following screenshot for a PayPal API request;



Next:

1.5.4 Attachments/Inline Files

Working with Attachments / Inline Files

soapUI supports the following technologies for working with files and attachments:

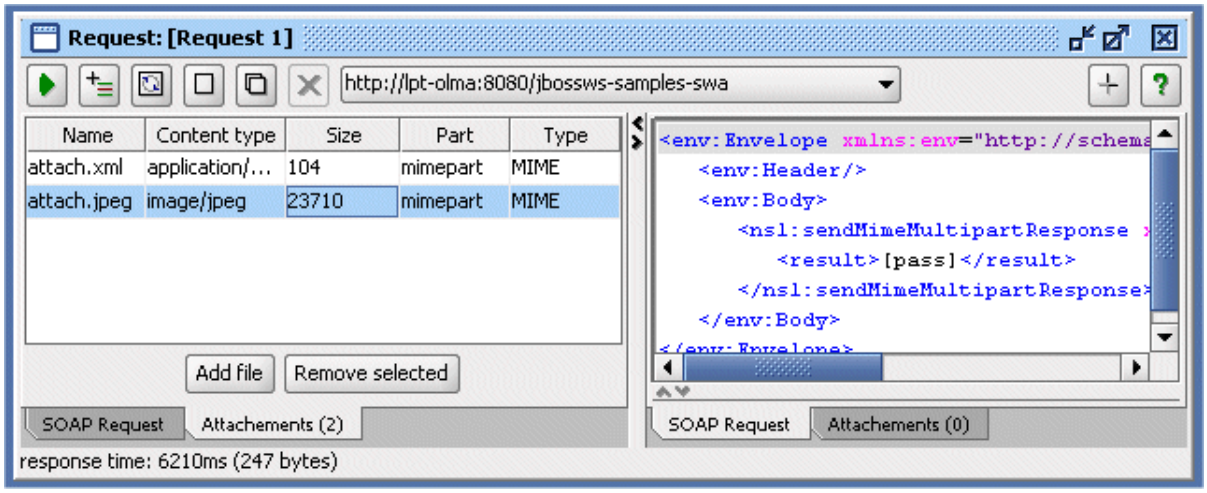
- MTOM - a technology for optimized transfer of binary data in SOAP Messages
- SOAP with Attachments in accordance with Attachments Profile - a MIME-based attachment mechanism for the SOAP/HTTP binding
- Inline files for binary content - soapui-specific functionality for simplifying handling of binary message content

(since the industry (for now) seems to be moving towards MTOM, we currently have no plans for supporting any other attachment technology, for example DIME)

Both MTOM and Inlining of files require internal processing and can be disabled for better performance in the Request Details Tab . Also, when disabling this feature, soapUI will no longer be required to load the WSDL Definition (either cached or remote) before sending a request.

The Attachments Tab

The request/response editors contain tabs for adding/accessing attachments for the current request/response messages. For request messages, this tab contains a table of files that have been attached to the current request. For the response, the table contains attachments in the response. The tabs are always available since they can always be used for both MTOM attachments and File Inlining as described below



The attachment-table contains the following columns:

- **Name** (read-only) - The filename of the attachment
- **Content-Type** - The mime content-type of the attachment
- **Size** (read-only) - The size of the attached file
- **Part** - The part this attachment should be associated with (see below)
- **Type** (read-only) - The type of attachment (depends on the attachments Part, see below)
- **ContentID** - Allows overriding of the content ID taken from the mime-part definition in the correspondning WSDL

Double-clicking an attachment in either message will attempt to save the attachment to a temporary file (unless it is relative as described below) and open that file in the systems web-browser.

Attaching files

A file can be attached to a request message either by dragging it from the file-system into the request attachments table or by selecting the "Add File" button which will open a file-selection dialog. When adding a file, soapUI will prompt if the file should be cached by soapUI or added as a file reference;

- Caching the attachment will result in soapUI creating a local copy of the attachment in the attachment folder configured in the soapUI Preferences dialog (see more below)
- Not caching the attachment will result in soapUI storing only the absolut file path to the selected file in the project

Once attached, a file must be assigned to a corresponding message part in the tables Part column:

- If the operation is defined to use MIME attachmentes in its WSDL, the dropdown displayed in the Part column will contain all defined mime-parts. Just select which part the attachment should be assigned to.
- If the request message contains SOAP-with-Attachments swaRef elements containing a "cid:XXX" value, that XXX value will be available in the part-dropdown allowing assocation of swaRefs with attached files.
- If the request message contains binary (base64 or hex) elements which contain a "cid:XXX" value, that XXX value will be available in the part-dropdown allowing assocation of binary elements with attached files.

(all/any of these may be possible depending on message definition and content)

Selecting an attachments part will also update the attachments type column, which will display one of MIME (for a mime attachment), XOP (for a MTOM attachment), CONTENT (for an inline file attachment), SWAREF (for a swaRef attachment), UNKNOWN (for unassigned attachments).

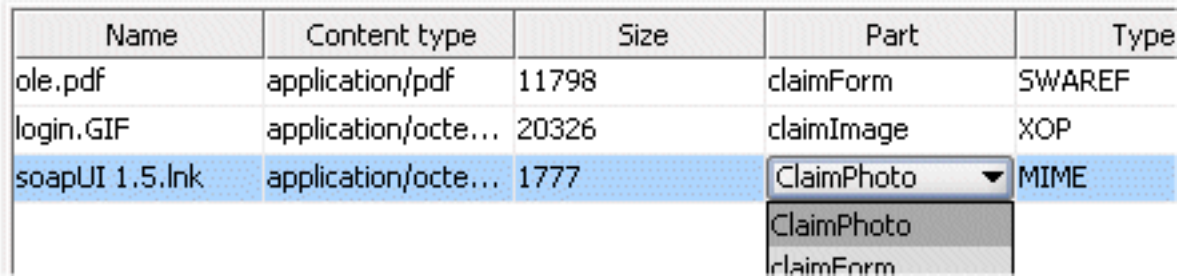
MIME Attachments

MIME attachments have been defined in the operations WSDL-defined as described in the SOAP with Attachments specification. When adding a file it must be associated with its corresponding Mime-Part as described above.For example, if the request operations' binding defines the following:

```
<wsdl:definitions xmlns:ref="http://ws-i.org/profiles/basic/1.1/xsd"
...
```

```
<wsdl:operation name="SendClaim">
          <soapbind:operation soapAction="http://example.com/soapaction"/>
          <wsdl:input>
              <mime:multipartRelated>
                  <mime:part>
                      <soapbind:body use="literal"
                                     parts="ClaimDetail"
                                     namespace="http://example.com/mimetypes"/>
                  </mime:part>
                  <mime:part>
                      <mime:content part="ClaimPhoto"
                                    type="image/jpeg"/>
                  </mime:part>
              </mime:multipartRelated>
          </wsdl:input>
          <wsdl:output>
              <soapbind:body use="literal"
                             namespace="http://example.com/mimetypes"/>
          </wsdl:output>
      </wsdl:operation>
```

This would result in the following option in the Part combo-box:



swaRef Attachments

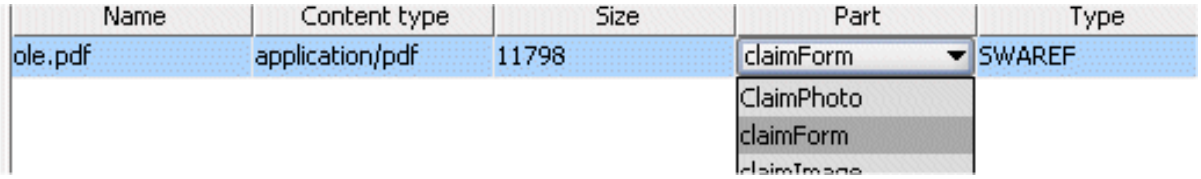
The WS-I Attachments Profile defines a special `swaRef` datatype which can be used in a schema for designating attached binary content. In soapUI, elements of the swaRef type should contain a value in the form of `cid:somename`, which will result in "somename" being available in the attachment part dropdown. For example, if the request operations' schema defines the following:

```
<wsdl:definitions xmlns:ref="http://ws-i.org/profiles/basic/1.1/xsd"
...
  <wsdl:types>
  ...
         <xsd:element name="ClaimForm" type="ref:swaRef"/>
```

and the corrsponding message would contain for example:

```
<ClaimForm>cid:claimForm</ClaimForm>
```

this would result in the following option in the Part combo-box:



MTOM Attachments

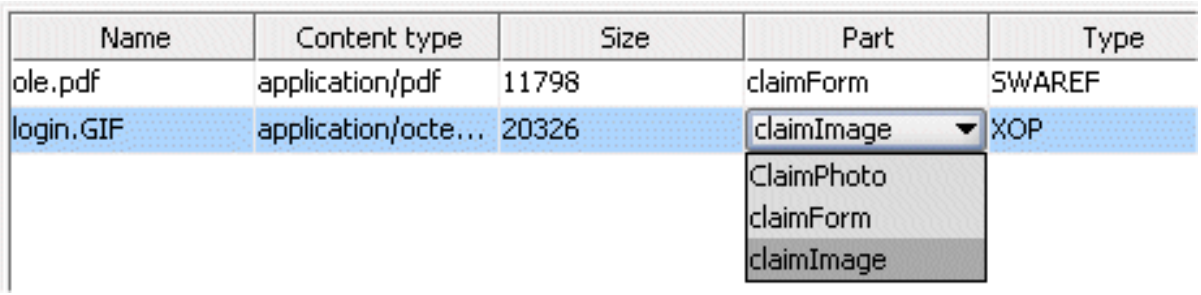
The MTOM specification describes optimized transport of binary data in SOAP-messages. The corresponding data-type used for the binary-datas message element must be one of the xmime datatypes, for example if the request operations' schema defines the following:

```
<wsdl:definitions xmlns:xmime="http://www.w3.org/2005/05/xmlmime
...
  <wsdl:types>
  ...
         <xsd:element name="ClaimImage" type="xmime:base64Binary"/>
```

and the corrsponding message would contain for example:

```
<ClaimImage xm:contentType="image/gif">cid:claimImage</ClaimImage>
```

this would result in the following option in the Part combo-box:



Since not all servers support MTOM, you can disable this feature in the Request Details Tab which will result in the file-data being inlined in the outgoing message in the same way as described for Inline Files below.

Anonymous Attachments

Sometimes you may want/need to attach files although this is not specified in the WSDL, for this soapUI allows "anonymous" attachments by selecting the "<anyonymous>" part from the part dropdown. For these you will also need to set a content id in the ContentID column if required.

Inline files

soapUI also has the option of "inlining" binary files into request messages when they are sent; if a message contains base64Binary or hexBinary data elements, these can be "filled" using an attached file in

2 ways:

1. By specifying a value of `cid:somename` and associating an attached file with the "somename" part in the same way as in the swaRef/MTOM examples above
2. By specifying the value `file:filepath` where filepath points to a local file

In both cases, the associated file will be inlined into the message with either base64 or hex encoding. For example if the request operations' schema defines the following:

```
<wsdl:definitions xmlns:xmime="http://www.w3.org/2005/05/xmlmime
...
  <wsdl:types>
  ...
        <xsd:element name="ClaimData" type="xsd:hexBinary"/>
```

and the corrsponding message would contain for example:

```
<ClaimData>file:c:\data\mydata.zip</ClaimData>
```

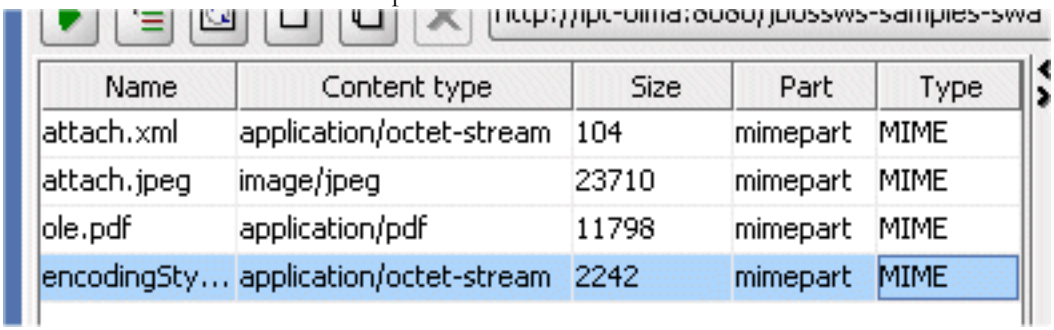
The c:\data\mydata.zip file would be inlined into the outgoing message using hex encoding.

Attachment Caching

When adding an attachment as described above, soapUI gives an option to cache the attachment locally. If selected, the file will be saved in the soapUI project file (compressed), otherwise the project file will keep an absolute path reference to the selected file.

Multipart Attachments

For anonymous, MIME, swaRef and MTOM attachments it is possible to associate multiple files with the same attachment part. This will result in soapUI first building a MIME Multipart message containing the associated files and then attaching that multipart-message to the corresponding part definition using the "multipart/mixed" content-type, a feature that can be turned off in the Requests Details tab ("Enable Multiparts"). If you need to set the ContentID for both the multipart attachment and the contained attachments, set the first attachments ContentID to "<attachment contentid> <multipart contentid>", ie a space-seperated string where the first token is the id of the attachment inside the multipart and the second token is the id of the multipart attachment itself.



The resulting multipart part can be seen in the httplog:

```
8 : DEBUG : >> "  </soapenv:Body>[\n]"
8 : DEBUG : >> "</soapenv:Envelope>[\r][\n]"
8 : DEBUG : >> "------=_Part_0_10514061.1152742987251[\r][\n]"
8 : DEBUG : >> "Content-Type: multipart/mixed; boundary="----=_Part_1_817174.1152742987301"[\r][\n]"
8 : DEBUG : >> "Content-Transfer-Encoding: binary[\r][\n]"
8 : DEBUG : >> "Content-ID: <mimepart=172513635203915@soapui.org>[\r][\n]"
8 : DEBUG : >> "[\r][\n]"
8 : DEBUG : >> "------=_Part_1_817174.1152742987301[\r][\n]"
8 : DEBUG : >> "Content-ID: <mimepart=172513633623826@soapui.org>[\r][\n]"
8 : DEBUG : >> "[\r][\n]"
8 : DEBUG : >> "<?xml version="1.0" encoding="ISO-8859-1"?>[\r][\n]"
```

Response Attachments

The response attachments table lists all attachment available in the response with their corresponding name, content-type, size, etc. Double-click an attachment to view it with the local web-browser.

Next: Functional Testing

1.6 Functional Testing

Functional Testing

soapUI supports functional testing of Web Services by providing a TestCase metaphor where a number of TestSteps can be executed in sequence. There are currently six types of TestSteps available providing for rich testing possibilities. TestCases are further organized into TestSuites of which an arbitrary number can be created within each project.

Functional testing in soapUI can be used for a variety of purposes:

- Unit testing : validate that each Web Service operation functions as stated
- Compliance testing : validate that the Web Service returns results compliant with its definition
- Process testing : validate that a sequence of web service invocations fullfil a required business process
- Data Driven testing : validate that any of the above works as required with data input from external sources (for example a database or another web service).

Next: TestSuites

1.6.1 Getting Started

Getting Started with Functional Testing

Now that you have imported some WSDLs and tries some requests its time to create your first TestCase. In soapUI, functional testing can be used to validate required functionality both for each web service invocation on its own (="unit testing") or for a number of requests sequence (="integration testing"). Further, you can add scripts (using the Groovy Language) that can enhance your tests in any way you want, for example interact with a database or perform complex test-flow logic.

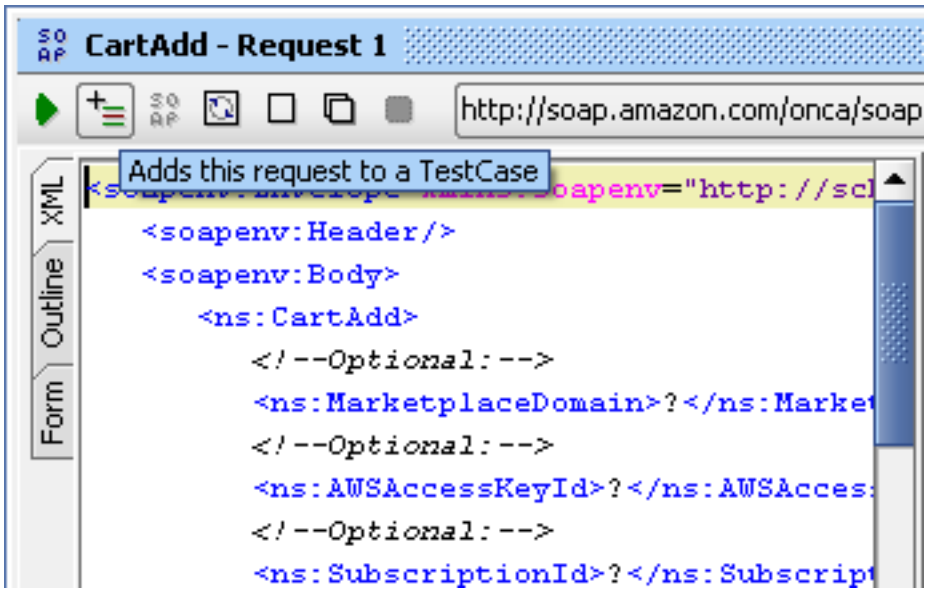
We will continue where we left off in the first "Getting Started" document where we just had created a project and imported the Amazon Web Service. The next step is to create a TestSuite/TestCase and add some TestSteps:

Create a TestCase from some request

Once you have some requests working as you want, you can create a testcase that verifies their behaviour

- Select the second toolbar button in the request editor window ("Add this request to a test-case").
- If there are no testsuites/testcases in your project, soapUI will prompt you for names of these, start out by specifying something like "Amazon TestSuite" and "Amazon TestCase"
- soapUI will prompt you for the name of your test request, call it "Step 1"
- Corresponding testsuite/testcases will be created and the request will be added as a Test-Request which is a copy of the original request (so you can keep playing with that without changing the test request)
- A test-request-editor almost identical to the previous request-editor will be opened with your new test request; it differs by adding assertions functionality

Read more about testsuites/testcases in the User Guide

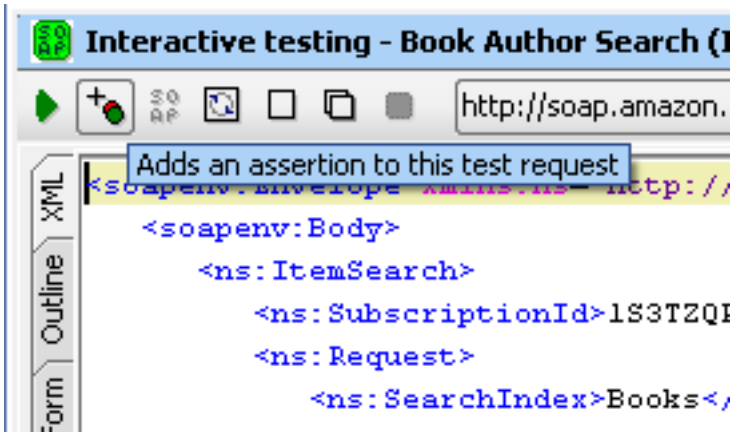


Add Assertions

Now that you have your first test-request you should add some assertions to verify that it works correctly

- Select the second toolbar button in the request editor window ("Add an assertion to this test-request").
- Start by adding a "Schema Compliance" assertion, this will check that the response is compliant with the associated WSDL/Schema definition. The assertion will be shown in the assertion list under the request/response editors (see image below)
- Post the request with the green submit-button, soapUI will run the request and validate the response. If all goes well the test-request should be shown with a green background in the navigation tree

Read more about assertions in the User Guide

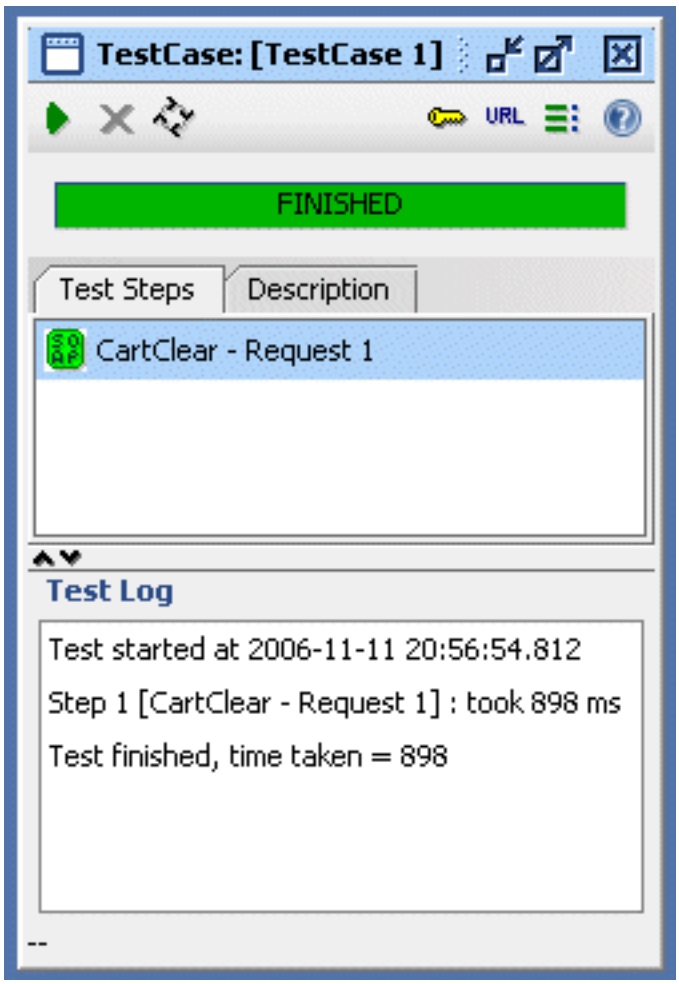


Now run your test!

Once you have all the test- request and their assertions you want you can run the entire testcase

- Double-click on the test-case node in the left navigation pane, this will open the test-case runner.
- Run all tests by selecting the green-arrow button labeled "Run this testcase", soapUI will submit each test-request and validate accordingly, the results will be displayed during the execution.

Run your tests from the command-line using one of the command line tools available.



Next: Getting Started with LoadTesting

1.6.2 TestSuites

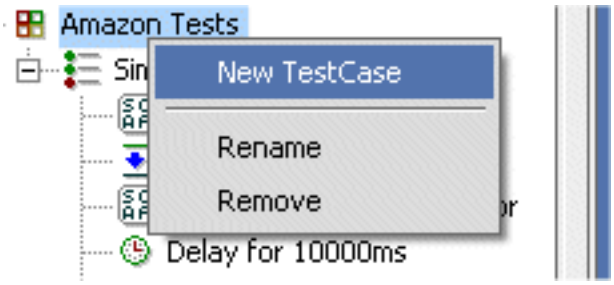
TestSuites

A TestSuite serves as container for an arbitrary number of TestCases. When running a TestSuite the contained TestCases can be executed either in sequence or in parallel as described below.

TestSuite Actions

The following actions are available from the TestSuite nodes' right-button menu:

- **Open TestSuite Editor** - Opens the TestSuite Runner described below
- **New TestCase** - prompts to create a new TestCase in the TestSuite
- **Clone TestSuite** - prompts to clone the entire TestSuite, including all TestCases/TestSteps
- **Rename** - prompts to rename the TestSuite
- **Remove** - prompts to remove the TestSuite from its project. All contained TestCases will be removed also.
- **Online Help** - Displays this page in an external browser



TestSuite Runner

Double Clicking a TestSuite in the navigator opens the TestSuite Runner containing a list of the contained TestCases and a toolbar. A Progress Bar is displayed for each TestCase, double-clicking a TestCase opens the associated TestCase editor. If a TestCase is currently being load tested its Progress Bar displays this and the TestCase will not be run during TestSuite execution.

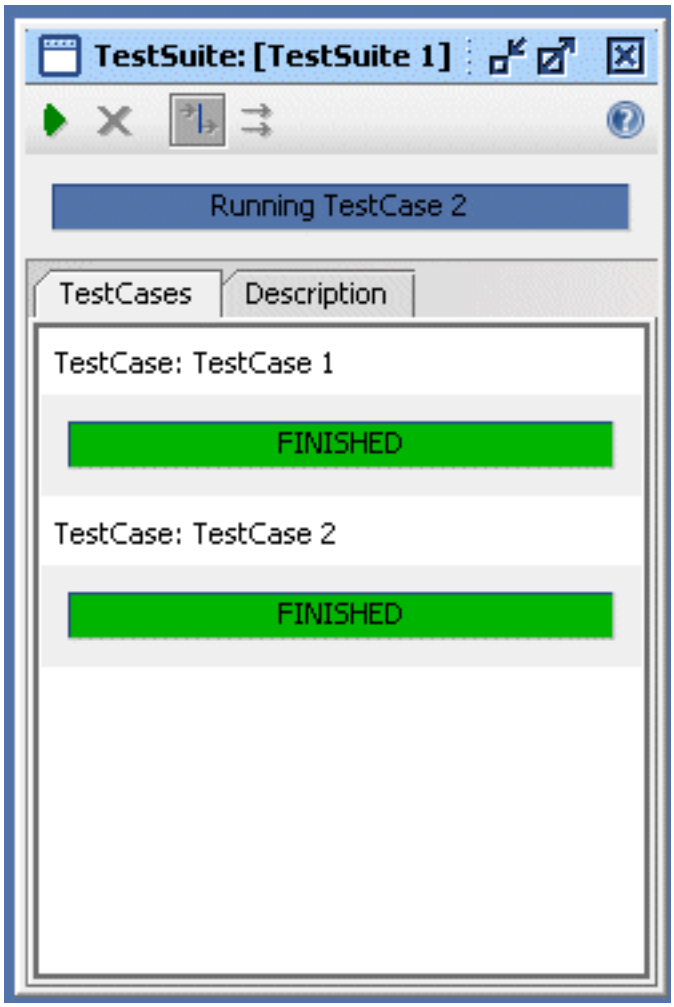
The available toolbar buttons are (left to right):

- **Run** : Runs selected TestCases
- **Cancel** : Cancels ongoing runs
- **Run in Sequence** : Toggles if the TestCases should be run in sequence
- **Run in Parallel** : Toggles if the TestCases should be run in parallel

TestCases can be selected/unselected by clicking them in the list, only selected TestCases are executed when running the TestSuite. If no TestCases are selected all are run.

The run in sequence/parallel state is preserved and also applied when running a TestSuite using one of the command line tools or the maven plugin. TestCase selection is *not* preserved internally, all TestCases are always executed in this case.

The **Description Tab** contains a single text area for arbitrary documentation for this TestSuite



Next: TestCases

1.6.3 TestCases

TestCases

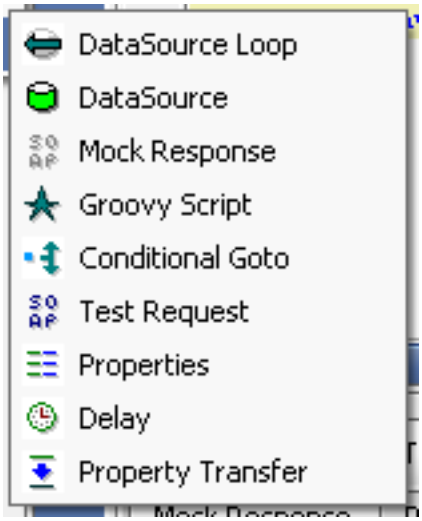
soapUI supports functional testing of web services by providing a TestCase metaphor where a number of TestSteps can be executed in sequence. Also, an arbitrary number of LoadTests can be associated with a TestCase for running the TestCase under different load scenarios.

Each TestStep in a TestCase exposes a number of properties which can be read/written/modified by other TestSteps, for example a Groovy Script step can read the "response" property of a Request Step and take some action depending on its value, see Property Expansion for details and examples.

TestStep Types

Currently the following types of steps are available;

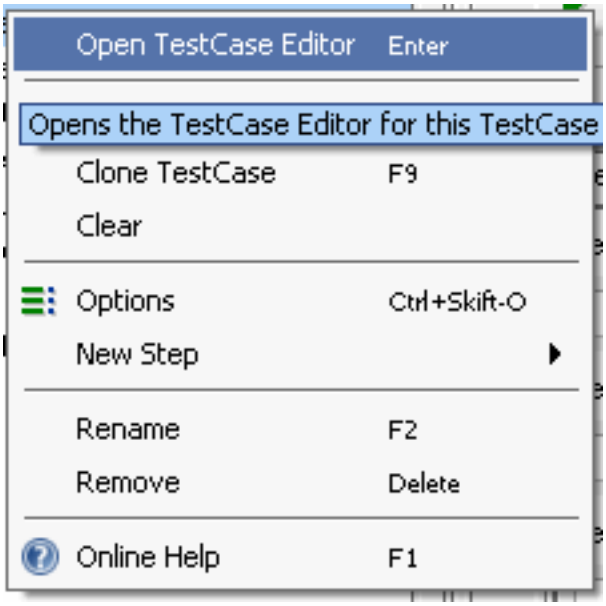
Step Type	Short Description
Request	Sends a SOAP request and allows the response to be validated using a variety of assertions.
Property Transfer	Used for transferring property values between two test steps.
Groovy Script	Runs a Groovy script that can do more or less "anything".
Properties	Used for defining global properties that can be read from an external source.
Conditional Goto	Allows any number of conditional jumps in the testcase execution path. Conditions are specified as xpath expression and applied to the previous request steps response.
Delay Step	Pauses a TestCase run for the specified number of milliseconds
DataSource Step	Reads external data to be used as input to requests, etc - soapUI pro only
DataSourceLoop Step	Used together with a DataSource to specify looping for external data rows - soapUI pro only
MockResponse Step	Waits/Listens for an incoming SOAP Request that can be validated and return a mock response - soapUI pro only



TestCase Actions

The following actions are available from the TestCase nodes' right-button menu:

- **Open TestCase Editor** - opens the TestCase Editor described below
- **New LoadTest** - prompts to create a new LoadTest for the TestCase
- **Clone TestCase** - prompts to clone the entire TestCase, optionally into another TestSuite
- **Clear** - prompts to remove all TestSteps from the TestCase
- **Options** - show the TestCase Options dialog described below
- **New Step** - adds a new TestStep to the TestCase
- **Rename** - prompts to rename the TestCase
- **Remove** - prompts to remove the TestCase from its TestSuite
- **Online Help** - Displays this page in an external browser



The TestCase Editor

Double-clicking a TestCase node in the navigator or selecting its **Open TestCase Editor** menu option opens the TestCase editor/runner. If the TestCase is currently being load-tested the editor will be mostly disabled.

The editor is divided into the following 4 parts (from top to bottom):

1. A toolbar at the top containing the following actions (from left to right)

- **Run TestCase** - runs the testcase (see below)
- **Cancel TestCase** - cancels a running TestCase
- **Run Continuously** - toggles if the TestCase is to be run continuously When selected, the TestCase will run repeatedly until cancelled with the Cancel TestCase button
- **TestCase Credentials** - prompts to set the credentials to be used by all requests in the TestCase. This is useful if you want to run your tests with different credentials
- **TestCase Endpoint** - prompts to set the endpoint to be used by all requests in the TestCase. This is useful if you want to run your tests against different servers, etc. The available URL's are collected from the test-requests' operations' interfaces. The selected endpoint is assigned to each request in the TestCase
- **TestCase Options** - opens the TestCase Options dialog described below
- **Online Help** - Displays this page in an external browser

2. A progress indicator showing how far a TestCase run has proceeded

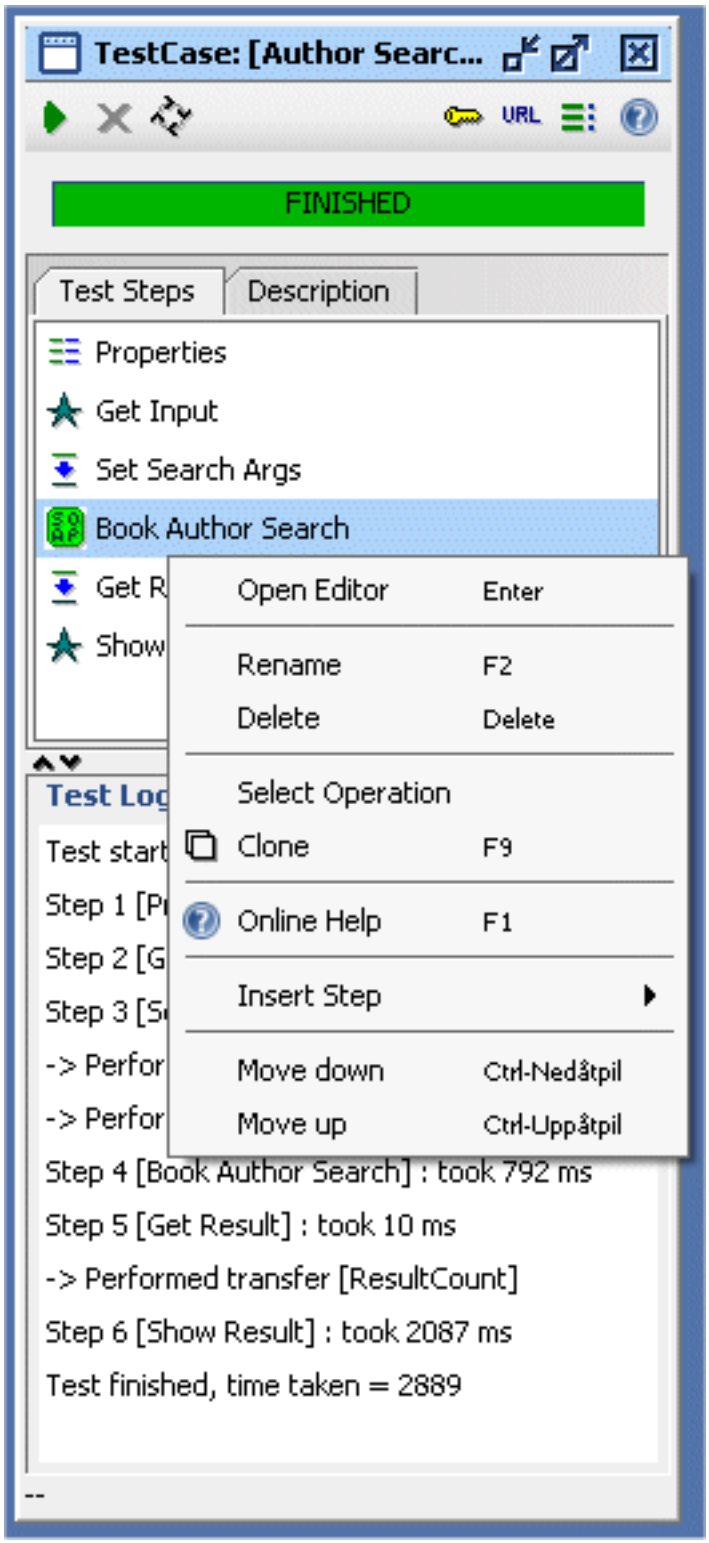
3. A "TestSteps" tab showing the TestSteps that this TestCase contains. Double-clicking an item in the list opens that items editor view. Right-clicking an item shows a popup-menu with the following actions:

- **Open Editor** - opens the associated TestStep editor (if available)
- **Rename** - prompts to rename the selected step
- **Delete** - prompts to delete the selected step
- **Clone** - prompts to clone the selected step and to which TestCase the step should be cloned. The cloned step will be added at the end of the selected TestCase
- **Insert Step ->** - shows a list of insertable test-steps at the current position
- **Move down** - moves the selected step down one position in the list (this can also be performed with Ctrl-Down)
- **Move up** - moves the selected step up one position in the list (this can also be performed with Ctrl-Up)

4. A "TestLog" list showing log-information when running the TestCase:

- When the TestCase started
- An entry for each executed TestStep specifying how long it took
- Optional errors and/or messages reported by each TestStep
- How long the TestCase took

The "Description" tab contains a single text area for arbitrary documentation for this TestCase

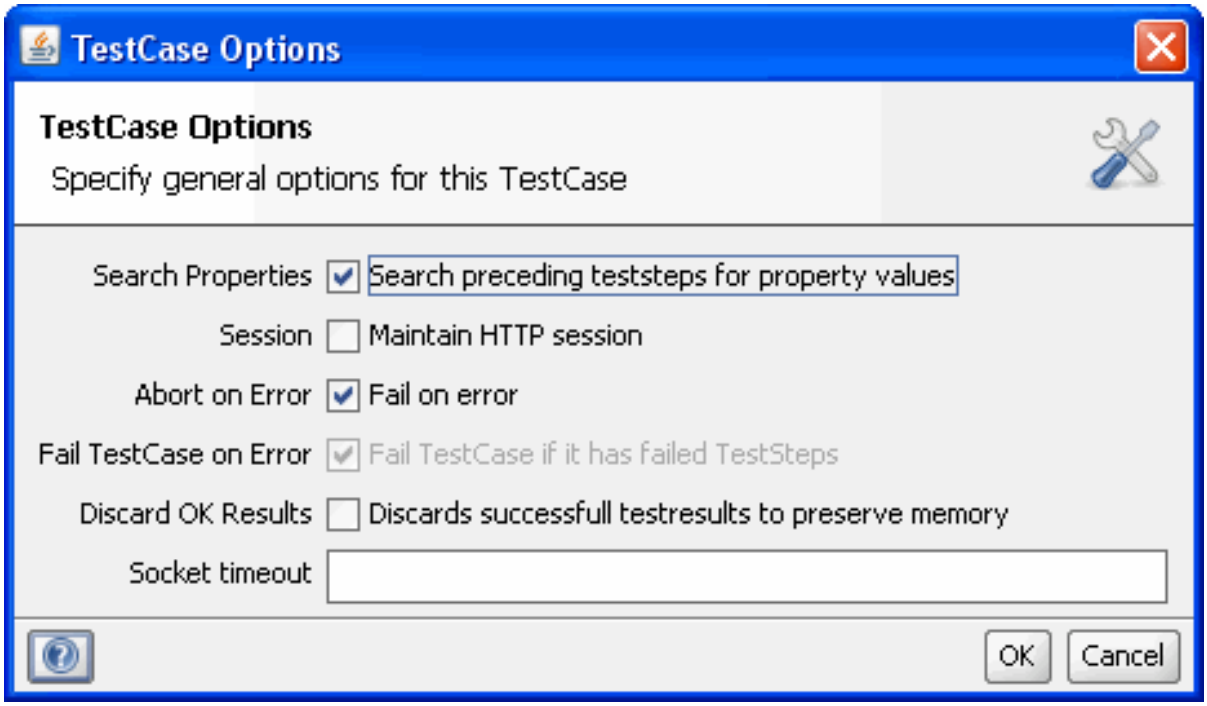


Double-clicking a TestSteps' entry in the log opens a TestStep result viewer for that step if available and as described on each TestStep documentation page (for example the Request Result Viewer).

TestCase Options

Selecting TestCase Options from either a TestCase nodes popup menu in the navigator or from the TestCase Editors toolbar opens a dialog with the following options.

- **Search Properties** : When looking för property values without step-specifications (see Property Expansion), check all steps before the current one for the named property.
- **Maintain HTTP Session** : Controls if a HTTP Session is to maintained for all requests in the TestCase. Selecting this will reuse cookies, authentication headers, etc
- **Fail on error** : Controls if the TestCase is to be cancelled when a TestStep fails with an error, (for example if a contained RequestStep has failed assertions)
- **Fail TestCase if it has failed TestSteps** : Controls if the TestCase is to fail if it the "Fail on error" option is not selected and the TestCase ends with one or more TestStep.
- **Discard OK Results** : Long-running testcases will eventually consume substantial amount of memory since all teststep results are internally cached for later viewing/reporting. Checking this option will make soapUI only save non-successfull teststep results which will save substantial amounts of memory.
- **Socket timeout** : The timeout (in milliseconds) to be used for all requests in the TestCase



Next: Working with TestRequests

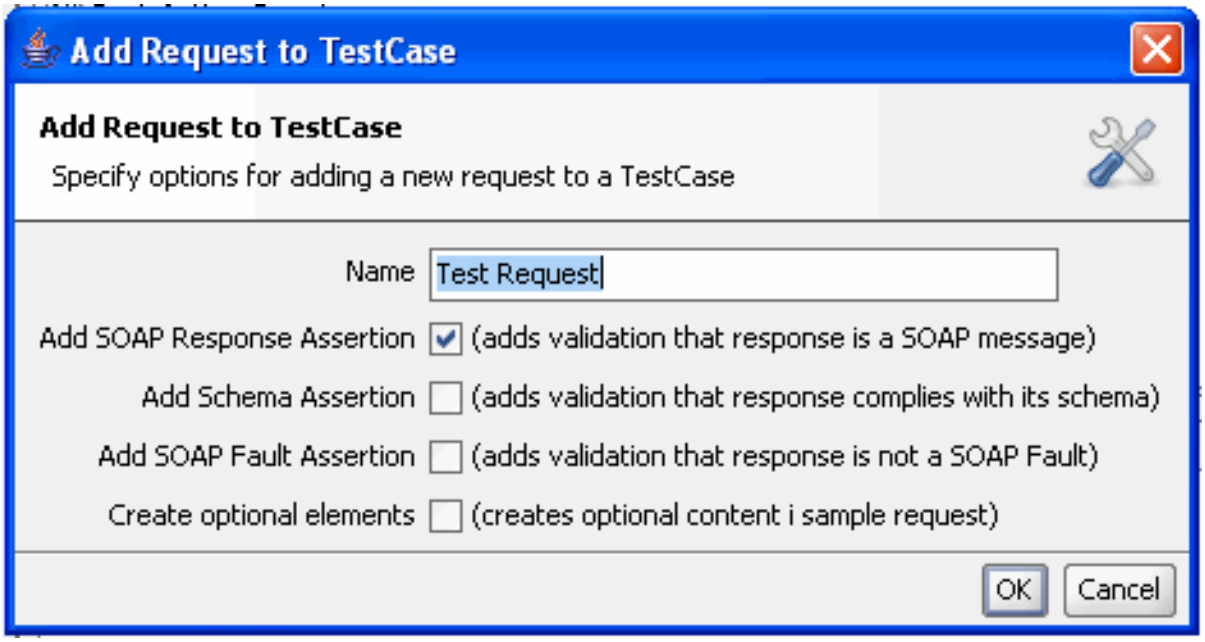
1.6.3.1 Test Requests

TestRequests

Test-Requests extend standard requests with the possibility to add any number of "Assertions" that will be applied to the response received for the request.

Test-Requests are either created from standard requests using their "Add to Testcase" action or from the TestCase Editors popup menu with the "Insert Step -> Test Request" option (shown to the right), which will prompt for which Interface/Operation the request should be create for.

In either case, a dialog will also prompt to add certain standard assertions so this must not be done manually for each request;



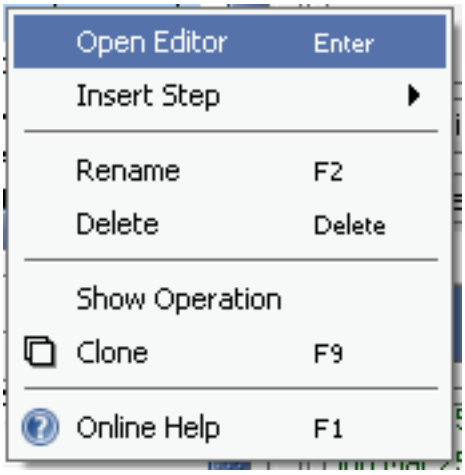
Test-Requests are submitted either manually through their editors submit actions or when running the TestCase containing the request. The requests response is validated against a requests assertions and the requests icon changes to reflect the result of the validations; green=all validations ok, red=some validation(s) failed. A grey background icon indicates that the request has not yet been submitted for validation, a white background indicates that the TestRequest lacks assertions

TestRequest Actions

The following actions are available from the test-request nodes' right-button menu:

- **Open Editor** - opens the Test Request Editor described below
- **Rename** - prompts to rename the test-request

- **Delete** - prompts to remove the test-request from its TestCase
- **Select Operation** - selects the interface operation in the Navigator that this request comes from
- **Clone** - prompts to clone the RequestStep, the cloned step will be appended to the containing TestCase
- **Online Help** - Displays this page in an external browser



TestRequest Details Tab

The "Details" tab shown in the bottom-left shows the same properties when a TestRequest node is selected in the navigation tree as for a standard request; see the Request Details Tab for more detailed information

The TestRequest Editor

Double-clicking a TestRequest in either the navigator or the TestCase Editor Pane opens the requests editor which is more or less a copy of the standard Request Editor with the following exception:

- The second toolbar-button "Add to Testcase" has been replaced with "Add Assertion" which prompts to add an assertion to the TestRequest
- The clone action now clones the TestRequest and appends the cloned request to the containing TestCase
- Under the request/response panes there is a now a tabbed pane containing 2 tabs; the "Assertions" tab and the "Request Log" tab (both described below).

All other editing/submittal/validation functionality is the same as in the request editor

The Assertions Tab

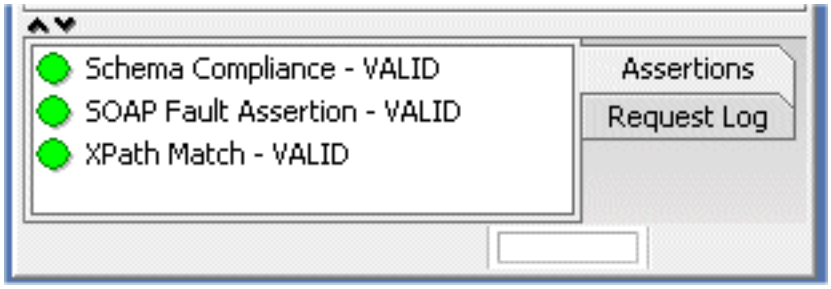
The Assertions tab lists the assertions that have been configured for the TestRequest. Double-clicking an assertion in the list opens that assertions' configuration dialog (if available). Any number of assertions can be added, often it may be relevant to add the same type of assertion multiple times with different configurations.

A colored circle next to the assertion indicates the status of the assertion in regard to the last received

response; red=assertion failed together with error message(s), green=assertion ok, grey=assertion has not been performed.

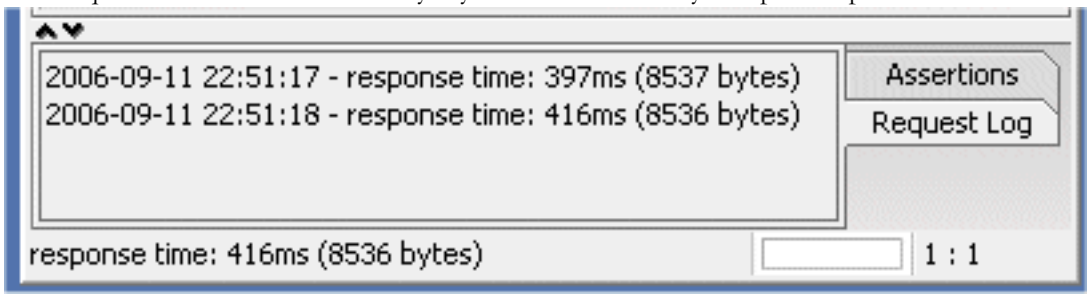
The following actions are available from the assertion lists right-button menu:

- **Add Assertion** - prompts to add a new assertion to the list.
- **Rename** - prompts to rename the assertion.
- **Remove** - prompts to remove the assertion.
- **Configure** - opens the assertions' configuration dialog (if available).



The Request Log Tab

The Request Log tab simply shows a history of submit/response times and response-sizes for the TestRequest. This can come in handy if you want to manually compare response times or sizes over time.

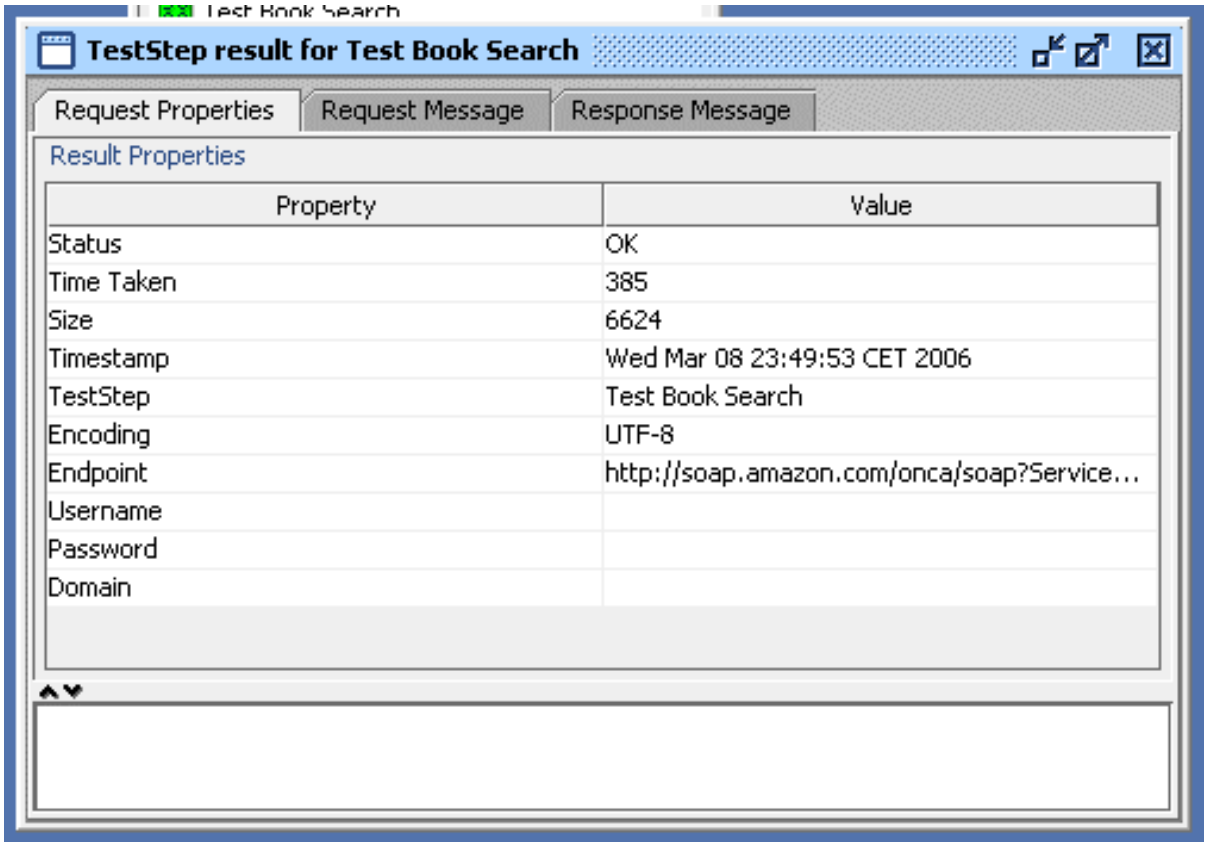


Request Result Viewer

When executing a Request Step from within a TestCase (or associated LoadTest) the actual request/result and associated properties for that request can be viewed by opening a "Request Result Viewer" either from the TestCase editors log list or from a LoadTest editors log by double-clicking the associated log entry. Sent/Received attachments are currently not saved to preserve memory.

The viewer shows 3 tabs:

- **Request Properties** - shows request and reponse properties for the request
- **Request Message** - shows the actual request message sent, including expanded properties, inline files and MTOM/XOP Includes..
- **Response Message** - shows the response message received



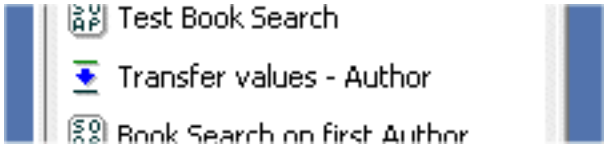
Next: Assertions

1.6.3.2 Property Transfers

Property Transfers

Property Transfers are TestSteps that transfer properties between TestSteps, each containing an arbitrary number of "property transfers" and each containing a source and destination step and property specification with optional XPath expressions.

Property Transfers use the same Saxon XPath engine as described for the XPath Assertion.

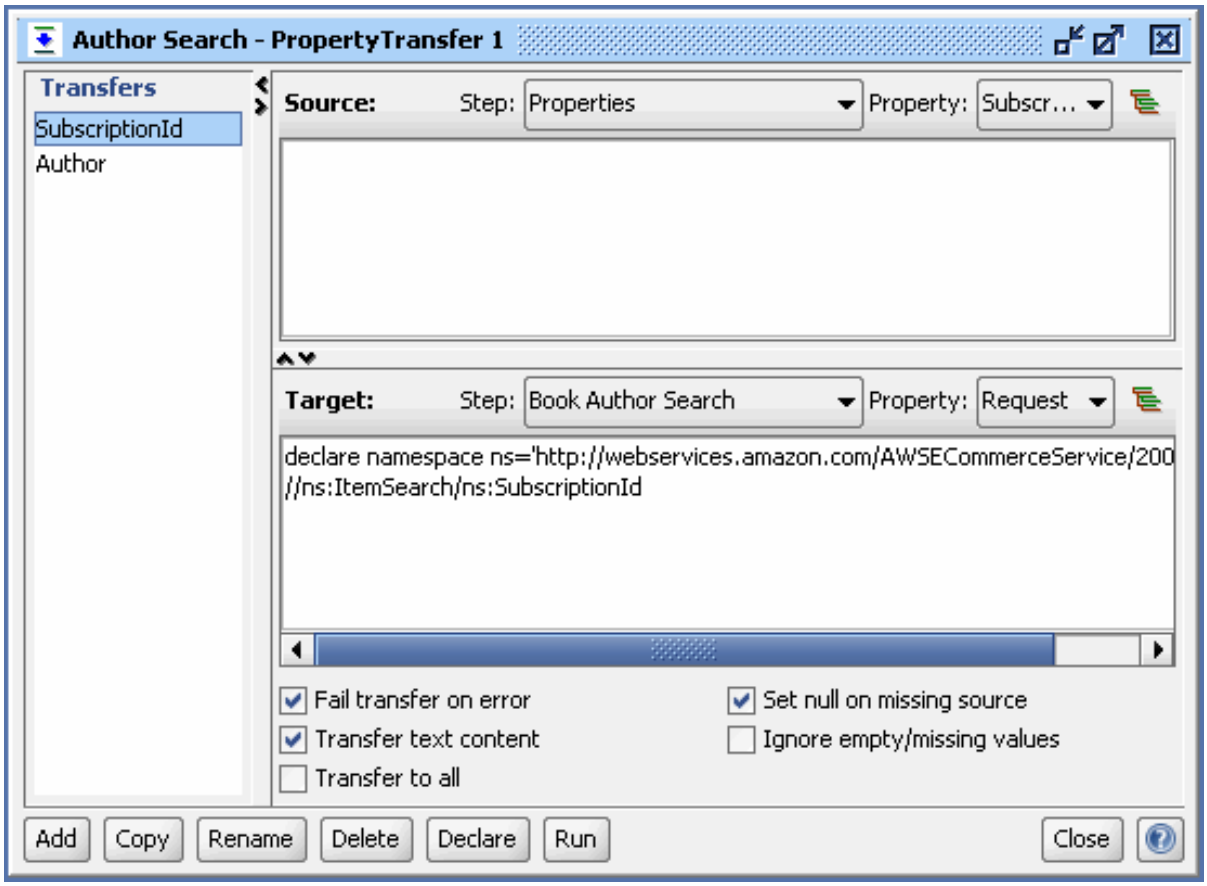


The PropertyTransfer Editor

The Property Transfer editor is opened by double-clicking a PropertyTransfer step either in the navigator or in the TestCase editors' TestStep-list.

The editor contains a list of configured transfers to the left, selecting a transfer in the list will displays that transfers source and destination XPath expressions to the right.

Use the top-right comboboxes to specify the source step/property to be transferred, the target step/property is specified with the combo-boxes in the middle. If the properties contain XML an accompanying XPath expression can be specified to further select the value to transfer from/to. soapUI Pro adds an XPath Selector button to the right of these combo boxes for easily selecting the source/target XPath expression from the selected source/target property.



The following actions are available from the bottom toolbar

- **Add** : prompts to add a new transfer to the list
- **Copy** : prompts to create a copy of the selected transfer
- **Rename** : prompts to rename the selected transfer
- **Delete** : prompts to delete the selected transfer
- **Declare** : declares namespaces in both the source and destination XPath fields. Namespaces in the source field are extracted from the selected TestRequests current response message, namespaces in the target field are extracted from the subsequent TestRequests request message. If either of these are not available, soapUI will prompt to define all namespaces available in the associated schemas.
- **Run** : runs the transfers, ie transfers the values specified. For this to work properly the source and target properties must be available (for example a TestRequest response).
- **Close** : closes the dialog
- **Online Help** - Displays this page in an external browser

Transfer Execution

Upon execution during a TestCase run, each transfer in the Property Transfer is performed by selecting the property specified by the transfers source step, property and optional XPath expression and copying their value(s) to the destination steps specified property using an optional XPath expression.

If XPath expressions are specified, soapUI will try to replace the target node with the source node if they are of the same type. If not (for example when assigning text() to an @attribute), soapUI will do its best to copy the value as possible.

Source and target XPath expressions must both point to *existing nodes* in their respective properties, the source property obviously requires the node so it can be selected, the target property requires the node so it can be found and overwritten.

If any of the transfers fail due to missing matches of any of the XPath expression, an error is printed and the step will either fail or go on, depending on of the "fail on error" option has been selected for that transfer. TestCase execution is only aborted if the TestCases' "Fail on error" option has been set as described for under TestCase Options .

The following options are available for each transfer:

- **Fail transfer on error** - Fails the property-transfer if an error occurs (for example a missing source property)
- **Set null on missing source** - Overides errors for missing source values and sets the target property to null in these cases
- **Transfer text content** - When the xpath expression point at element nodes, their text content is transferred instead of the elements themselves (required for backward-compatibiliy with soapUI 1.5)
- **Ignore empty/missing values** - Overrides errors for missing source values and just ignores corresponding transfers
- **Transfer to all** - If the target XPath expression selects multiple nodes, the source property value will be set in all these nodes instead of only the first one.

Working with Property Transfers

A Property Transfer can be created as follows:

1. Begin by creating the two TestSteps that the PropertyTransfer should transfer between.
2. Create the PropertyTransfer and first add a transfer in the configuration dialog using the **Add** button
3. Select the source and target TestStep and property in their respective combo-boxes
4. If any of the properties contains XML, proceed by defining namespaces in the XPath expressions using the **Define** button, then add the actual XPath expressions that specify what to select and where to copy it. In the screenshot above both expressions are `//ns1:SessionId` which will result in the sessionId element being copied from the preceding response to the following request (where it must be available but preferably empty)
5. Test the transfer(s) by selecting the **Run** button and check the following request message that the values have been copies correctly. Any errors will be shown either in the main soapUI log (at the bottom) or in a designated popup
6. Repeat steps 3-5 for each transfer added to the ValueTransfer

with soapUI Pro creating property transfers is greatly simplified using the corresponding Response Wizards which perform most of the above steps automatically.

PropertyTransfer Example

The following sample is also included in the sample project included in the offline distribution:

Set up 2 requests to the Amazon Web Service defined at (http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl);

```
<soapenv:Envelope
xmlns:ns="http://webservices.amazon.com/AWSECommerceService/2006-02-15"
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
     <ns:ItemSearch>
        <ns:SubscriptionId>- your subscription id here-</ns:SubscriptionId>
        <ns:Request>
           <ns:SearchIndex>Books</ns:SearchIndex>
           <ns:Title>Oxford</ns:Title>
        </ns:Request>
     </ns:ItemSearch>
  </soapenv:Body>
</soapenv:Envelope>
```

and

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ns="http://webservices.amazon.com/AWSECommerceService/2006-02-15">
  <soapenv:Body>
     <ns:ItemSearch>
        <ns:SubscriptionId>- your subscription id here -</ns:SubscriptionId>
        <ns:Request>
           <ns:Author>?</ns:Author>
       <ns:SearchIndex>Books</ns:SearchIndex>
        </ns:Request>
     </ns:ItemSearch>
  </soapenv:Body>
</soapenv:Envelope>
```

and then create the following transfer which moves the first author from the first requests "response" property to the `author` request element in the following steps "request" property;

Source XPath:

```
declare namespace ns='http://webservices.amazon.com/AWSECommerceService/2006-02-15';
(//ns:Author)[1]
```

Target XPath:

```
declare namespace ns='http://webservices.amazon.com/AWSECommerceService/2006-02-15';
(//ns:Author)[1]
```

Next: Conditional Goto Steps

1.6.3.3 Conditional Gotos

Conditional Gotos

Conditional Goto steps evaluate an arbitrary number of xpath conditions on the previous requests response message and transfer TestCase execution to the TestStep associated with the first condition that evaluates to true. This allows for conditional TestCase execution paths, where the result of some request controls how to move on in the TestCase. If no condition matches the current response, TestCase execution continues after the Goto Step as normal.

Sample scenarios could be:

- Branching depending on results returned by a request
- Restarting after a "longer" delay (minutes) for surveillance testing
- Repeatedly waiting and checking for a status value before moving on (for example on a batch-process)

Conditions use the same Saxon XPath engine as described for the XPath Assertion, remember that a condition *must* evaluate to a Boolean value to be valid (see examples below)

The Conditional Goto Editor

The ConditionalGoto Editor is opened by double clicking a ConditionalGoto test step either in the navigator or in the TestCase Editors' test-step-list.

The editor contains a list of configured conditions to the left, selecting an existing condition in the list will display that condition's expression and a target TestStep ComboBox to the right. The **Test Condition** button will evaluate the current condition against the current response and display the result (a response message must be available for the preceding TestRequest).

soapUI Pro adds an XPath Selector button to the right of the Test Condition button for easily selecting the XPath that should be used for evaluation.

The following actions are available from the bottom toolbar

- **Add** : prompts to add a new condition to the list
- **Copy** : prompts to create a copy of the selected condition
- **Delete** : prompts to delete the selected condition
- **Declare** : declares namespaces in the selected conditions expression field. Namespaces are extracted from the preceding TestRequests current response message (if available).
- **Run** : runs the conditions and displays which condition that matched the current response (a response message must be available for the preceding TestRequest)
- **Close** : closes the dialog
- **Online Help** - Displays this page in an external browser

Condition Examples

Conditions must evaluate to a Boolean value, for example the following expression checks that there are hits in the search result returned from the Amazon web service:

```
declare namespace SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/';
declare namespace
ns1='http://webservices.amazon.com/AWSECommerceService/2005-10-05';
declare namespace SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/';
count(//ns1:Item)>0
```

A generic condition that checks for a soap body would be:

```
declare namespace SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/';
count(//SOAP-ENV:Body)=1
```

and the corresponding check for a soap fault would be:

```
declare namespace SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/';
count(//SOAP-ENV:Fault)=1
```

Next: Groovy Script Steps

1.6.3.4 Groovy Scripts

Groovy Scripts

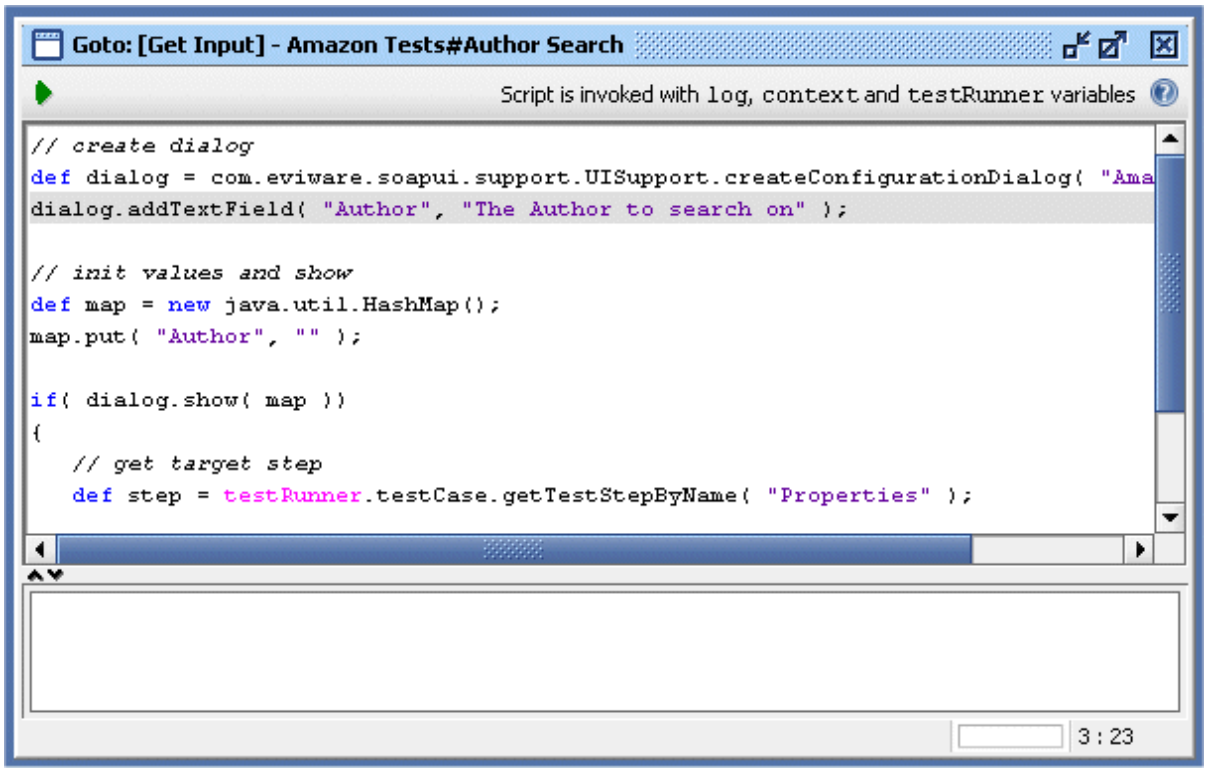
The Groovy Script step allows you to specify an arbitrary Groovy script during the execution of a TestCase. The script has full access to the soapUI object model and can thus perform a variety of tasks, for example:

- Read data from an external data source and write it to another steps properties
- Control TestCase flow based on outcome of some previous step or external invocation
- Trigger the execution of other TestSteps and/or TestCases
- Perform advanced response validations not possible with existing assertions
- Write the results of some previous TestStep to an external source (reporting, etc...)
- etc..

If you need to add external libraries to the soapUI classpath for your groovy scripts (for example jdbc drivers), you need to modify the soapUI.bat/.sh file manually and add the respective .jar files there (not currently possible in the Java WebStart version of soapUI).

The Groovy Script Editor

The Groovy Script editor is a straight forward text-editor without any bells-and-whistles except undo/redo functionality;



The editor has three components (from top to bottom):

- **A toolbar** : currently only containing a "Run" button
- **A script editor** : standard text-editor
- **A log** : can be written to using the script contexts `log` object
- **A statusbar** : displays caret position and a progressbar during script execution

Script Execution

When a groovy script is executed, the following context variables are set for the script to use:

- `testRunner` : the TestRunner running the current TestCase, this will for now always be an instance of WsdlTestCaseRunner
- `context` : the TestRunContext running the current TestCase, this will for now always be an instance of WsdlTestRunContext
- `log` : a standard log4j `Logger` object available for logging arbitrary information

When the script is run from inside the editor using the toolbars "Run" button, the first 2 objects will be set to mock implementations allowing limited access to their actual functionality. The `log` object will in this case write to the log window in the editor, when running from inside a TestCase the log will write to a "groovy log" tab in the main soapUI log.

If you want to fail the script due to some internal error throw an Exception from the script containing the error message (see example below)

Context Properties

The following properties are available from within a groovy-script:

- **ThreadIndex** - the index of the created thread when running under a LoadTest. This value will never change for a given TestCase during its run time. New threads will simply get an incremented index, the mod of this value could for example be used as an index into a data-file (to handle changes in number of threads). When not running under a LoadTest the value of this property will be "0".
- **RunCount** - tells how many times the TestCase has been run by its thread (not in total) when running under a LoadTest. When not running under a LoadTest the value of this property will be "0".
- **LoadTestRunner** - the current LoadTestRunner when running under a LoadTest, null otherwise.
- **#HTTP_STATE** - the HttpState maintained by the TestCase if it has been set to do so in the TestCase Options dialog.

Groovy Script Examples

Getting started with Groovy is not as easy as it may seem... here are some examples to get you started:

Modification of a Request xml:

```
// get request property
def request = testRunner.testCase.getTestStepByName( "Request 1" );
def property = request.getProperty( "request" );

// parse out textnodes to modify
def node = new groovy.util.XmlParser(false,false).parseText(property.value);
def textNodes = node["soapenv:Body"]["sam:getContactInfo"]["String_1"][0].children()

// modify
textNodes.clear();
textNodes.add( "test" + System.currentTimeMillis() );

// write back to string
def writer = new java.io.StringWriter();
def printer = new groovy.util.XmlNodePrinter( new PrintWriter( writer ));
printer.print( node );

// set property
property.setValue( writer.toString() )
```

Restarting a test after a certain time

```
// check if time is set
startTime = context.getProperty( "startTime" )
if( startTime == null )
{
   startTime = System.currentTimeMillis()
   context.setProperty( "startTime", startTime )
}
timePassed = System.currentTimeMillis() - startTime
if( timePassed < 60000 )
{
   // countdown and set name
   context.currentStep.name = "Groovy Step - " + (60000-timePassed) + "ms left"
```

```
   log.info "timePassed = " + timePassed
   Thread.sleep( 1000 )
   testRunner.gotoStep( 0 )
}
else
{
   // reset name and pass on
   context.currentStep.name = "Groovy Step"
   log.info "timePassed = " + timePassed + ", exiting.."
}
```

Reading properties from a file and assigning them to a Properties Step

```
// read the file
def properties = new java.util.Properties();
properties.load( new java.io.FileInputStream( "testprops.txt" ));

def targetStep = testRunner.testCase.getTestStepByName( "Properties" );

// assign single property
targetStep.setPropertyValue( "myproperty", properties.getProperty( "myproperty" ));

// assign all properties
def enum = properties.propertyNames();
while( enum.hasMoreElements() )
{
   def name = enum.nextElement();
   targetStep.setPropertyValue( name, properties.getProperty( name ));
}
```

Add a TestStep dynamically

```
// add a properties step
testRunner.testCase.addTestStep( "properties", "another step" );
```

Fail the script step

```
// fail randomly
if( Math.random() > 0.5 )
   throw new Exception( "A random error has occured!" );
```

Read/set cookies

If you select the "Maintain HTTP Session" option in the TestCase Options Dialog, an internal `HttpState` object will be maintained for the entire TestCase, which inludes any cookies set by the target server. The following script reads all cookies currently set in the HttpState and writes them to the groovy log:

```
def state = context.getProperty(
com.eviware.soapui.model.testsuite.TestRunContext.HTTP_STATE_PROPERTY )
assert state != null : "Missing HttpState.."

def cookies = state.cookies
assert cookies.length > 0 : "Missing cookies.."
```

```
for( c in 0..cookies.length-1 )
 log.info cookies[c].name + " = " + cookies[c].value
```

If you on the other hand want to set some cookie before making a request, do as follows:

```
def state = context.getProperty(
com.eviware.soapui.model.testsuite.TestRunContext.HTTP_STATE_PROPERTY )
assert state != null : "Missing HttpState.."

state.addCookie( new org.apache.commons.httpclient.Cookie(
"http://www.mydomain.com", "SessionID", "1234" ))
```

Want some more examples? Mail the soapUI forum for a request and we will add them!

Next: Property Steps

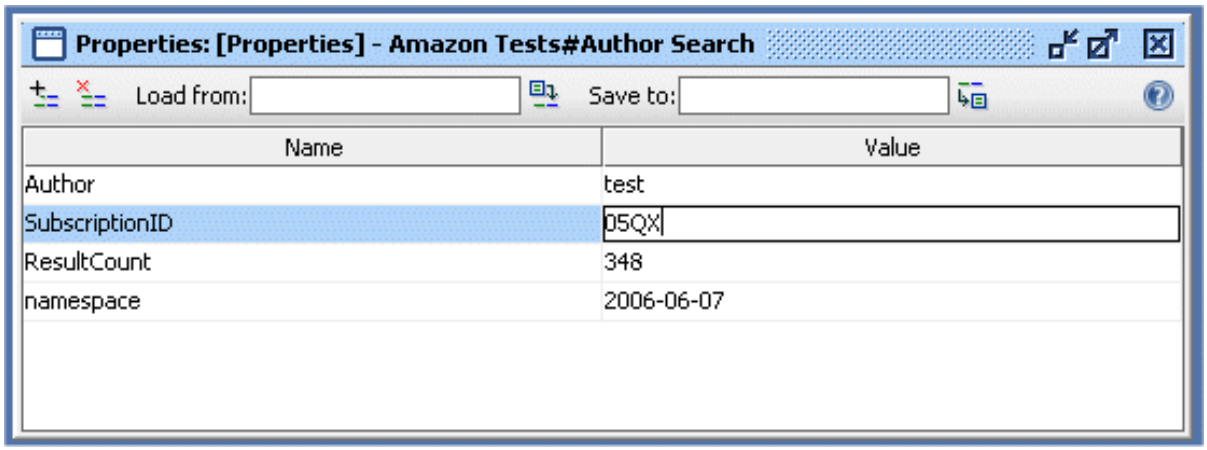
1.6.3.5 Properties Step

Properties Step

The Properties Step allows you to define an arbitrary number of properties that can be used from PropertyTransfer and GroovyScript steps. The properties can optionally be read from/written to a properties-file upon execution, for example if you want to specify some properties external (passwords, endpoints, etc) or want to write some results to a file for later reporting..

Properties Editor

The Property Step Editor is straightforward:



The toolbar at the top contains the following (left to right):

- **Add Property** : prompts to add a new property
- **Remove Property** : prompts to remove the selected property
- **Load from** : optional field containing a file/URL/system-property that contains the source of the properties. The specified file/URL will be read as a standard properties file and contained property values will be assigned to the steps properties. If the field is set to the name of a system property, this property must in turn specify a file or URL that will be subsequently read as described
- **Select Properties Source** : allows selection of a local file containing the properties to be read. The selected file will read and the contained properties values will be assigned to matching properties in the Properties Step (you will be prompted if unavailable properties are to be created)
- **Save to** : optional field containing a file/system-property that contains the target name of properties file. The specified file will be created as a standard properties file and contained property values will be written to it. If the field is set to the name of a system property, this property must in turn specify a file that will be subsequently created as described

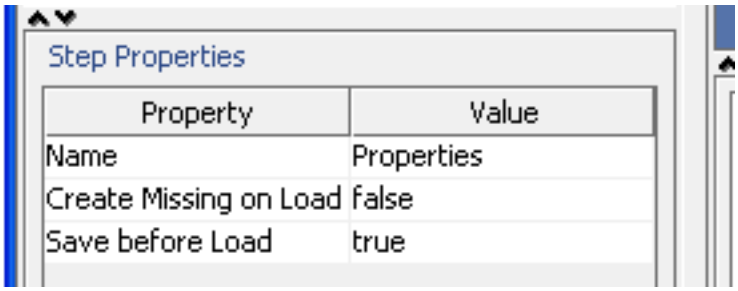
- **Select Properties Target** : allows selection of a local file to which the properties should be written

The table displayed under the toolbar displays the currently defined properties and their values, values and names can be changed by standard editing.

Details Tab

The Properties Step Details Tab (bottom left) contains 2 options for Property Step execution:

- Create Missing on Load - creates properties from source property files that are not currently defined
- Save before Load - specifies to save existing properties before loading new ones to/from the specified source and target property files



Step Execution

When a Properties Step is executed during a TestCase, the following actions are taken:

- The properties are read or written from a source if specified as described above (depending on "Save before Load" option)
- The properties are written or read to a target if specified as described above (depending on "Save before Load" option)
- The properties are all copied to the current TestRunContexts properties so they are available for Property Expansion

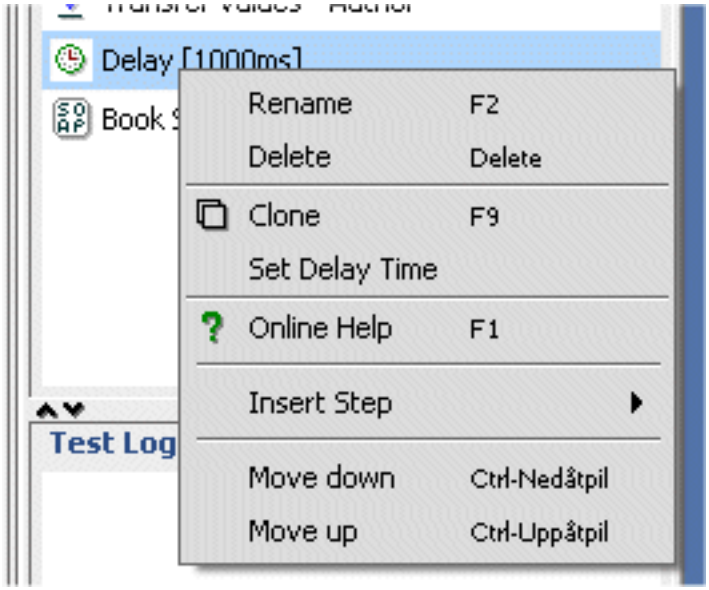
Next: Delay TestStep

1.6.3.6 Delay Step

Delay Steps

Delay Test Steps can be inserted at any position in a TestCase to pause the execution of the TestCase for a specified number of milliseconds.

Once inserted using the **Insert Step** menu option described above, right-click the delay step and select the **Set Delay Time** option to set the number of milliseconds to pause (default is 1000ms).



Next: Property Expansion

1.6.3.7 Mock Response

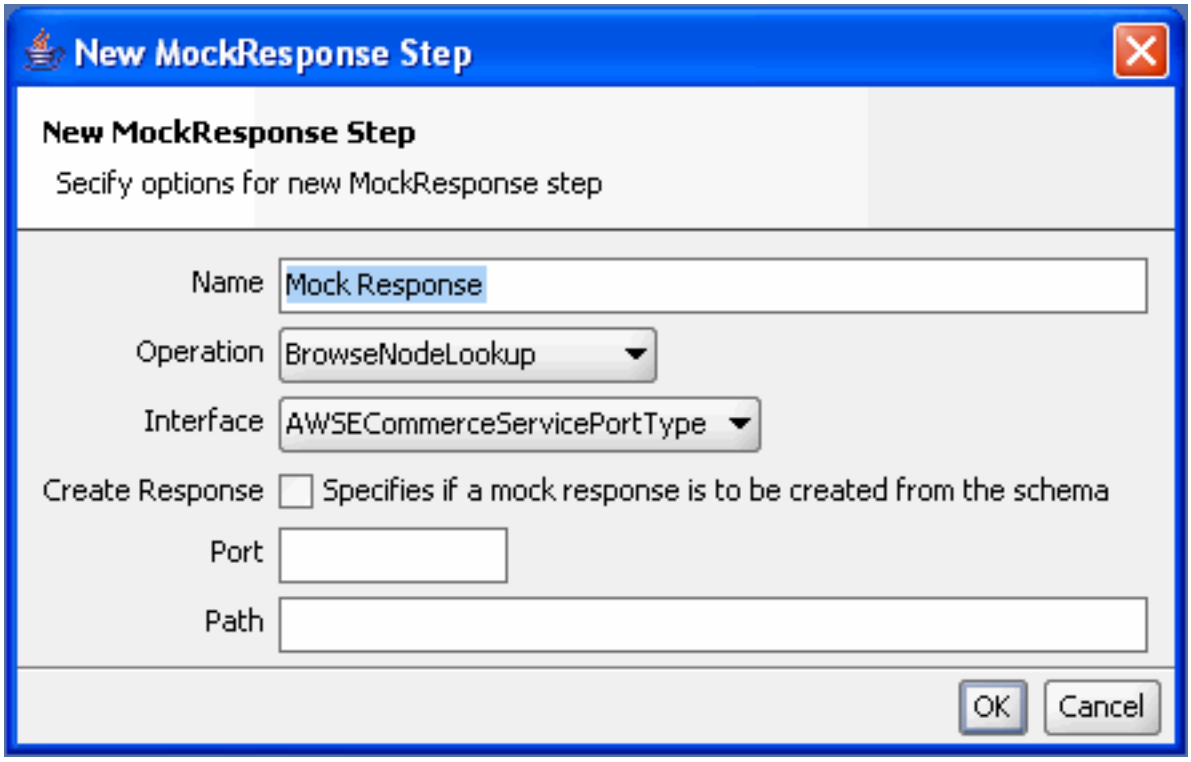
MockResponse Step

soapUI Pro includes a MockResponse step that listens/waits for a SOAP Request and returns a preconfigured response before moving on. The incoming request can be validated just as the response of a TestRequest Step with the same configurable assertions.

Usage scenarios for this TestStep are for example:

- Client testing - validating incoming requests and returning dummy/incorrect responses to test a clients behaviour
- Testing of asynchronous processes - for example starting some kind of job with an initial RequestStep and then waiting for a notification before moving on

When selecting to create a MockResponse Step from the TestCases Insert/Add Step menu, the following dialog will be displayed:



The dialog contains the following options:

- Name - The name of the created step
- Operation - Specifies which operation to mock

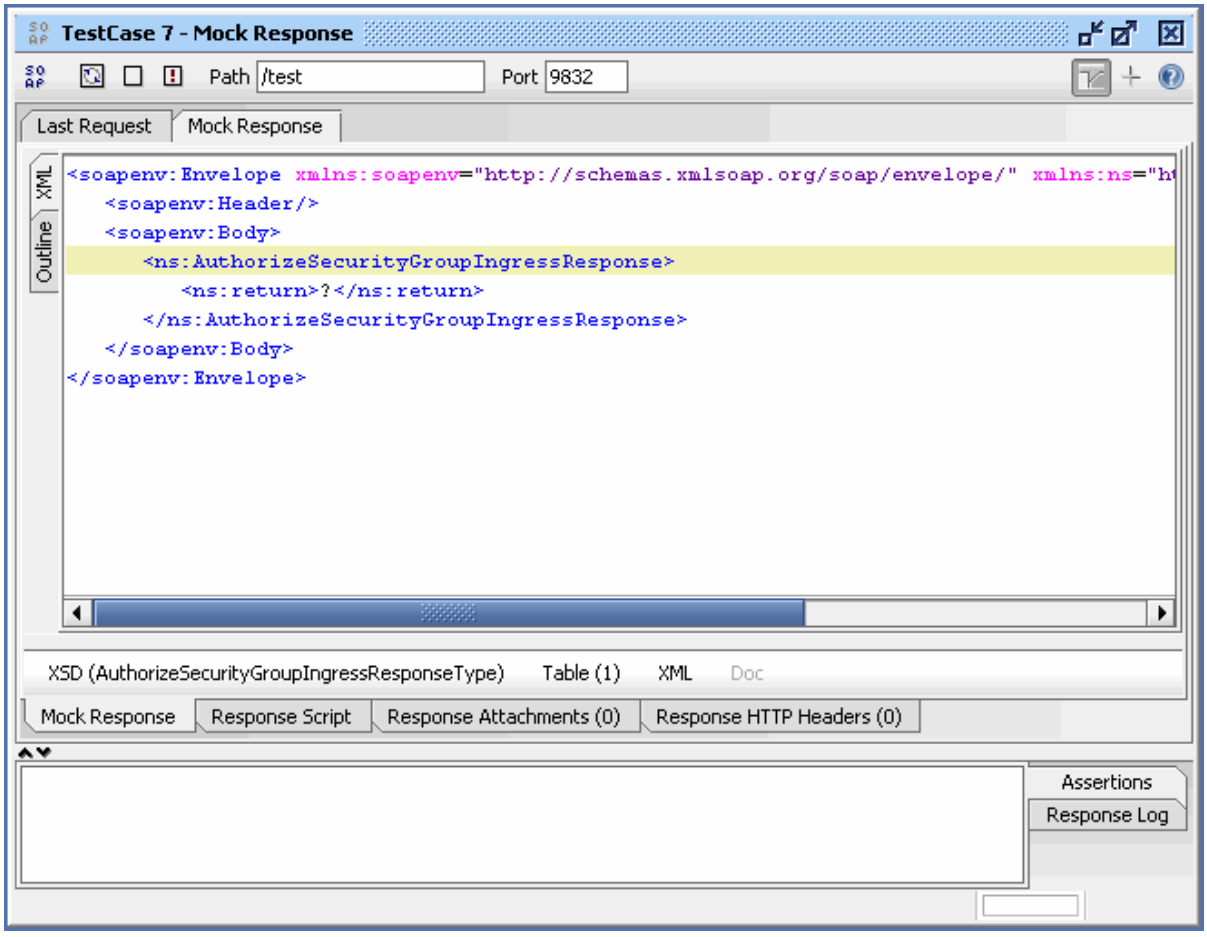
- Interface - Specifies which interface to mock, changing this will also update the list of available operations
- Create Response - if a default MockResponse message should be created from the schema
- Port - the port to listen on
- Path - the path to listen on

After setting the desired values and selecting OK, the step will be created and its editor opened

The MockResponse Step Editor

The MockResponse Step is the basically the same as the standard MockResponse editor with the addition of the same assertions/log tabs in the bottom as the TestRequest editor and a number of context-sensitive wizards in the Outline Editor for creating assertions and Property Transfers.

At the top there is a toolbar that adds input fields for the path and port to listen on during execution, the rest is the same as the MockResponse editor. These are grayed out during execution of the MockResponse step. Assertions can be added/managed just as for the TestRequest editor , the list of available assertions is the same except the SOAP-Fault-related assertions, which are not applicable to request messages.



MockResponse Step Execution

When the execution of a TestCase reaches a MockResponse Step, the step will start a local temporary MockService and wait for a request to the configured operation on the configured path and port. Onces a request has been received, it will be validated with the configured assertions and processed just like a standard MockResponse. After returning the MockResponse result, the MockService will be closed down. and execution will move to the next step in the TestCase.

Property Transfers can be used with MockResponse Steps just as with TestRequests, ie properties can be transferred from incoming requests and to outgoing responses.

Although running a LoadTest containing MockResponse steps will work, these steps are not designed for LoadTesting due to the following:

- A MockService is started and stopped for each time the MockResponse step executes. If the request to the temporary MockService comes in before it has time to start, it will get a standard 404 error message back.
- If there are several threads running/waiting for a request, in which order should soapUI dispatch incoming requests to these? Currently soapUI only starts one MockService at a time, the first thread to get to the MockResponse step gets to wait first. Susbsequent steps will need to wait until "their turn".

Next:

1.6.3.8 DataSource

DataSource Steps

soapUI Pro includes a specialized DataSource TestStep that greatly eases the task of creating datadriven tests. DataSource steps can further be nested in a TestCase allowing one DataSource to be input to another, for example if directly contains a number of xml-datafiles, the Directory DataSource could be used to iterate the files, and an XML DataSource could be used to extract data from the files.

The following DataSources are currently supported:

- JDBC DataSource - reads data from a JDBC datasource
- XML DataSource - extracts data from an XML property
- File DataSource - extracts data from a columnar data file
- Directory DataSource - reads files into properties
- Groovy DataSource - opens for any kind of DataSource

A complementary DataSourceLoop step is required and thus available for specifying whend and where to loop for each row of data available from the datasource.

The DataSource Editor

The DataSource editor is divided into 4 parts;

- A toolbar for selecting a DataSource and setting properties and options
- A list of properties to the left for specifying which properties the DataSource reads and exposes
- A DataSource specific-configuration panel
- A Test log displaying rows returned when either testing the datasource or when running its containing TestCase

The list of Properties must contain the properties that will be "exposed" by the DataSource step, they are added/removed using the Add/Remove toolbar buttons. Each DataSource then has its own way of reading data into the properties, as described below.

The DataSource dropdown select with DataSource to use, and the "Test" button will prompt to select a configurable number of rows from the currently selected and configured DataSource. Returned rows will be displayed in the TestLog at the bottom. The "Preload" and "Share" options are described in the "DataSource Exection" section below.

Depending on the value of the "Preload" option in the toolbar, the DataSource will initialize its external data either before running the TestCase (preload = true) or when first running the DataSource (preload = false). Depending on which kind of DataSource that has been selected and configured, it may not make much sense to set preload to true, for example when using the XML Datasource together with the

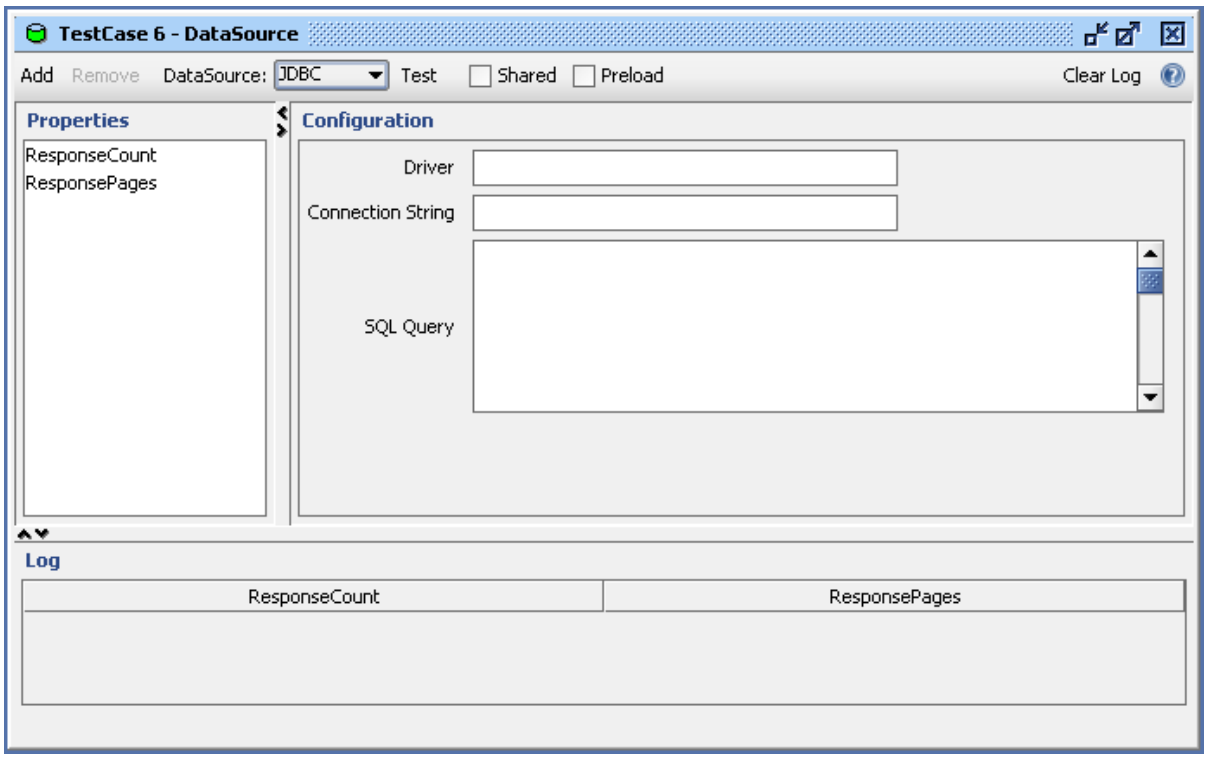
response property of a request, setting preload to true will fail since the data is first available when the request has been executed.

Consecutive rows are retreived and read into the exposed properties either when TestCase execution passes the DataSource step or when a DataSourceLoop Step checks for the next available row.

The "Shared" option in the toolbar control if the DataSource should be shared between threads during a LoadTest, meaning that all running threads will read from the same source of data instead of each creating their own.

JDBC DataSource

The JDBC DataSource has the following configuration panel:

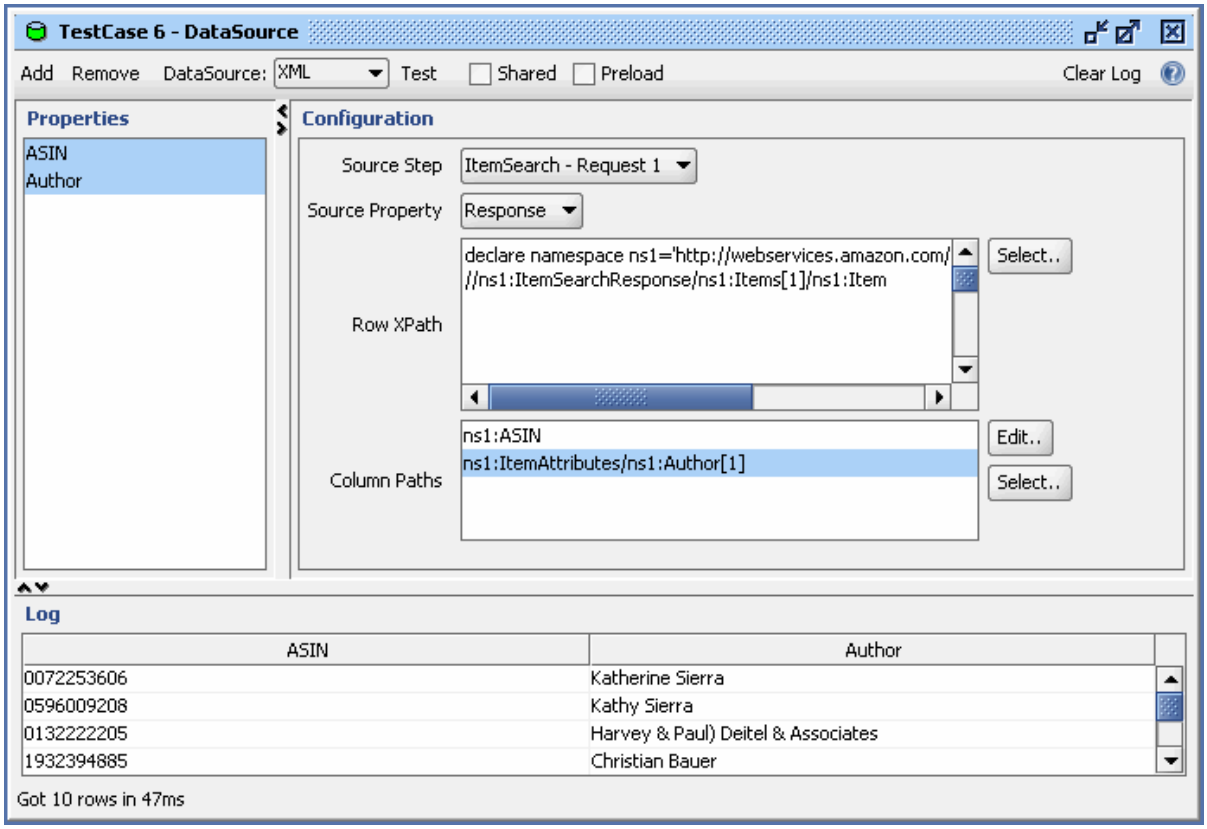


The 3 available options are pretty straight-forward:

- Driver - the JDBC Driver to load and use for this DataSource. Must be available in the soapUI classpath either by placing it in a soapui/bin/ext directory, the jre/lib/ext directory or by directly modifying the soapui.bat/.sh files to include the required jar files
- Connection String - the connection-string to use
- SQL Query - the query to issue for the created connection. It must return at least the same number of rows as specified in the Properties List where there must be a column with the same name as its corrsponding property.

XML DataSource

The XML DataSource has the following configuration panel:



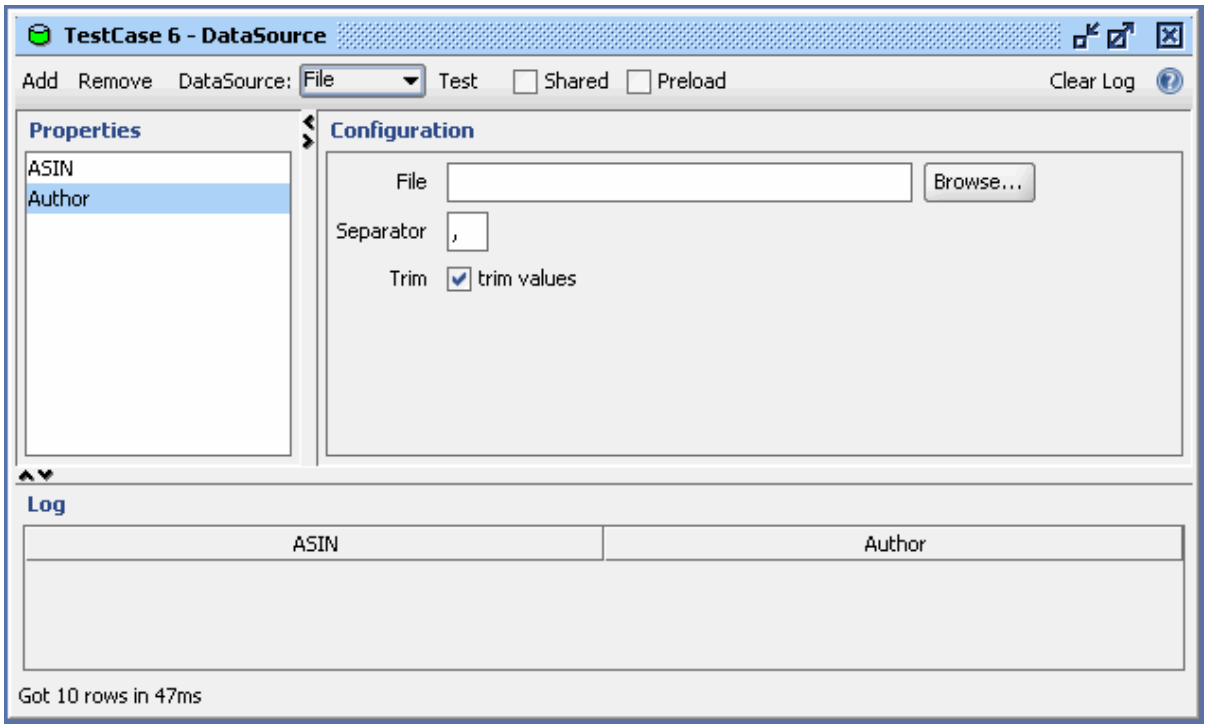
The available options are:

- Source Step - the TestStep containing the XML property to read from (could be another DataSource)
- Source Property - the Property containing the XML to use
- Row XPath - the XPath expression to use for selecting the "Rows" for the data
- Column XPaths - one XPath expression for each DataSource property relative to the Row XPath

During execution, the Row XPath will be selected during startup and each rows property values will be selected relative to the current row node (as shown in the above screenshot)

File DataSource

The File DataSource has the following configuration panel:

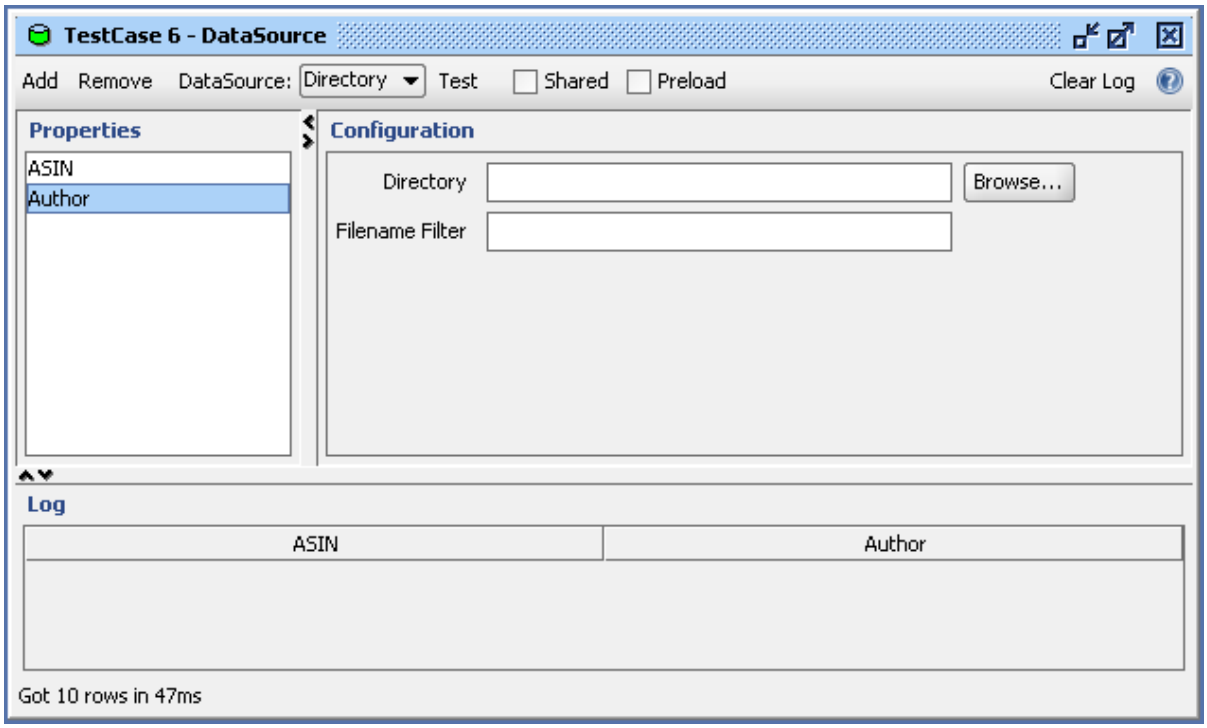


Files are expected to be in a simple row format (for example comma-separated), where each row contains the supplied data values seperated by a configurable separator. The available options are:

- File - the file to read
- Separator - the seperator for the columns in each row
- Trim Values - values will be trimmed after reading

Directory DataSource

The Directory DataSource has the following configuration panel:



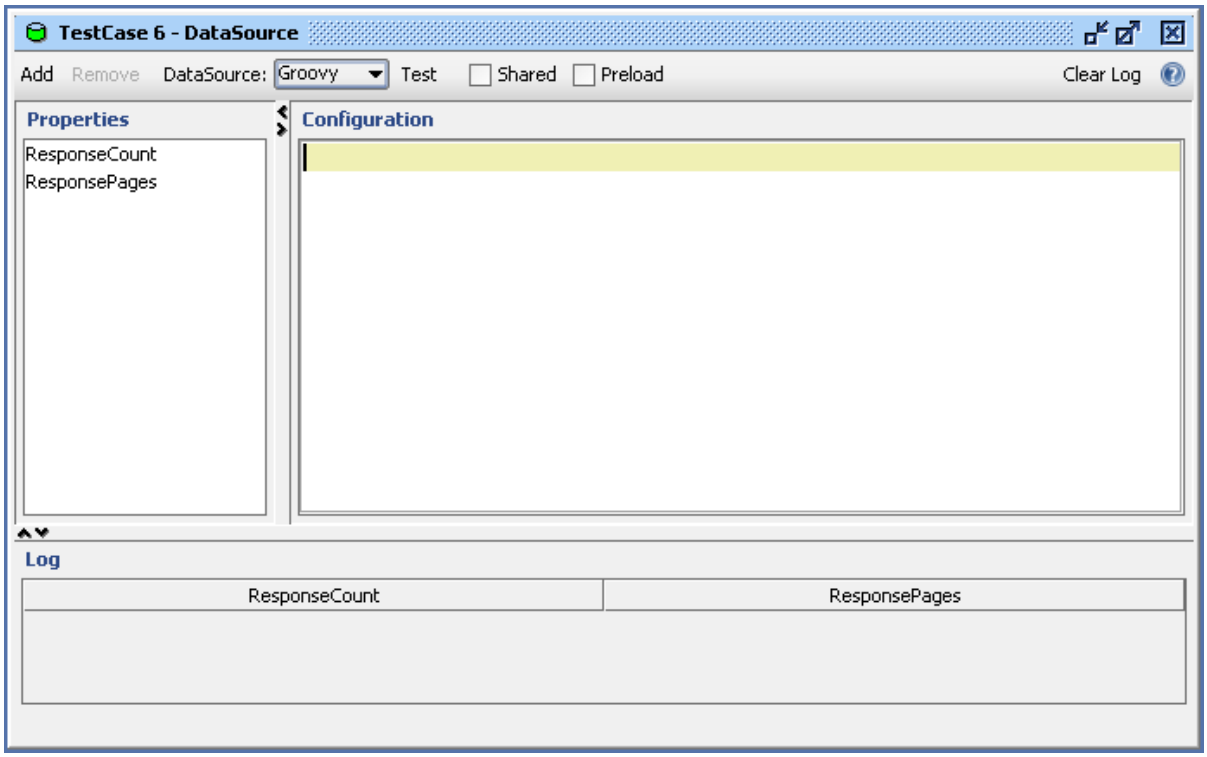
The directory DataSource has 2 options:

- Directory : the Directory to scan for files
- Filename Filter : a standard FileName filter used to narrow down which files to read

A DataSource Step with a Directory DataSource need only to define one property which will contain the entire file as read from disk. This could then be input to a request body or a nexted DataSourceStep that uses the XML DataSource to extract data from the read file.

Groovy DataSource

The Groovy DataSource has the following configuration panel:



This step allows any kind of DataSource to be created for a DataSource Step. The groovy script must set desired properties in the available "result" object (a StringToStringMap), available context variables are:

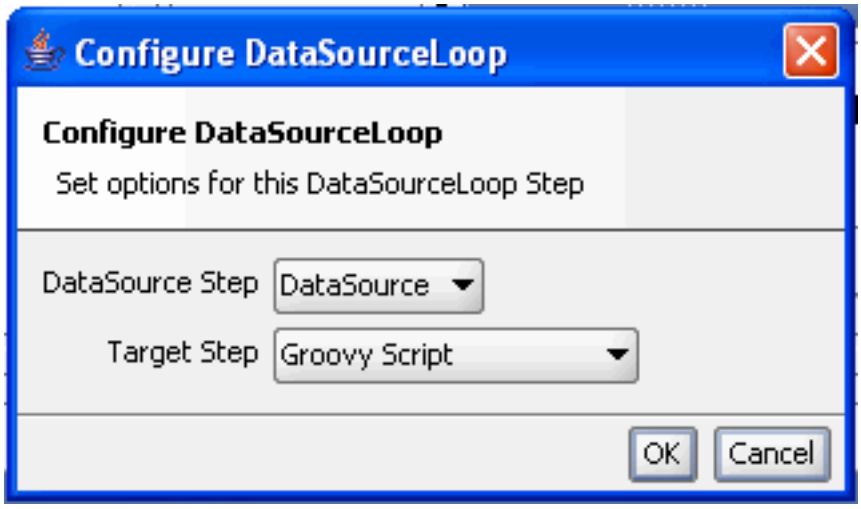
- "result" - for specifying result values
- "context" - the current TestRunContext for storing context-specific values
- "testRunner" - the current TestRunner object

Next:

1.6.3.9 DataSource Loop

DataSourceLoop Step

The DataSourceLoop Step is used together with the DataSource Step to create a sequence of steps that should be executed for each row of data. It must be placed after its corresponding DataSource step and is configured by either double-clicking or selecting "Configure" from its popup menu, which displays the following dialog:



The dialog contains 2 options:

- DataSource Step - selects the DataSource Step to check for more data
- Target Step - selects the Step to loop to if more data is available

For example, if a DataSource step is used to read data from an external file and the followed by a request using that data, a DataSource Loop should be added after the request specifying the DataSource Step and the Request step as its target step.. (ie loop back to the request if more data is available).

Next:

1.6.4 Property Expansion

Property Expansion

soapUI provides a common `${[testStepName#]propertyName[#xpath-expression]}` syntax to dynamically insert values into a number of values during processing. Currently, properties are expanded in:

- request messages
- mock response messages (from the MockRunContext see Response Scripts), testStepName is ignored.
- xpath assertions and their matching values
- xpath source/target property transfers
- contains/not-contains content assertions
- request endpoints
- custom request/mock-response http headers
- property-transfer xpath expressions
- datasource step configuration values (soapUI Pro only)
- mockresponse step response messages (soapUI Pro only)

When properties are expanded, soapUI will look for the named property as follows:

1. if the property contains the name of a testStep, query that step directly
2. in the current run/submitcontext where properties can have been set from a groovy-script
3. in all teststep properties starting from the current step backwards to the first if the "Search Properties" option has been selected in the TestCase Options dialog)

For example, the following request gets the Author property as described above, either from a preceding PropertiesStep containing the corresponding property or from the current context set by a preceding Groovy Script :

```
<ns:Request>
   <ns:Author>${Properties#Author}</ns:Author>
</ns:Request>
```

Since the value of the property will be set just before it is used, the actual used value will not be seen in the corresponding editors.

If the property expansion further includes an xpath expression, this will be selected from the property value, for example the above example could "extract" the author value from a preceding response with:

```
<ns:Request>
   <ns:Author>${Search
Request#Response#//ns1:Item[1]/n1:Author[1]/text()}</ns:Author>
</ns:Request>
```

Which would first get the "Response" property of the "Search Request" step and then select the value of the first Items' first Author element. Note that the namespace prefix must match those used in the response message, otherwise the expansion will fail.

Property Transfers are a more tedious way of accomplishing the same functionality as with property-expansion. On the other hand, property transfers provide the possibility to transfer complex content between request/response messages. Also, the result of a Property Transfer is visible directly in the corrsponding request/response editors.

Available Properties

The table below lists all properties available for property-expansion and property-transfers

Request	The configures request message
Response	The last response message (read-only)
Endpoint	The current endpoint for the request
Username	The current username for the request
Password	The current password for the request
Domain	The current domain for the request
Properties Step	
<any defined property>	the properties' value
GroovyScript Step	
result	the value returned by the script from its last run (read-only)
DataSource Step (soapUI Pro only)	
<any defined property>	the properties' value (read-only)
MockResponse Step (soapUI Pro only)	
Request	The last request message (read-only)
Response	The configured response message

Next: LoadTesting

1.7 Load Testing

Load Testing

soapUI provides extensive load-testing functionality allowing you to do the following:

- Functional LoadTesting : validate functionality under load using standard TestCase methods
- Behavioral LoadTesting : analyze performance behaviour under varying load with different load strategies
- Performance LoadTesting : find maximal performance available using thread strategies and commandline execution
- Requirements-Driven LoadTesting : define performance requirements and continuously validate using load-test assertions

Any number of LoadTests can be created for a TestCase (using the TestCases "New LoadTest" popup-menu action), each with different strategies, assertions, etc. to validate/assess a TestCases and its TestSteps performance under different circumstances.

The documentation for load-testing in soapUI has been split into the following documents:

- This document gives a background on the soapUI approach to loadtesting and an overview of the LoadTest Editor
- Configuration : describes limits and strategies
- Execution : describes the execution of LoadTests
- Assertions : specifies the available LoadTest assertions and how they are used
- Diagrams : describes the available diagrams during LoadTesting
- JMeter Comparison : a comparison with the popular JMeter tool

Requirements-Driven Load Testing

One of our main objectives with the load-testing functionality in soapUI was to implement a "requirement-driven" approach to load-testing. Far too often, load-testing (in our experience) is performed to see "how fast" a certain web service / business process is. Although this may be interesting from a technical (and sometimes business) point of view, more often it is important the web service is "fast enough" for the actual business processes being realized, which is usually far below the actual performance achievable. Although this "fast enough" is usually very hard to define by the business owner, we believe that this is none-the-less the best approach when assessing the performance of a web service and/or its environment.

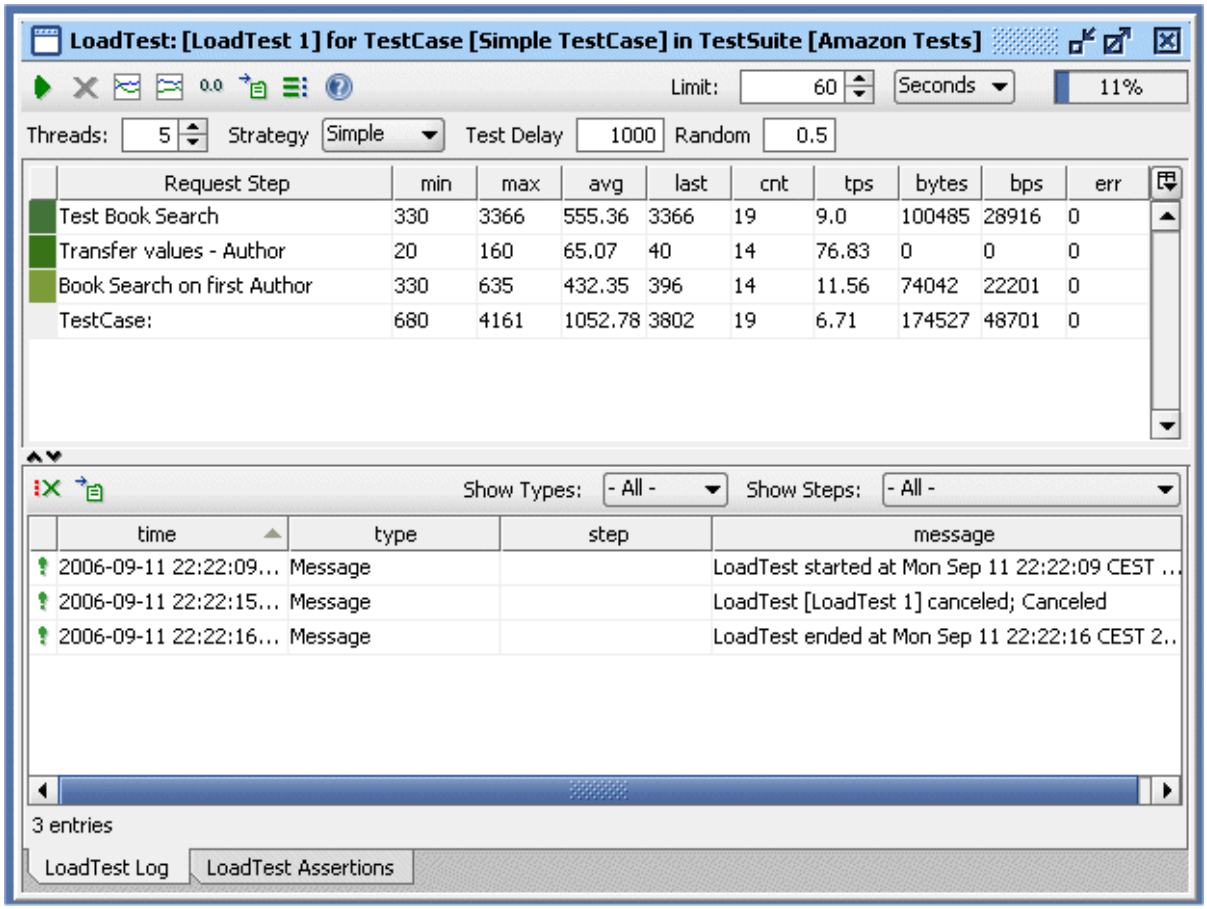
Based on this approach, soapUI allows you to define a number of LoadTest Assertions that are continuously applied to an ongoing LoadTest to validate that it performs "as required", for example the average response time of a request can be asserted to not go under a specified value for a longer period of

time. Other assertions available include max-response time, TPS, etc...

This functionality can further be used for "surveillance testing", where a number of LoadTests are run periodically using some scheduling tool to assess that individual web services (for example in a SOA or an integration API) continuously perform as required. A setup for this is described in the Surveillance Testing document.

The LoadTest Editor

The LoadTest Editor gives an overview of the current LoadTest configuration, results and events:



The editor contains the following components (top-down)

- The LoadTest toolbar containing a number of actions and the Test Limit settings for this LoadTest
- The Load Strategy toolbar containing settings for the selected Load Strategy
- The Statistics Table showing up-to-date statistics on the current LoadTest
- A Tabbed Pane containing two tabs: a "LoadTest Log"-tab showing log-events for the current LoadTest and a "LoadTest Assertions"-assertions where the assertions for this LoadTest can be configured

The LoadTest Toolbar

The main toolbar contains the following actions (left to right):

- **Run** : Starts the LoadTest as described under LoadTest Execution
- **Cancel** : Cancels an ongoing LoadTest
- **Statistics Graph** : Show the Statistics Graph for the LoadTest
- **Statistics History Graph** : Show the Statistics History Graph for the LoadTest
- **Reset Statistics** : Resets the statistics for an ongoing LoadTest
- **Export Statistics** : Prompts to export the current LoadTest Statistics to a comma-separated file
- **Options** : Shows the LoadTest Options dialog
- **Limit Settings** : Sets the limit for the LoadTest as described in the Execution document
- The far right contains a Progress Bar displaying the progress (in percent) of the current LoadTest exection

The LoadStrategy toolbar is described in the Execution document.

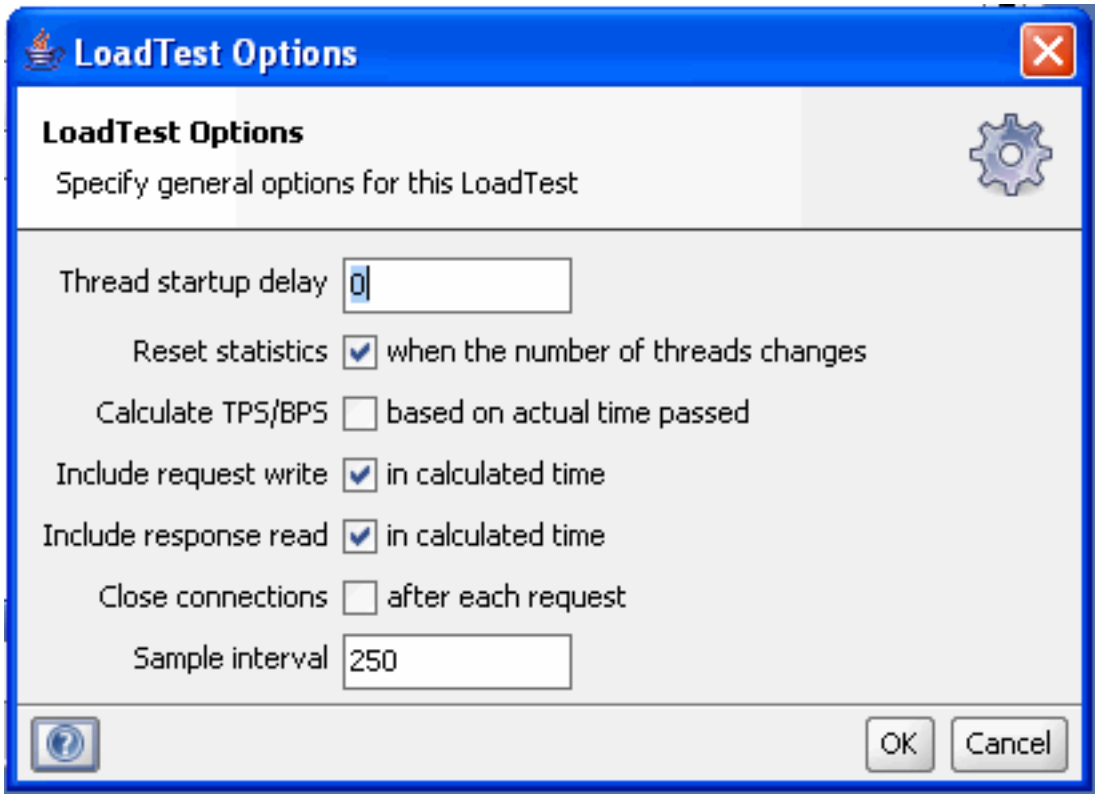
LoadTest Options

The LoadTest Options dialog contains the following settings:

- **Thread Startup Delay** : Sets the startup delay for each thread (in milliseconds), setting to 0 will start all threads simultaneously
- **Reset Statistics when thread-count changes** : Automatically resets statistics when the number of threads changes. Since (for example) the average is calculated using the number of threads, this value will be "influenced" by previous results from a different thread-count, which can be avoided by resetting the statistics when the number of threads changes
- **Calculate TPS/BPS based on actual time passed** : By default, TPS (Transactions per second) is calculated with ((1000/avg)*threadcount), see Calculation of TPS/BPS . When setting a testcase delay using the Simple LoadStrategy, the avg will generally be very low, but the actual transactions per second will not be equivalently high (since there is a delay). Selecting this option will calculate TPS using (time-passed/cnt) instead
- **Include request write in calculated time** : When selected, the time to establish the connection with the target endpoint and write the HTTP request will be included in the calculated time for Request test steps. Select this option if you want the "actual" request time measured (includes proxy, requests, authentication challenges, etc)
- **Include response read in calculated time** : When selected, the time to read the HTTP response will be included in the calculated time for Request test steps. If not selected, only the time for reading the response HTTP headers will be included (the response content will still be read for assertions and eventual results viewing)
- **Close Connections after each request** : Select this if you want to disable Keep-Alives/Connection reuse, which will result in a loadtesting scenario which resembles an environment with many different Web Service clients
- **Sample Interval** : Sets the sample-interval for the LoadTest Statistics table (in milliseconds), default is 250ms.

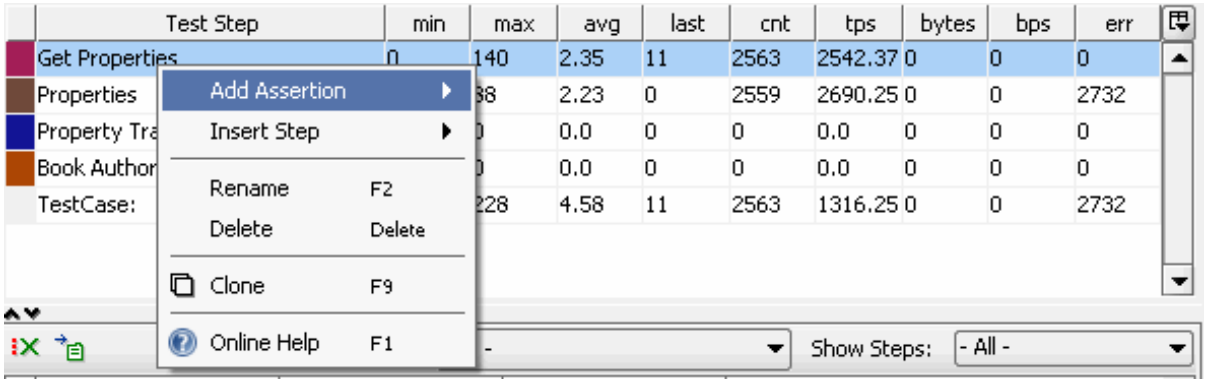
The effect of these last three options on LoadTest results can be rather substantial for response times

(depending on network configuration, etc) and is illustrated and discussed in the JMeter Comparison document.



The Statistics Table

The statistics table shows real time statistics for each TestStep in the underlying TestCase and for the TestCase total.



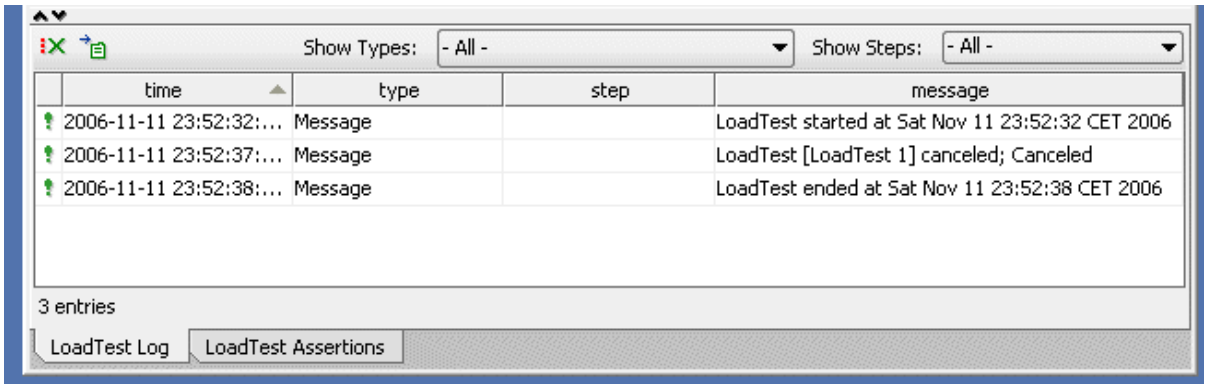
The Statistics currently collected are described in detail in the LoadTest Execution document.

Double-clicking a TestStep row in the table opens that TestSteps associated editor.

Right-clicking on a TestStep shows a popup-menu show TestStep specific actions and actions to add LoadTest Assertions for the selected TestStep as described under LoadTest Assertions .

The LoadTest Log

The LoadTest Log displays execution status messages and errors reported by the LoadTest Assertions. Double-clicking an error will open the errors result viewer (if available), for example allowing you to see the actual request/response that generated the error.



The top toolbar contains the following actions (left-to-right):

- **Remove Errors** : removes all errors from the LoadTest Log
- **Export** : prompts to export the current LoadTest log to a file
- **Show Types filter** : filters which type of errors/messages that should be shown in the log
- **Show Steps filter** : filters which steps that should be shown in the log

The table is sortable by clicking the column-header for the column to be sorted on. A label under the table displays the number of rows currently in the table.

Next: LoadTest Configuration

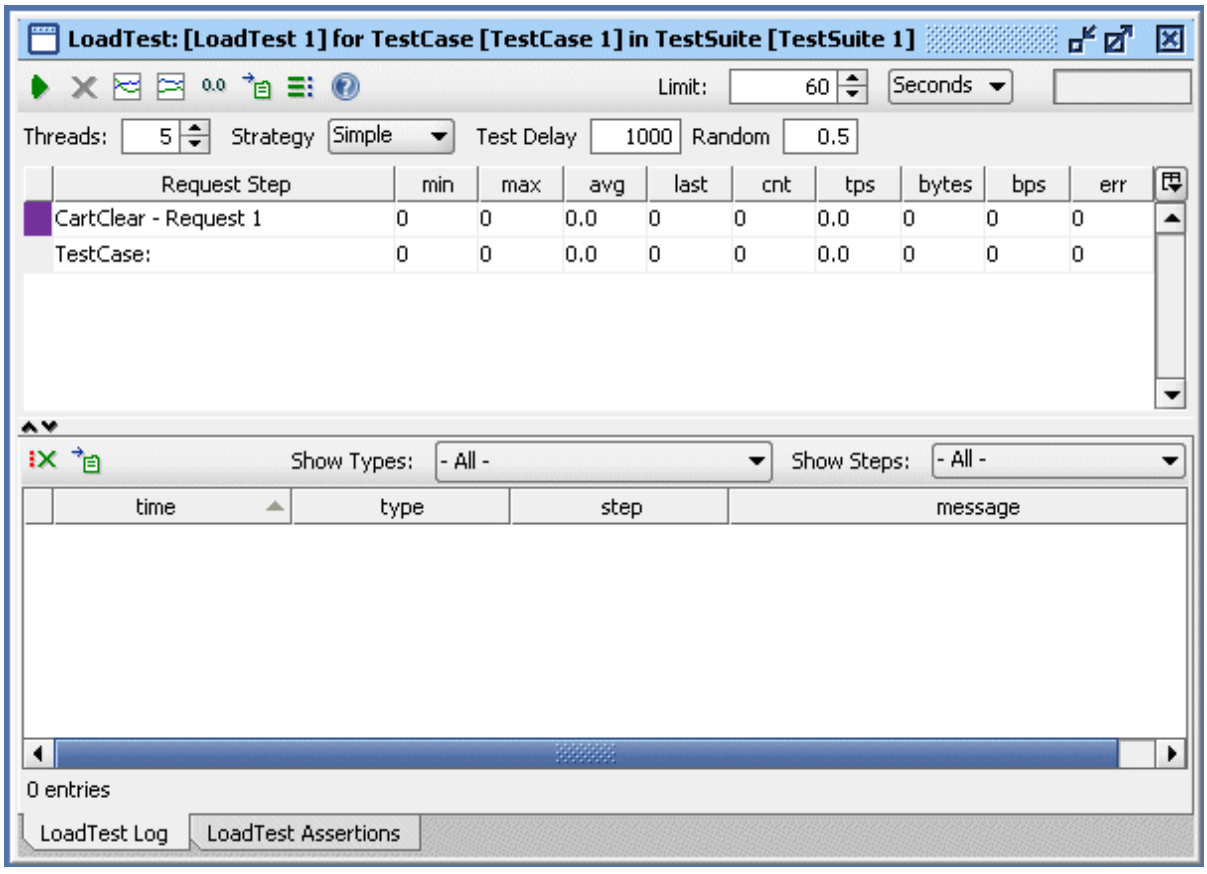
1.7.1 Getting Started

Getting Started with Load Testing

soapUI 1.5 introduces fairly powerfull loadtesting functionality allowing you to:

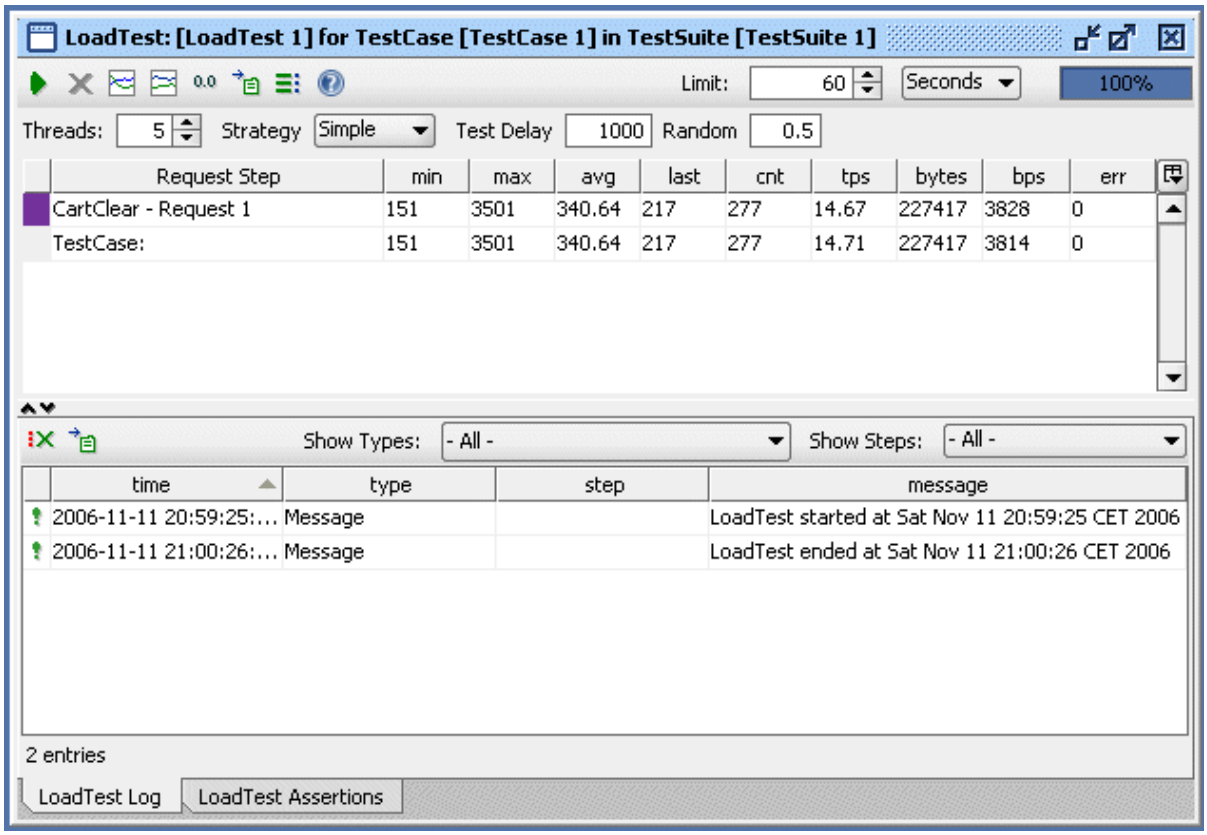
- Validate a web servics performance under different Load scenarios
- Maintain Functional validations to see that they dont "break" under load
- Run several load-tests simultaneously to see how they affect each othter

Here we will continue from the Functional TestCase created in the previous "Getting Started" document. Start by creating a LoadTest for the created TestCase using its "New LoadTest" popup menu option. The opened LoadTest Editor should be something like the following (depending on which steps you created in your TestCase):



The LoadTest is preconfigured to run for 60 seconds (top right) with 5 threads using the Simple LoadStrategy. Modify these values as desired (read more about LoadTest Configuration) and run the test. You will see the statistics table in the middle beginning collecting data and after 60 seconds should have a

finished LoadTest as follows (read more about LoadTest Execution):



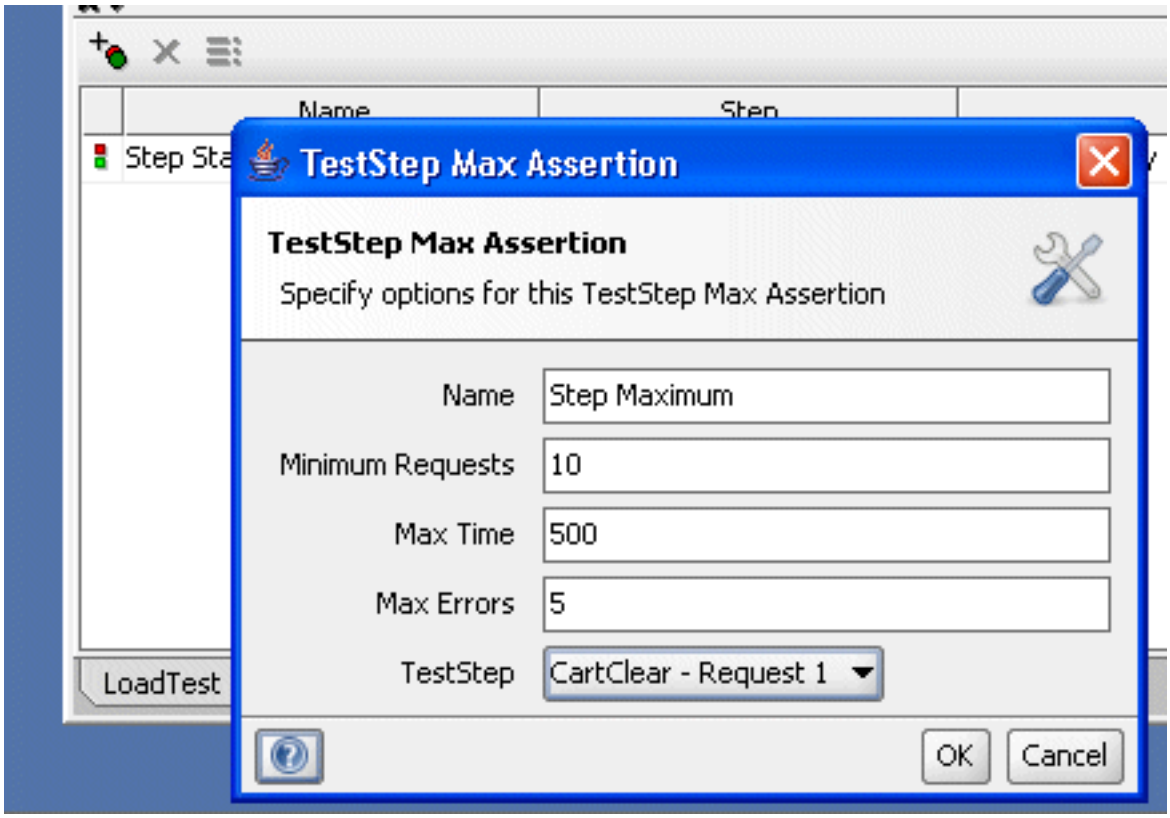
Add Assertions

soapUI allows you to add LoadTest Assertions to LoadTests just as to Functional TestSteps. When creating a LoadTest, soapUI will always add a TestStep Status Assertion for us, we will add a TestStep Max Assertion to validate that our TestCase never exceeds a certain exection time.

Select the "LoadTest Assertions" tab at the bottom of the LoadTest Editor and press the displayed "Add Assertion" button. Select the "Step Maximum" assertion and configure it as follows in the displayed dialog:

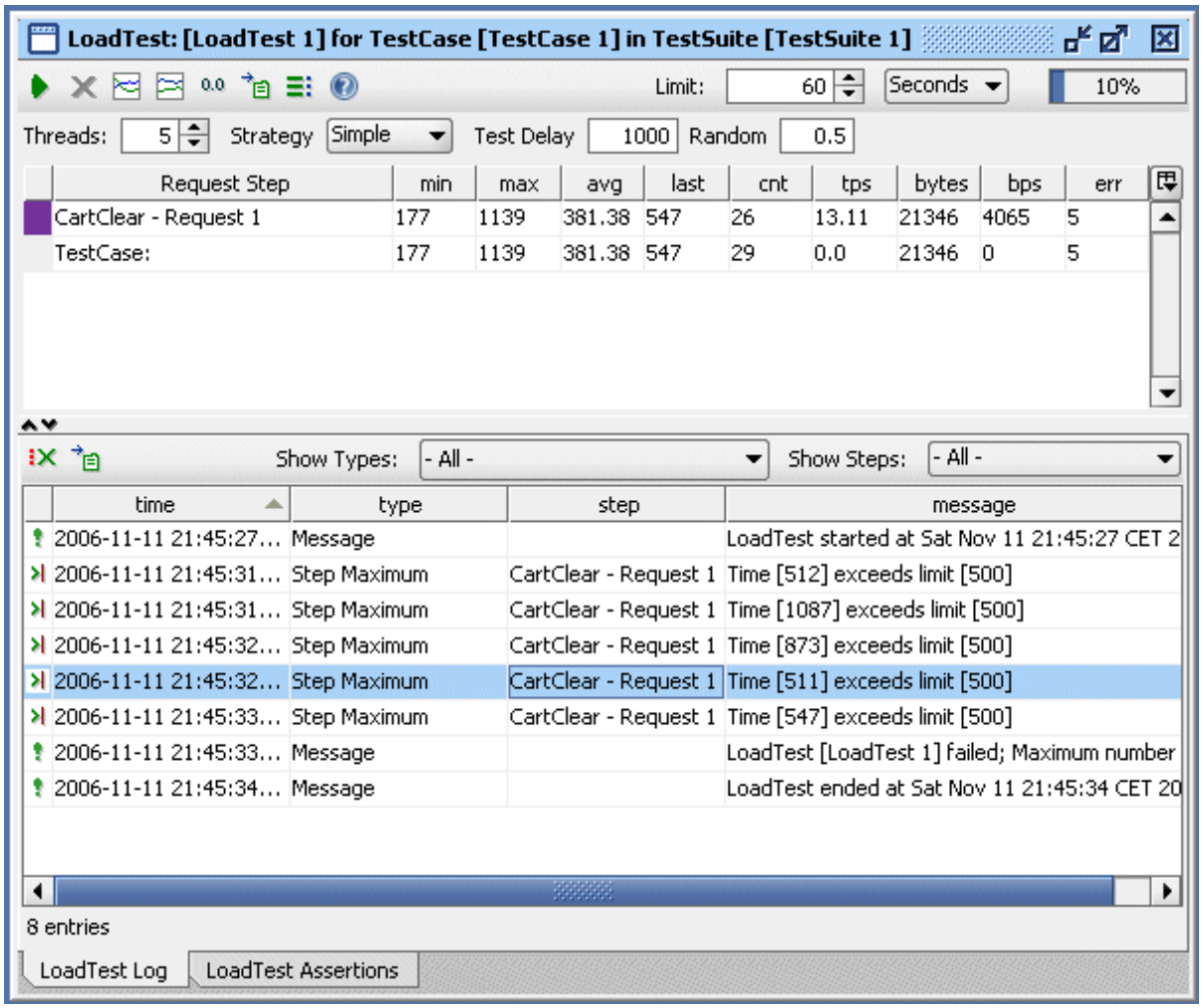
The Step Maximum assertion checks that a steps max-time does not exceed a specified value.

- **Name** - the name of the assertion, leavt it for now
- **Minmum Requests** - the minimum number of runs that must have been executed before applying this assertion. We will allow the first 10 TestCase runs to pass.
- **Max Time** - the maximum allowed step time, we set it to 500 ms
- **Max Errors** - the maximum number of errors to allow before cancelling the LoadTest, we will allow 5 "breaches" of the 500ms limit before the LoadTest should fail
- **TestStep** - the target step to assert. We select the request in our TestCase

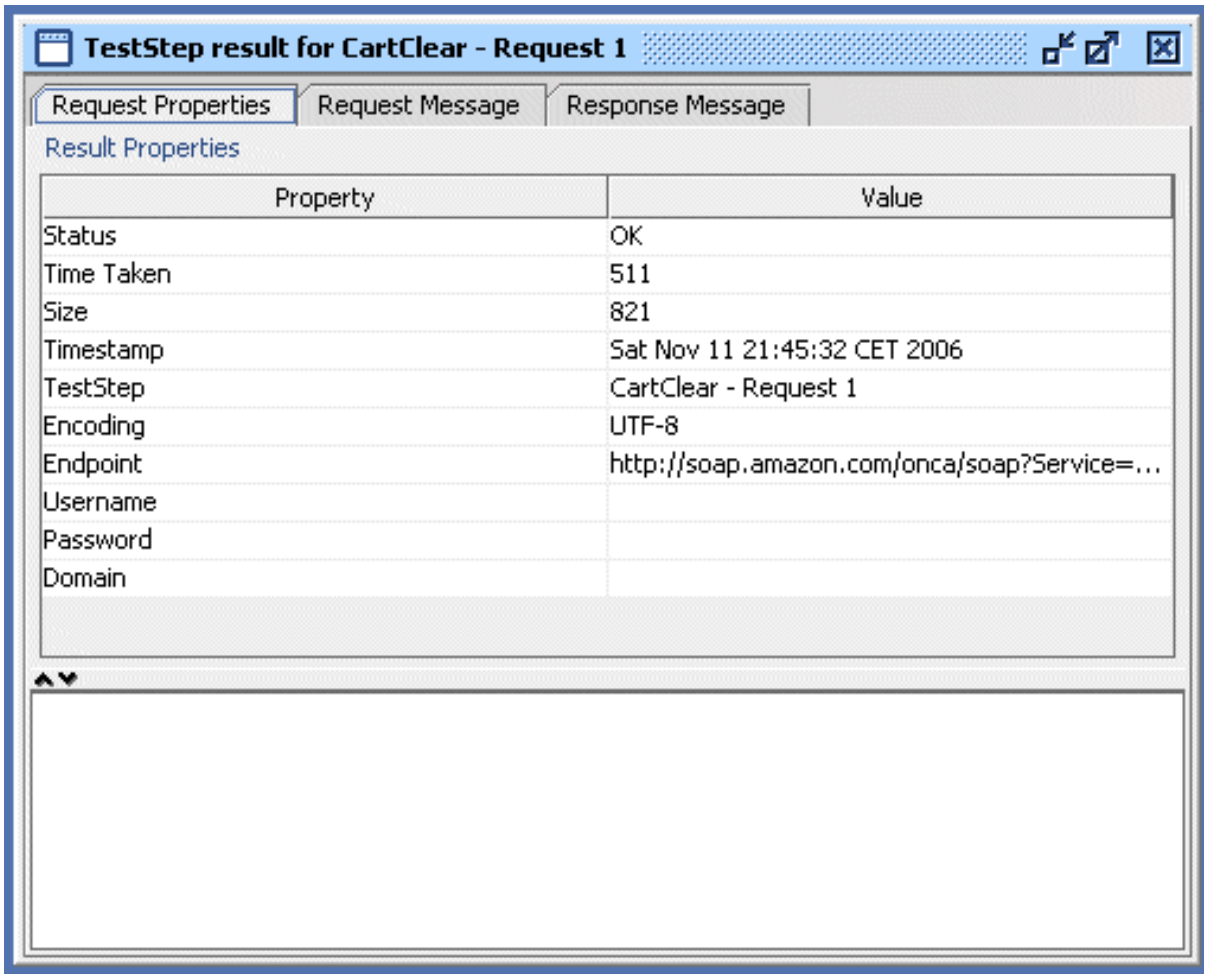


Run again!

Now running the loadtest might give you a result like to following where the TestCase 5 times exceeded the limit of 500ms and failed the LoadTest.



You can double-click an assertion failure in the log and view the actual request that failed, allowing you to debug the request/response message, etc.. Double-clicking one of the errors above shows the following:



Now What?

Try playing around with the different LoadTest Strategies, Diagrams , etc..

Next: Getting Started with Mocking

1.7.2 Limit & Strategies

LoadTest Configuration

The actual execution of a LoadTest is configured in two dimensions:

- The LoadTest Limit controls how long a LoadTest should run
- The LoadTest Strategies control how TestCases are scheduled and threaded (ie run in parallel)

LoadTest Limit

The LoadTest Limit is specified on the top right of the LoadTest Editor and consists of two settings:

- **Limit** : specifies for how many TestCase runs or seconds the LoadTest should run
- **Limit Type** : specifies how to limit the LoadTest; **Seconds** will run the LoadTest for the specified time, **Runs** will run the underlying TestCase for the specified number of times

Setting the Limit to 0 with either Limit Types will run the LoadTest indefinitely. A progress-bar to the right of the settings will show the current LoadTest progress.

LoadTest Strategies

A LoadTest can in soapUI be run using a number of strategies for how TestCases are executed. All strategies have a thread-count allowing the specification of how many TestCases that should be run in parallel. The current strategy is selected using the available Strategy combo, which will show that strategies settings to its right. Some strategies allow interactive changing of the thread-count while executing the LoadTest, others don't.

The following LoadTest Strategies are currently available:

- Simple : TestCase execution with a configurable delay
- Variance : TestCase execution varying the number of threads over time
- Burst : TestCase execution in "bursts"
- Thread : TestCase execution with a fixed thread-count modification

Since multiple LoadTests with different strategies can be created one could for example create one LoadTest with a simple "base" load using the Simple strategy and another LoadTest using the Burst strategy to monitor how base performance is affected when running both LoadTests simultaneously within soapUI or from the command-line.

Simple

The Simple Strategy has the following settings:

- **Test Delay** : sets a delay between each TestCase run
- **Random** : sets how much of the delay to randomize

Use this for simple load testing. For example setting the delay/random to 1000/0.5 will delay each threads TestCase execution by 500 to 1000 ms. Setting the delay to 0 will ignore the random setting and run without any delays.

If setting a delay be aware that the TPS statistic will give misleading results unless the "Calculate TPS on actual time passed" option is selected in the LoadTest Options dialog

Variance

The variance strategy has the following settings:

- **Interval** : sets the duration of each variance cycle in seconds
- **Variance** : sets how much the thread-count should be varied during the cycle

Use this strategy for simulating a linearly varying number of threads over time. The strategy will start at the currently configured thread-count, increase to current + (current*variance), decrease to current - (current*variance) and increase back to the current thread-count.

For example setting the thread-count to 20, duration to 480 and variance to 0.5, the strategy will increase from 20 to 30 threads in the first 2 minutes, decrease to 10 threads of the next 4 minutes and then increase back to the original 20 in the last 2 minutes.

Burst

The Burst strategy has the following settings:

- **Burst Delay** : sets the delay between each burst
- **Burst Duration** : sets duration for each burst

Use this strategy to simulate a "burst" load and for example monitor system behaviour during the "recovery" periods

For example setting the limit-type to seconds and limit to 600, the thread-count to 30, burst delay to 50 and burst duration to 10 will pause all 30 threads for 50 seconds and run them for 10 seconds repeatedly under 10 minutes.

Thread

The Threads Strategy has the following settings:

- **Start Threads** : sets the number of threads to start at
- **End Threads** : sets the number of threads to end at

Use this strategy to vary the number of threads from a start to end value linearly during the entire LoadTest. This can for example be useful to find the thread-count at which the best TPS is achieved.

For example setting the limit-type to seconds and limit to 600, the start threads to 1 and end threads to 30 will increase the number of threads from 1 to 30 over 10 minutes.

Next: LoadTest Execution

1.7.3 Execution

LoadTest Execution

When running a LoadTest soapUI internally creates a complete copy of the underlying TestCase for each thread, allowing each TestCase to maintain its own state and properties. Depending on which limit and strategy has been selected, loadtesting will proceed as configured until the LoadTest terminates due to one of the following:

- It has reached its configured limit
- It has been cancelled by the user with the "Cancel" button on the LoadTest toolbar
- It has been cancelled by a LoadTest Assertion when the maximum number of allowed errors for that assertion has been passed

During execution, the following statistics are periodically collected and displayed in the Statistics Table:

- `min` - the shortest time the step has taken
- `max` - the longest time the step has taken
- `avg` - the average time for the test step
- `last` - the last time for the test step
- `cnt` - the number of times the test step has been executed
- `tps` - the number of transactions per second for the test step, see Calculation of TPS/BPS below.
- `bytes` - the number of bytes processed by the test step
- `bps` - the bytes per second processed by the test step
- `err` - the number of assertion errors for the test step

Collection/calculation of this data is performed asynchronously (ie independently from the actual TestCase executions) in soapUI, so it should not affect the actual loadtesting performance.

The **Total** row shows the statistics for the TestCase itself.

Step Time calculation

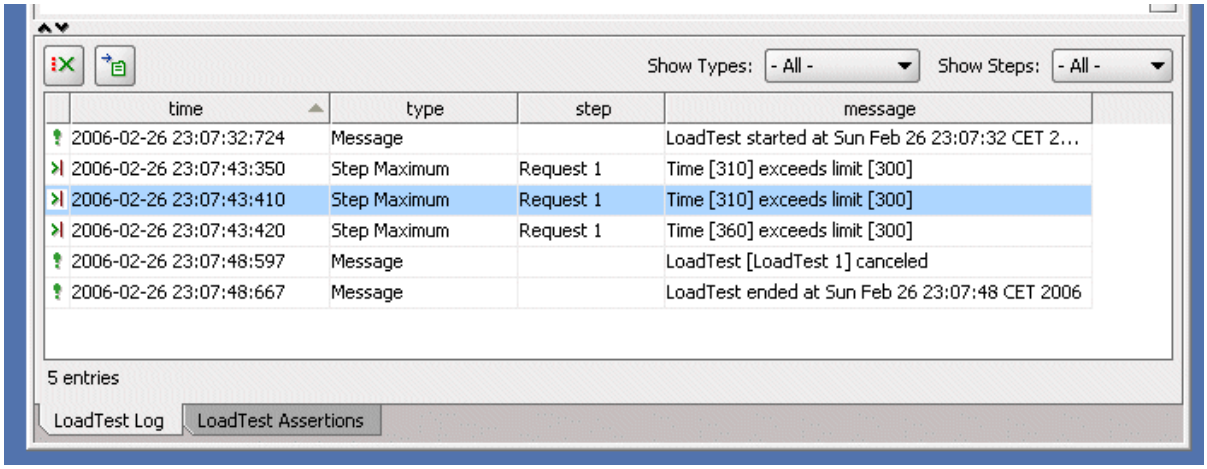
soapUI internally uses the `System.nanoTime()` method for determining the actual time taken by a test-step. The following table describes how these times are calculated for the available teststep types:

Step Type	Duration
Request Step	The request duration is calculated depending on the settings in the LoadTest Options dialog (as described there).

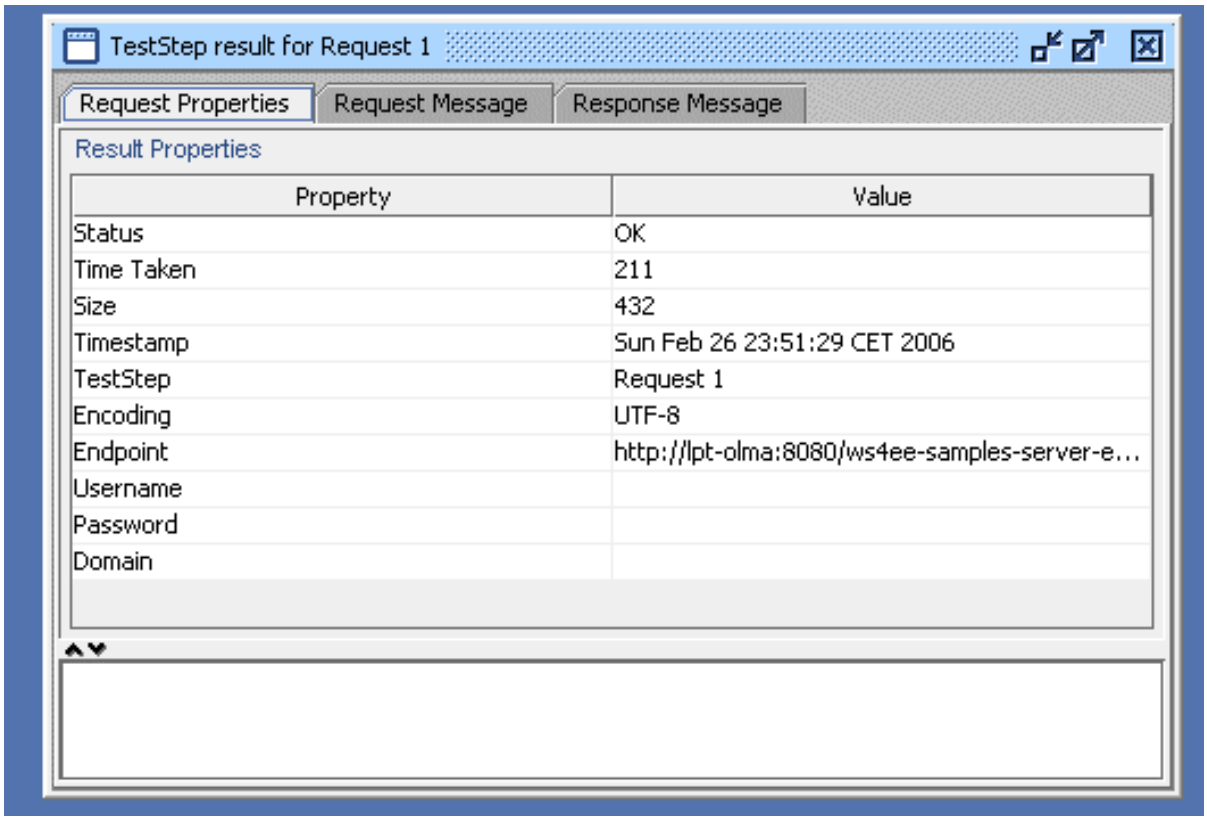
Step Type	Duration
Groovy Script	The actual script evaluation
Properties Step	The time to load/save the properties (if configured)
Delay Step	The steps delay
Property Transfer	The time it took to transfer all properties
Goto Step	The time it took to evaluate the goto-conditions

LoadTest Assertions

When running a LoadTest, the configured assertions are applied and will generate an error in the log if not met:



Assertion errors can be double-clicked and will show a view displaying the underlying TestStep state that failed the assertion. For example the following will be shown for Request that failed an assertion:



Calculation of TPS/BPS

Depending on the setting of the "Calculate TPS.." option in the LoadTest Options dialog, TPS and BPS are calculates as follows:

- NOT based on actual time passed (default):
  - TPS : (1000/avg)*threadcount, for example avg = 100 ms with ten threads will give a TPS of 100
  - BPS : (bytes/cnt) * TPS, ie the average number of bytes per request * TPS. For example a total number of received bytes of 1000000 for 100 requests with a TPS of 100 would give (100000/100 * 100) = 1000000 BPS
- Based on actual time passed:
  - TPS : Time passed in TestCase / CNT / 1000, ie a TestCase that has run for 10 seconds hand handled 100 request will get a TPS of 10
  - BPS : Bytes / Time passed / 1000, ie a TestCase that has run for 10 seconds and handled 100000 bytes will get a BPS of 10000.

  Please note the time-passed value used is the time for the entire testcase, not each step for itself. This can bias the TPS/BPS values quite a lot for those teststeps that take a small amount of the TestCases total time

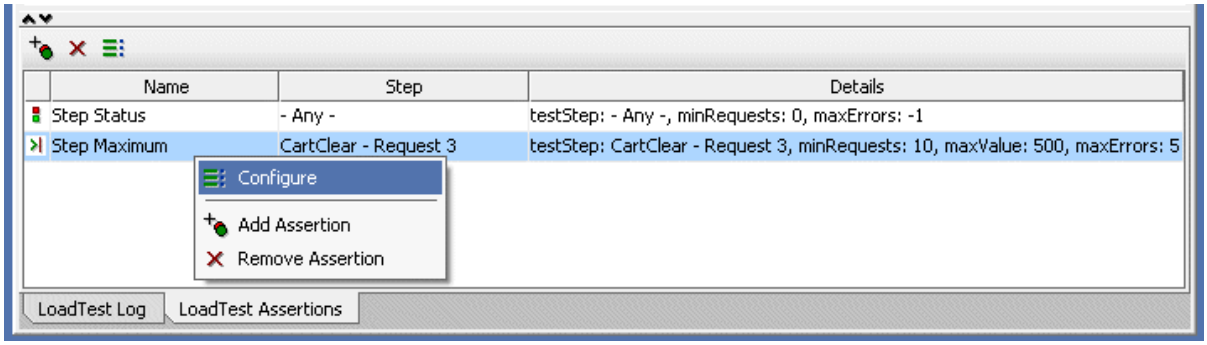
The option to base on actual time passed can be usefull when setting a testcase delay using the Simple LoadStrategy, which will generally give a lov average, but the actual transactions per second will not be equivalently high (since there is a delay). Selecting this option will calculate TPS using (time-passed/cnt) instead.

Next: LoadTest Assertions

1.7.4 Assertions

LoadTest Assertions

soapUI allows the creation of an arbitrary number of LoadTest Assertions for a LoadTest either from the LoadTest Statistics Popup Menu or from the "LoadTest Assertions" tab at the bottom of the LoadTest editor:



The toolbar at the top of the tab contains the following actions (left-to-right):

- **Add Assertion** - prompts to add and configure a new LoadTest Assertion
- **Remove Assertion** - prompts to remove the currently selected assertion
- **Assertion Options** - shows the options for the currently selected assertion

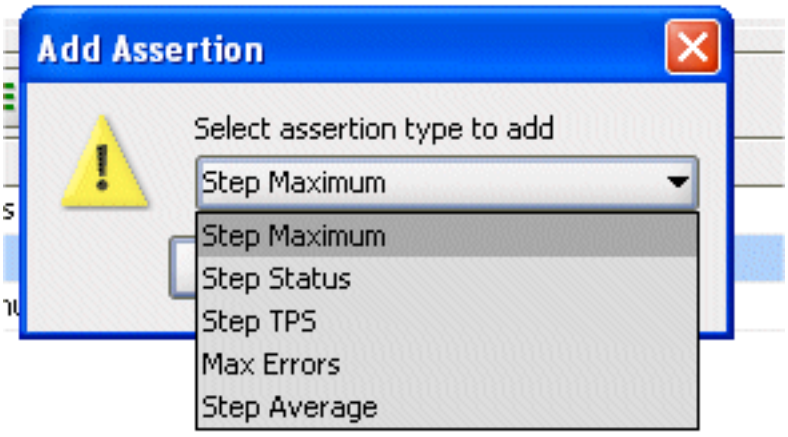
Below the toolbar a table containing the configuration assertions is shown with the following columns:

- **Name** : the configured name of the assertion
- **Step** : the target step for the assertion
- **Details** : detailed information on the assertion

A popup menu containing the same actions as in the toolbar is also available.

soapUI currently provides the following assertion types:

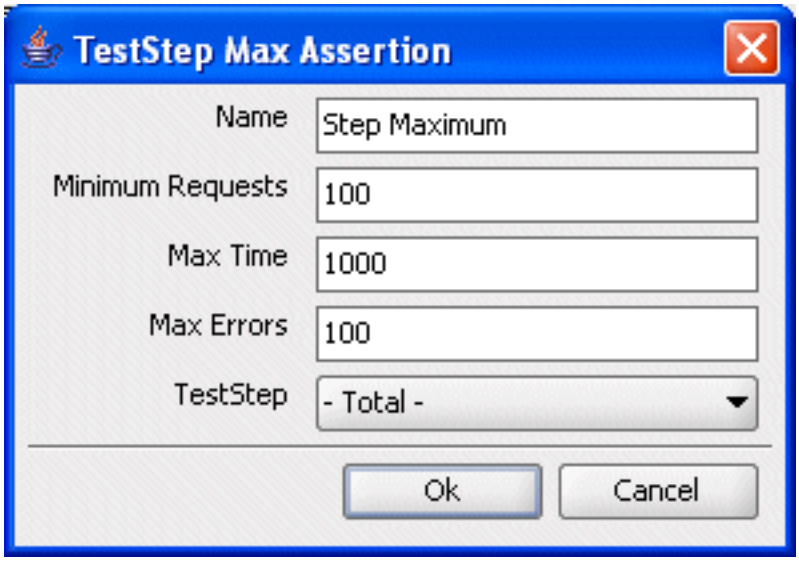
Type	Short Description
Step Maximum	Asserts a steps max time
Step Status	Asserts a steps status
Step TPS	Asserts a steps TPS
Max Errors	Asserts the total number of errors
Step Average	Asserts a steps average



Step Maximum Assertion

The Step Maximum assertion checks that a steps max-time does not exceed a specified value. It has the following settings:

- **Name** - the name of the assertion
- **Minimum Requests** - the minimum number of runs that must have been executed before applying this assertion. Use this for avoiding assertion errors during startup of a LoadTest
- **Max Time** - the maximum allowed step time. If this time is exceeded, an assertion error is logged to the LoadTest log
- **Max Errors** - the maximum number of errors to allow before cancelling the LoadTest
- **TestStep** - the target step to assert. Selecting "- Any -" will assert all steps, selecting "- Total -" will assert the total time taken for the entire TestCase run

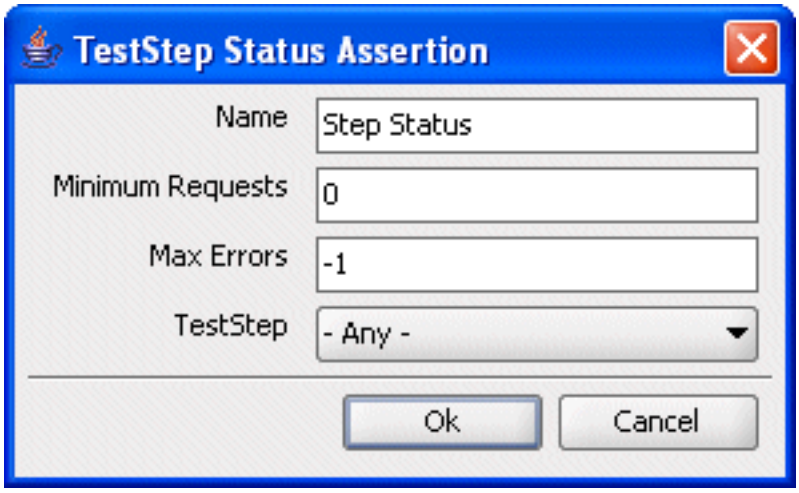


Step Status Assertion

The Step Status Assertion checks that a step has not failed with an error, for example a Request steps or a Groovy script step. It has the following settings:

- **Name** - the name of the assertion

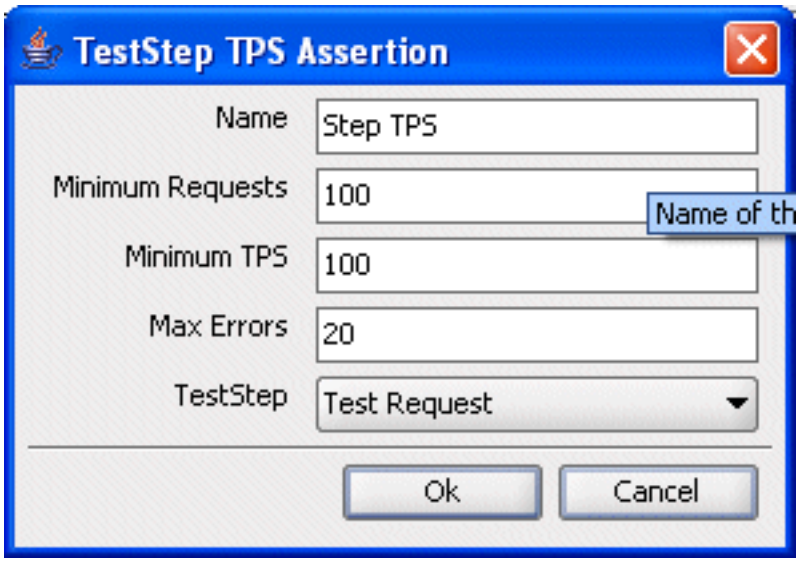
- **Minimum Requests** - the minimum number of runs that must have been executed before applying this assertion. Use this for avoiding assertion errors during start up of a LoadTest
- **Max Errors** - the maximum number of errors to allow before cancelling the LoadTest
- **TestStep** - the target step to assert. Selecting "- Any -" will assert all steps



Step TPS Assertion

The Step TPS Assertion checks that a steps TPS does not go below a specified value. It has the following settings:

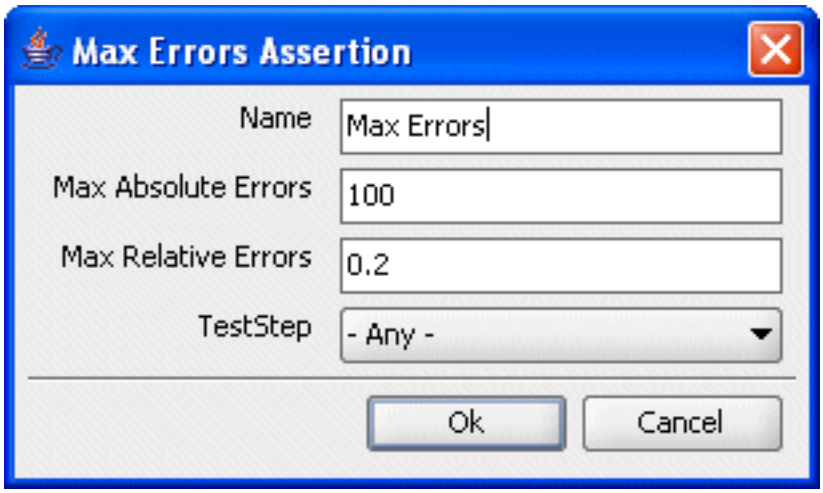
- **Name** - the name of the assertion
- **Minimum Requests** - the minimum number of runs that must have been executed before applying this assertion. Use this for avoiding assertion errors during start up of a LoadTest
- **Minimum TPS** - the minimum required TPS. If the actual TPS is lower, an assertion error is logged to the LoadTest log
- **Max Errors** - the maximum number of errors to allow before cancelling the LoadTest
- **TestStep** - the target step to assert. Selecting "- Any -" will assert all steps, selecting "- Total -" will assert the total TPS for the entire TestCase run



Max Errors Assertion

The Max Errors Assertion checks that the total number of errors for a specified TestStep does not exceed a specified value. It has the following settings:

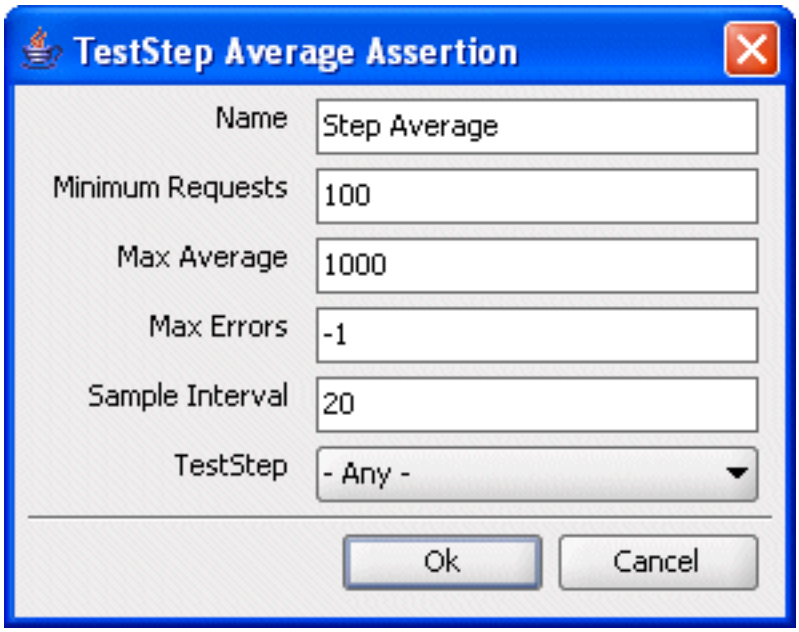
- **Name** - the name of the assertion
- **Max Absolute Errors** - the absolute maximum allowed number of errors for the specified TestStep
- **Max Relative Errors** - the relative maximum allowed number of errors for the specified TestStep, for example 0.2 will assert that at most 20% of the runs for the specified TestStep result in an error
- **TestStep** - the target step to assert. Selecting "- Any -" will assert all steps, selecting "- Total -" will assert the total number of errors for the entire TestCase run



Step Average Assertion

The Step Average Assertion checks that a steps average does not go over a specified value. It has the following settings:

- **Name** - the name of the assertion
- **Minmum Requests** - the minimum number of runs that must have been executed before applying this assertion. Use this for avoiding assertion errors during start up of a LoadTest
- **Max Average** - the maximum allowed average. If the actual average is higher, an assertion error is logged to the LoadTest log
- **Max Errors** - the maximum number of errors to allow before cancelling the LoadTest
- **Sample Interval** - the sample interval
- **TestStep** - the target step to assert. Selecting "- Any -" will assert all steps, selecting "- Total -" will assert the total average for the entire TestCase run



Next: LoadTest Diagrams

1.7.5 Diagrams

Diagrams

soapUI provides 2 diagrams for behavioural analysis of LoadTest Statistics over time (neither provides any exact values/legends). These are:

- A **Statistics** diagram showing all statistics for a selected TestStep
- A **Statistic History** diagram showing a selected Statistic for all TestSteps

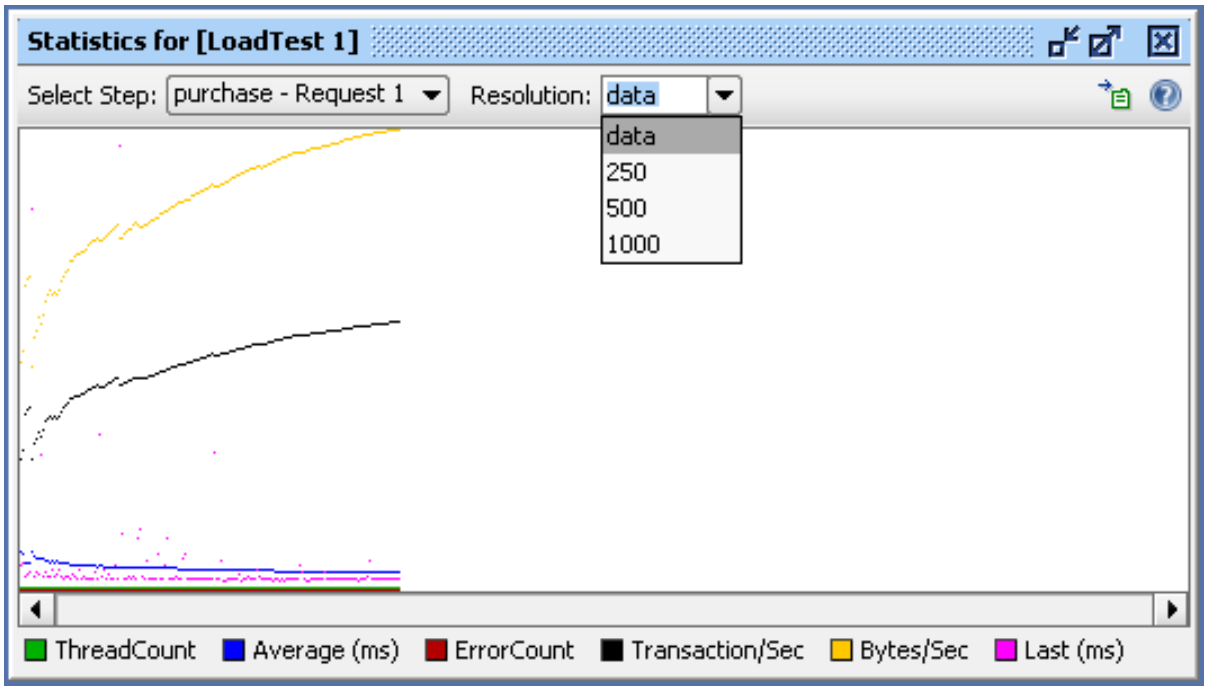
Both diagrams share a "Resolution" setting which controls how often the diagram should be updated. This can be set either to "Data" or a number of milliseconds:

- **Data** - means that the diagram will be updated whenever the main LoadTest Statistics are updated, which in turn is controlled by the "Sample Interval" setting in the LoadTest Options dialog. This is the default setting and sufficient if you have a high throughput of TestCase runs (which will constantly update the Statistics. If you have a LoadTest that has long-running TestCases, this option will result in a diagram that updates very infrequently which may both be confusing and not what is desirbed.
- **XXX** - an interval in milliseconds, the diagram will be updated every XXX ms regardless of if the underlying Statistics have changed or not.

Be aware of the fact that all statistics for these diagrams are always collected internally, ie you can open the diagram after running your LoadTests and see the results. This is recommended as it does not take any performance from the UI during LoadTesting.

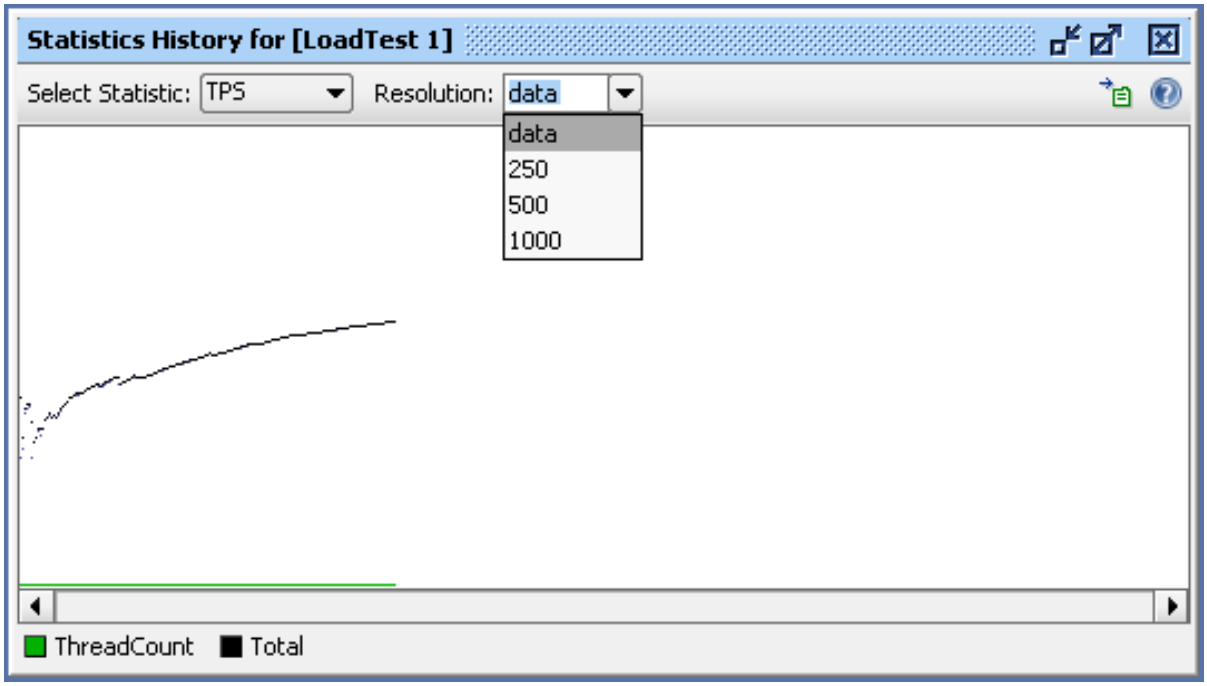
The Statistics Diagram

The statistics diagram shows a number of statistics for a selected TestStep or the TestCase Total. The toolbar at the top contains a combo-box for selecting which step to display and a button to the far right for exporting the currently selected steps data to a comma-separated file.



The Statistics History Diagram

The statistics history diagram shows a selected statistic for all TestSteps in the TestCase. The toolbar at the top contains a combo-box for selecting which statistic to display and a button to the far right for exporting the currently selected statistics data to a comma-separated file.



Next: JMeter Comparison

1.7.6 JMeter comparison

JMeter Comparison

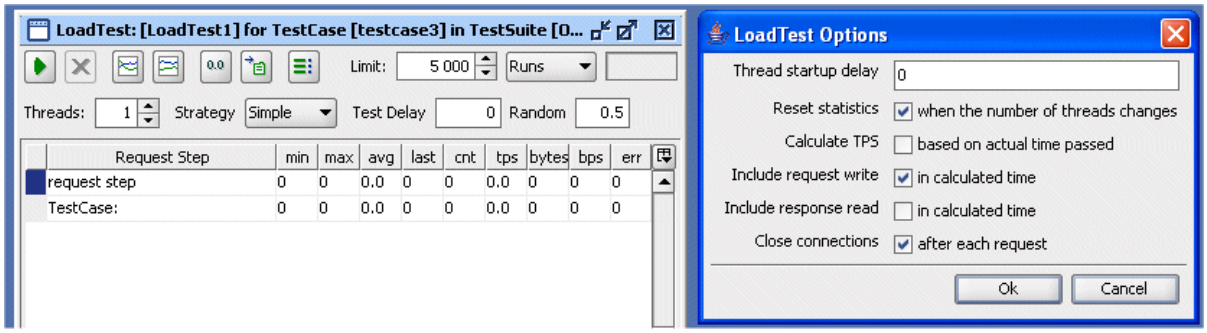
This document aims mainly to compare the results obtained with the popular JMeter tool from apache with those obtained with soapUI and to show and discuss some of the differences in the results obtained with the 2 tools. If you think/know we are doing something wrong in our tests or conclusions, don't hesitate to contact us and we will try to sort things out :-)

A feature comparison will between the two tools will not be done in any detail, generally one could say that JMeter is much better at load-testing in general (ie HTTP, JDBC, JMS, etc, etc) while soapUI is better at load-testing web services specifically. Most things you can do with soapUI can probably be done in JMeter, but since JMeter has a more "generic" approach to almost everything, it won't always be as "intuitive" as it is (or at least should be) in soapUI. On the other hand, many things that can be done in JMeter are not possible in soapUI (for example distributed load testing).

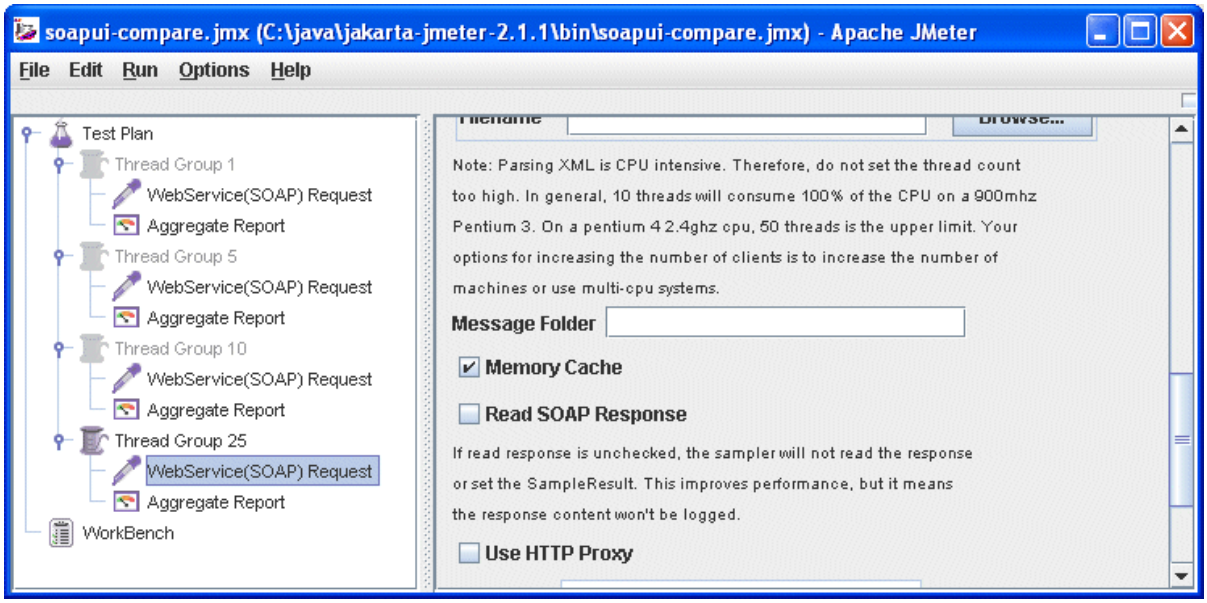
In general one can say that Load-testing web services is extremely difficult since so many factors can affect the measured results; network latency, host-load, hardware, jvms, etc.. Both soapUI and JMeter will be a "victim" to this, although soapUI has slightly more options to control exactly what and how results are to be measured (in the TestCase options dialog).

Comparison Setup

The setup for the comparison is a simple request to a web service running locally under an improved version of JBoss 4.0.3SP1 and running a slightly modified sample Web Service from the jbossws 4.0.3 samples. A simple request will be run 5000 times under 4 different loads; 1 thread, 5 threads, 10 threads and 25 threads all using the simple strategy with no delay, the TestCase Options are configured to include only request write times, and will be run with and without closing connections after each request.



This is in JMeter is configured equivalently using one ThreadGroup with no ramp-up and varying the loop count and number of threads accordingly; 1/5000, 5/1000, 10/500 and 25/200. The "WebService(SOAP) Request" sampler is used and configured as shown below. An "Aggregate Report" listener is configured for collecting the results.



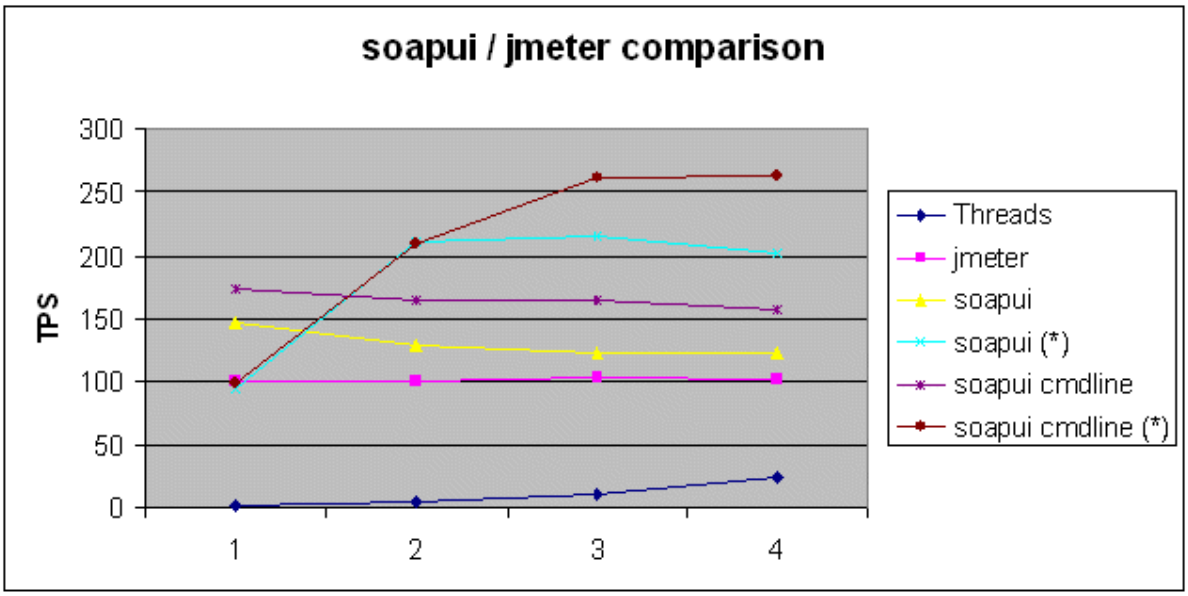
My laptop is running WinXP SP2 on a Pentium M 1.8 with 1Gb of RAM. JRE 1.5.0_06 is used for running JBoss, JMeter and soapUI.

Results

The table shows the results, for each test the average and TPS times are shown, (*) tests are run without closing connections

Threads	jmeter	soapUI	soapUI (*)	soapUI cmdline	soapUI cmdline (*)
1	8 ms, 105 TPS	6.78 ms, 147 TPS	10.7 ms, 94 TPS	5.75 ms, 174 TPS	10 ms, 99 TPS
5	43 ms, 110 TPS	38.7 ms, 128 TPS	23.7 ms, 211 TPS	30.4 ms, 164 TPS	24 ms, 210 TPS
10	86 ms, 112 TPS	82 ms, 122 TPS	46.5 ms, 215 TPS	61 ms, 164 TPS	38 ms, 262 TPS
25	214 ms, 114 TPS	204 ms, 123 TPS	124 ms, 202 TPS	159 ms, 157 TPS	95 ms, 263 TPS

This can be plotted as follows:



As can be seen in the graph and numbers, soapUI consistently gives a higher TPS value than JMeter, except when reusing connections and running only one thread. This may be a bit surprising, evidently it is faster for HttpClient and the JVM to create new connections than to pool existing ones.

The biggest difference between the two tools can be seen when switching to connection-reuse and running from the command-line. This is expected, connection reuse should have a big impact when running many threads, and running from the command-line uses no UI resources at all.

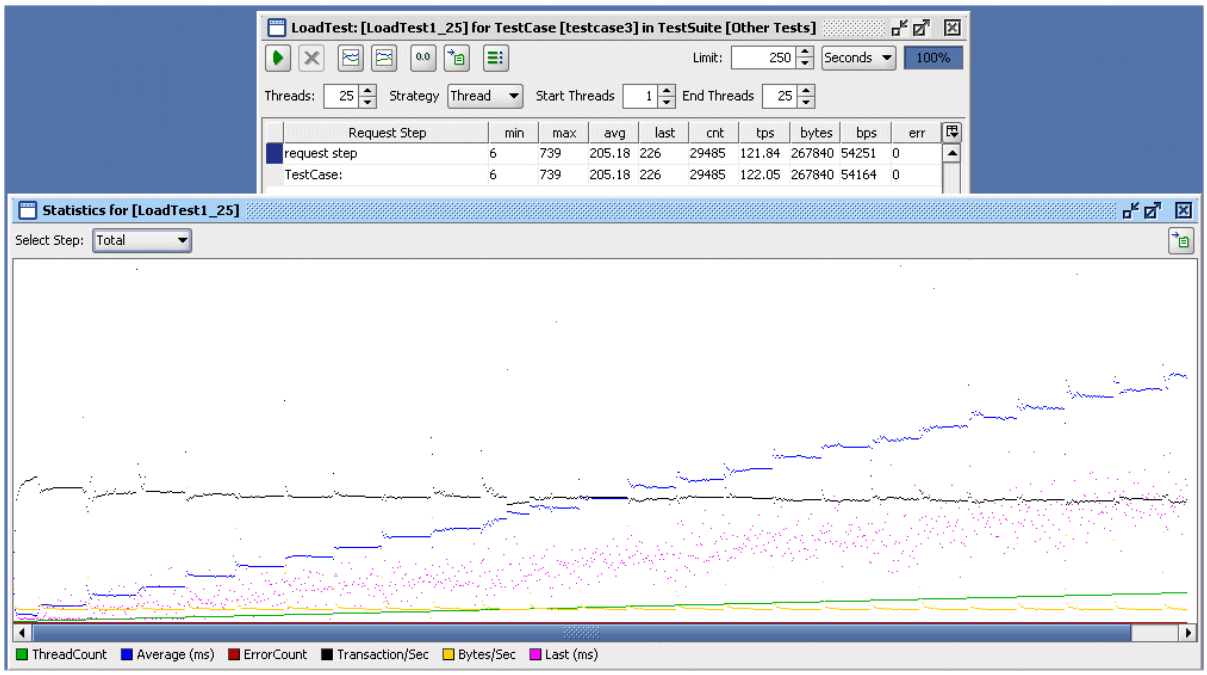
One could argue that closing connections after each request is a more "realistic" scenario for web services that serve a large number of different clients, for example an online weather service. Web services running internally in a SOA may be more likely to serve a smaller number of clients, so these could be tested with connection reuse enabled.

Looking through the JMeter sources, it seems as if JMeter uses HTTP 1.0 and does not maintain any internal connection pool for Web Service requests. Also, it uses `System.currentTimeMillis()` for measuring elapsed time instead of the `System.nanoTime()` feature available in Java 1.5 and used by soapUI.

Further Analysis

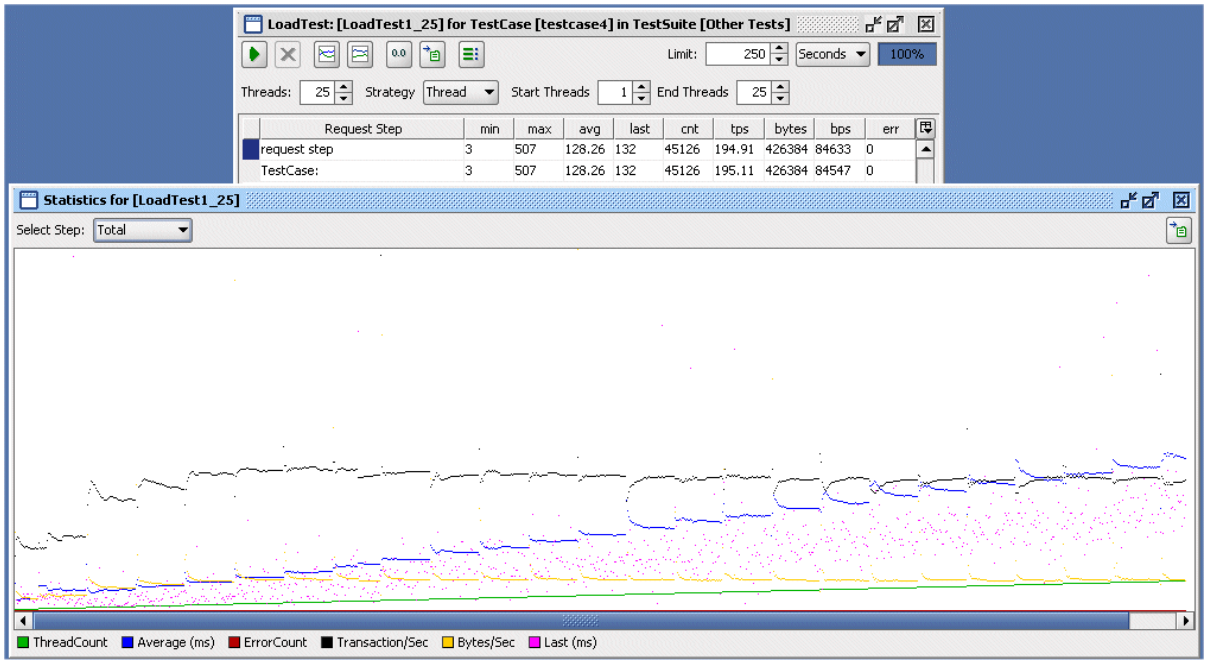
Since the above results indicate an improved result when reusing connections, I ran a LoadTest in soapUI using the "Threads" strategy increasing the number of threads from 1 to 25 over a period of 250 seconds, both with and without connection reuse. The graphs below show the results:

First without connection reuse (click on image for full-size):



This confirms the previous results; TPS does not vary much when increasing the number of threads, the average request time "takes" the entire increased load.

And then with connection reuse (click on image for full-size):



The results here also confirmed those above; when reusing connections the TPS value increases steadily up to about 6 threads and then lies still. Also, one can see that the average request time starts increasing steadily first when the TPS has hit its peak, the average then "takes" all the increased load just as when not reusing connections at all.

Conclusion

The values obtained with soapUI are almost consistently "higher", if this is *better* is a matter of discussion. One clear advantage of soapUI seems to be the possibility to reuse connections, which also makes a big difference when running many threads (ie simulating many clients).

Based on the above results my recommendation would therefore be the following:

- For behavioural and functional load testing of Web Services; start with soapUI, move to jmeter if you need its "high profile" features.
- For performance load testing of Web Services: use soapUI in conjunction with another tool (for example JMeter) and weigh the results!

Files

If you want to run the tests yourself, the following files are available in comparison-files.zip :

File	Description
comparison-soapui-project.xml	soapUI project file containing all tests run in the comparison
soapui-compare.jmx	jmeter project file containing all tests run in the comparison
ws4ee-samples-server-ejb.jar	precompiled jar file containing tested Web Service, just drop this in the JBoss 4.0.3 deploy folder and the above tests should run as configured (you may need to change the endpoint)

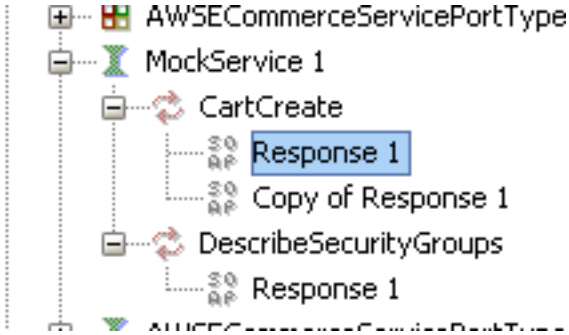
You can download JBoss 4.0.3SP1 from here , but you can also use the 4.0.4RC2 release using the new JBossWS stack which gives slightly different results than those shown above (not shown here since this comparison is not meant to compare jboss versions)

Next: Web Service Mocking

1.8 Mocking

Web Service Mocking

soapUI 1.7 introduces the possibility to create Mock Implementations (called "MockServices") of any WebService from its WSDL contract. A soapUI "MockService" can expose an arbitrary number of operations ("MockOperations") from differentWSDLs which can further be configured quite extensively in regard to which response(s) ("MockResponse") they should return, including the possibility to create custom groovy scripts for both dispatching and response creation. MockServices can be hosted/run either directly in soapUI or via the command-line MockServiceRunner and be invoked from any client (including soapUI itself of course) as usual.



Mocking of Web Services opens for a number of interesting usage-scenarios:

- Client development / testing : create a MockService for an existing contract that returns a number of predictable results which can be used by client developers/integrators during the development phase.
- TestDriven development from WSDL contract (both server and client) : Starting from only a WSDL, generate both client and server stubs (using any of the soapUI tool integrations) together with a MockService which can be used to build entire functional TestSuites in soapUI before the actual service implementation is ready.
- Web Service rapid prototyping : Quickly create simple WSDL implementations for discovering optimal message exchanges, etc
- etc..

soapUI Pro further introduces a MockResponse test step which can be added to any functional TestCase to simulate the handling of a SOAP request as part of the test, which opens of testing of asynchronous Web Services, etc.

Next: Mock Services

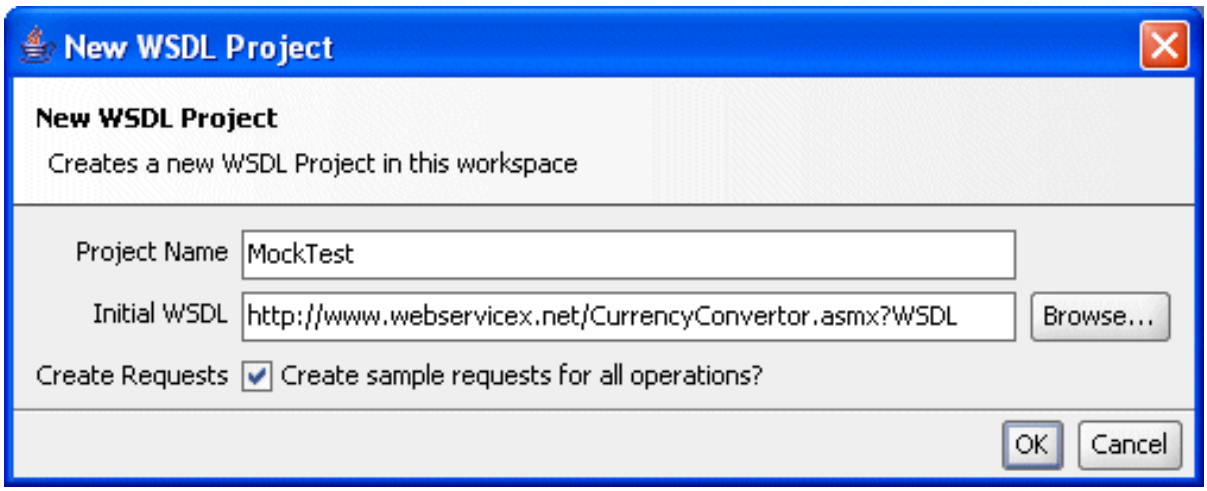
1.8.1 Getting Started

Getting Started with Mocking

soapUI 1.7 introduces the possibility to create mock implementations of operations and entire services with a combination of static response messages combined with groovy scripts, attachments and custom http-headers. This opens for a number of usage scenarios, including:

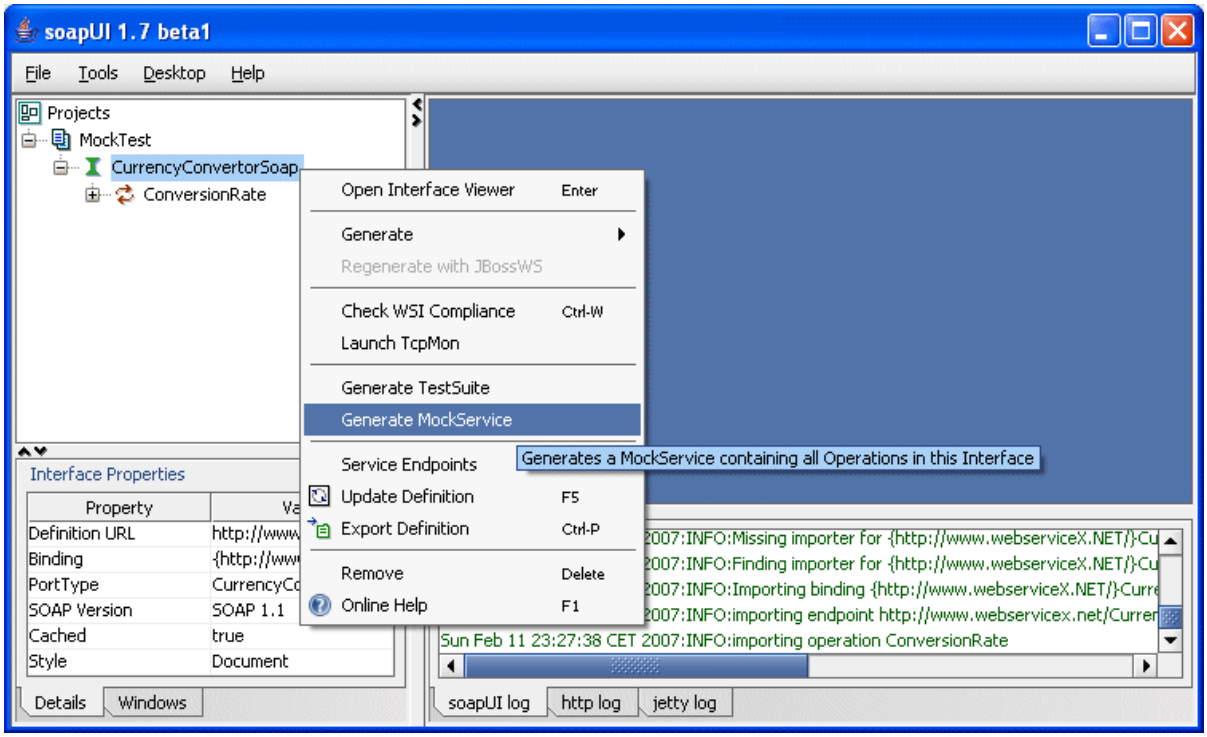
- Rapid Prototyping of Web Services; generate a complete static mock implementation from a WSDL in seconds and add dynamic functionality using Groovy.
- Client testing/development; create mock implementations of desired operations and set up a number of alternative responses (including scripts, attachments and custom http-headers). Clients can be developed/tested without access to the "live" services. Responses can be cycled, randomized or selected with XPath expression from incoming request
- Test-Driven Development; Create soapUI TestSuites/TestCases against MockServices before/during the actual services have been/are being implemented
- etc..

In the following example we will create a mock implementation of an existing service, tailor the mock response with some groovy and call the mock service from within soapUI. Start by creating a new project for the publicaly available CurrencyConverter at http://www.webservicex.net/CurrencyConvertor.asmx?WSDL

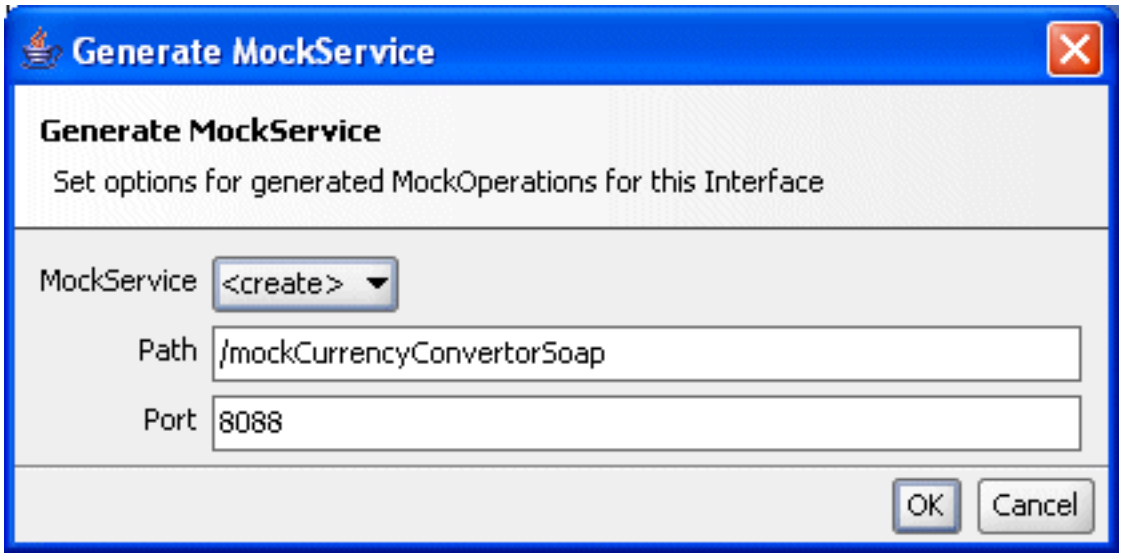


Create a MockService

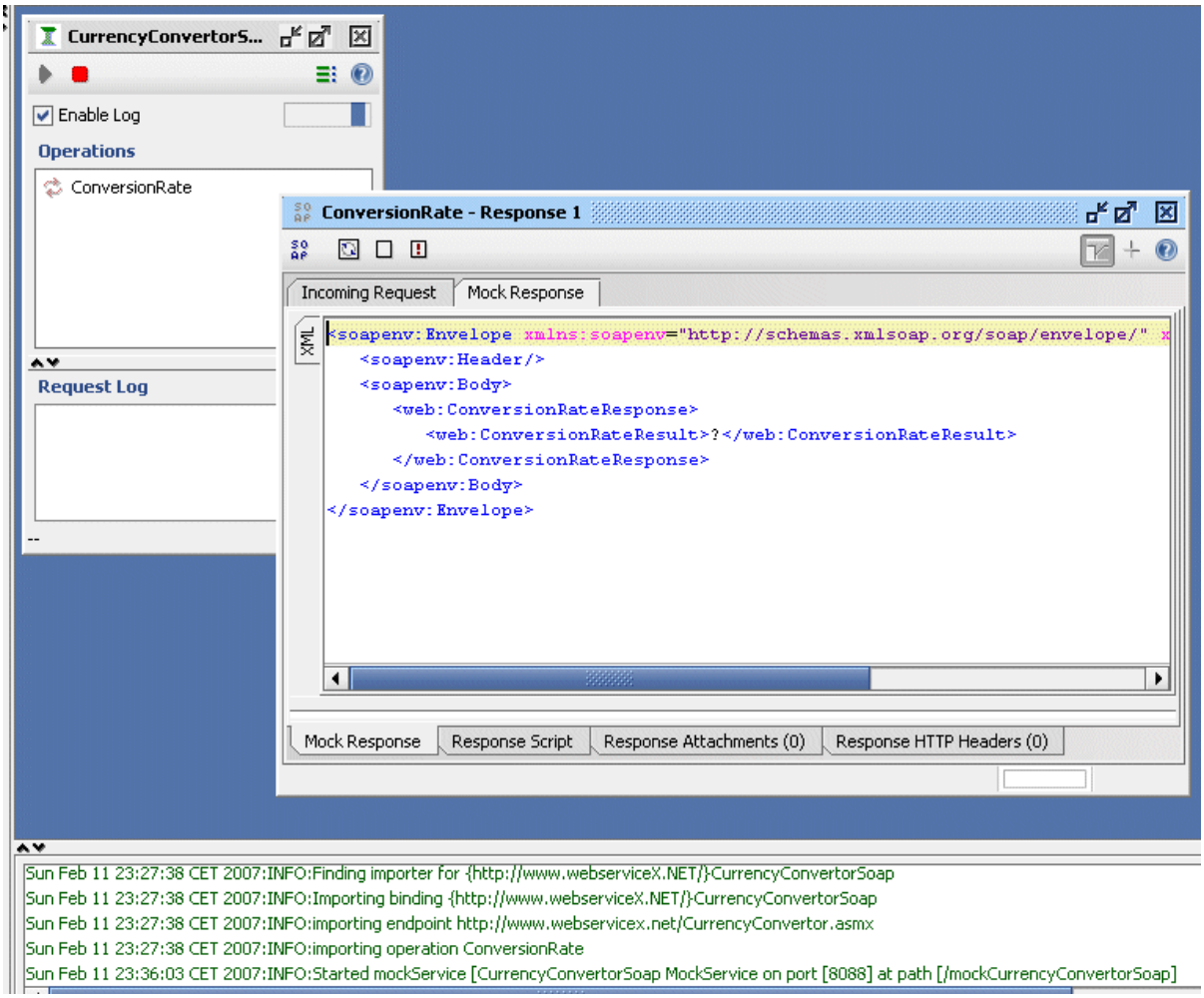
After importing, you should get an interface with one operation and request. Right-click the interface and select the "Generate MockService" menu option:



This opens the following dialog which lets you specify the local port/path for the create service (can of course be changed later..);



Just leave the suggested values, for this and coming dialogs.. When the MockService has been created, you will get a new tree-hierarchy similar to the the initial Interface, but with grayed icons.. The MockService contains a MockOperation corresponding to the operation in the original interface and soapUI has also created one MockResponse for this operation from its schema.. Double click both the MockService and "Response 1" tree nodes to open their corresponding editors, after some moving/resizing arround you should be able to arrive at the following:



The MockService editor (top left) contains a toolbar, a list of all operations for the service and a request/response log which logs all incoming requests that have been handled by the MockService.

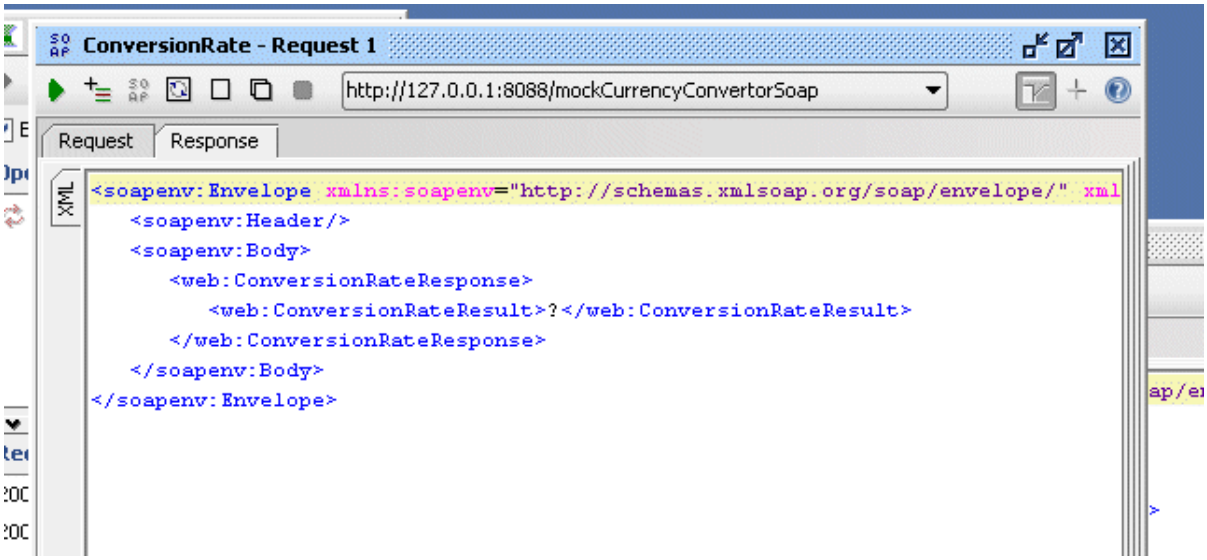
The MockResponse editor is very similar to the standard soapUI Request editor (which can/will be confusing at first ;-), only the "focus" has been shifted to the response-tabs of the editor, since this is where the editing of the mock response will be done. The "Incoming Request" tab/view shows the last received request to the operation (usefull for debugging client calls), including attachments nad http-headers.

Invoke the MockService from within soapUI

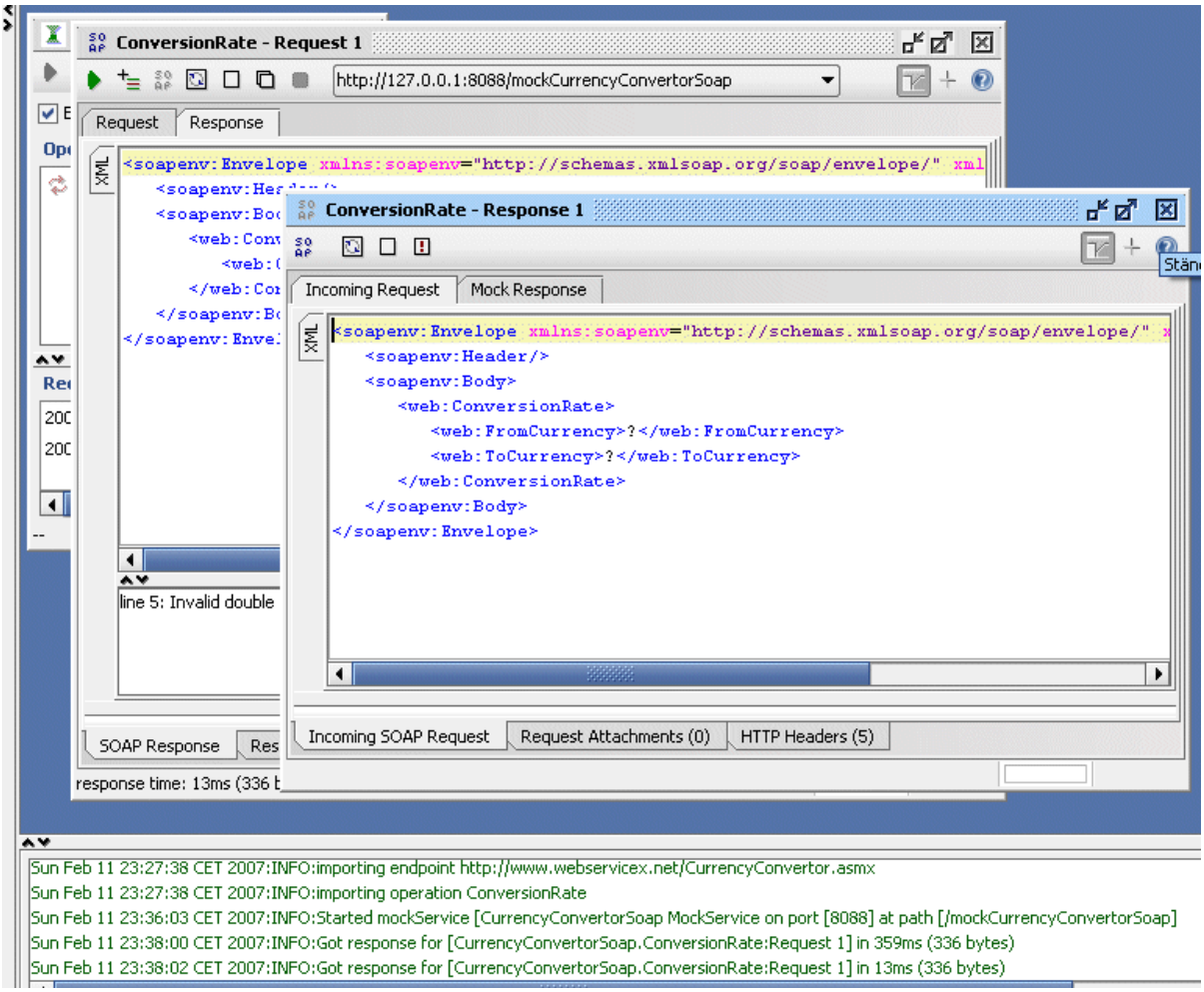
In the screen above you can see the default response that has been generated by soapUI from the schema. Lets continue by just invoking this mock operation from within soapUI:

1. Click the green arrow button in the MockService editor.. this will start the service on the configured port/path which can also be seen in the log (this has already been done in the screenshot above..)
2. In the MockResponse editor, select the left-most button in the toolbar, which will prompt to open one of the existing requests for the corresponding operation in your project. When opening the request, soapUI will automatically change its endpoint to the endpoint of the now locally running MockService

3. Submit the opened request! If all goes well you should get the previously create MockResponse in the request editors response window:



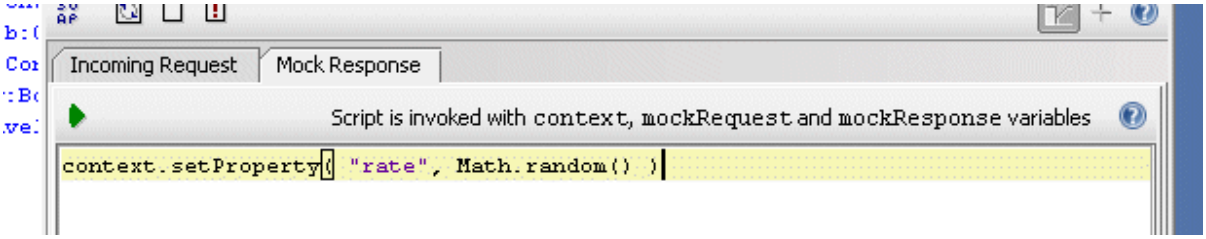
Switching back to the MockResponse editor and selecting its "Incoming Request" tab/view, you should be able to see the request that was posted to the MockService and handled by the corresponding MockOperation:



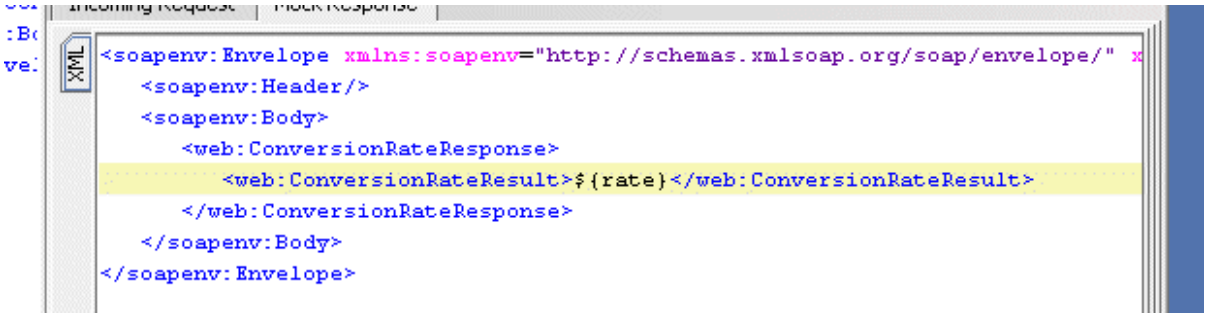
Congratulations! Hopefully all went well and you now have your first MockService up and running.. Feel free to modify the created MockResponse content and resubmitting the request to see that your changes are instantly available (no need to restart the MockService)

Customize the MockResponse

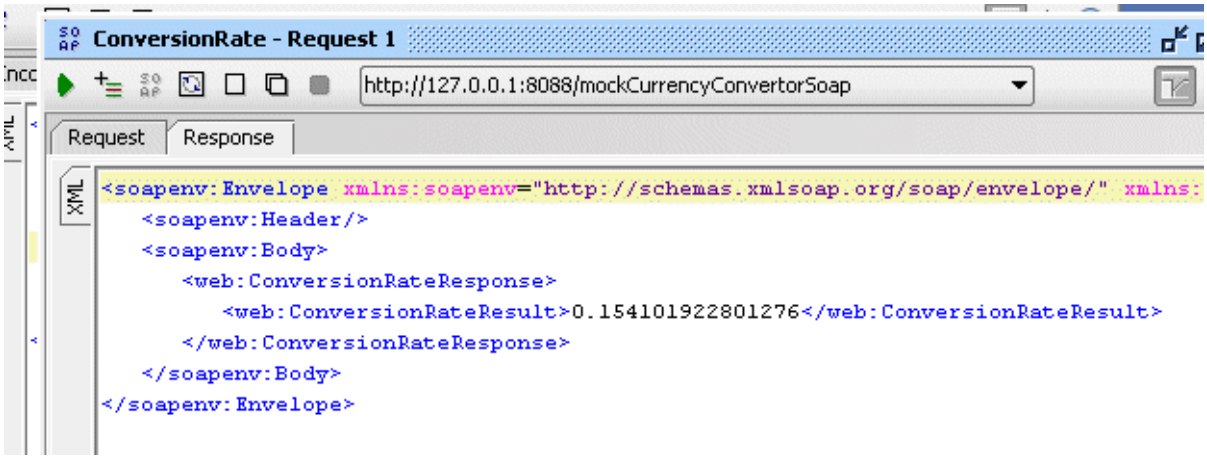
Let's continue by making the created mock response a bit more dynamic with a groovy script. Switch to the "Response Script" tab in the MockResponse editor and enter the following:



Go back to the Mock Response tab and change the response body to



Now, when submitting a request to this MockResponse, the script will be evaluated and the generated rate value will be inserted into the outgoing response message;



Ok.. this should get you started with your first mock service.. now you can continue with:

- Add some more responses to the MockOperation (via its right-button menu). Change the dispatch method through the MockOperation Options dialog
- Add more dynamic script behaviour (database/file access, input processing, etc..), attachments, custom response http headers, etc..
- Generate a web-service client with one of soapUI's tool integrations and call your MockService from there to see what message the client actually sends (and that it handles your response correctly)
- etc..

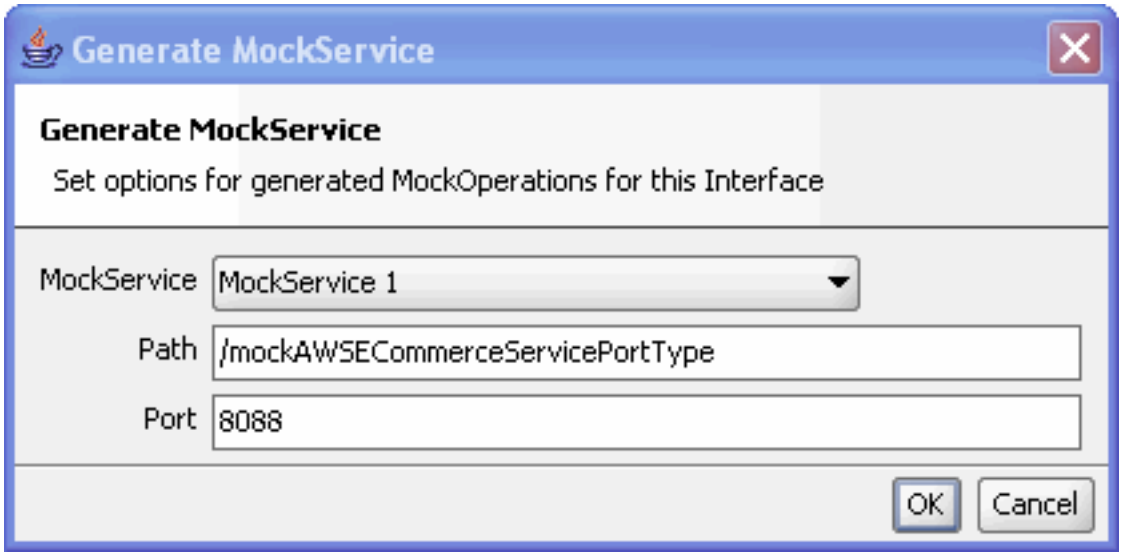
Next: soapUI User Guide

1.8.2 Mock Services

Mock Services

MockServices are displayed in the navigator under their containing project node. They can be created in a number of ways;

- From the Project popup menu with the "New MockService" action. Creates an empty MockService with no operations.
- From the Interface popup menu with the "Generate MockService" action. This will prompt as follows:



After specifying a path and port to listen on, soapUI will create a MockService with a MockOperation corresponding to each of the specified interfaces operation. Each MockOperation will further be configured with a default MockResponse which will be created from the associated WSDL/Schema definition in the same manor as when importing interfaces and creating requests.

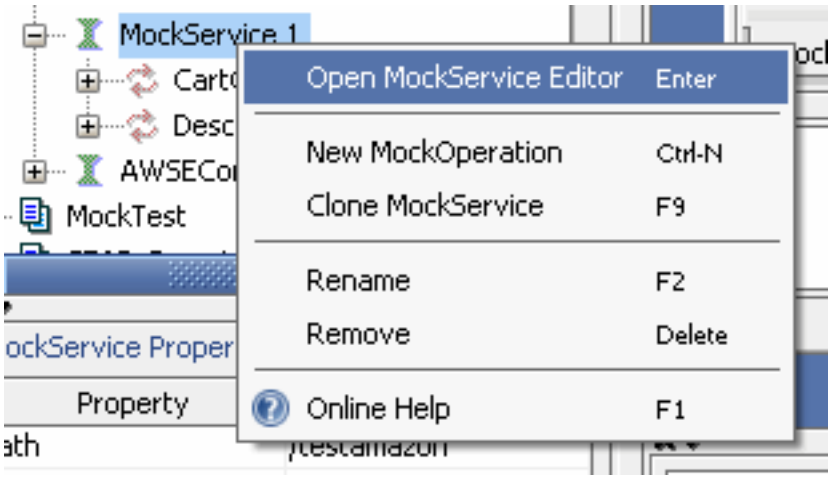
- From an existing request by selecting the "Create MockResponse" toolbar button which will prompt for which MockService to add the selected requests operation and current response as a MockResponse.

Once created, an arbitrary number of MockOperations can be added to a MockService, there is no requirement that all operations come from the same interface or that all operations from an interface be mocked. Just mock those operations that are required.

MockService Actions

Right-clicking a MockService node in the navigator shows a popup menu with the following actions:

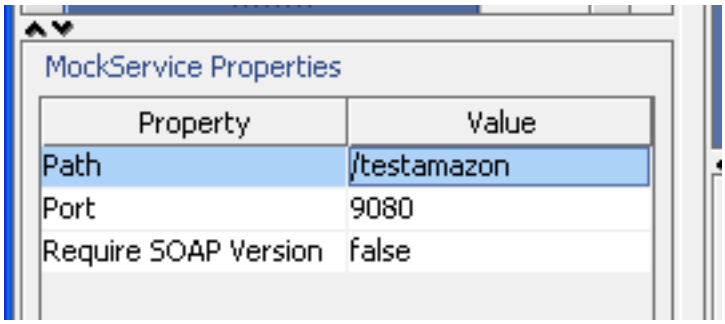
- **Open MockService Editor** - opens the MockService Editor, see below
- **New MockOperation** - prompts to create a new MockOperation in the MockService
- **Clone MockService** - clones the entire MockService
- **Rename** - prompts to rename the MockService
- **Remove** - prompts to remove the MockService
- **Online Help** - displays this page in an external browser



MockService Details Tab

The bottom left details tab for a MockService displays the following properties:

- **Path** - the path this MockService listens on (read only)
- **Port** - the port this MockService listens on (read only)
- **Require SOAP Version** - controls if incoming requests must match the SOAP Version of a corresponding MockOperation in a MockService.



The MockService Editor

Double-clicking a MockService in the navigator opens the MockService editor as seen to the right. From the top down the editor has the following parts:

A Toolbar with the following options:

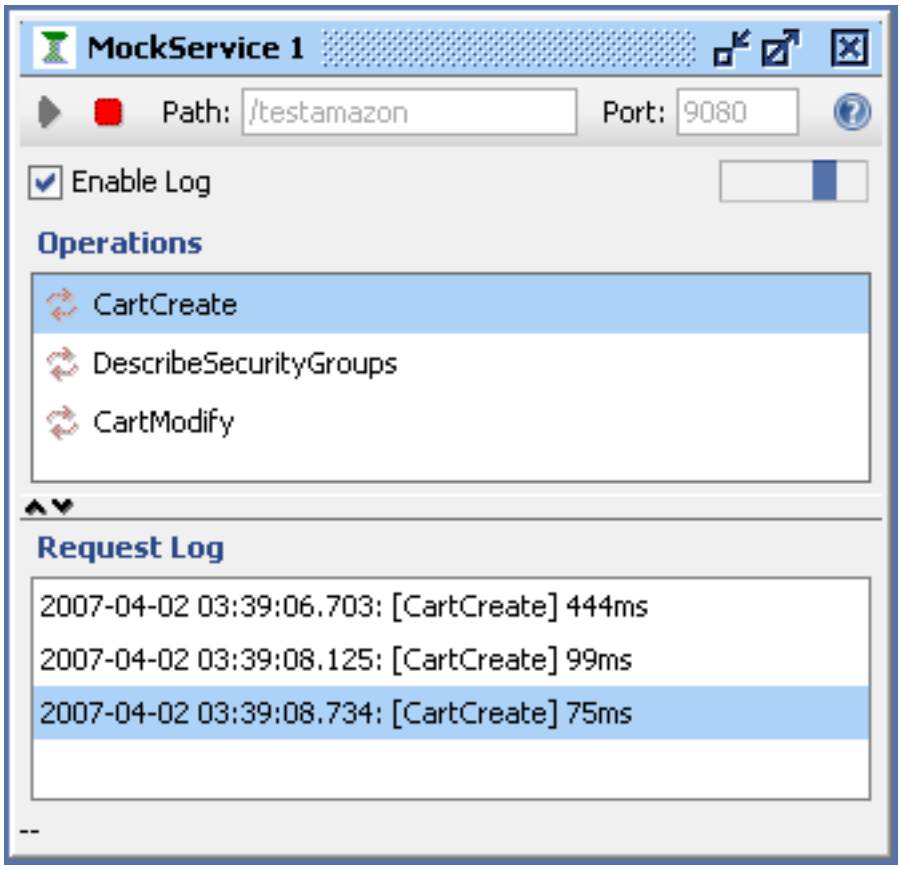
- **Run** - Starts the MockService on the configured path/port and waits for requests (see below)
- **Stop** - Stops a running MockService
- **Path** - The path to listen on, disabled when running

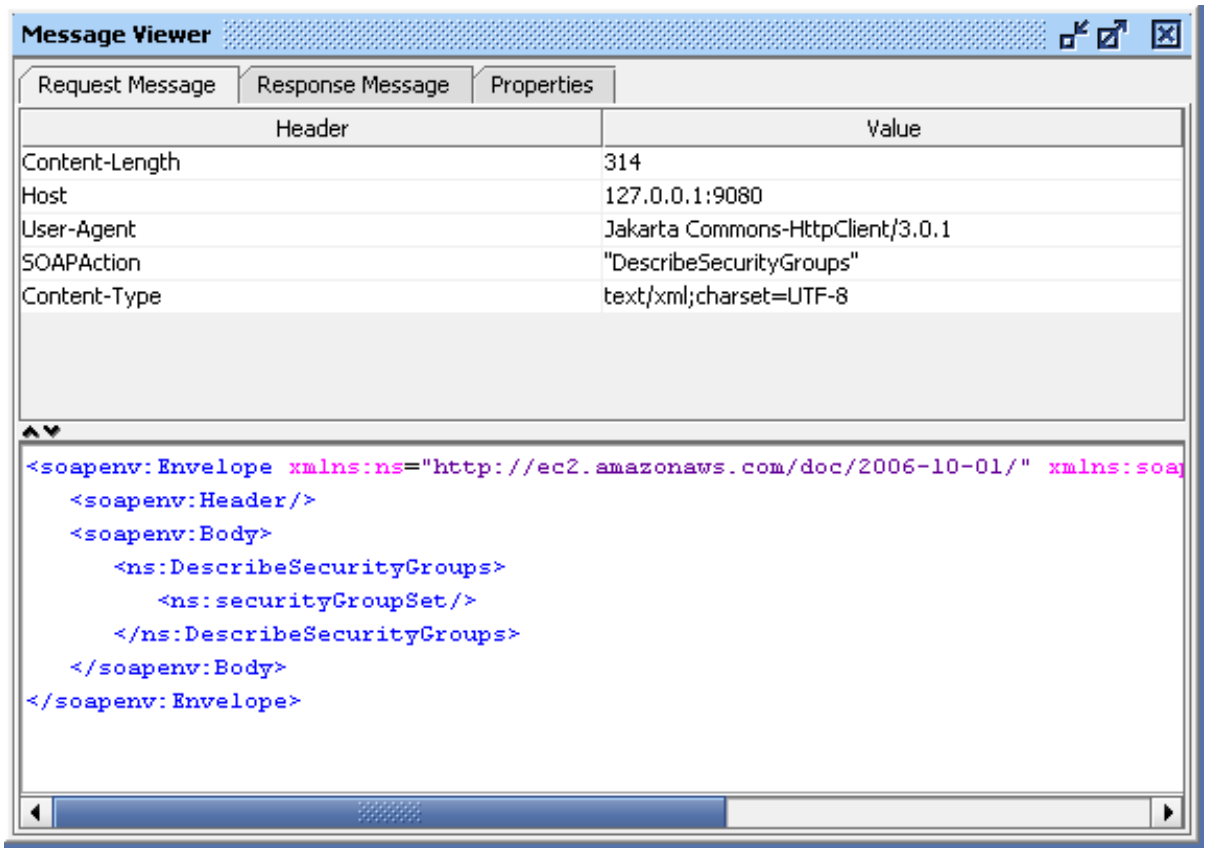
- **Port** - The port to listen on, disabled when running
- **Help** - Opens this page in a browser

A status indicator with the possibility to turn on/off logging

The list of MockOperations currently configured for this MockService. Right-clicking show the MockOperations available actions, double-clicking opens the MockOperation editor

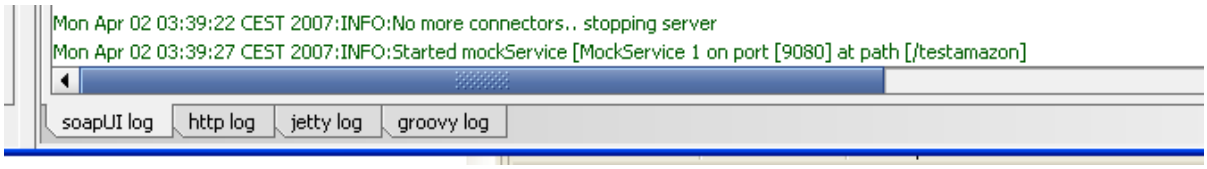
A Log of handled requests. Double-clicking a request displays the entire message exhange for that request as shown below:





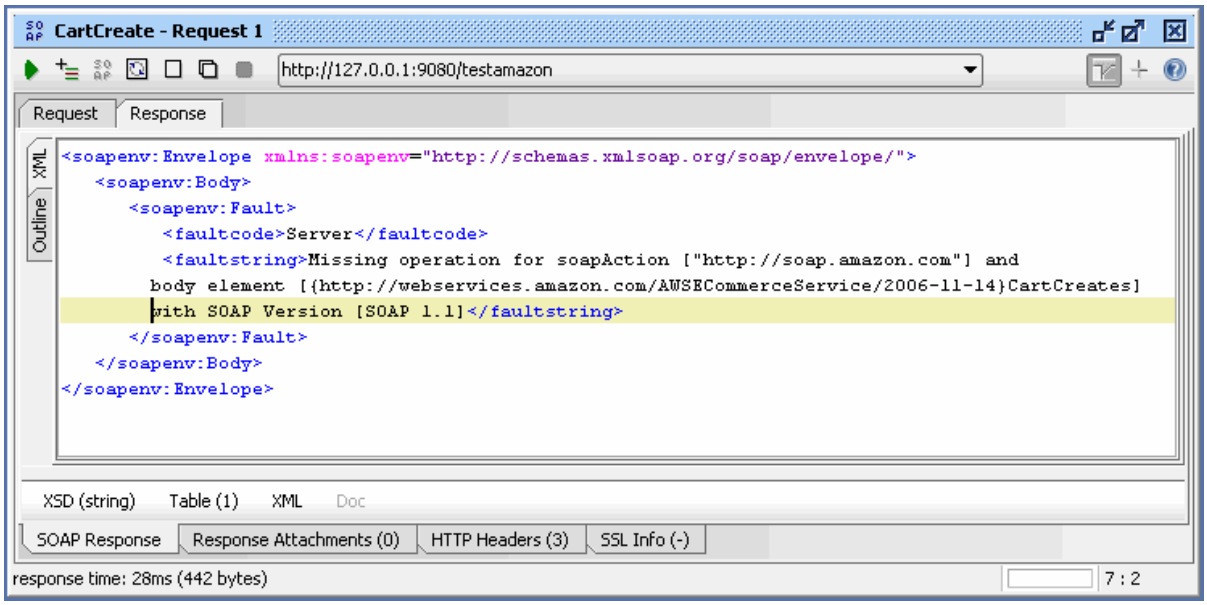
MockService execution

When starting the MockService using the Run toolbar button, soapUI starts a local http server (if not already running for another MockService) and makes the MockService available on the configured port and path. This can be seen in the soapUI-log as follows:



SOAP Requests can now be issued to the server from any client as if it were a "real" Web Service, the incoming requests will be dispatched to the matching MockService / MockOperation which will further dispatch as configured in Request Dispatching . While the MockService is running, it is still possible to add/remove MockOperations, MockResponses, etc.. The only values that can not be changed are the path and port of the MockService, for this it is required to first stop the MockService and then start it again after changing as desired.

If an error occurs during request processing a SOAP Fault will be returned, for example in the following situation a request was issued from within soapUI that could not be matched to any available MockServices MockOperation:



Next: Mock Operations

1.8.3 Mock Operations

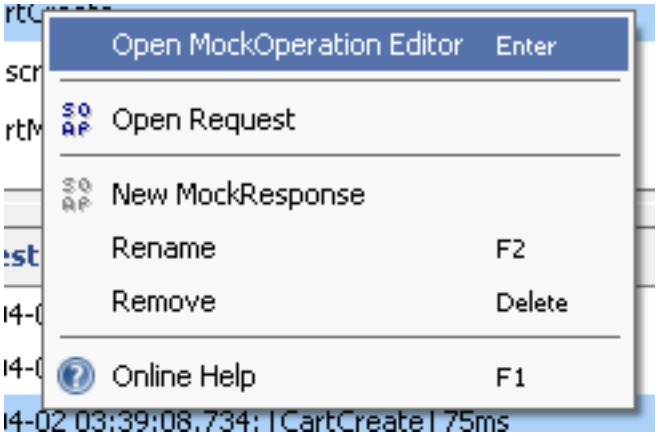
Mock Operations

A MockOperation corresponds to an operation in one of the current projects interfaces and will dispatch incoming requests if they match the operations SOAP Action and arguments. A MockOperation contains a variable number of MockResponses that contain actual response messages to be returned to a caller. Dispatching of requests to a specific MockResponse can be done in a number of ways as described below.

MockOperation Actions

Right-clicking a MockOperation node in the navigator or MockService editor shows a popup menu with the following actions:

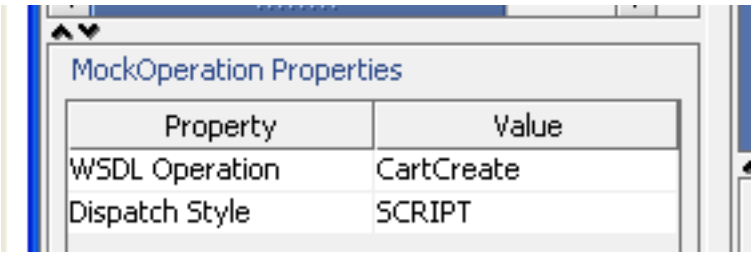
- **Open MockOperation Editor** - opens the MockOperation Editor, see below
- **Open Request** - prompts to open an existing request for the operation being mocked with the endpoint set to the MockServices endpoint. This is the preferred way of quickly testing a MockService from within soapUI after the MockService has been started.
- **New MockResponse** - prompts to add a new MockResponse to the MockOperation
- **Rename** - prompts to rename the MockOperation
- **Remove** - prompts to remove the MockOperation
- **Online Help** - displays this page in an external browser



MockOperation Details Tab

The bottom left details tab for a MockService displays the following properties:

- **WSDL Operation** - shows which Interface Operation that is being mocked
- **Dispatch Style** - shows how incoming requests to this MockOperation are dispatched.



The MockOperation Editor

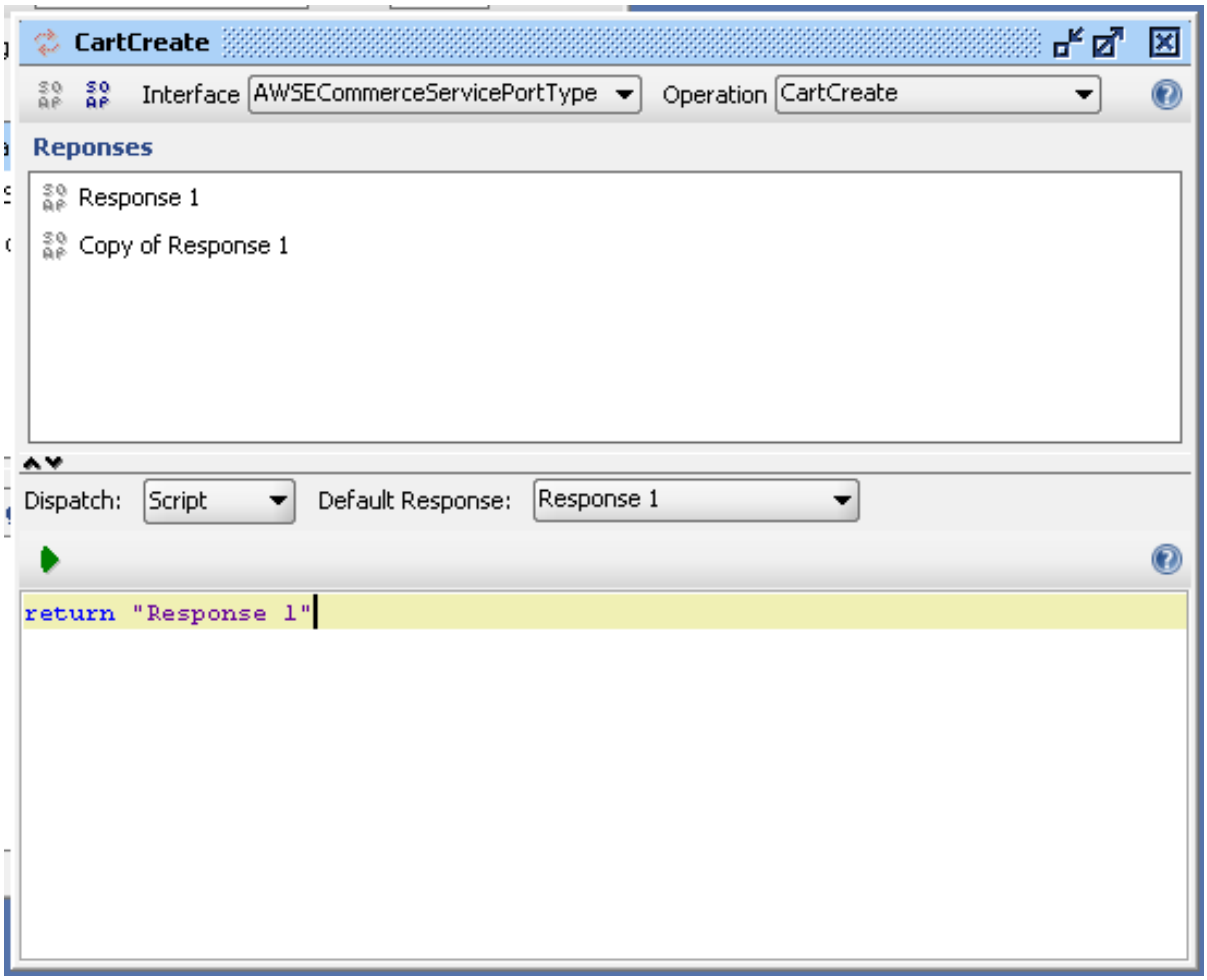
Double-clicking a MockOperation in the navigator opens the MockOperation editor as seen below. From the top down the editor has the following parts:

A Toolbar with the following options:

- **Add MockResponse** - Prompts to add a new MockResponse to the MockOperation
- **Open Request** - prompts to open an existing request for the operation being mocked as described above
- **interface** - The Interface containing the operation to be mocked
- **Operation** - The operation to be mocked
- **Help** - Opens this page in a browser

The list of MockResponse currently configured for this MockOperation. Right-clicking show the MockResponses' available actions, double-clicking opens the MockResponse editor

A Dispatch Toolbar and configuration are below whose content depends on the selected dispatch style (see below)



Request Dispatching

Incoming requests to a MockOperation can be dispatched in one of 4 ways using the dispath combo-box:

- **Sequence** - dispatches to the configured responses in sequence, starting from the beginning when the last has been reached.
- **Random** - dispatches randomly between existing MockResponses
- **XPath** - dispatches based on a specified XPath expression that will be used to select the name of the MockResponse step to dispatch to.
- **Script** - allows a groovy script for dispatching based on any arbitrary criteria

Both the XPath and Script dispath methods allow specification of a default response to be returned if there is no matching response for the XPath or script expression. When selecting XPath or Script, a corresponding editor is shown below the dispatch toolbar where the XPath expression or groovy script can be entered and tested. When running soapUI Pro, the XPath editor also includes a button for the XPath Selector .

Groovy Script Dispatching

Using groovy scripts as a controller to dispatch incoming is a very flexible way of creating MockOperations. The script has access to the following variables:

The script has access to the following objects:

- `context` - an instance of MockRunContext that can hold dynamically user-defined properties. The context is service scoped, ie shared between all MockOperations/MockResponses for a MockService during its lifetime. Since this object implements the `Map` interface it can be accessed using groovy's built-in collection support (see example screenshot below)
- `mockRequest` - an instance of WsdlMockRequest which provides access to request-related objects, including the underlying HttpServletRequest/HttpServletResponse objects
- `mockResponse` - an instance of WsdlMockResponse which provides access to the current MockResponse object (for example for dynamic manipulation of attachments)
- `log` - a standard log4j Logger that logs to the groovy log tab

The script must return the name of the MockResponse to dispatch to, ie the script shown in the screenshot above always dispatches to a MockResponse named "Response 1"

Next: Mock Responses

1.8.4 Mock Responses

Mock Responses

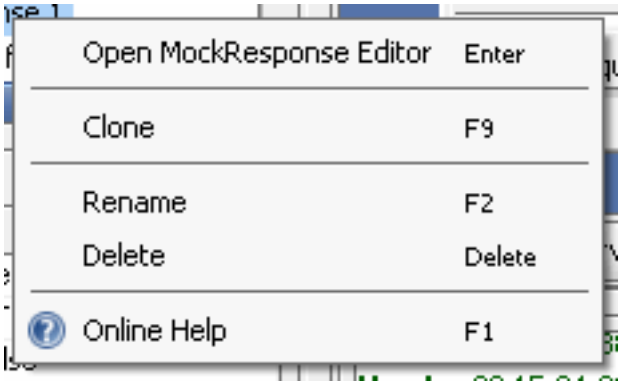
A MockResponse holds an actual response to returned from a MockOperation. The response can be configured in regard to response content, http headers, attachments and dynamic processing using groovy (for example to process the input or read data from some external source).

MockResponses are added to a MockOperation from their popup menu or from the Request popup/editor with the "Add to MockService" action/button.

MockResponse Actions

Right-clicking a MockResponse node in the navigator or in the MockOperation editor shows a popup menu with the following actions:

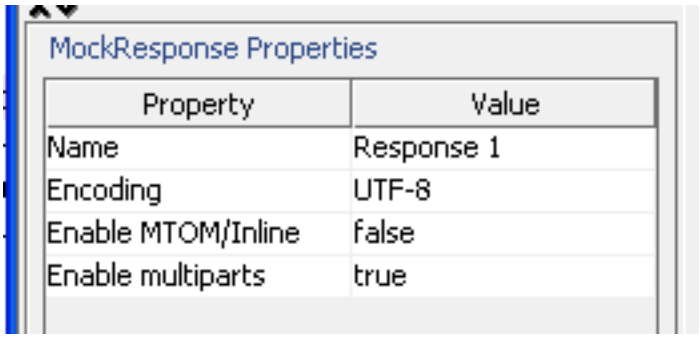
- **Open MockResponse Editor** - opens the MockResponse Editor, see below
- **Clone** - prompts to clone the MockResponse
- **Rename** - prompts to rename the MockResponse
- **Remove** - prompts to remove the MockResponse
- **Online Help** - displays this page in an external browser



MockResponse Details Tab

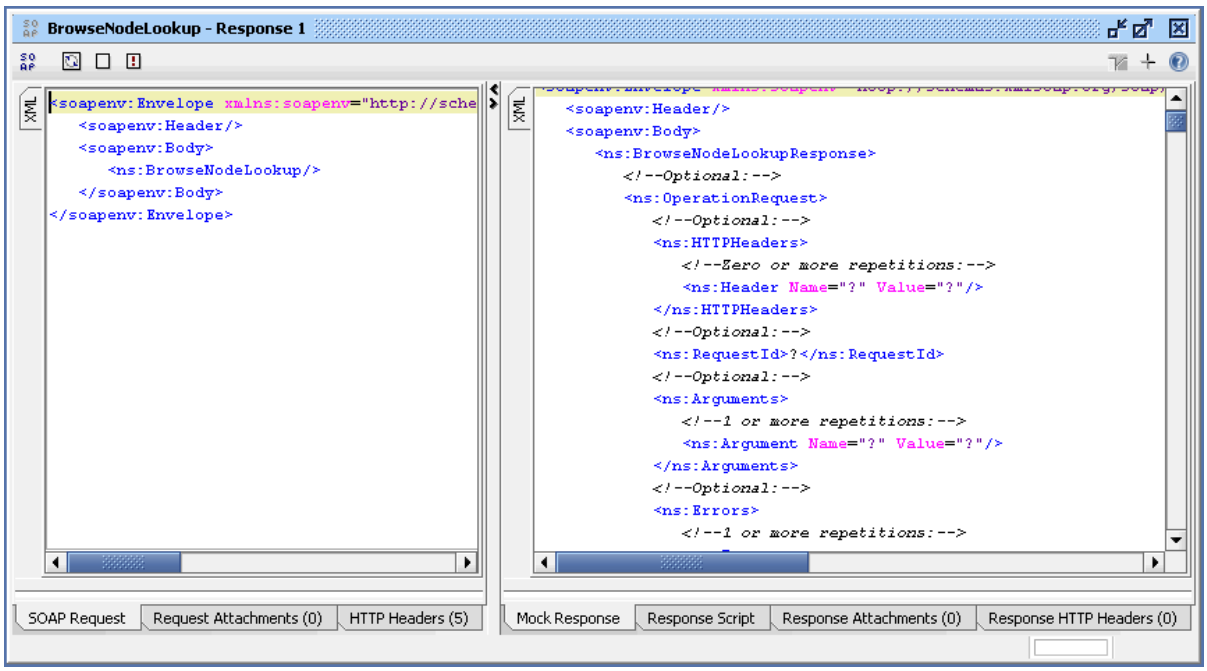
The bottom left details tab for a MockResponse displays the following properties:

- **Name** - the name of the MockResponse
- **Encoding** - the encoding to use for the content of the MockResponse message.
- **Enable MTOM/Inline** - enables MTOM/Inline processing for MockResponse messages in the same way as descrbibed for Request Attachments
- **Enable Multiparts** - enables multiparts in the same way as described for Multipart Attachments



The MockResponse Editor

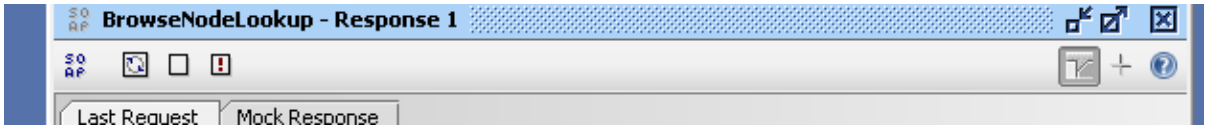
Double-clicking a MockResponse in the navigator or MockOperation editor opens the MockResponse editor as seen below:



This editor is more or less a copy of the standard request editor, but shifts focus to the right-hand response area where the MockResponse is configured. The left hand request area shows the latest request handled by this MockResponse, including its HTTP Headers and Attachments.

The Response Area contains the same XML Source editor , HTTP Headers tab and Attachments tab as in the Request Editor (but now all editable). Additionally, a "Response Script" tab is available that displays a groovy script editor for creating scripts that should be executed before returning the actual response content (see more below).

MockResponse Editor Toolbar

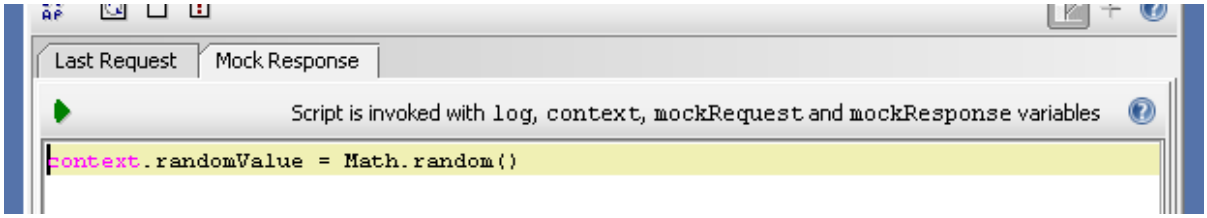


The toolbar at the top of the MockResponse editor contains the following actions (left to right):

- **Open Request** - Opens a new/existing request that can be issued to this MockOperation when its MockService is running, the request will have the correct local endpoint set automatically.
- **Recreate Response** - Prompts to create a default response message from the associated WSDL/Schema definition
- **Create Empty Response** - Prompts to create an empty SOAP response message
- **Create SOAP Fault** - Prompts to creates an empty SOAP Fault, if the operation being mocked defines any faults, soapUI will prompt for which of these to generate en default detail element;
- **Toggle Layout** - toggles between the available split/tab layouts as described in Editor Layouts .
- **Online Help** - shows this document in an external browser.

Response Scripts

The "Response Script" tab shows a standard groovy editor for a script that will be executed prior to returning the configured response message, opening for the possibility to create response specific scripts that can dynamically create content of the outgoing response.



The script has access to the following objects:

- `context` - an instance of MockRunContext that can hold dynamically user-defined properties. The context is service scoped, ie shared between all MockOperations/MockResponses for a MockService during its lifetime. Since this object implements the `Map` interface it can be accessed using groovy's built-in collection support (see example screenshot below)
- `mockRequest` - an instance of WsdlMockRequest which provides access to request-related objects, including the underlying HttpServletRequest/HttpServletResponse objects
- `mockResponse` - an instance of WsdlMockResponse which provides access to the current MockResponse object (for example for dynamic manipulation of attachments)
- `log` - a standard log4j Logger that logs to the groovy log tab

The Run button in the script toolbar will attempt to run the script, setting the context to either the currently available context (if the MockService is running) or an empty one. The mockRequest will be set to the last handled request if available.

Properties set in the context can the be used using standard property expansion in the response, for example the following script:

```
context.randomValue = Math.random()
```

creates a "randomValue" property which can then be "used" in the outgoing response:

```
<detail>${randomValue}</detail>
```

Next: Usage Scenarios

1.9 Usage Scenarios

Usage Scenarios

This section describes some more "complete" soapUI usage scenarios, currently the following are described:

Scenario	Description
Data-Driven Testing	Shows how to use a combination of TestSteps and/or Groovy Scripts to allow test-data (query parameters, username/password, etc) to be provided from an external source
Template-Driven Testing	Extends Data-Driven testing to show how to read a series for test-values from an external source and run the entire TestCase for each "row" of values.
Interactive Testing	Shows how to use groovy scripts to create an input dialog for a testcase and for displaying the result.
Surveillance Testing	Describes how to use a scheduling tool in conjunction with soapUI to set up ongoing test-processes that continuously validate functionality and performance.

If you have any suggestions or requests for more scenarios please let us know!

Next: Data-Driven Testing

1.9.1 Data-Driven Testing

Data-Driven Testing

Data-Driven testing is useful if you want to provide input for a test from an external source, for example from a database or properties file. This document will outline a standard approach with a complete example.

A simple approach to Data-Driven Testing in a TestCase is the following:

- Create the Request step(s) you want to be "data-driven"
- Create a Properties Step and define the properties you will read from some external source
- Create a Property Transfer Step that transfers these properties to their destination steps
- Now you need to specify the actual reading of the external properties, the easiest is to specify in the "Properties Step" which source properties file to read from. A more flexible approach is to create a preceding Groovy Step that reads the properties from some source and assigns them to the Properties Step. Both these approaches will be shown below

Below follows a complete example of the above using the Amazon Search Service and specifying the author and subscriptionId externally for a book search. You need to create a project and import the Amazon WSDL (http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl) before digging in..

Create The TestCase and TestRequest

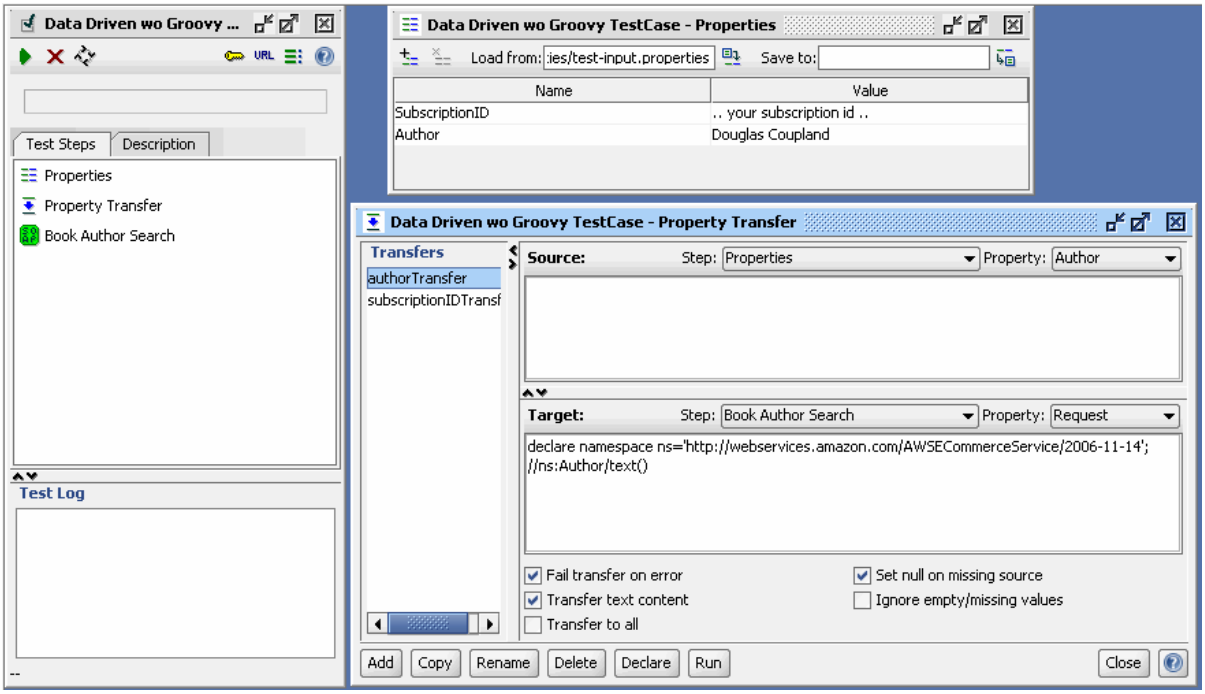
Create a new TestCase and add a new Request Step containing the following request:

```
<soapenv:Envelope
xmlns:ns="http://webservices.amazon.com/AWSECommerceService/2006-02-15"
                xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns:ItemSearch>
         <ns:SubscriptionId>?</ns:SubscriptionId>
         <ns:Request>
            <ns:SearchIndex>Books</ns:SearchIndex>
            <ns:Author>?</ns:Author>
         </ns:Request>
      </ns:ItemSearch>
   </soapenv:Body>
</soapenv:Envelope>
```

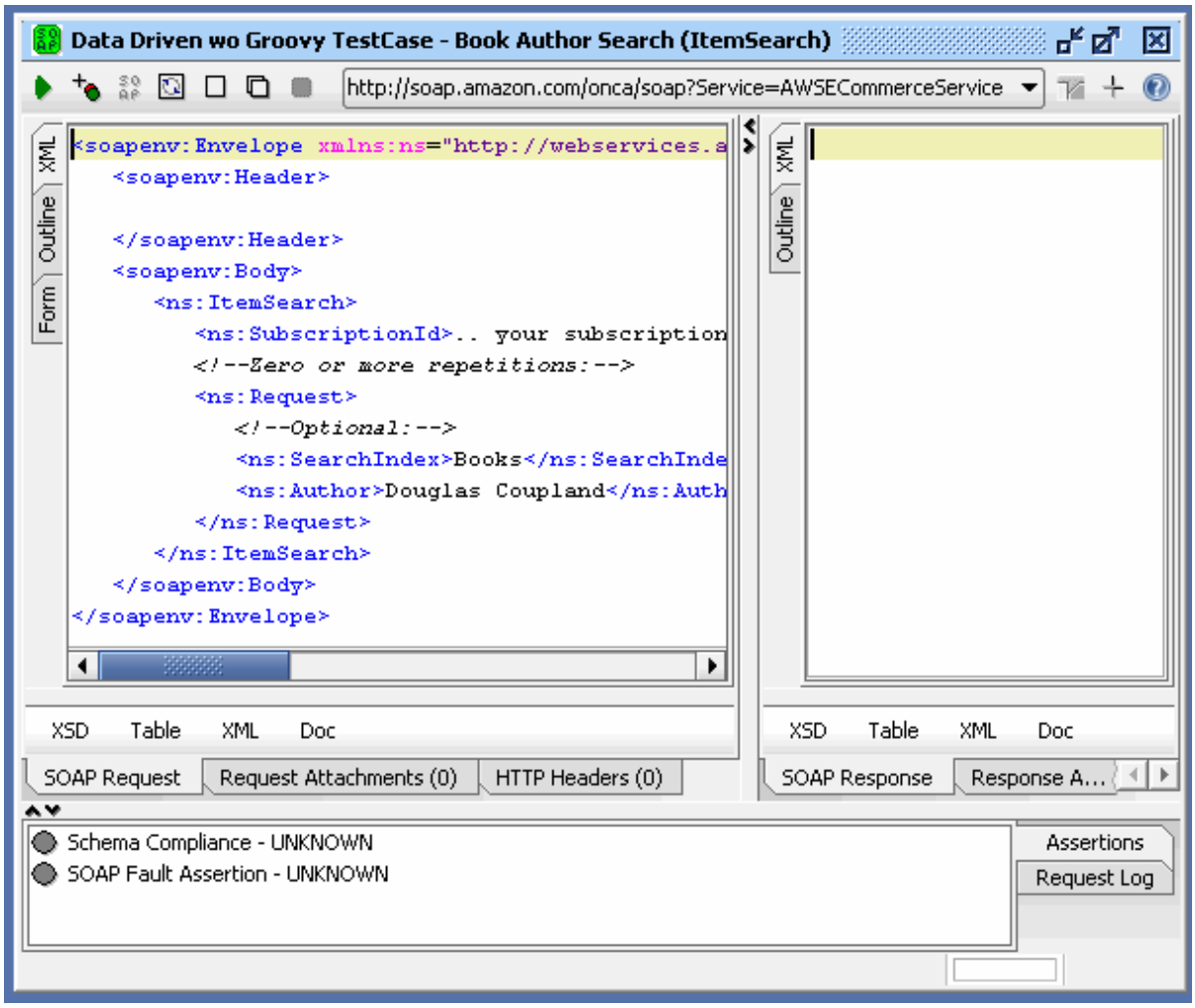
Add a "Schema Compliance" Assertion to validate the response and a "SOAP Fault" Assertion to catch unexpected errors.

Create Properties and Property Transfer

Insert a "Property Step" and add 2 properties named "SubscriptionID" and "Author" and give them some default value. After that insert a "Property Transfer" step and define 2 transfers each transferring the respective property to the target request. You should now have something like the following setup:



If you select the "Run" button in the Property Transfer Editor and open the Request editor for the search request you should see something like the following:



(the default values of the properties have been transferred to the request)

Reading the properties from an external properties file

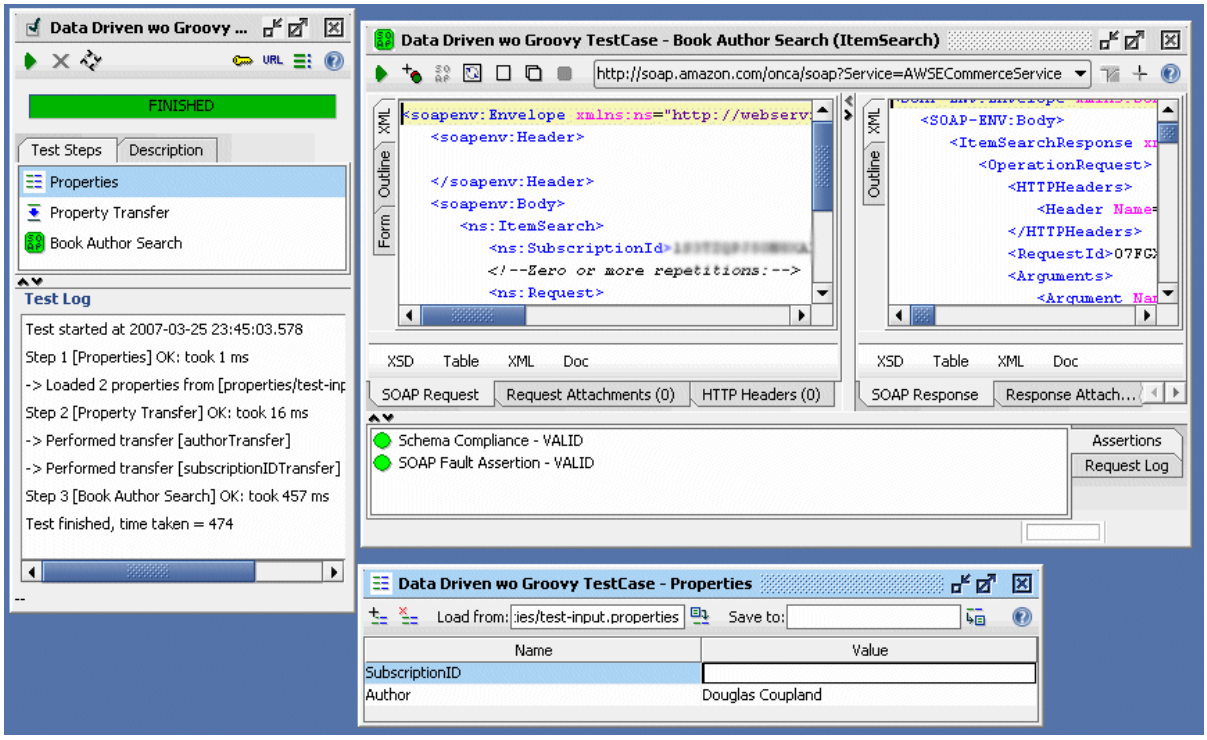
If you have a standard properties file containing the input you want to use, specify its name in the Property Step Editors source file field. In this example we will create a file in the current directory containing the following:

```
SubscriptionID=.. your subscription id ..
Author=Douglas Coupland
```

Save this to a "test-input.properties" file and specify that file in the Property Step Editor:

You're all set! Open the TestCase editor and run the TestCase and open the Request Editor to see the

results:



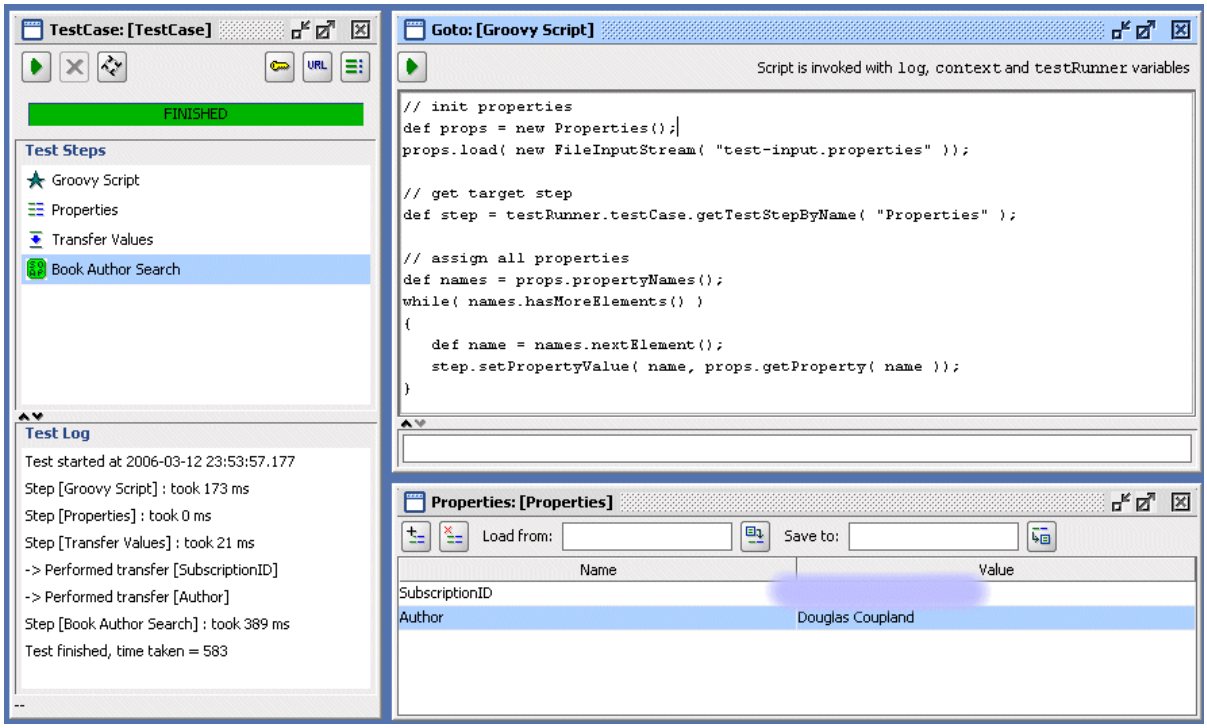
Setting the properties with Groovy

Instead of reading the properties from a properties file we may need to get them "somewhere else", for example from a database. The Groovy Script step will provide any flexibility you may need in this regard.. here we will just create a Groovy Step that "manually" reads the same properties file as created above.

Begin by inserting a Groovy Script step in the beginning of the TestCase and opening the Groovy Script Editor. Enter the following script:

```
// init properties
def props = new Properties();
props.load( new FileInputStream( "test-input.properties" ));

// get target step
def step = testRunner.testCase.getTestStepByName( "Properties" );

// assign all properties
def names = props.propertyNames();
while( names.hasMoreElements() )
{
   def name = names.nextElement();
   step.setPropertyValue( name, props.getProperty( name ));
}
```

The script reads the properties file and assigns all its contained properties to the target "Properties" Step. When running this you should get the exact same result as above:



Next: Template-Driven Testing

1.9.2 Template-Driven Testing

Template-Driven Testing

Template-Driven testing is an extension to Data-Driven testing where the same TestCase is run for a variable number of input values read from some external source. The example given here extends the Data-Driven Testing example to read a list of authors and their expected minimum books from a text file and then run the TestCase for each author and validate the number of books found.

To achieve this, we need to extend/modify the Data-Driven example as follows:

- Create an initial Groovy step that reads the external values and stores them in the current TestRunContext together with an index telling which author is currently tested
- Create another groovy step that assigns a set of values to the Properties Step for each TestCase run
- Create a groovy step at the end of the TestCase that checks the number of hits and loops back to the beginning if there are more test values to test

Here we go!

Read external values

The following example script reads all lines of the "testdata.txt" file into a list stored in the TestRunContext. Each line contains an author name and a number of minimum hits seperated by a comma, for example "King,230"

```
def list = []
new File( "testdata.txt" ).eachLine { line -> list.add( line ) }

log.info( "Read " + list.size() + " test values.." )

context.setProperty( "values", list )
context.setProperty( "index", 0 )
```

After reading the file into the list, it is stored in the TestRunContext together with an index telling which "row" is currently tested

Init properties with test values

This script reads the current row and assigns that rows author to the Properties Steps "Author" property

```
def values = context.getProperty( "values" )
def index = context.getProperty( "index" );

def str = values[index]
```

```
def ix = str.indexOf( "," )
def author = str.substring( 0, ix )

def props = testRunner.testCase.getTestStepByName( "Properties" )
props.setPropertyValue( "Author", author )

log.info( "set author to [" + author + "]" )
```

After this, the previously avaialable Property Transfers and Requests are run as configured:

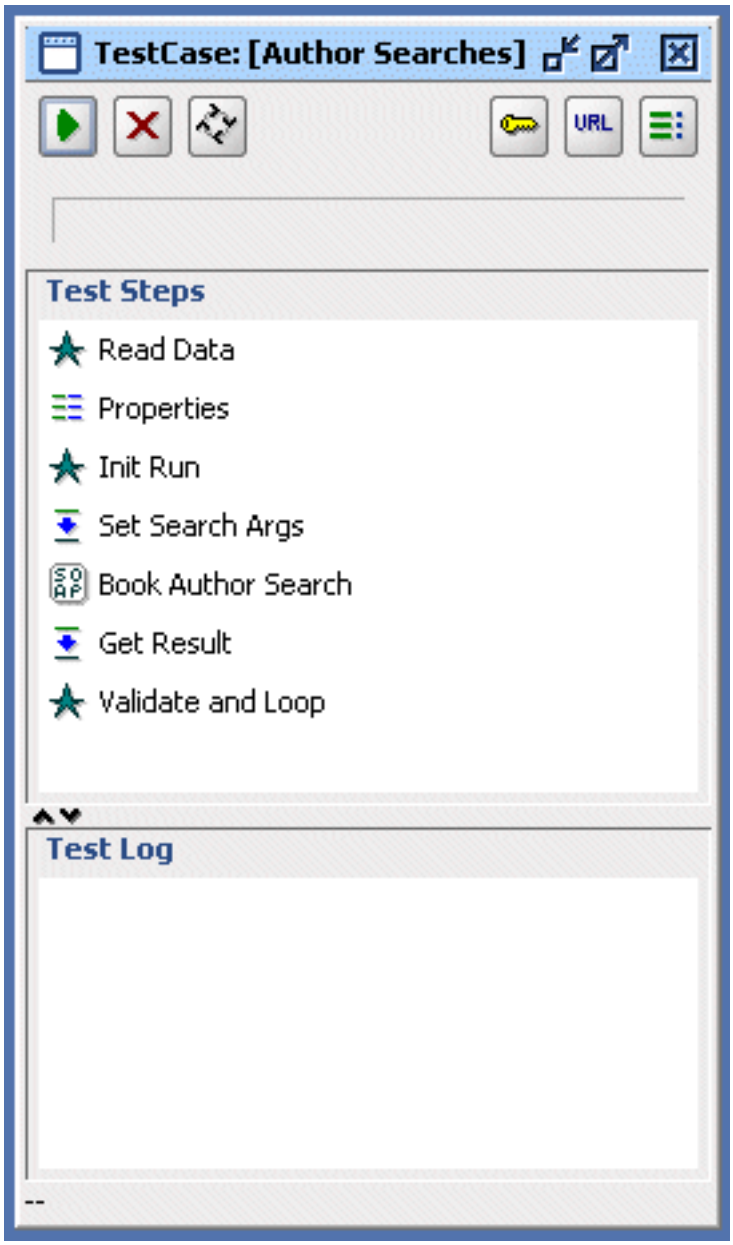
- The first transfer copies the author and subscription id to the search request
- The request performs the book search
- The second transfer copies the number of hits to the "ResultCount" property

Validate and move to next

This step checks the number of hits against the provided text value and fails if the number of hits is not sufficient. Otherwise, the script advances the test index to the next row and loops back if that row exists

```
def values = context.getProperty( "values" )
def index = context.getProperty( "index" );

def str = values[index]
def ix = str.indexOf( "," )

def props = testRunner.testCase.getTestStepByName( "Properties" )
def resultCount = props.getPropertyValue( "ResultCount" )
def count = str.substring( ix+1 )
def author = props.getPropertyValue( "Author" )

if( count > resultCount )
{
   throw new Exception( "not enough hits for author [" + author +
                        "], expected " + count + ", got " + resultCount )
}
else
{
   log.info "got " + resultCount + " hits for [" + author + "], required " + count
   if( ++index < values.size() )
   {
      context.setProperty( "index", index )
      testRunner.gotoStepByName( "Init Run" )
   }
   else
   {
      log.info "Finished TestCase, tested " + values.size() + " values"
   }
}
```

Running the TestCase



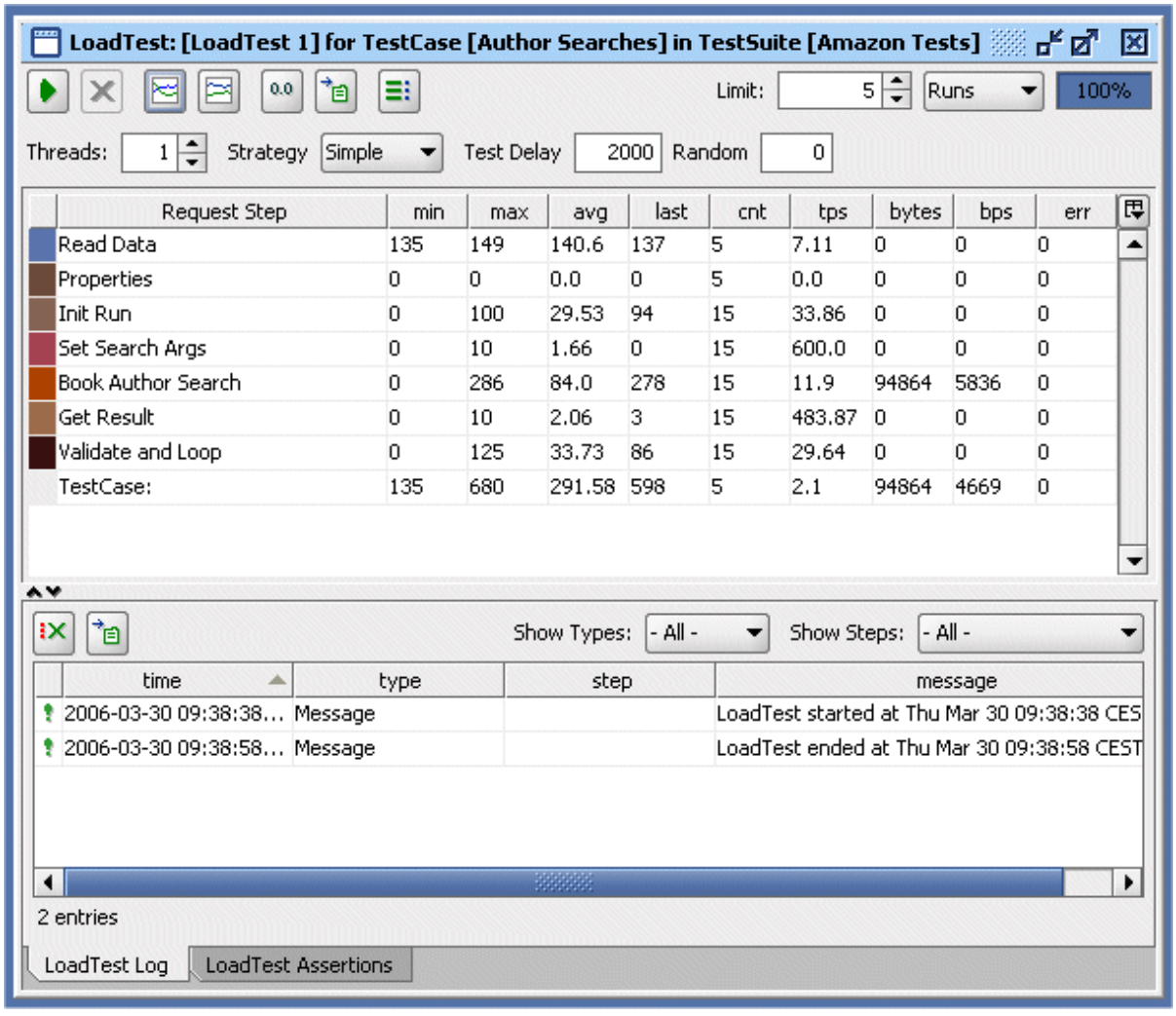
and running it with the following testdata.txt

```
coupland,150
king,150
shakespeare,150
```

shows the following result in the groovy log:

```
09:35:07 : INFO : Read 3 test values..
09:35:07 : INFO : set author to [coupland]
09:35:17 : INFO : got 170 hits for [coupland], required 150
09:35:17 : INFO : set author to [king]
09:35:17 : INFO : got 17787 hits for [king], required 150
09:35:18 : INFO : set author to [shakespeare]
09:35:18 : INFO : got 6098 hits for [shakespeare], required 150
09:35:18 : INFO : Finished TestCase, tested 3 values
```

The following image shows a simple LoadTest which ran the TestCase 5 times, as you can see each step in the loop was called 15 times in total, ie once for each test value



Next: Interactive Testing

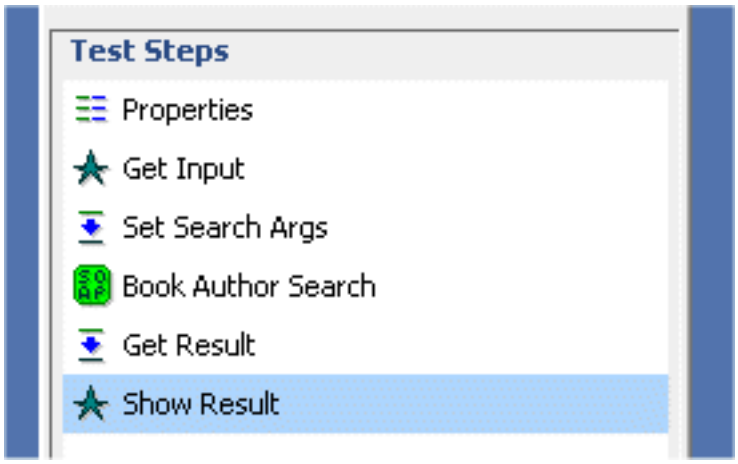
1.9.3 Interactive Testing

Interactive Testing

Using the Groovy TestStep, it is possible to create both input and output dialogs, which can be useful if you for example want to make it easy to run your tests with user-supplied indata or if you want to provide a soapUI project containing TestCases as "demonstrations" of your web services for your clients.

In this example we will create a TestCase that performs an Amazon Book search on Author and displays the number of hits. We need the following TestSteps:

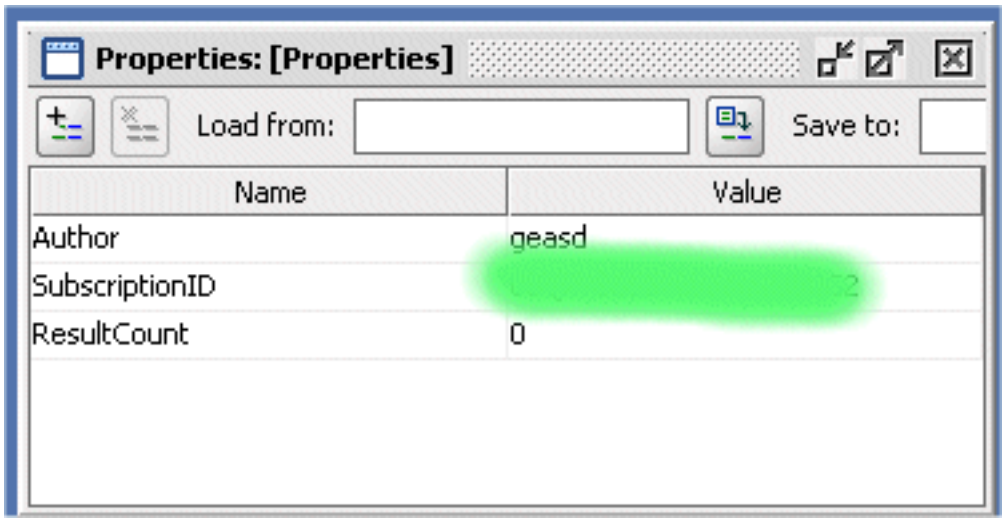
- A Properties Step to hold global properties
- A Groovy Step for showing an input dialog
- A Property Transfer that transfers the input value to the search request
- A Request Step that performs the search
- A Property Transfer that gets the result to a global property
- A Groovy Step for displaying the result



Properties

The following properties will be defined:

- **Author** : holds the author entered by the user
- **SubscriptionID** : holds the Amazon subscription id
- **ResultCount** : holds the result count for display



Groovy Input

The following script will be used to ask the user for the author to search on:

```
// create dialog
def dialog = com.eviware.soapui.support.UISupport.createConfigurationDialog( "Amazon
Query" );
dialog.addTextField( "Author", "The Author to search on" );

// init values and show
def map = new java.util.HashMap();
map.put( "Author", "" );

if( dialog.show( map ))
{
   // get target step
   def step = testRunner.testCase.getTestStepByName( "Properties" );

   // assign
   step.setPropertyValue( "Author", map.get( "Author" ));
}
else testRunner.cancel( "No author to search on!" );
```

When running the script, the user will be prompted with a dialog and the entered value will be set in the Global Properties Step.

Search Args Transfer

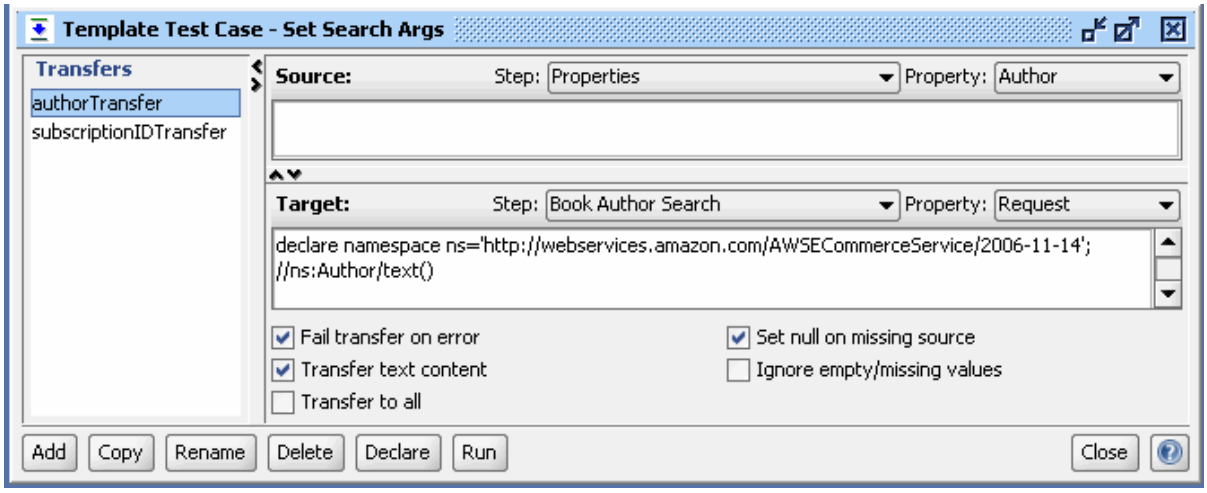
The Search Args Transfer contains 2 property transfers, one for the SubscriptionID and one for the Author property. Each is transferred from the Global Properties step to the Search Request using an XPath epression. For the SubscriptionID this is:

```
declare namespace ns='http://webservices.amazon.com/AWSECommerceService/2006-03-08';
//ns:SubscriptionId/text()
```

and for the Author it is

```
declare namespace ns='http://webservices.amazon.com/AWSECommerceService/2006-03-08';
//ns:Author/text()
```

(See the following Request Step to see the target XML)



Using Property Replacement

As of soapUI 1.6, it is possible to use the common `${propertyName}` to expand properties as described in Property Expansion . Using this feature we can remove the "Search Args Transfer" step and instead use the following request:

```
<ns:ItemSearch>
   <ns:SubscriptionId>${SubscriptionId}</ns:SubscriptionId>
   <ns:Request>
      <ns:SearchIndex>Books</ns:SearchIndex>
      <ns:Author>${Author}</ns:Author>
   </ns:Request>
</ns:ItemSearch>
```

This will result in the values for SubscriptionId and Author being copies into the request just before it is sent (the inserted values will not be written into the request seen in the editor).

Search Request

The following search request will be used:

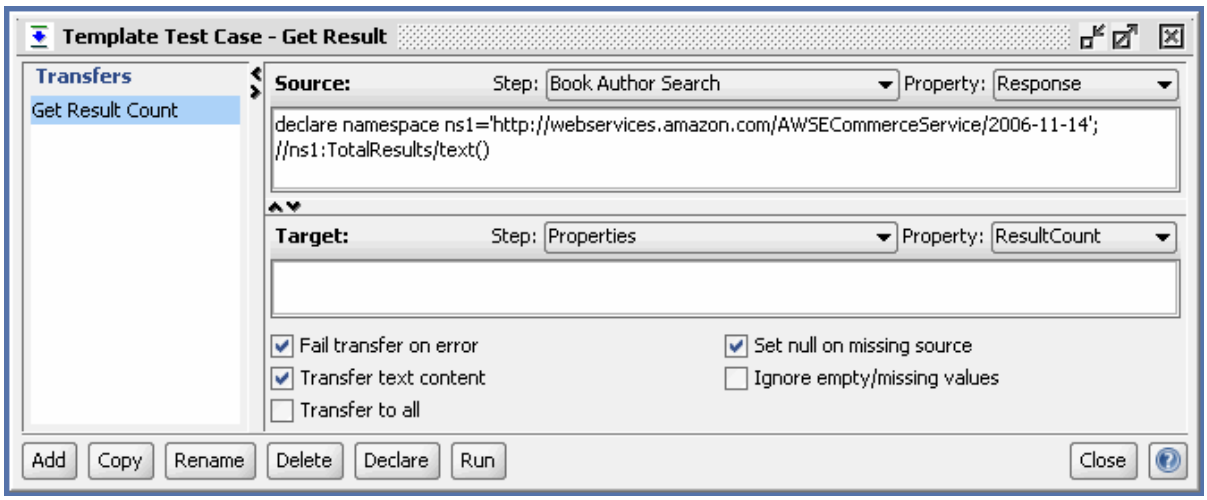
```
<soapenv:Envelope
xmlns:ns="http://webservices.amazon.com/AWSECommerceService/2006-03-08"
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns:ItemSearch>
         <ns:SubscriptionId>?</ns:SubscriptionId>
```

```
         <ns:Request>
            <ns:SearchIndex>Books</ns:SearchIndex>
            <ns:Author>?</ns:Author>
         </ns:Request>
      </ns:ItemSearch>
   </soapenv:Body>
</soapenv:Envelope>
```

Result Transfer

This step extracts the number of hits in the result and transfers it to the Global Properties "ResultCount" property using the following XPath expression:

```
declare namespace
ns1='http://webservices.amazon.com/AWSECommerceService/2006-03-08';
//ns1:TotalResults/text()
```



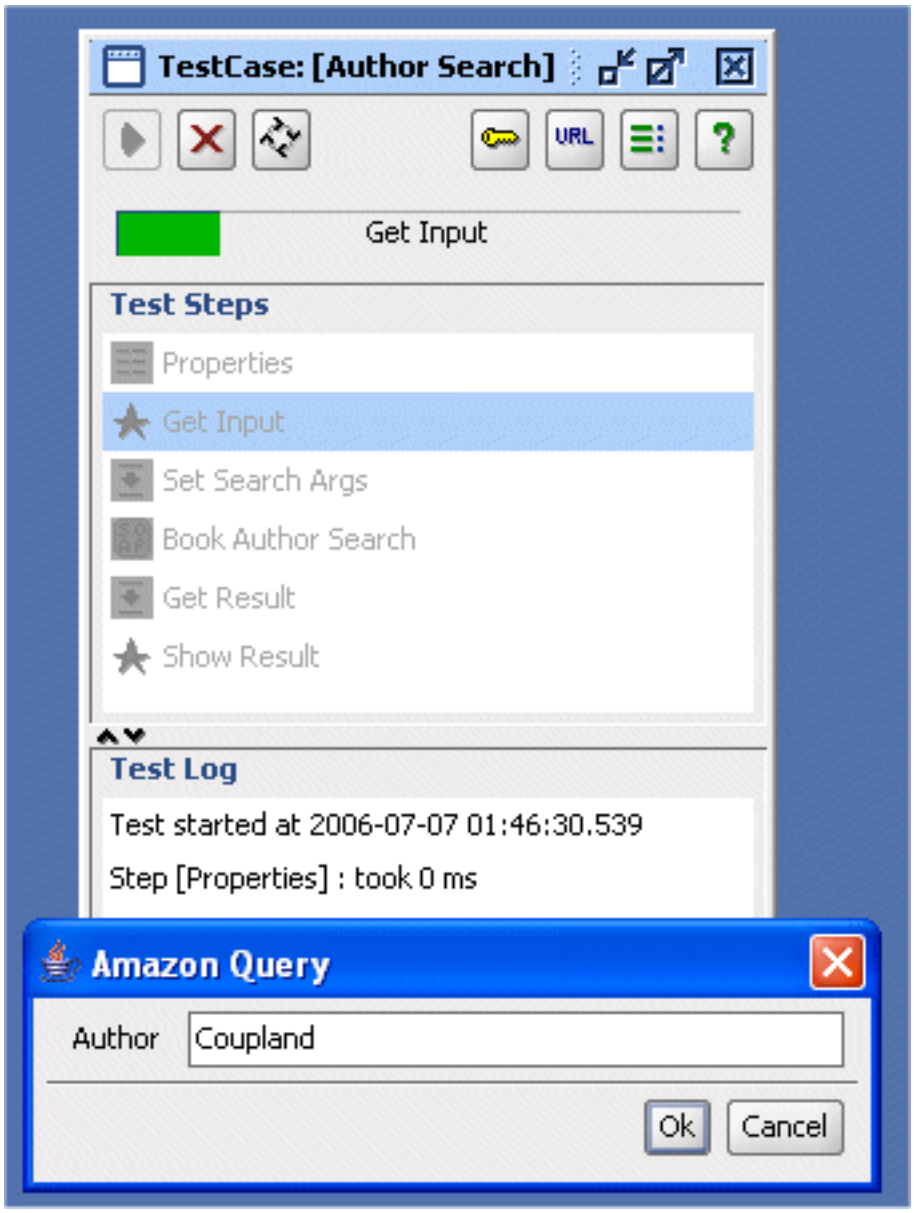
Groovy Display Result

The following script will be used to display the result:

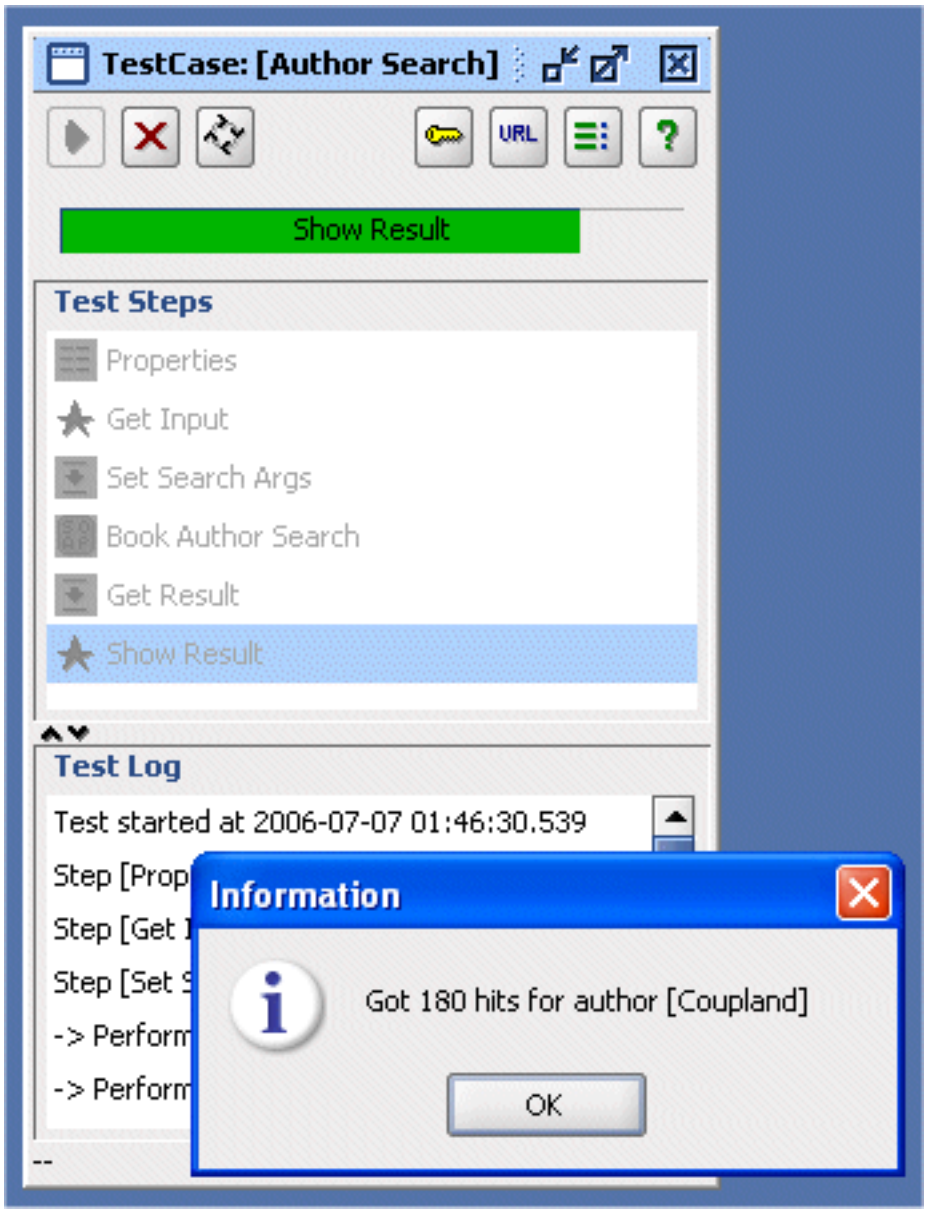
```
// get target step
def step = testRunner.testCase.getTestStepByName( "Properties" );
com.eviware.soapui.support.UISupport.showInfoMessage(
   "Got " + step.getPropertyValue( "ResultCount" ) + " hits for author [" +
   step.getPropertyValue( "Author" ) + "]" );
```

Running the TestCase

Running the TestCase produces the following input dialog:



and shows the following result:



Summary

The example above is very simple but shows the possibilites, creating more complex UI's and flows is definitely possible, but maybe questionable as soapUI is not (yet?) a BPEL tool and has been created for other purposes. But then again... if it works and helps you out... ;-)

Next: Surveillance Testing

1.9.4 Surveillance Testing

Surveillance Testing

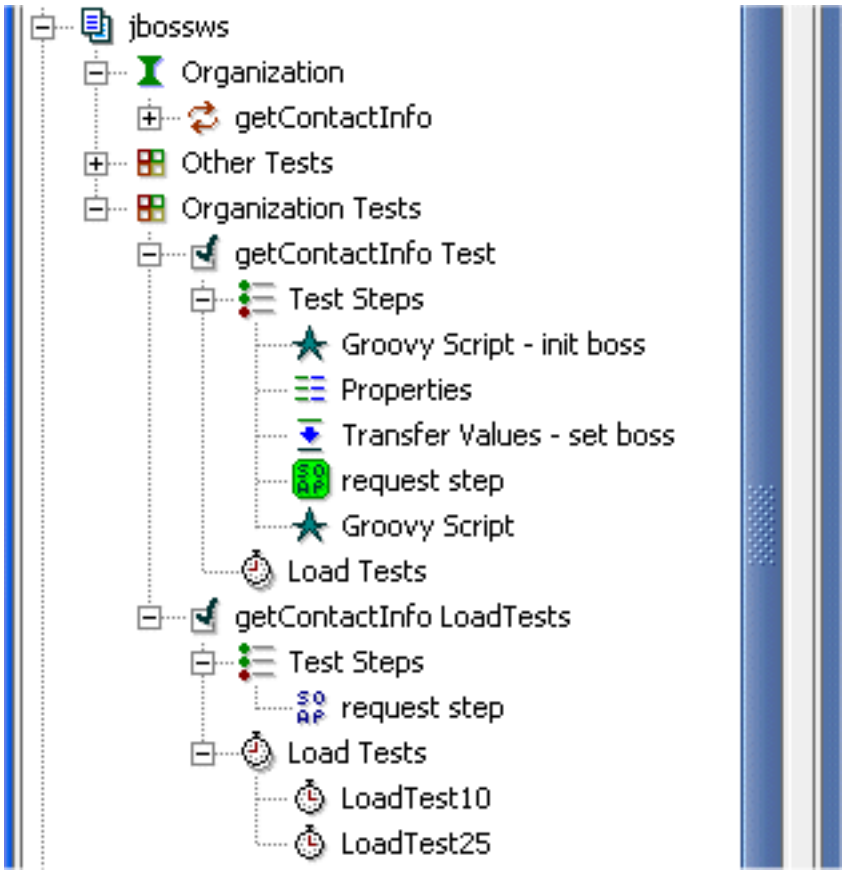
Surveillance testing is here used to describe a setup where a number of both functional and load tests are run using some kind of scheduling tool to continuously validate the availability to some web service. I this example, we will use the open-source "luntbuild" tool for setting up both functional and load test to be run regularly and validated. A notification will be sent if any of the tests fail.

Creating the tests

The tests used in the sample will test one of the apps included in with jbossws (the JBoss web-service stack). They consist of:

- A "getContactInfo" TestCase which validates that the response returned by a random input value contains that value back. The value is generated and validated using 2 groovy steps.
- A number of LoadTests named LoadTestXX, where XX is the number of threads, each validating that the TPS value never goes under 100. The LoadTests use the Simple Strategy with no delay and both run for 5000 requests.

The image to the right shows this setup in the navigator



Running from the command-line

Before we automate the running of the tests using luntbuild, let's just check that they run ok from the command line. We will use the command-line runners in this example, let's start by running the functional TestCase from within the folder where the soapUI project file is located:

```
testrunner.bat "C:\Documents and Settings\ole.matzura\My
Documents\jbossws-soapui-project.xml"
                  -s"Organization Tests" -c"getContactInfo Test" -r
```

This produces the following output:

```
SoapUI SNAPSHOT TestCase Runner
10:47:36,703 INFO  [SoapUITestCaseRunner] setting projectFile to [C:\Documents and
Settings\ole.matzura\My Documents\jbossws-soapui-project.xml]
10:47:36,703 INFO  [SoapUITestCaseRunner] setting testSuite to [Organization Tests]
10:47:36,703 INFO  [SoapUITestCaseRunner] setting testCase to [getContactInfo Test]
10:47:37,284 INFO  [WsdlProject] Loaded project from [C:\Documents and
Settings\ole.matzura\My Documents\jbossws-soapui-project.xml]
10:47:38,616 INFO  [SoapUITestCaseRunner] Running soapui tests in project [jbossws]
10:47:38,616 INFO  [SoapUITestCaseRunner] Running soapui suite [Organization Tests],
runType = SEQUENTIAL
10:47:38,626 INFO  [SoapUITestCaseRunner] Running soapui testcase [getContactInfo
Test]
10:47:38,636 INFO  [SoapUITestCaseRunner] runing step [Groovy Script - init boss]
```

```
10:47:39,176 INFO  [SoapUITestCaseRunner] runing step [Properties]
10:47:39,176 INFO  [SoapUITestCaseRunner] runing step [Transfer Values - set boss]
10:47:39,557 INFO  [SoapUITestCaseRunner] runing step [request step]
Retrieving document at
'http://lpt-olma:8080/ws4ee-samples-server-ejb/Organization?wsdl'.
10:47:40,398 INFO  [SchemaUtils] Loading schema types from
[http://lpt-olma:8080/ws4ee-samples-server-ejb/Organization?wsdl]
10:47:40,398 INFO  [SchemaUtils] Getting schema
http://lpt-olma:8080/ws4ee-samples-server-ejb/Organization?wsdl
10:47:40,568 INFO  [SoapUITestCaseRunner] Assertion [Schema Compliance] has status
VALID
10:47:40,568 INFO  [SoapUITestCaseRunner] runing step [Groovy Script]
10:47:40,689 INFO  [log] boss name matched [testsd1143708459106]
10:47:40,689 INFO  [SoapUITestCaseRunner] Finished running soapui testcase
[getContactInfo Test], time taken: 1065ms, status: FINISHED
10:47:40,689 INFO  [SoapUITestCaseRunner] Skipping testcase [getContactInfo
LoadTests], filter is [getContactInfo Test]
10:47:40,689 INFO  [SoapUITestCaseRunner] soapui suite [Organization Tests] finished
in 2073ms

SoapUI SNAPSHOT TestCaseRunner Summary
------------------------------
Time Taken: 2070ms
Total TestSuites: 1
Total TestCases: 1
Total TestSteps: 5
Total Request Assertions: 1
Total Failed Assertions: 0
Total Exported Results: 0
```

Likewise, running the LoadTests is done as follows:

```
loadtestrunner.bat "C:\Documents and Settings\ole.matzura\My
Documents\jbossws-soapui-project.xml"
   -s"Organization Tests" -c"getContactInfo LoadTests" -r
```

Producing the following:

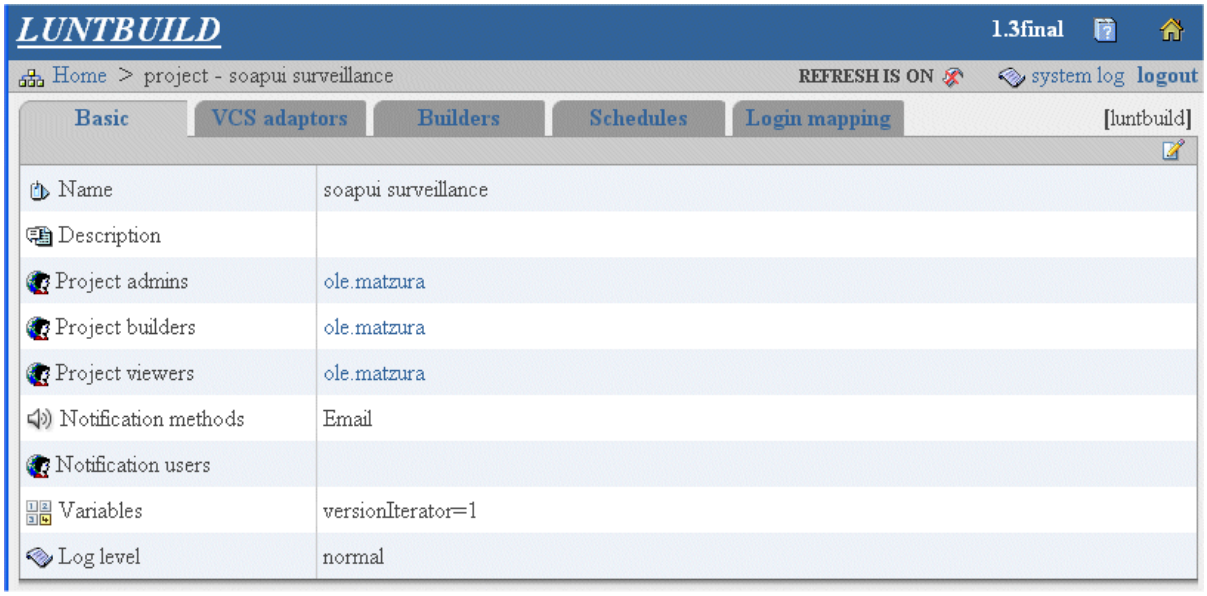
```
SoapUI 1.5beta2 LoadTestRunner
21:16:07,912 INFO  [SoapUILoadTestRunner] setting projectFile to [C:\Documents and
Settings\ole.matzura\My Documents\j..
21:16:07,932 INFO  [SoapUILoadTestRunner] setting testSuite to [Organization Tests]
21:16:07,942 INFO  [SoapUILoadTestRunner] setting testCase to [getContactInfo
LoadTests]
21:16:09,134 INFO  [WsdlProject] Loaded project from [C:\Documents and
Settings\ole.matzura\My Documents\jbossws-soapu
21:16:11,458 INFO  [SoapUILoadTestRunner] Skipping testcase [getContactInfo Test],
filter is [getContactInfo LoadTests
21:16:11,458 INFO  [SoapUILoadTestRunner] Running LoadTest [LoadTest10]
21:16:11,848 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest10] progress: 0.0
21:16:12,850 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest10] progress: 0.0144
.. etc ..
21:16:54,399 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest10] progress: 0.9452
21:16:55,391 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest10] progress: 0.9744
21:16:56,392 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest10] finished with status
FINISHED
21:16:56,392 INFO  [SoapUILoadTestRunner] Exporting log and statistics for LoadTest
```

```
[LoadTest10]
21:16:56,512 INFO  [SoapUILoadTestRunner] Exported 1 log items to
[LoadTest10-log.txt]
21:16:56,512 INFO  [SoapUILoadTestRunner] Exported 0 error results
21:16:56,512 INFO  [SoapUILoadTestRunner] Exported 2 statistics to
[LoadTest10-statistics.txt]
21:16:56,602 INFO  [SoapUILoadTestRunner] Running LoadTest [LoadTest25]
21:16:57,113 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest25] progress: 0.0046
21:16:58,135 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest25] progress: 0.0348
.. etc ..
21:17:37,742 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest25] progress: 0.9506
21:17:38,773 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest25] progress: 0.9748
21:17:39,845 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest25] finished with status
FINISHED
21:17:39,865 INFO  [SoapUILoadTestRunner] Exporting log and statistics for LoadTest
[LoadTest25]
21:17:40,095 INFO  [SoapUILoadTestRunner] Exported 1 log items to
[LoadTest25-log.txt]
21:17:40,095 INFO  [SoapUILoadTestRunner] Exported 0 error results
21:17:40,095 INFO  [SoapUILoadTestRunner] Exported 2 statistics to
[LoadTest25-statistics.txt]
21:17:40,105 INFO  [SoapUILoadTestRunner] soapui suite [Organization Tests] finished
in 88647ms
```

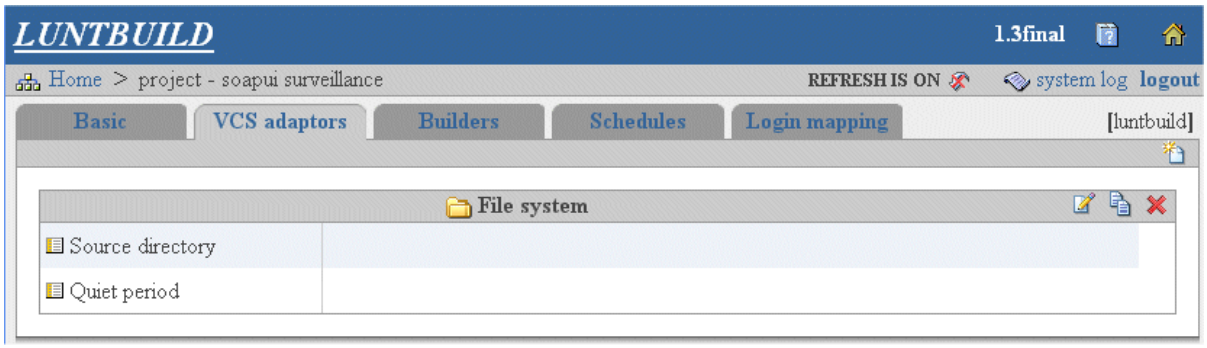
Setting up the Luntbuild Project

Download and install luntbuild from the sourceforge site. Once you have it "figured out" and running, do the following:

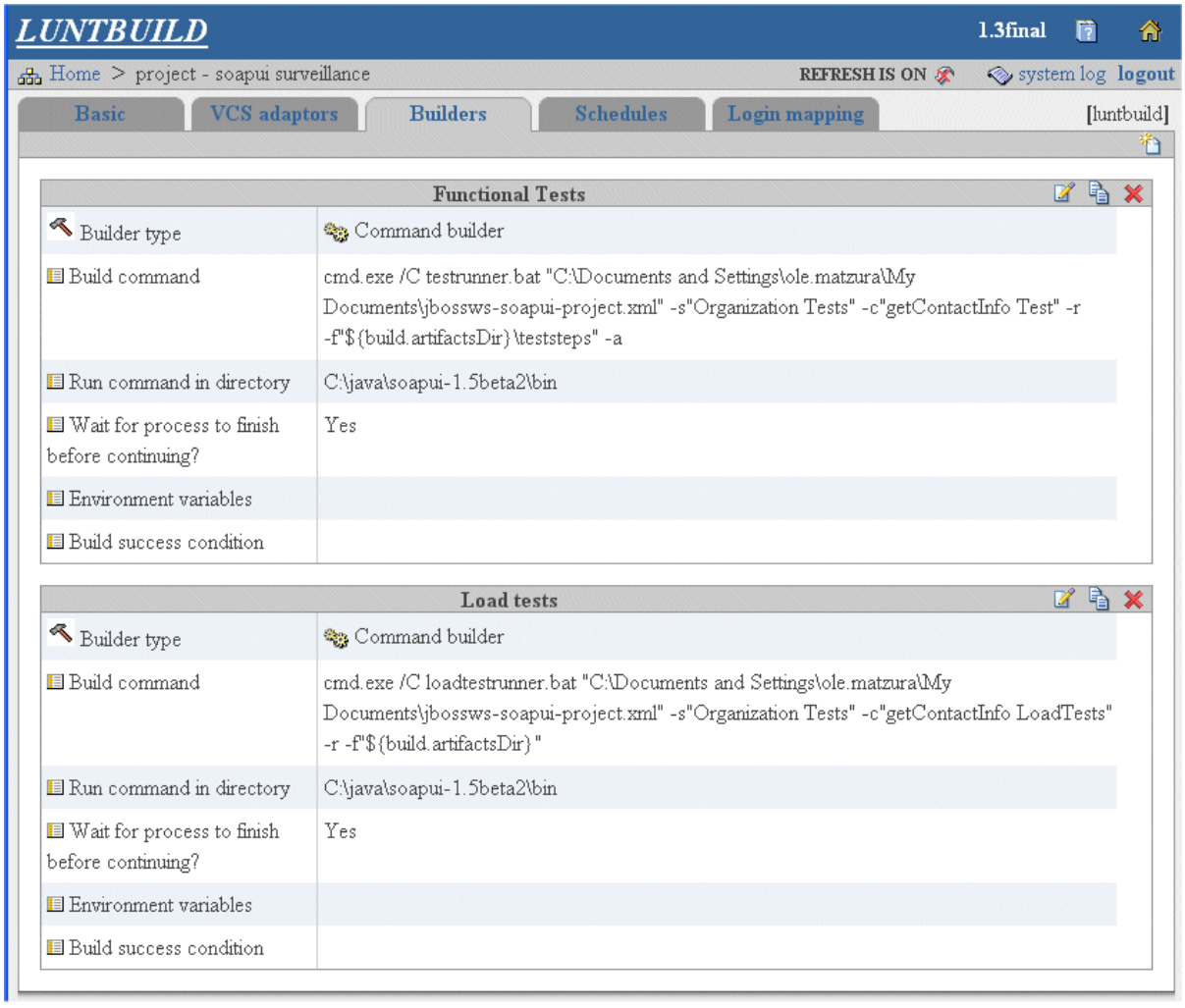
Login to luntbuild and create a new project:



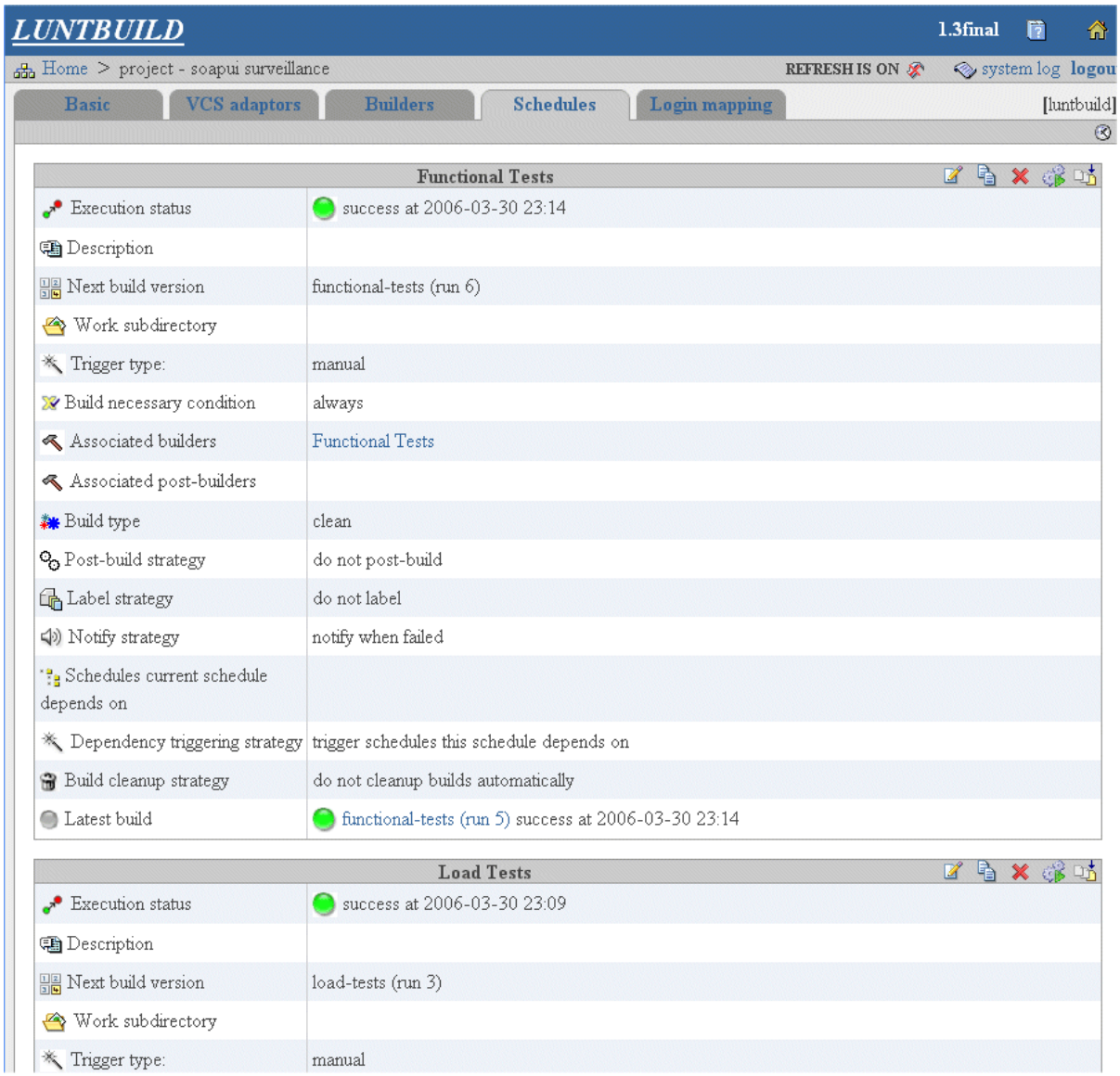
Create an empty VCS File System Adapter (we will host the soapUI project file for our tests localy in this example, better would of course be to have them in a version-control system):



Create one builder for each testrunner, set the build command to the entire command-line invocation of one of the soapUI testrunners prefixed with "cmd.exe /C". Remember to set the directory to the soapui/bin directory.

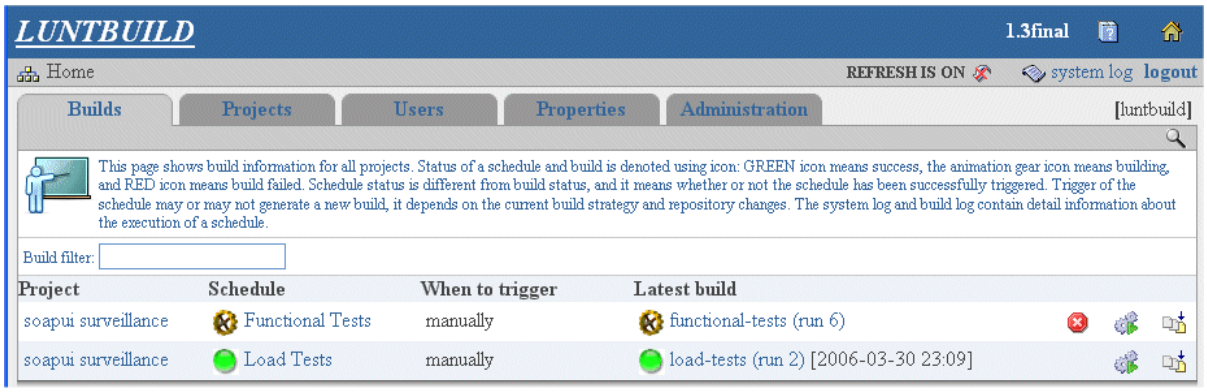


Create one schedule for each builder and schedule accordingly (in the example below the trigger is set to manual but should be set to some more repetitive timing)

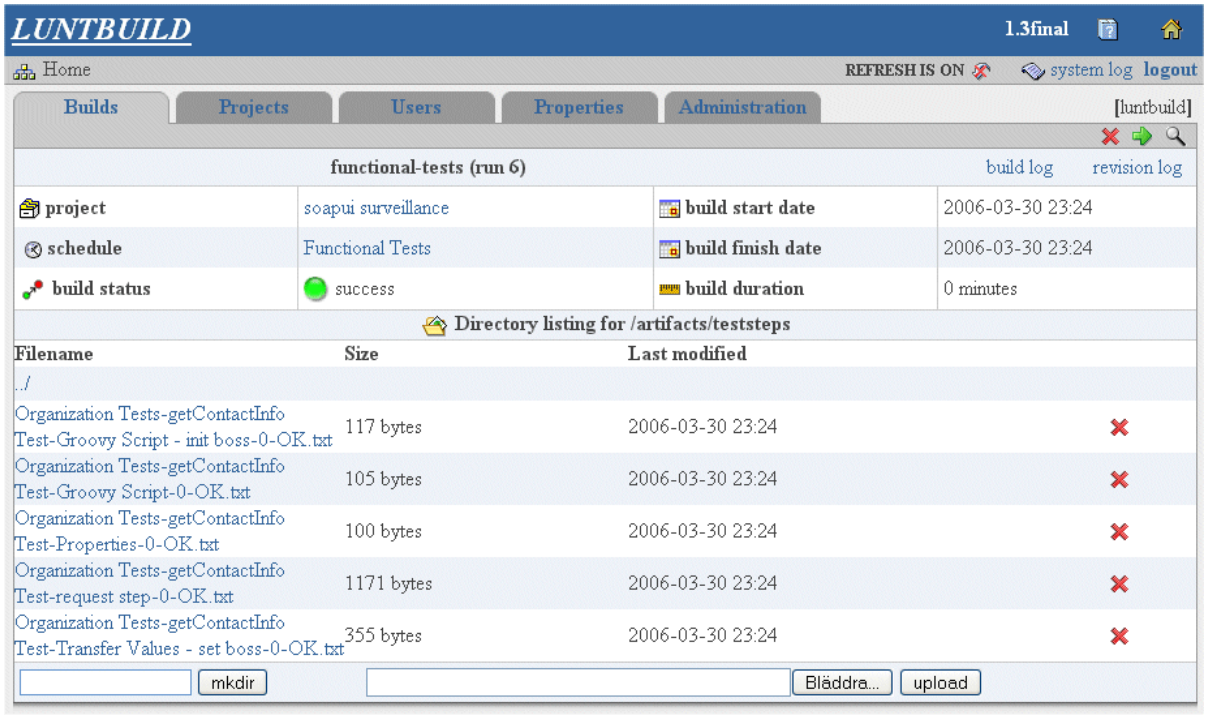


Run the Tests..

Go back to the main page and run the first schedule (press the green arrow to the right of the schedule and select "Save" at the bottom of the run page):

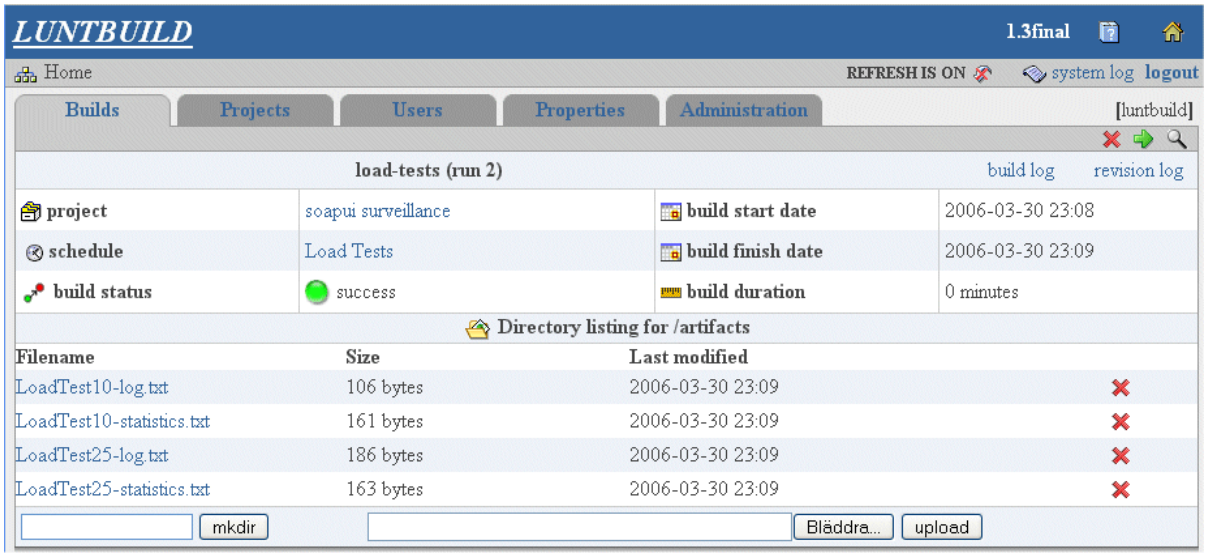


Click on the schedules Latest Build and into the teststeps directory (since we added the -f option in our builder the output teststep results are written to that folder):



To view the entire output written to system.out select the "build log" link on the top right:

Go back to the main page and run the second schedule and click on its Latest Build, the created statistics will be in the output folder:



Click on one of the statistics files to see its contents:

```
Request Step,min,max,avg,last,cnt,tps,bytes,bps,err
request step,3,1189,95.25,76,5000,262.45,2160000,84454,0
Total:,3,1189,95.25,76,5000,262.45,2160000,84454,0
```

Summary

The above shows (rather quickly) how to set up a complete "Continuous Testing" environment which can be used for surveillance testing of Web Services... play around with the above and add schedules, more tests, notifications, etc to get the setup that you want.

Next: Tool Integrations

1.10 Tool Integrations

Tool Integrations

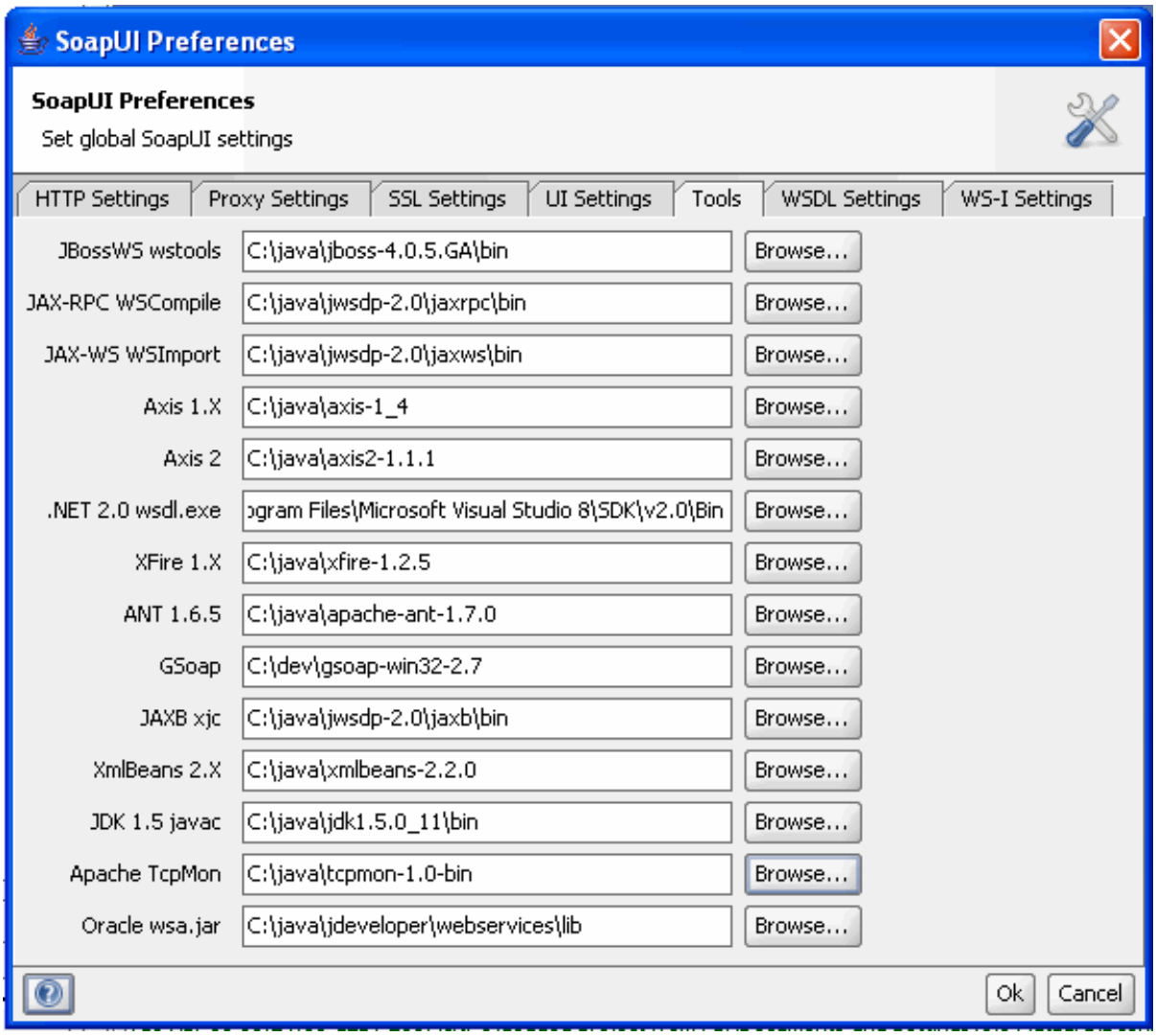
soapUI provides integrated support for a number of open-source webservice toolkits and utilities, including;

1. **Code Generation Tools** allow generation of client/implementation-artifacts from an existing WSDL definition in a soapUI project.
2. **WSDL Generation Tools** allow generation of WSDL's from code which will be automatically added to a soapui-project.
3. **WS-I Tools** integrates WS-I Basic Profile validation tools into soapUI
4. **Apache Tcp-Mon** provides TCP-traffic monitoring functionality

If you would like us to integrate any other tools (AJAX, PHP, etc..), dont hesitate to send us a link to the documentation for the command-line utility that should be invoked together with some examples and we will consider integrating the tool for you.

Configuration

Before any of the integrations can be used, the respective tools need to be installed seperately (they are not bundled with soapUI) and their location configured on the "Integrated Tools" tab in the soapui preferences:



(the image shows possible configuration values for all integrations)

Next: Code Generation Tools

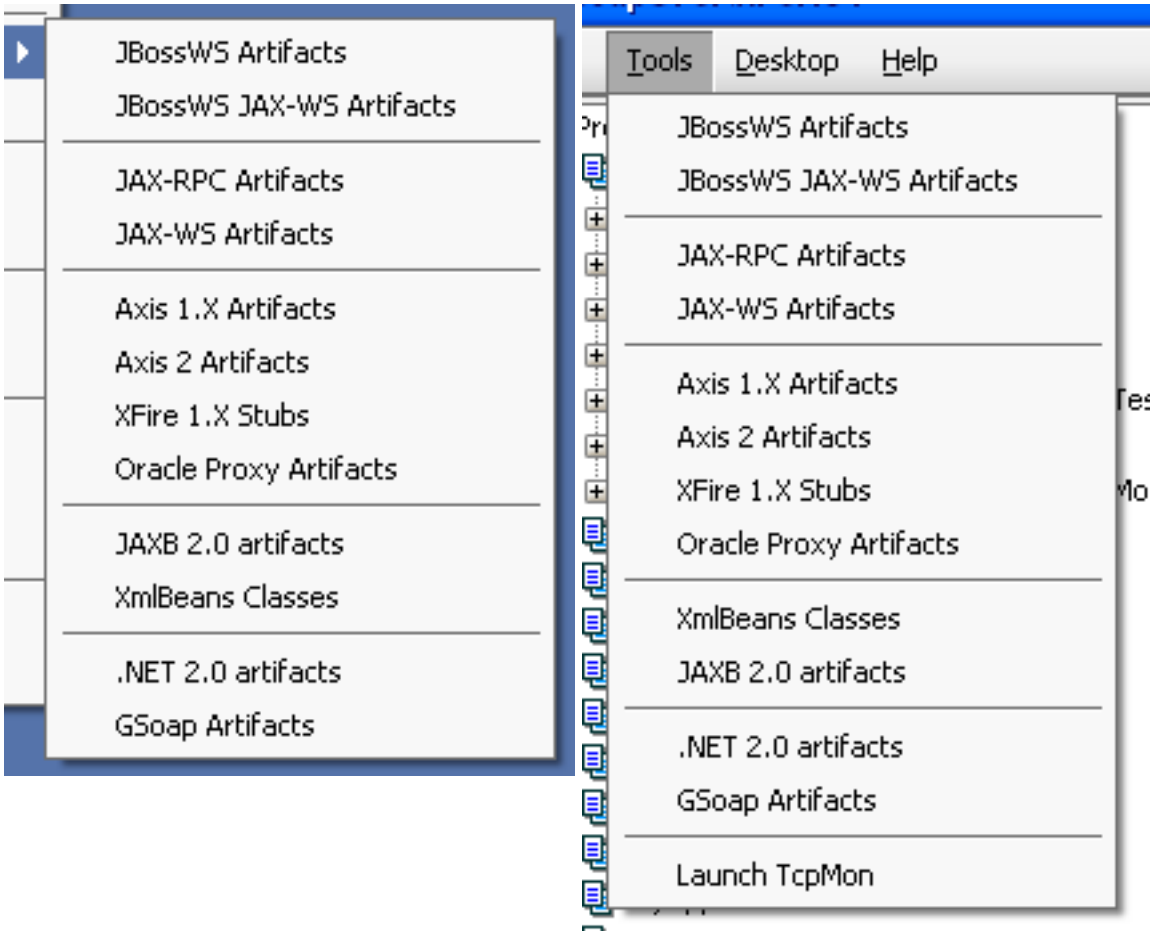
1.10.1 Code Generation

Code Generation Tools

The following code generation tools have currently been integrated and are available from the main Tools menu or form the Interface right-button menu: JBossWS wstools , JBossWS wsconsume , JAX-RPC , JAX-WS , Axis 1.X , Axis 2 , XFire 1.X , Oracle wsa , XmlBeans 2.X , JAXB 2.X , GSoap 2.X and .NET 2.0 .

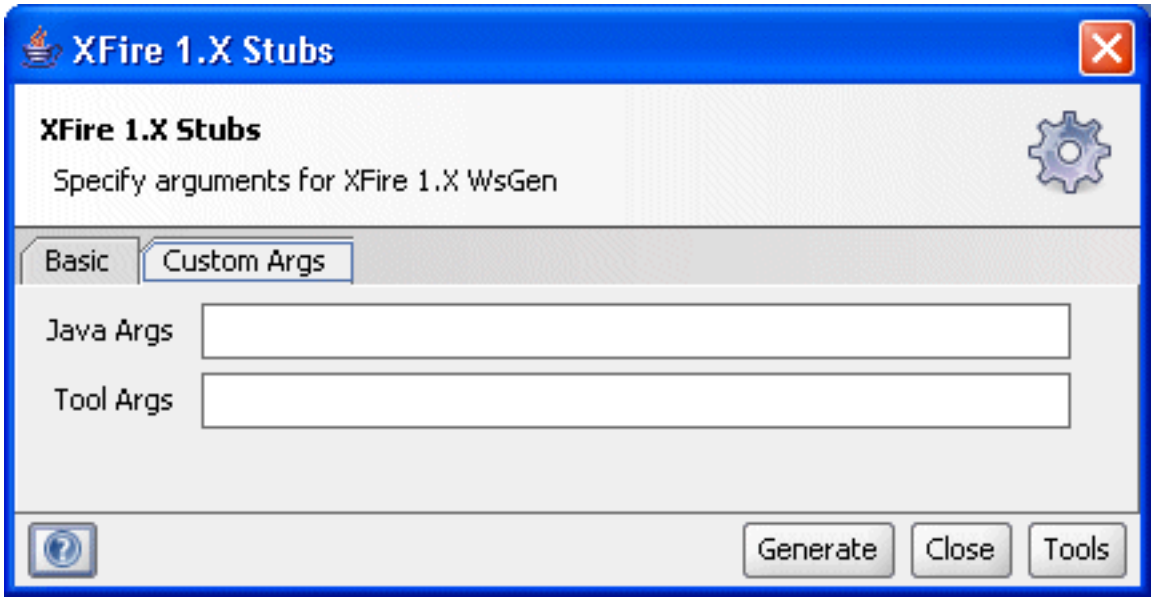
Each integration has a number of tabs for configuring the corresponding tools options and adding arbitrary command-line arguments. If the tool is invoked from the Interface menu, relevant values will be pre-entered into the dialogs (WSDL url, Namespaces, etc.) and the integrations remember their settings, making it easy to rerun the desired tools if changes are made to the external source(s) or WSDL(s).

If the WSDL for the current interface has been cached, an option will be available for using either the cached WSDL or the "online" WSDL. If selecting to use the cached WSDL, soapUI will first export the WSDL to a temporary directory and specify this location as the WSDL path.



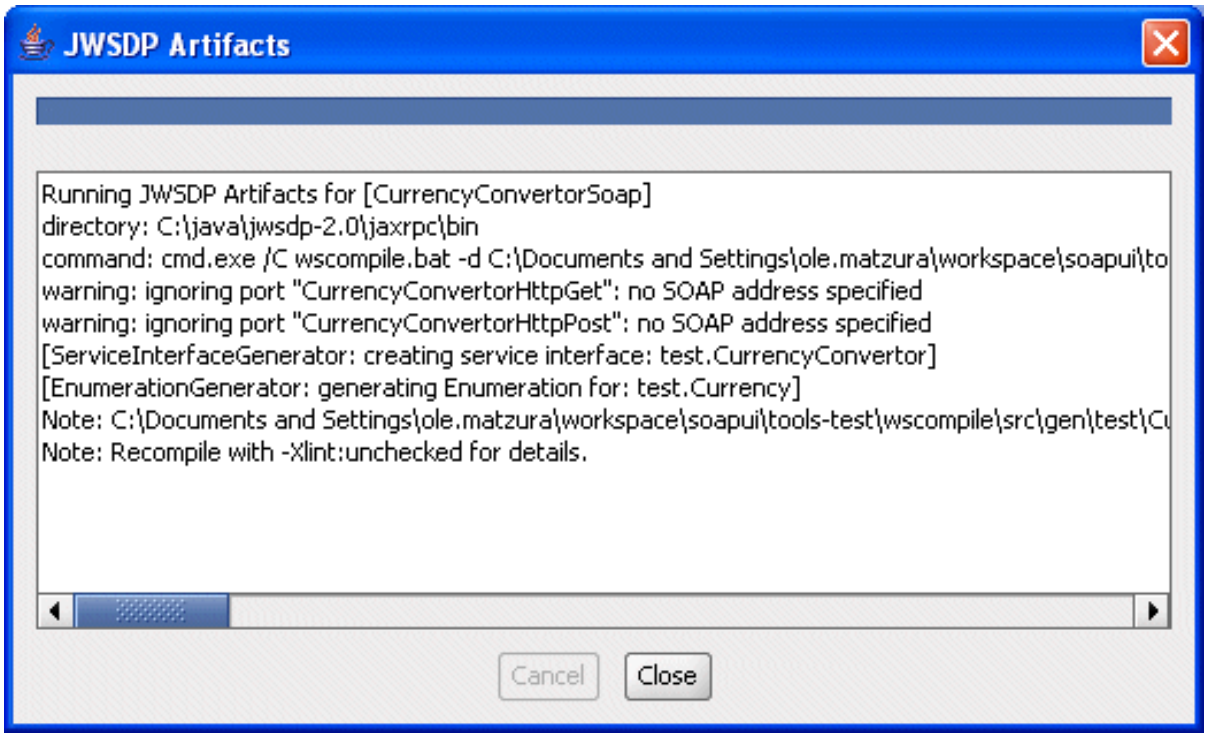
Custom Arguments

All integrations have a "Custom args" tab allowing the specification of custom arguments to the invoked tool(s), for example those that are not supported by soapUI in intermediate versions (before we catch up :-). The following dialog for the XFire integration allows both cutsom args to java and to the WsGen tool itself:



Running a tool

Once the tool is configured as required by its dialog, a "Generate" option will open a seperate window showing the input/output of the invoked tools process. During the process execution a "Cancel" button will be available for cancelling the ongoing process, otherwise a success/error message will be shown when the tool is finished. The log window will be available until it is closed allowing detailed viewing of the process output (for debugging purposes..). The following image shows the process runner window after a successfull invocation to the JWSDP wscompile tool:



JBossWS wstools

Tool	Version(s)	Documentation	Installation / Usage
JBossWS wstools	versions 1.0.X	JBossWS User Guide	Install the latest JBossWS stack and install/configure as required. Specify the directory containing the wstools script in the Tool Integrations settings dialog. The dialog has a "Show Config" button that displays the wstools config file generated for the current settings.

JBossWS wsconsume

Tool	Version(s)	Documentation	Installation / Usage
JBossWS wstools	versions 1.0.X	JBossWS User Guide	Install the latest JBossWS 1.2 or newew stack and install/configure as required. Specify the directory containing the wstools script in the Tool Integrations settings dialog. The dialog has a "Show Config" button that displays the wstools config file generated for the current settings.

JWSDP JAX-RPC/wscompile

Tool	Version(s)	Documentation	Installation / Usage
JWSDP JAX-RPC/wscompile , tested with versions 1.6 and 2.0	tested with versions 1.6 and 2.0	wscompile	Install the desired JWSDP version and install/configure as required. Specify the directory containing the wscompile script in the Tool Integrations settings dialog.

JWSDP JAX-WS/wsimport

Tool	Version(s)	Documentation	Installation / Usage
JWSDP JAX-WS/wsimport , tested with versions 1.6 and 2.0	tested with versions 1.6 and 2.0	wsimport	Install the desired JWSDP version and specify the jaxws/bin directory in the soapUI Tool Integrations settings

Apache Axis 1.X

Tool	Version(s)	Documentation	Installation / Usage
Apache Axis 1.X	tested with versions 1.2.1, 1.3 and 1.4	wsdl2java	Install Axis 1.X and specify the installation directory in the soapUI Tool Integrations settings

Apache Axis 2

Tool	Version(s)	Documentation	Installation / Usage
Apache Axis 2	tested with 0.9X and 1.0	wsdl2java	Install Axis 2 and specify the installation directory in the soapUI Tool Integrations settings

XFire 1.X

Tool	Version(s)	Documentation	Installation / Usage
XFire 1.X WsGen	tested with 1.X	WsGen	Install XFire 1.X and specify the installation directory in the soapUI Tool Integrations settings

Oracle wsa

Tool	Version(s)	Documentation	Installation / Usage
Oracle wsa.jar	tested with latest jdeveloper release	Oracle WebServices (PDF)	Install JDeveloper and specify <jdeveloper-root>/webservices/lib in soapUI Tool Integrations settings

XMLBeans 2.x

Tool	Version(s)	Documentation	Installation / Usage
XMLBeans 2.X	tested with 2.X	scomp	Install XmlBeans 2.X and specify the installation directory in the soapUI Tool Integrations settings

JAXB 2.x

Tool	Version(s)	Documentation	Installation / Usage
JWSDP JAXB 2.0	tested with JWSDP 2.0	xjc	Install JWSDP as required and specify the jaxb\bin installation directory in the soapUI Tool Integrations settings

GSoap 2.X

Tool	Version(s)	Documentation	Installation / Usage
GSoap 2.X	tested with GSoap 2.7 on win32	wsdl2h , soapcpp2	Install GSoap as required and specify the installation directory in the soapUI Tool Integrations settings

.NET 2.0

Tool	Version(s)	Documentation	Installation / Usage
.NET 2.0	.NET 2.0	wsdl.exe	Install Visual Studio and .NET 2.0 and specify the directory containing wsdl.exe in the soapUI Tool Integrations settings

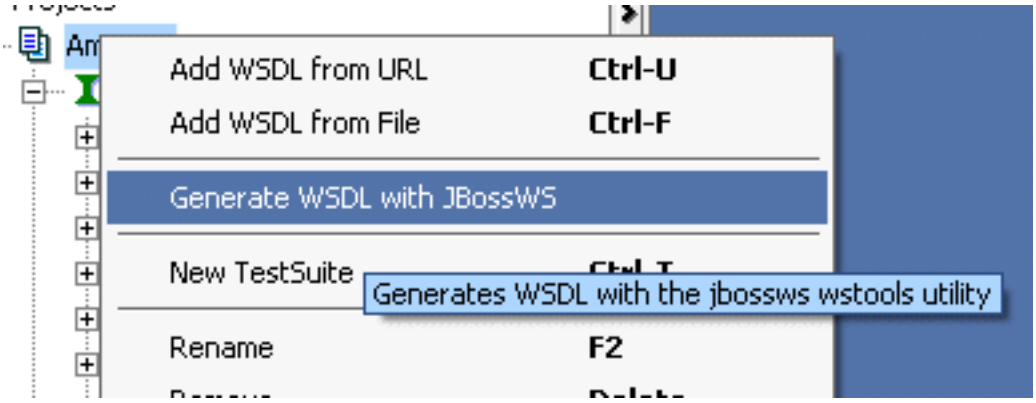
Next: WSDL Generation Tools

1.10.2 WSDL Generation

WSDL Generation Tools

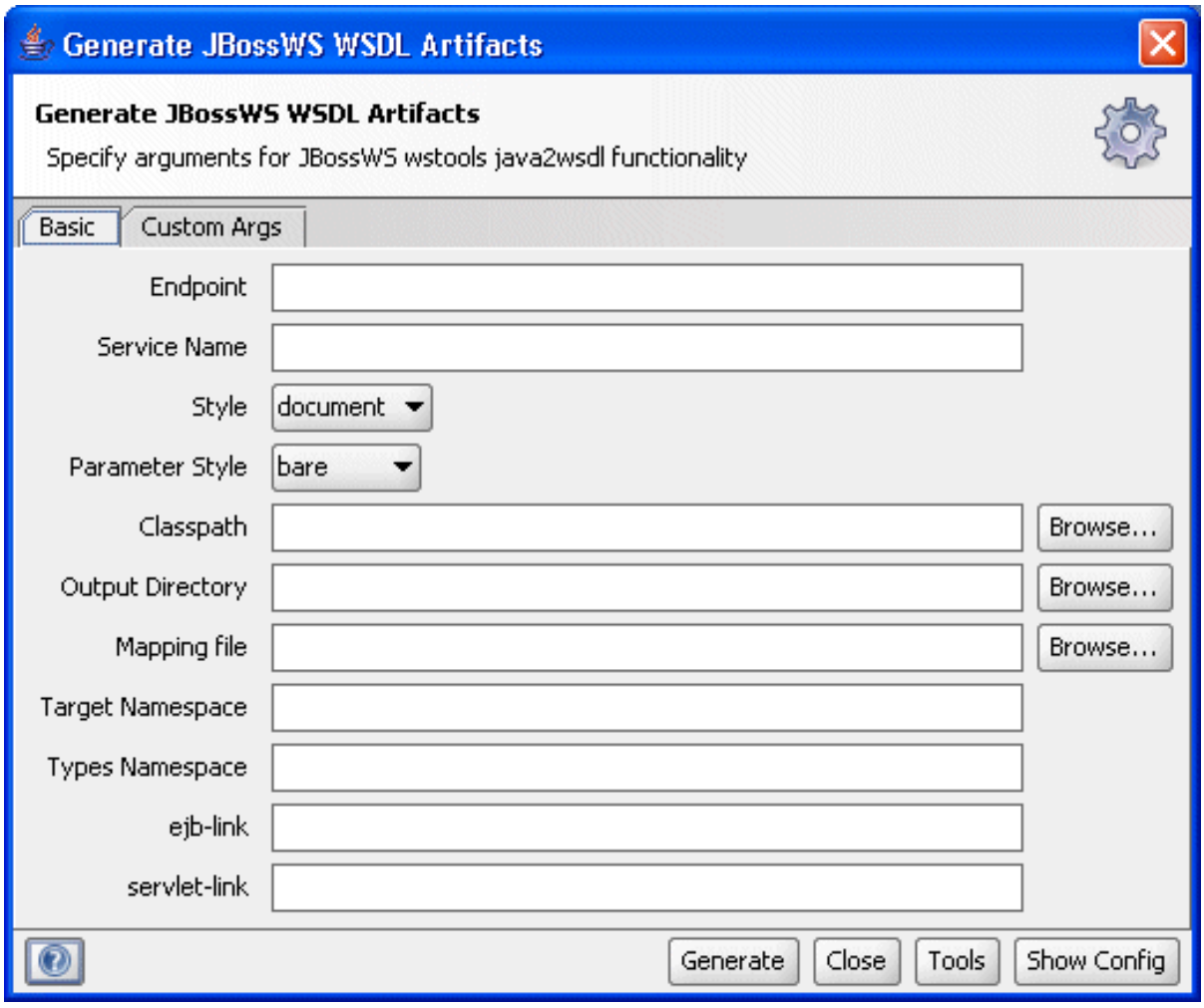
The following WSDL-Generation tools have currently been integrated and are available from the Project right-button menu:

Tool	URL	Documentation	Prerequisite
JBossWS wstools	JBossWS	JBossWS User Guide	Install the latest JBossWS stack release and configure as required. Extract the wstools script from the lib/jdk1.X/jbossws.jar file into some directory and specify that directory in the soapUI Tool Integrations settings



JBossWS wstools

The recently released JBossWS stack includes a utility for generating a WSDL from existing/annotated java-code; wstools. soapUI integrates with wstools allowing you to run the tool and automatically add the generated WSDL to the selected soapUI project. The added interface will have a "Regenerate with JBossWS" menu option allowing you to rerun the tool as needed if the underlying java-code changes or if you want to change some generation option.



Next: WS-I Integrations

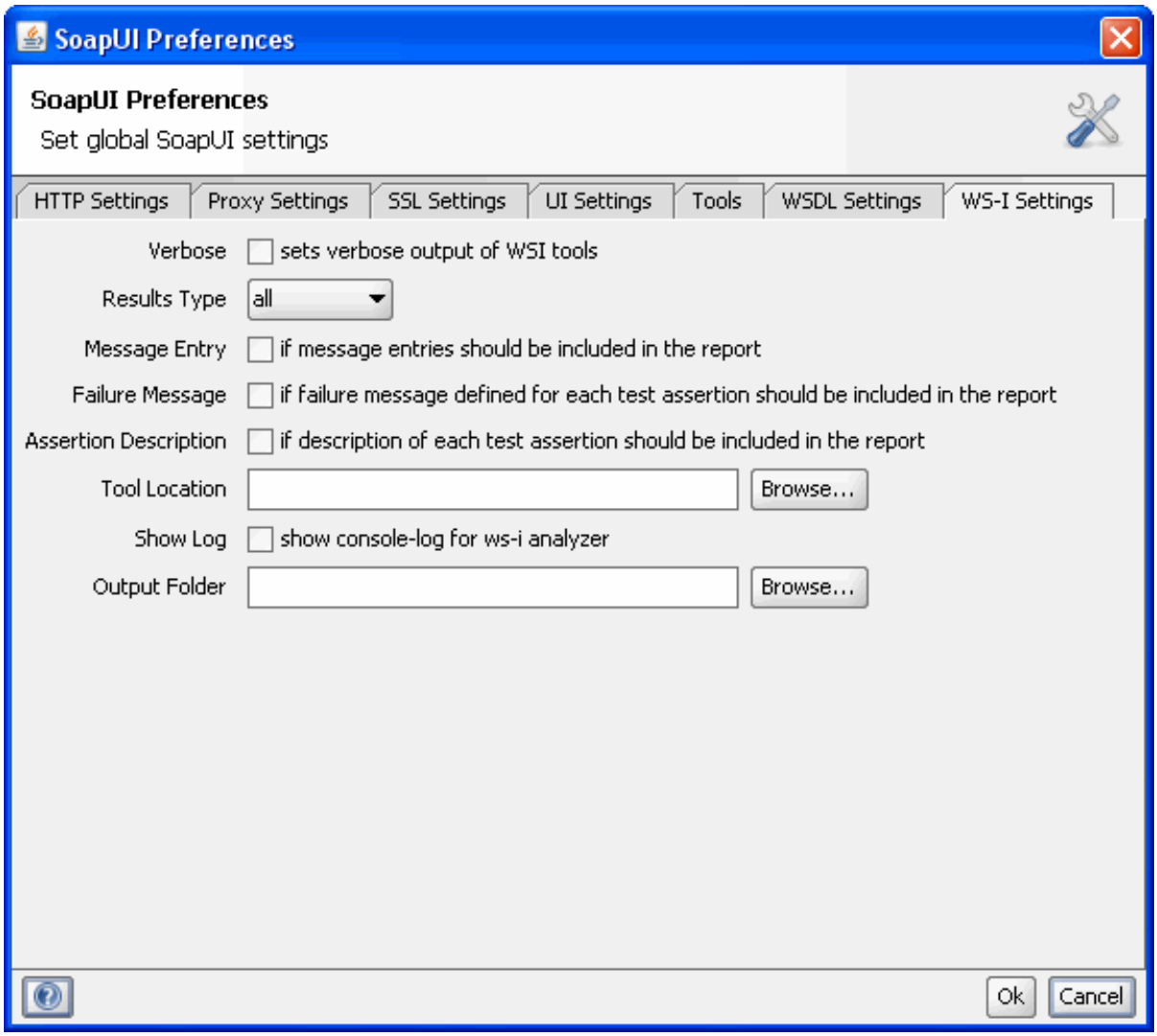
1.10.3 WS-I Tools

WS-I Tools Integration

soapUI includes integrated support for the WS-I organizations Basic Profile validation tools for 2 situations:

1. **Validating WSDL definitions** - from the Interface Menu with the "Check WSI Compliance" option. This will run the WS-I Test Tools and validate the WSDL definition accordingly.
2. **Validating SOAP request/response messages** - from within the Request Editors response popup with the "Check WSI Compliance" option (as described under Message Validation)

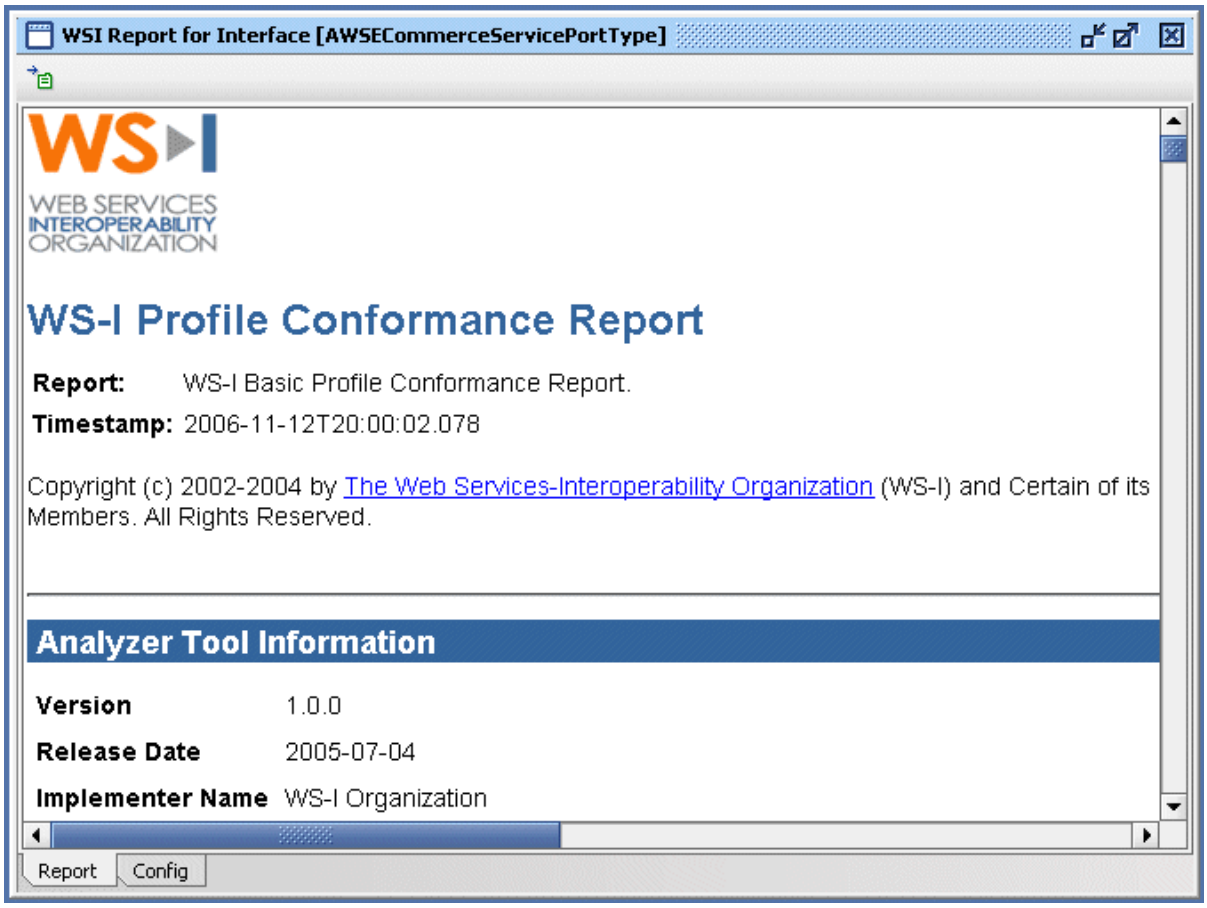
In either case, you first need to download either the java or C# version (soapUI will use whichever is available) of the WS-I Interoperability Testing Tools 1.1 from the WS-I deliverables page. Once downloaded, unzip the file into a local directory and specify the contained "wsi-test-tools" directory in the soapUI together with the desired settings;



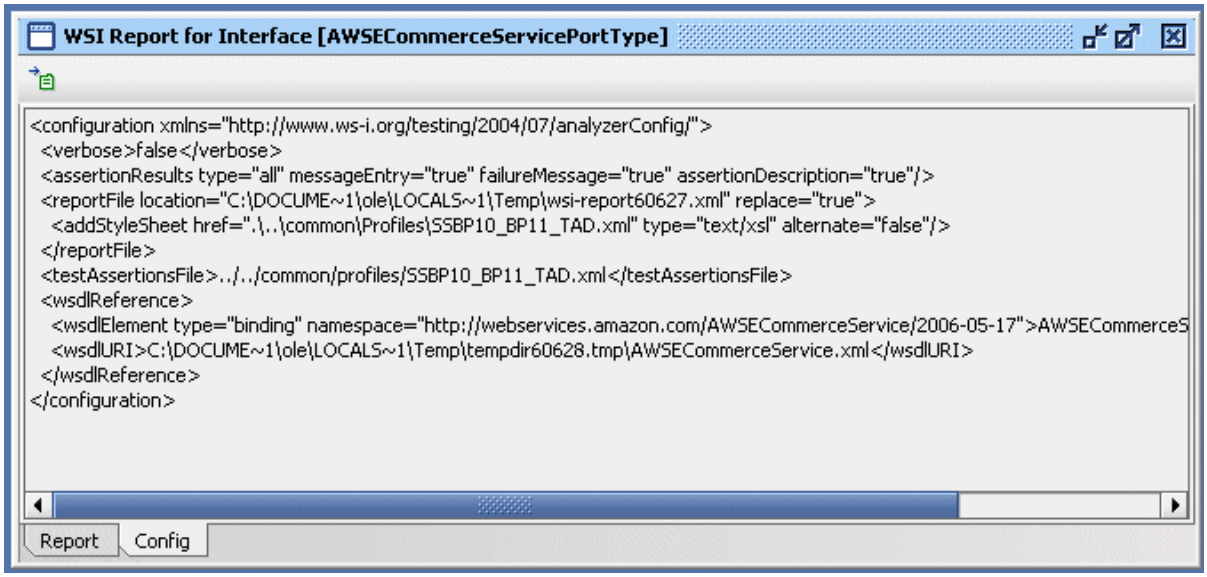
WS-I Reports

When running one of the integrated validations described above, the corresponding WS-I Analyzer configuration file will be generated and the WS-I Analyzer will be invoked in a seperate process; a progress dialog will be shown while the tool is running. If all goes well, you will be presented with a WS-I Compliance Report in a seperate desktop window, otherwise an error will be displayed.

A generated report looks as follows:

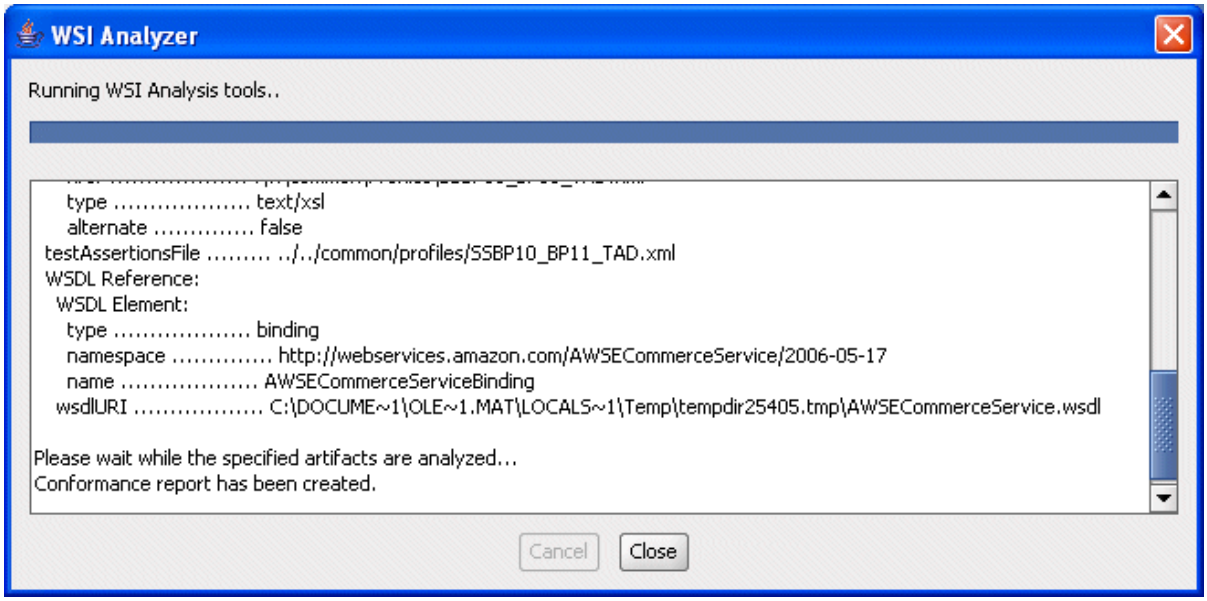


The top toolbar contains a button for exporting the report to an HTML file. Also, a "Config" tab is available for displaying the configuration file used to invoke the WS-I tools, for example:



Displaying Tool Output

Selecting the "Show Log" option in the WSI Settings Tab will display ongoing output from the command-line tools which can be usefull for debugging purposes;



Next: Apache TcpMon

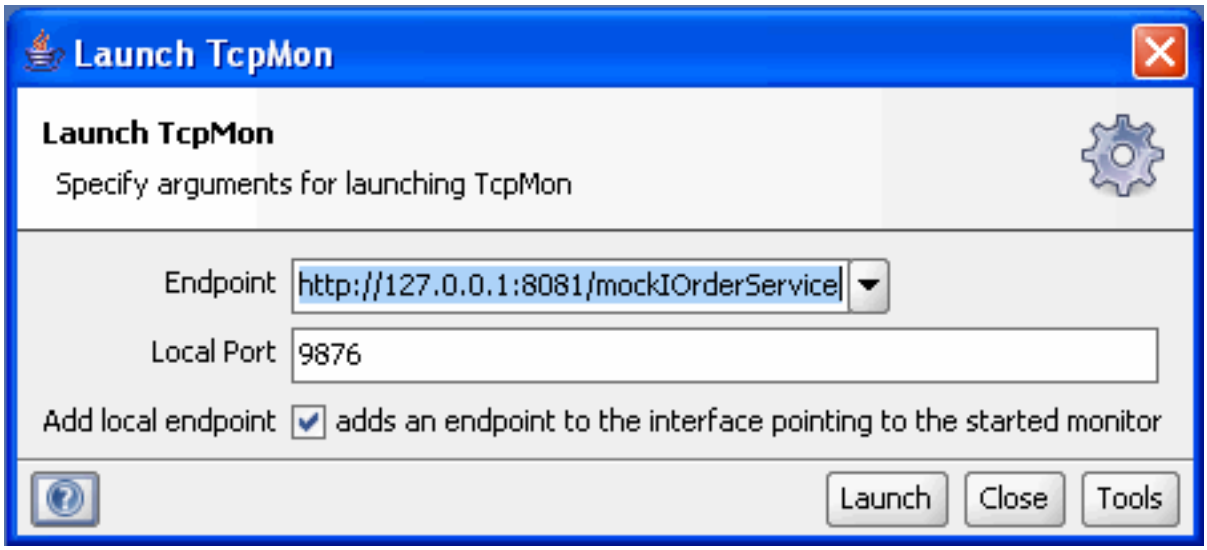
1.10.4 Apache Tcp-Mon

Apache Tcp-Mon Integration

The Apache Tcp-Mon utility can be used to "listen in" on any http-traffic, including of course SOAP request/response messages. Download and install TcpMon as described on their site and specify the installation directory in the soapUI Integrated Tools Settings.

soapUI provides 2 simple integrations with this usefull tool:

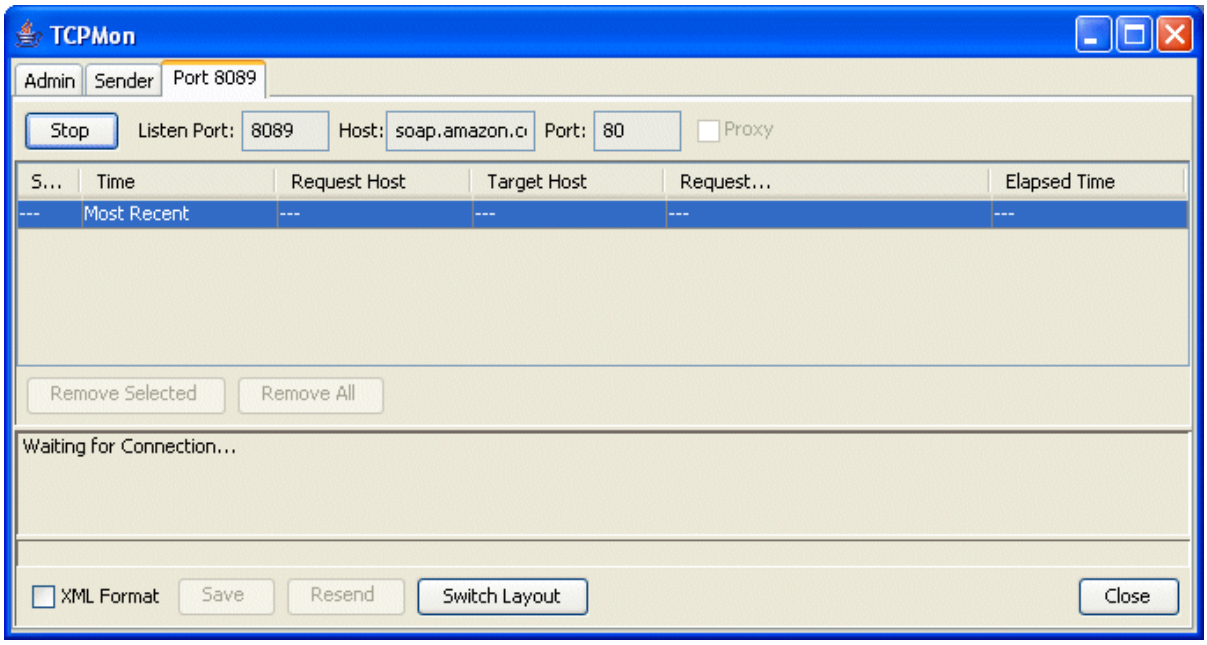
1. From the main Tools menu which will just start TcpMon in a seperate process
2. From the Interface Menu which can be used to create a local proxy for the designated service. This will bring up the following dialog:



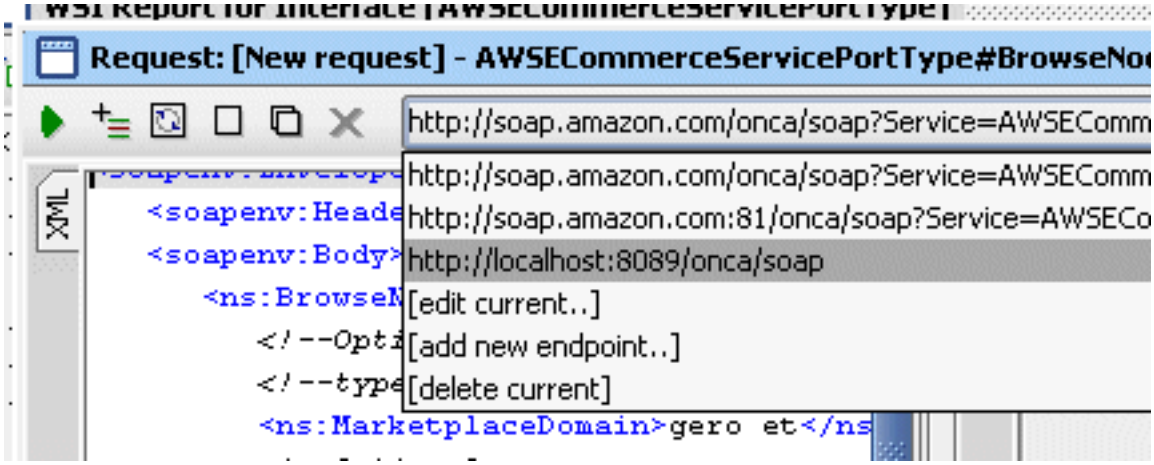
The dialog contains the following options:

- **Endpoint** - the target endpoint that TcpMon should invoke
- **Local Port** - the local port to that TcpMon should listen on
- **Add Local Endpoint** - adds an endpoint to the current service pointing at the created TcpMon proxy

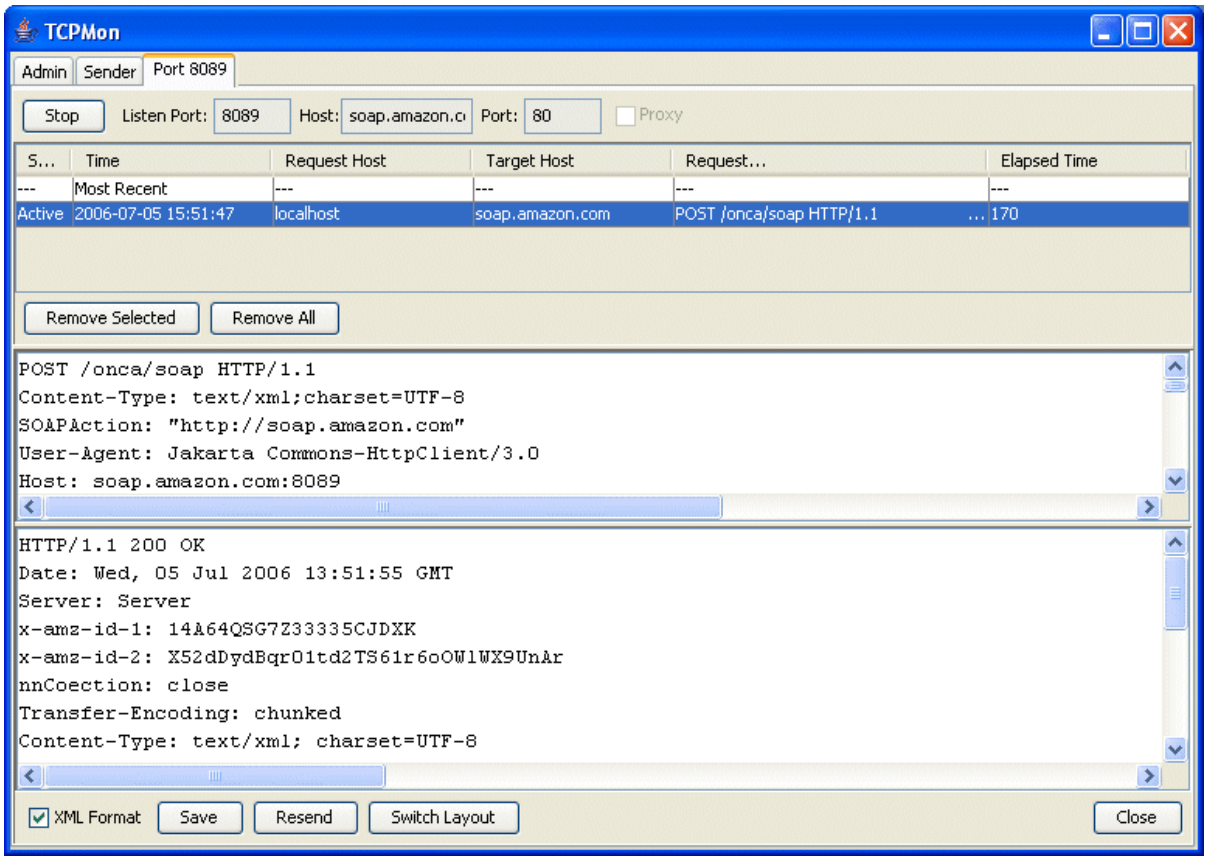
Launching with the above settings will start TcpMon with the following monitor:



In soapUI, an endpoint has been added to the current service that can be used to invoke TcpMon instead of the service directly (this endpoint must be removed manually):



Using this endpoint results in the SOAP request being "proxied" through TcpMon which will be shown as follows:



Next: Command-Line Tools

1.11 CommandLine Tools

soapUI CommandLine tools

soapUI currently comes with the following commandline tools for running TestCases and LoadTests

Tool	Description
SoapUITestCaseRunner	Runs specified TestCases and reports/exports results as configured
SoapUILoadTestRunner	Runs specified LoadTests and reports/exports results as configured
SoapUIMockServiceRunner	Runs specified MockServices
SoapUIToolRunner	Runs any of the configured Code Generation tools for specified project and interface
soapUI Maven 1.X/2.X plug-ins	Allows execution of the above two from within a Maven 1.X/2.X build environment

Next: SoapUITestCaseRunner

1.11.1 TestCaseRunner

SoapUITestCaseRunner

The runner is defined in the `com.eviware.soapui.tools.SoapUITestCaseRunner` class and takes the path to the soapUI project file containing the tests and a number of options:

switch	description
e	The endpoint to use when invoking test-requests, overrides the endpoint set in the project file
h	The host:port to use when invoking test-requests, overrides only the host part of the endpoint set in the project file
s	The TestSuite to run, used to narrow down the tests to run
c	The TestCase to run, used to narrow down the tests to run
u	The username to use in any authentications, overrides any username set for any testrequests
p	The password to use in any authentications, overrides any password set for any testrequests
w	Sets the WSS password type, either 'Text' or 'Digest'
d	The domain to use in any authentications, overrides any domain set for any testrequests
r	Turns on printing of a small summary report (see below)
f	Specifies the root folder to which test results should be exported (see below)
j	Turns on exporting of junit-compatible reports, see below
a	Turns on exporting of all test results, not only errors

The distribution contains a testrunner.bat script for running tests in the bin directory, for example;

```
testrunner.bat -hlocalhost:8080 -a -fresults c:\projects\my-soapui-project.xml
```

will run all the tests defined in the my-soapui-project.xml file against the specified host and export all test results to the "results" folder (will be created if not available).

JUnit Integration

It is fairly easy to invoke the testrunner from your own junit-tests;

```
public void testRunner() throws Exception
{
        SoapUITestCaseRunner runner = new SoapUITestCaseRunner();
        runner.setProjectFile( "src/dist/sample-soapui-project.xml" );
        runner.run();
}
```

The `runner.run()` call will throw an exception if an error occurs. If you want more control over your integration / error-reporting, a specific TestCase could be run as follows:

```
public void testTestCaseRunner() throws Exception
{
        WsdlProject project = new WsdlProject( "src/dist/sample-soapui-project.xml"
);
        TestSuite testSuite = project.getTestSuiteByName( "Test Suite" );
        TestCase testCase = testSuite.getTestCaseByName( "Test Conversions" );

        // create empty properties and run synchronously
        TestRunner runner = testCase.run( new PropertiesMap(), false );
        assertEquals( Status.FINISHED, runner.getStatus() );
}
```

Reporting and Exporting

The SoapUITestCaseRunner has basic reporting functionalities, including the possibility to create junit-compatible xml-reports using the -j switch. Also, it will during execution print diagnostic information and if the -r switch was specified print a small summary:

```
testrunner.bat jbossws-soapui-project.xml -stest -ctesting -r -a -fmyresults
```

produced the following output:

```
SoapUI 1.5beta2 TestCase Runner
12:33:10,042 INFO  [SoapUITestCaseRunner] setting projectFile to
[jbossws-soapui-project.xml]
12:33:10,042 INFO  [SoapUITestCaseRunner] setting testSuite to [test]
12:33:10,042 INFO  [SoapUITestCaseRunner] setting testCase to [testing]
12:33:10,583 INFO  [WsdlProject] Loaded project from [jbossws-soapui-project.xml]
12:33:11,915 INFO  [SoapUITestCaseRunner] Running soapui tests in project [jbossws]
12:33:11,915 INFO  [SoapUITestCaseRunner] Running soapui suite [test], runType =
SEQUENTIAL
12:33:11,925 INFO  [SoapUITestCaseRunner] Running soapui testcase [testing]
12:33:11,935 INFO  [SoapUITestCaseRunner] runing step [Groovy Script - init boss]
12:33:12,335 INFO  [SoapUITestCaseRunner] runing step [Properties]
12:33:12,335 INFO  [SoapUITestCaseRunner] runing step [Transfer Values - set boss]
12:33:12,716 ERROR [SoapUITestCaseRunner] Transfer Values - set boss failed,
exporting to
 [myresults\test\testing\Transfer-FAILED.txt]
12:33:12,716 INFO  [SoapUITestCaseRunner] runing step [request step]
Retrieving document at
'http://lpt-olma:8080/ws4ee-samples-server-ejb/Organization?wsdl'.
12:33:13,407 INFO  [SchemaUtils] Loading schema types from
```

```
[http://lpt-olma:8080/ws4ee-samples-server-ejb
 /Organization?wsdl]
12:33:13,407 INFO  [SchemaUtils] Getting schema
http://lpt-olma:8080/ws4ee-samples-server-ejb/Organization?wsdl
12:33:13,787 INFO  [SoapUITestCaseRunner] Assertion [Schema Compliance] has status
VALID
12:33:13,807 INFO  [SoapUITestCaseRunner] Finished running soapui testcase
[testing], time taken = 1882ms
12:33:13,807 INFO  [SoapUITestCaseRunner] Skipping testcase [testcase2], filter is
[testing]
12:33:13,807 INFO  [SoapUITestCaseRunner] Skipping testcase [Copy of testing],
filter is [testing]
12:33:13,807 INFO  [SoapUITestCaseRunner] Skipping testcase [Copy of Copy of
testing], filter is [testing]
12:33:13,807 INFO  [SoapUITestCaseRunner] soapui suite [test] finished in 1892ms

SoapUI 1.5beta2 TestCaseRunner Summary
----------------------------
Time Taken: 1895ms
Total TestSuites: 1
Total TestCases: 1
Total TestSteps: 4
Total Request Assertions: 1
Total Failed Assertions: 0
Total Exported Results: 4
```

By default, the testrunner exports only failed results to a text file, the `-a` option will export all results instead. For example the file for a Request TestStep will be as follows (slightly modified to fit):

```
Status: OK
Time Taken: 55
Size: 448
Timestamp: Sun Mar 12 12:45:57 CET 2006
TestStep: request step
--------------------------------------------------
Encoding: UTF-8
Endpoint: http://lpt-olma:8080/ws4ee-samples-server-ejb/Organization
Username: asd
Password: dfsdfdsf
Domain: asdasd
---------------- Request --------------------------
<soapenv:Envelope xmlns:sam="http://org.jboss.test.webservice/samples"
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <sam:getContactInfo>
         <String_1>testsd1141581163341</String_1>
      </sam:getContactInfo>
   </soapenv:Body>
</soapenv:Envelope>
---------------- Response -------------------------
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <ns1:getContactInfoResponse
xmlns:ns1="http://org.jboss.test.webservice/samples">
         <result>The 'testsd1141581163341' boss is currently out of office, please
call again.</result>
      </ns1:getContactInfoResponse>
```

```
   </soapenv:Body>
</soapenv:Envelope>
```

Exported files are written to a file named <TestSuite>-<TestCase>-<TestStep Name>-<Count>-<Status>.txt in the current or specified folder. An example filename is "TestSuite 1-TestCase 1-Request Step 1-0-OK.txt" (the count is added since a TestStep can be invoked several times within the run of a TestCase).

JUnit Reports

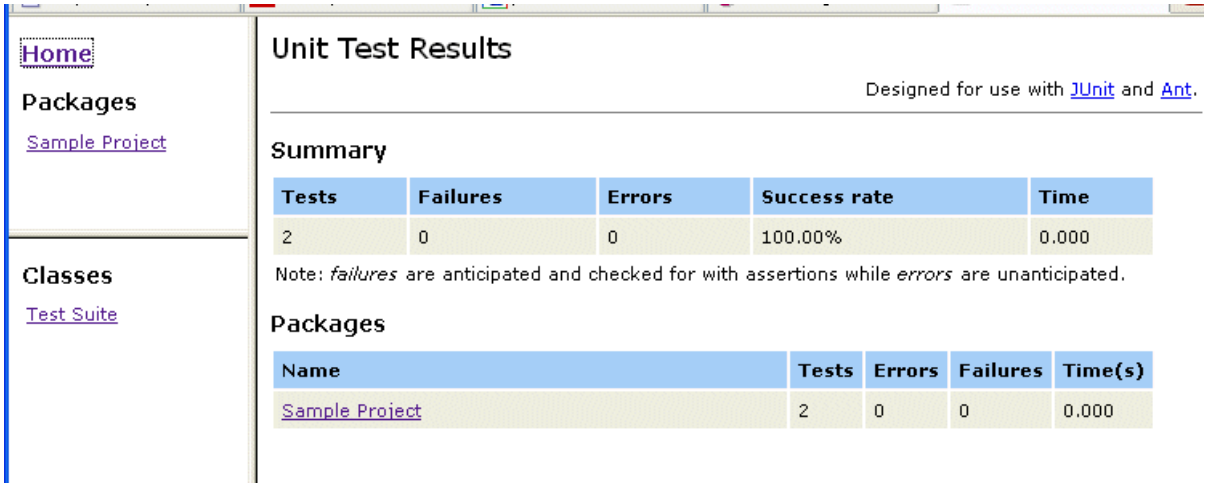
Adding the -j switch produces xml-reports in the same format that are exported by the junit ant task with the xml-report format. Currently, soapUI maps TestSuites to report-packages and TestCases to report TestCases. For example running the following:

```
testrunner.bat -j -ftestresults sample-soapui-project.xml
```

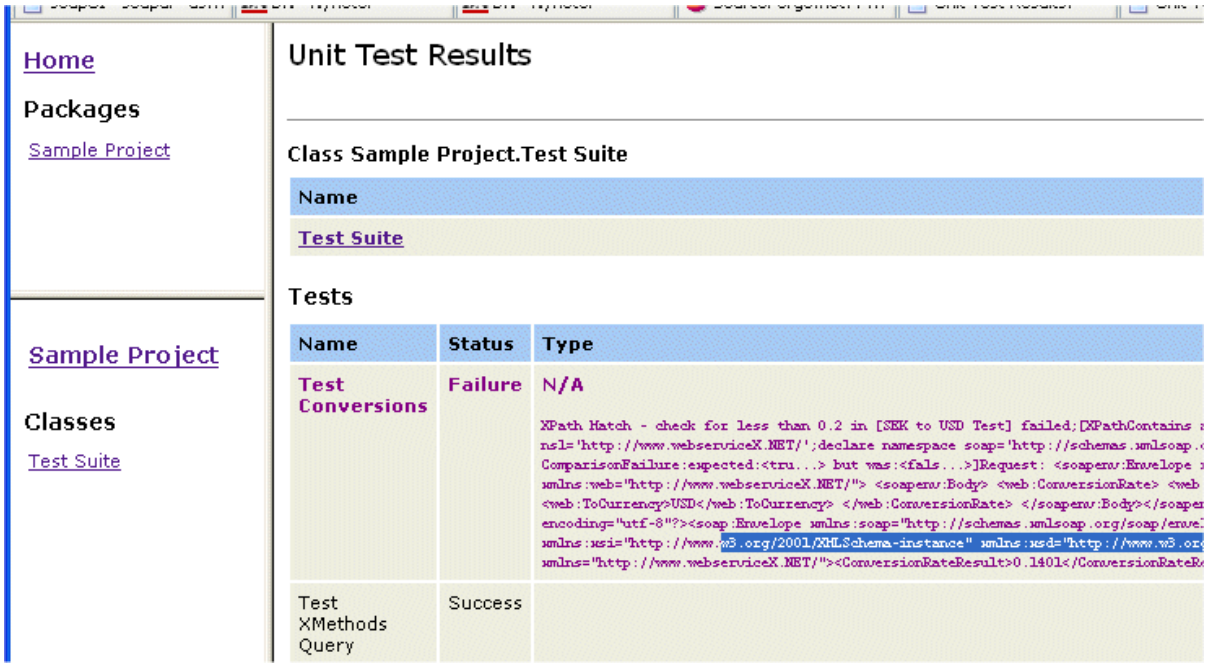
produces a "TEST-Test Suite.xml" file in the specified testresults folder. This result can be further transformed using the ant junitreport task:

```
<junitreport todir="./testresults">
   <fileset dir="./testresults">
      <include name="TEST-*.xml"/>
   </fileset>
   <report format="frames" todir="./testresults/html"/>
 </junitreport>
```

Which produces an output as follows:



Errors are shown as follows:



Wrapping everything into one ant-target for win32 would be as follows:

```
<target name="soapui-report">

        <exec dir="C:\\Program Files\\eviware\\soapUI-1.6-beta1\\bin"
executable="cmd.exe">
                <arg line="/c testrunner.bat -j -fg:\\reports
g:\\projects\\sample-soapui-project.xml"/>
        </exec>

        <junitreport todir="g:\\reports">
          <fileset dir="g:\\reports">
            <include name="TEST-*.xml"/>
          </fileset>
          <report format="frames" todir="g:\\reports\\html"/>
        </junitreport>

</target>
```

Next: SoapUILoadTestRunner

1.11.2 LoadTestRunner

SoapUILoadTestRunner

The SoapUILoadTestRunner can be used to run LoadTests from the command line, which can be used for surveillance testing and maximizing local testing. The runner will run each specified LoadTest in turn, be carefull not to run LoadTests with an indefinite test limit.

The runner is defined in the `com.eviware.soapui.tools.SoapUILoadTestRunner` class and takes the path to the soapUI project file containing the loadtest(s) to run and a number of options:

switch	description
e	The endpoint to use when invoking test-requests, overrides the endpoint set in the project file
h	The host:port to use when invoking test-requests, overrides only the host part of the endpoint set in the project file
s	The TestSuite to run, used to narrow down the tests to run
c	The TestCase to run, used to narrow down the tests to run
l	The LoadTest to run, used to narrow down which loadtests to run
u	The username to use in any authentications, overrides any username set for any testrequests
p	The password to use in any authentications, overrides any password set for any testrequests
d	The domain to use in any authentications, overrides any domain set for any testrequests
r	Turns on exporting of a loadtest statistics summary report (see below)
f	Specifies the root folder to which test results should be exported (see below)

The distribution contains a loadtestrunner.bat script for running tests from within the bin directory, for example;

```
loadtestrunner.bat -ehttp://localhost:8080/services/MyService
c:\projects\my-soapui-project.xml
```

will run all the loadtests defined in the my-soapui-project.xml file against the specified service-endpoint.

Reporting and Exporting

The SoapUILoadTestRunner has limited reporting functionalities, it will during execution print diagnostic and progress information and if the -r switch was specified write the LoadTest statistics and LoadTest Log to a file after execution. Error Result in the LoadTest Log will be exported likewise, in the same manner as for the TestCase runner.

```
loadtestrunner -stest -ctestcase4 -l"LoadTest 1" -r "C:\Documents and
Settings\ole.matzura\My Documents\jbossws-soapui-project.xml"
```

produced the following output:

```
SoapUI 1.5beta2 LoadTestRunner
15:31:50,397 INFO  [SoapUILoadTestRunner] setting projectFile to [C:\Documents and
Settings\ole.matzura\
 My Documents\jbossws-soapui-project.xml]
15:31:50,517 INFO  [SoapUILoadTestRunner] setting testSuite to [test]
15:31:50,517 INFO  [SoapUILoadTestRunner] setting testCase to [testcase3]
15:31:52,901 INFO  [WsdlProject] Loaded project from [C:\Documents and
Settings\ole.matzura\My Documents\
 jbossws-soapui-project.xml]
15:31:59,941 INFO  [SoapUILoadTestRunner] Skipping testcase [testing], filter is
[testcase3]
15:31:59,941 INFO  [SoapUILoadTestRunner] Skipping testcase [testcase2], filter is
[testcase3]
15:31:59,941 INFO  [SoapUILoadTestRunner] Running LoadTest [LoadTest1]
15:32:00,141 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress: 0.0025
15:32:01,153 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress: 0.019533332
15:32:02,154 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress: 0.036383335
15:32:03,155 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress: 0.053066667
..etc..
15:32:57,944 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress: 0.9662167
15:32:58,956 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress: 0.9830833
15:32:59,977 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress: 1.0001
15:33:00,979 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] finished with status
FINISHED
15:33:00,979 INFO  [SoapUILoadTestRunner] Exporting log and statistics for LoadTest
[LoadTest1]
15:33:01,039 INFO  [SoapUILoadTestRunner] Exported 2 log items to [LoadTest
1-log.txt]
15:33:01,039 INFO  [SoapUILoadTestRunner] Exported 0 error results
15:33:01,119 INFO  [SoapUILoadTestRunner] Exported 2 statistics to [LoadTest
1-statistics.txt]
15:33:01,119 INFO  [SoapUILoadTestRunner] Skipping testcase [Copy of Copy of
testing], filter is [testcase3]
15:33:01,119 INFO  [SoapUILoadTestRunner] soapui suite [test] finished in 61178ms
```

The exported statistics file is the same as exported from within the LoadTest Editor and looks as follows:

```
Request Step,min,max,avg,last,cnt,tps,bytes,bps,err
request step,1,407,17.63,26,11591,283.49,5192768,86696,0
Total:,1,407,17.63,26,11591,283.49,5192768,86696,0
```

Next: SoapUIMockRunner

1.11.3 MockServiceRunner

SoapUIMockServiceRunner

The runner is defined in the `com.eviware.soapui.tools.SoapUIMockServiceRunner` class and takes the path to the soapUI project file containing the MockService and a number of options:

switch	description
m	The name of the MockService to run
p	The local port to listen on, overrides the port configured for the MockService
a	The local path to listen on, overrides the path configured for the MockService
s	The soapui-settings.xml file to use

The distribution contains a mockservicerunner.bat script for running mockservices in the bin directory, for example;

```
mockservicerunner.bat -m"IOrderService MockService" "C:\Documents and Settings\Ole
Matzura\My Documents\demo2-soapui-project.xml"
```

will start the specified MockService as follows:

```
SoapUI SNAPSHOT MockService Runner
15:08:15,515 INFO  [SoapUI] Added
[file:/C:/workspace/core/ext/mysql-connector-java-5.0.4-bin.jar] to classpath
15:08:16,375 INFO  [SoapUI] initialized soapui-settings from [soapui-settings.xml]
15:08:16,406 INFO  [WsdlProject] Loaded project from [C:\Documents and Settings\Ole
Matzura\My Documents\demo2-soapui-project.xml]
15:08:17,609 INFO  [SoapUIMockServiceRunner] Running MockService [IOrderService
MockService] in project [demo]
15:08:17,609 INFO  [SoapUIMockServiceRunner] Press any key to terminate
15:08:17,953 INFO  [MockEngine] Started mockService [IOrderService MockService] on
port [8081] at path [/mockIOrderService]
15:08:17,953 INFO  [SoapUIMockServiceRunner] MockService started on port 8081 at
path [/mockIOrderService]
Progress: 1 - Loading definition from cache
15:09:18,625 DEBUG [WsdlContext] Loading definition from cache
15:09:18,640 DEBUG [WsdlLoader] Returning baseInputSource
[http://evitop:8080/OrderServiceImpl?wsdl]
15:09:18,687 DEBUG [WsdlLoader] Returning baseURI
[http://evitop:8080/OrderServiceImpl?wsdl]
Retrieving document at 'http://evitop:8080/OrderServiceImpl?wsdl'.
15:09:18,796 DEBUG [WsdlContext] Loaded definition: ok
15:09:18,812 INFO  [SchemaUtils] Added default schema from
```

```
/C:/workspace/core/target/classes/xop.xsd with targetNamespace
http://www.w3.org/2004/08/xop/include
15:09:18,890 INFO  [SchemaUtils] Added default schema from
/C:/workspace/core/target/classes/XMLSchema.xsd with targetNamespace
http://www.w3.org/2001/XMLSchema
15:09:18,890 INFO  [SchemaUtils] Added default schema from
/C:/workspace/core/target/classes/xml.xsd with targetNamespace
http://www.w3.org/XML/1998/namespace
15:09:18,890 INFO  [SchemaUtils] Added default schema from
/C:/workspace/core/target/classes/swaref.xsd with targetNamespace
http://ws-i.org/profiles/basic/1.1/xsd
15:09:18,890 INFO  [SchemaUtils] Added default schema from
/C:/workspace/core/target/classes/xmime200505.xsd with targetNamespace
http://www.w3.org/2005/05/xmlmime
15:09:18,890 INFO  [SchemaUtils] Added default schema from
/C:/workspace/core/target/classes/xmime200411.xsd with targetNamespace
http://www.w3.org/2004/11/xmlmime
15:09:18,890 WARN  [SchemaUtils] Failed to open schemaDirectory
[C:\workspace\soapui-pro\schemas]
15:09:18,890 INFO  [SchemaUtils] Loading schema types from
[http://evitop:8080/OrderServiceImpl?wsdl]
15:09:18,890 INFO  [SchemaUtils] Getting schema
http://evitop:8080/OrderServiceImpl?wsdl
15:09:18,906 INFO  [SchemaUtils] schema for [http://www.example.org/OrderService/]
contained [{}] namespaces
15:09:19,359 INFO  [SoapUIMockServiceRunner] Handled request 1; [purchase] with
[MockResponse 1] in [9ms] at [2007-04-03 15:09:17.968]
15:09:20,734 INFO  [SoapUIMockServiceRunner] Handled request 2; [purchase] with
[MockResponse 2] in [0ms] at [2007-04-03 15:09:20.734]
15:09:21,296 INFO  [SoapUIMockServiceRunner] Handled request 3; [purchase] with
[MockResponse 1] in [0ms] at [2007-04-03 15:09:21.296]
15:09:21,937 INFO  [SoapUIMockServiceRunner] Handled request 4; [purchase] with
[MockResponse 2] in [0ms] at [2007-04-03 15:09:21.937]
15:09:22,343 INFO  [SoapUIMockServiceRunner] Handled request 5; [purchase] with
[MockResponse 1] in [0ms] at [2007-04-03 15:09:22.343]
```

Which can now be invoked from soapUI or any other client. Terminate the runner by pressing the return key in the console, which will shutdown as follows:

```
15:20:30,703 INFO  [MockEngine] Stopping connector on port 8081
15:20:30,703 INFO  [MockEngine] No more connectors.. stopping server
15:20:30,703 INFO  [SoapUIMockServiceRunner] MockService stopped, handled 7 requests
15:20:30,703 INFO  [SoapUIMockServiceRunner] time taken: 733117ms
```

Next: SoapUIToolRunner

1.11.4 ToolRunner

SoapUIToolRunner

The SoapUIToolRunner can be used to any of the integrated Code Generation tools from the command-line as they have been configured in soapUI.

The runner is defined in the `com.eviware.soapui.tools.SoapUIToolRunner` class and takes the path to the soapUI project file containing the interface and tool-configuration to run. It has the following options:

switch	description
i	the interface for which the tool should be run
t	the tool(s) to run, a comma-seperated list with the following tokens; axis1, axis2, dotnet, gsoap, jaxb, wstools, wsconsume, ora, wsi, wscompile, wsimport, xfire or xmlbeans
s	the soapui-settings.xml file to use (usually the one in the soapUI bin directory)

The distribution contains a toolrunner.bat script for running tools from within the bin directory, for example;

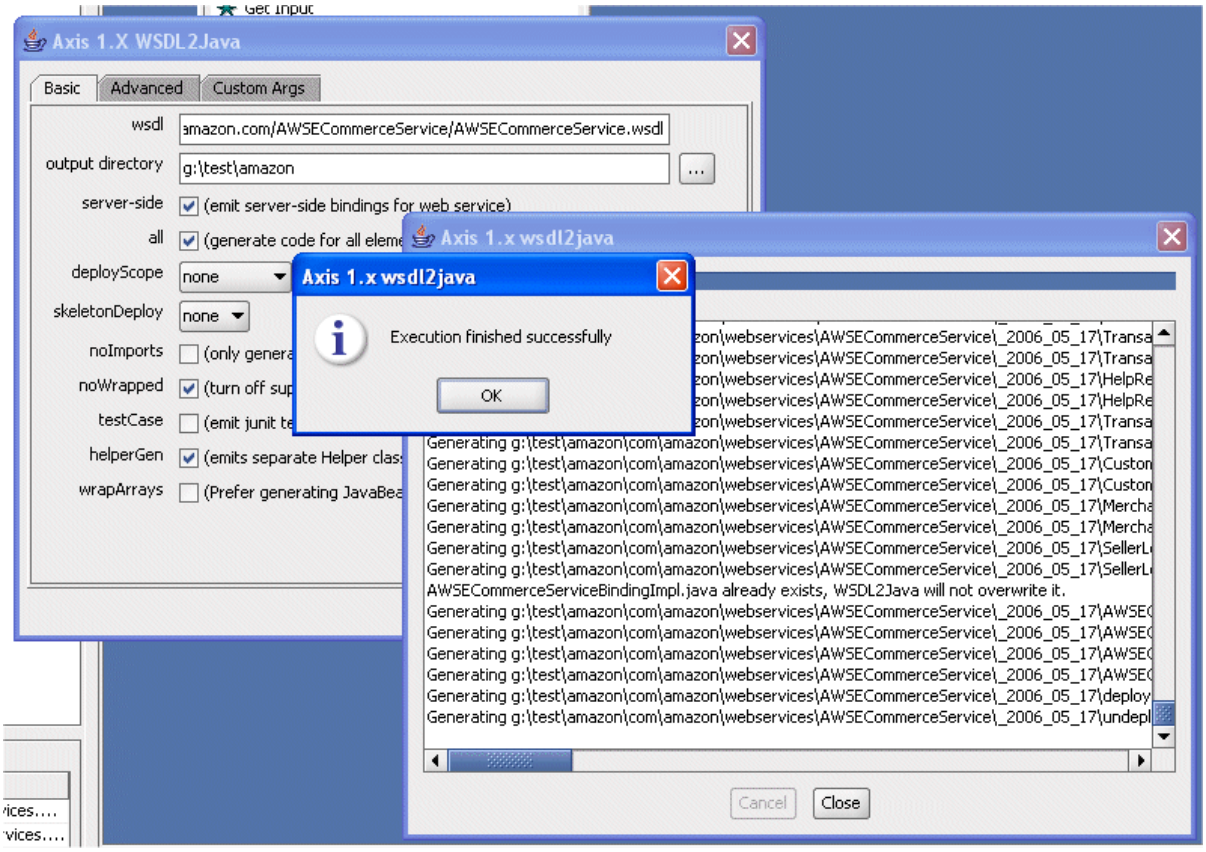
```
toolrunner.bat -iAWSECommerceServicePortType -taxis1
G:\test\Amazon-soapui-project.xml
```

will run all the axis 1.X code-generation as previously defined within soapUI and produce the following output:

```
SoapUI 1.6 beta1 Tool Runner
02:27:02,386 DEBUG [SoapUI] initialized soapui-settings from [soapui-settings.xml]
02:27:02,506 INFO  [WsdlProject] Loaded project from
[G:\test\Amazon-soapui-project.xml]
02:27:08,765 INFO  [SoapUIToolRunner] Running tools for project [Amazon]
directory: C:\java\axis-1_4\lib
command: java -cp
activation.jar;axis-ant.jar;axis.jar;bsf.jar;castor-0.9.5.2.jar;commons-codec-1.2.jar;
commons-discovery-0.2.jar;commons-httpclient-3.0-rc2.jar;commons-logging-1.0.4.jar;commons-net-1.0.0-dev.
httpunit.jar;ibmjsse.jar;javax.jms.jar;jaxrpc.jar;JimiProClasses.jar;junit-3.8.1.jar;log4j-1.2.8.jar;
 mailapi_1_3_1.jar;saaj.jar;servlet.jar;wsdl4j-1.5.1.jar
org.apache.axis.wsdl.WSDL2Java -v -W -s -a
 -H -T 1.2 -o g:\test\amazon
C:\DOCUME~1\OLE~1.MAT\LOCALS~1\Temp\tempdir8304.tmp\AWSECommerceService.wsdl
Parsing XML file:
C:\DOCUME~1\OLE~1.MAT\LOCALS~1\Temp\tempdir8304.tmp\AWSECommerceService.wsdl
Generating
```

```
g:\test\amazon\com\amazon\webservices\AWSECommerceService\_2006_05_17\CartGetRequest.java
...
Generating
g:\test\amazon\com\amazon\webservices\AWSECommerceService\_2006_05_17\deploy.wsdd
Generating
g:\test\amazon\com\amazon\webservices\AWSECommerceService\_2006_05_17\undeploy.wsdd
SoapUIToolRunner: Execution finished successfully
02:27:14,593 INFO  [SoapUIToolRunner] time taken: 5814ms
```

The above axis 1.X generation had been previously configured in soapUI as shown below:



Next: Keyboard shortcuts

1.12 IDE/Tool Plugins

IDE / Tool Plugins

soapUI comes with plug-ins for the following tools / IDE's

Tool/IDE	Functionality
Maven 1.X/2.X	Provides goals/properties corresponding to each of the soapUI command-line tools and its arguments
NetBeans 5.5/6.0	Provides full soapUI functionality from within the NetBeans IDE
IntelliJ IDEA 6+	Provides full soapUI functionality from within IntelliJ IDEA
Eclipse 3.2+	Provides full soapUI functionality from within Eclipse plus a soapUI Project Nature for integrated browsing/viewing of eclipse/soapUI projects in the Project Explorer.
JBossWS/JBossIDE 2.0.0+	Provides the same functionality as the standard Eclipse plug-in plus a number of JBossWS-related wizards/dialogs/actions.

Since the IDE's all have their own window/desktop management, some options in the soapUI Preferences / UI Settings will not be available.

1.12.1 Maven Plugins

soapUI maven plugins

soapUI includes plugins for both maven 1.x and maven 2.x allowing one to automate TestSuites / TestCases , LoadTests and code generation previously created/configured in soapUI. If any of the plugin tasks fail, an error-message will be printed to the console and the execution will exit with an error.

Read more:

- maven 1.X plugin
- maven 2.X plugin

Next: maven 1.X plugin

1.12.1.1 Maven 1.X Plugin

soapUI maven 1.X plugin

Download

Prior to installing a new version of the plugin, you should delete any old versions from you local cache/repository/maven installation.

Download the latest version of the plugin from sourceforge or install it directly from the command-line:

```
maven plugin:download -DgroupId=eviware -DartifactId=maven-soapui-plugin
-Dversion=1.6 -Dmaven.repo.remote=http://www.soapui.org/repository
```

Alternatively you can specify a dependency directly in your project.xml;

```
<dependency>
    <groupId>eviware</groupId>
    <artifactId>maven-soapui-plugin</artifactId>
    <version>1.6</version>
    <type>plugin</type>
</dependency>
```

In this case, don't forget to add the soapUI repository to your repository list in project.properties;

```
maven.repo.remote=http://www.ibiblio.org/maven,http://www.soapui.org/repository
```

Usage

Download/install the plugin as described above. By default, the plugin will look for a soapUI project file named `${pom.artifactId}-soapui-project.xml`, you can override this by setting with the `maven.soapui.project` property.

Run functional tests with

```
maven soapui:test
```

and loadtests with

```
maven soapui:loadtest
```

The plugin will load the specified project file and run all TestCases available in all TestSuites. If you want to narrow down which TestSuites/TestCases/LoadTest to run, use the `maven.soapui.test.testsuite`, `maven.soapui.test.testcase` and `maven.soapui.test.loadtest` properties for that purpose.

If you want to change the service endpoint, host, username, password or domain used by the TestRequests in the executed TestCases, set this with one of the `maven.soapui.test.XX` properties , for example

```
maven soapui:test
-Dmaven.soapui.test.endpoint=http://somehost.com:8080/services/MyService
```

Be aware that this will set the endpoint for *all* TestRequests executed, you might need to narrow down which TestCase to run if you have multiple TestCases testing multiple services.

For example if you have 2 TestSuites, each testing its own service in your project, you could run these as follows;

```
maven soapui:test -Dmaven.soapui.test.testsuite=TestSuite1
       -Dmaven.soapui.test.endpoint=http://somehost.com:8080/services/MyService1
```

followed by

```
maven soapui:test -Dmaven.soapui.test.testsuite=TestSuite2
-Dmaven.soapui.test.endpoint=http://someotherhost.com:8080/services/MyService2
```

Turn on creation of simple reports as described for the commandline SoapUITestCaseRunner and SoapUITestCaseRunner runners with the `maven.soapui.report.XX` properties.

Sample Outputs

Here comes an example output when running soapUI functional tests through maven as described above using the sample project included in the offline distribution:

```
maven soapui:test -Dmaven.soapui.project=sample-soapui-project.xml
 __  __
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / -_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0.2

build:start:

soapui:test:
    [echo] Running soapUI project sample-soapui-project.xml with endpoint []
    [java] SoapUI 1.6 TestRunner
    [java] 23:57:56,258 INFO  [SoapUITestCaseRunner] setting projectFile to
[sample-soapui-project.xml]
    [java] 23:57:56,838 INFO  [WsdlProject] Loaded project from
[sample-soapui-project.xml]
    [java] 23:57:57,219 INFO  [SoapUITestCaseRunner] Running soapui tests in project
```

```
[Sample Project]
    [java] 23:57:57,219 INFO  [SoapUITestCaseRunner] Running soapui suite [Test
Suite]
    [java] 23:57:57,219 INFO  [SoapUITestCaseRunner] Running soapui testcase [Test
Conversions]
    [java] 23:57:57,219 INFO  [SoapUITestCaseRunner] runing step [SEK to USD Test]
    [java] Retrieving document at
'http://www.webservicex.net/CurrencyConvertor.asmx?WSDL'.
    [java] 23:57:59,893 INFO  [SchemaUtils] Loading schema types from
[http://www.webservicex.net/CurrencyConvertor.asmx?WSDL]
    [java] 23:57:59,893 INFO  [SchemaUtils] loading schema types from
http://www.webservicex.net/CurrencyConvertor.asmx?WSDL
    [java] Assertion [Schema Compliance] has status VALID
    [java] Assertion [XPath Match - check for less than 0.2] has status VALID
    [java] Assertion [XPath Match - check for more than 0.1] has status VALID
    [java] Assertion [SOAP Fault Assertion] has status VALID
    [java] 23:58:01,025 INFO  [SoapUITestCaseRunner] runing step [USD to SEK Test]
    [java] Assertion [Schema Compliance] has status VALID
    [java] Assertion [XPath Match - check for less than 8] has status VALID
    [java] Assertion [XPath Match - check for more than 7] has status VALID
    [java] Assertion [SOAP Fault Assertion] has status VALID
    [java] 23:58:01,495 INFO  [SoapUITestCaseRunner] Finished running soapui
testcase [Test Conversions], time taken = 4276ms
    [java] 23:58:01,505 INFO  [SoapUITestCaseRunner] Running soapui testcase [Test
XMethods Query]
    [java] 23:58:01,505 INFO  [SoapUITestCaseRunner] runing step [Test
getAllServiceNames]
    [java] Assertion [XPath Match - check for Anagram Service] has status VALID
    [java] 23:58:03,488 INFO  [SoapUITestCaseRunner] runing step [Transfer values]
    [java] 23:58:03,688 INFO  [SoapUITestCaseRunner] runing step [Test
getServiceDetail]
    [java] Assertion [XPath Match] has status VALID
    [java] 23:58:03,949 INFO  [SoapUITestCaseRunner] Finished running soapui
testcase [Test XMethods Query], time taken = 2444ms
    [java] 23:58:03,959 INFO  [SoapUITestCaseRunner] soapui suite [Test Suite]
finished in 6740ms
BUILD SUCCESSFUL
Total time: 10 seconds
Finished at: Sun Oct 02 23:58:04 CEST 2005
```

Another sample with a failed loadtest:

```
maven soapui:loadtest -Dmaven.soapui.test.loadtest=LoadTest1
-Dmaven.soapui.report.folder=results
 __  __
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / -_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0.2

Attempting to download swingx-SNAPSHOT.jar.
build:start:

soapui:loadtest:
    [echo] Running soapui project jbossws-soapui-project.xml with endpoint []
    [java] SoapUI 1.6 LoadTestRunner
    [java] 22:48:13,116 INFO  [SoapUILoadTestRunner] setting projectFile to
[jbossws-soapui-project.xml]
    [java] 22:48:13,116 INFO  [SoapUILoadTestRunner] setting testSuite to [test]
    [java] 22:48:13,116 INFO  [SoapUILoadTestRunner] setting testCase to [testcase3]
```

```
    [java] 22:48:13,647 INFO  [WsdlProject] Loaded project from
[jbossws-soapui-project.xml]
    [java] 22:48:14,818 INFO  [SoapUILoadTestRunner] Skipping testcase [testing],
filter is [testcase3]
    [java] 22:48:14,818 INFO  [SoapUILoadTestRunner] Skipping testcase [testcase2],
filter is [testcase3]
    [java] 22:48:14,818 INFO  [SoapUILoadTestRunner] Running LoadTest [LoadTest1]
    [java] 22:48:14,878 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.0
    [java] 22:48:15,870 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.038
    [java] 22:48:16,871 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.102
    [java] 22:48:17,873 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.14
    [java] 22:48:18,884 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.204
    [java] 22:48:19,876 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.274
    [java] 22:48:20,887 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.35
    [java] 22:48:21,889 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.424
    [java] 22:48:22,890 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.488
    [java] 22:48:23,891 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.564
    [java] 22:48:24,893 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] progress:
0.642
    [java] 22:48:25,894 INFO  [SoapUILoadTestRunner] LoadTest [LoadTest1] finished
with status FAILED
    [java] 22:48:25,894 INFO  [SoapUILoadTestRunner] Exporting log and statistics
for LoadTest [LoadTest1]
    [java] 22:48:25,944 INFO  [SoapUILoadTestRunner] Exported 36 log items to
[results\LoadTest1-log.txt]
    [java] 22:48:26,265 INFO  [SoapUILoadTestRunner] Exported 33 error results
    [java] 22:48:26,265 INFO  [SoapUILoadTestRunner] Exported 3 statistics to
[results\LoadTest1-statistics.txt]
    [java] 22:48:26,265 INFO  [SoapUILoadTestRunner] Skipping testcase [testcase4],
filter is [testcase3]
    [java] 22:48:26,265 INFO  [SoapUILoadTestRunner] soapui suite [test] finished in
11447ms
    [java] 22:48:26,265 INFO  [SoapUILoadTestRunner] 1 load tests failed:
    [java] 22:48:26,265 INFO  [SoapUILoadTestRunner] LoadTest1: Maximum number of
errors [30] exceeded for step [Groovy Script]
    [java] 22:48:26,265 ERROR [SoapUILoadTestRunner]
com.eviware.soapui.support.SoapUIException: LoadTests failed

BUILD FAILED
File..... C:\Documents and
Settings\ole.matzura\.maven\cache\maven-soapui-plugin-1.5beta1\plugin.jelly
Element... ant:java
Line..... 104
Column.... 105
Java returned: 1
Total time: 18 seconds
Finished at: Sun Mar 12 22:48:26 CET 2006
```

Next: maven 1.X plugin goals

1.12.1.2 Maven 2.X Plugin

soapUI maven 2.X plugin

Usage

Prior to using the plugin, add the eviware maven 2 repository either to your project or settings.xml;

```
<pluginRepositories>
   <pluginRepository>
      <id>eviwarePluginRepository</id>
      <url>http://www.eviware.com/repository/maven2/</url>
   </pluginRepository>
</pluginRepositories>
```

Then, add the soapUI plugin to your pom.xml

```
<plugins>
  <plugin>
  <groupId>eviware</groupId>
  <artifactId>maven-soapui-plugin</artifactId>
  <version>SNAPSHOT</version>
  <configuration>
    <projectFile>sample-soapui-project.xml</projectFile>
    <host>http://127.0.0.1:8181</host>
  </configuration>
  </plugin>
</plugins>
```

Run functional tests with

```
mvn eviware:maven-soapui-plugin:test
```

loadtests with

```
mvn eviware:maven-soapui-plugin:loadtest
```

and MockServices with

```
mvn eviware:maven-soapui-plugin:mock
```

The plugin will load the specified project file and run all TestCases available in all TestSuites. If you want to narrow down which TestSuites/TestCases/LoadTest to run, use the `testSuite`, `testCase` and `loadTest` properties for that purpose.

Integrated Tools

If you configure any of the integrated code-generation tools in soapUI for an interface in your project, you can invoke this code-generation functionality with

```
mvn eviware:maven-soapui-plugin:tool
```

which could be configured in the maven 2 build process as part of the generate-sources step:

```
<plugins>
        <plugin>
                <groupId>eviware</groupId>
                <artifactId>maven-soapui-plugin</artifactId>
                <version>SNAPSHOT</version>
                <configuration>
                           <projectFile>sample-soapui-project.xml</projectFile>
                           <testSuite>OleTest</testSuite>
                           <iface>IOrderService</iface>
                           <tool>wsi,axis1,axis2</tool>
<settingsFile>C:\workspace\core\soapui-settings.xml</settingsFile>
                         </configuration>
                <executions>
                    <execution>
                              <phase>generate-sources</phase>
                              <goals>
                                <goal>generate</goal>
                              </goals>
                    </execution>
                  </executions>
        </plugin>
</plugins>
```

Several tools can be specified in a comma-seperated list, so the above example would run both WS-I reports and configured code generation for both Axis1 and Axis2. Also, the soapUI settings file has been specified since it contains tool paths and global WS-I settings

Next: maven 2.X plugin goals

1.12.2 NetBeans Plugin

soapUI NetBeans-Plugin

The soapUI NetBeans-Plugin provides all soapui-functionality directly from within NetBeans 5.5 allowing one integrated development and testing environment for c in java. Existing soapUI desktop components are made available through two docked tabs:

- A "soapUI Navigator" tab in the explorer contains the Workspace Navigator and Details Tab
- A "soapUI Log" tab at the bottom contains the same Log Tabs as in soapUI

All soapUI Desktop Panels are made available as editor tabs and will be opened/closed as usual. Here are some screenshots:

Currently, the plugin can be seen as a "level-1" integration, ie it is "merely" a packaging of soapUI components in the NetBeans environment.

Getting Started

Install the soapUI Plugin as described in the Installation document. The existing Getting Started applies also to the NetBeans Plugin, as does the User Guide . The only real difference is that the standalone soapUI Desktop has been replaced by the NetBeans tabbed interface for editors.

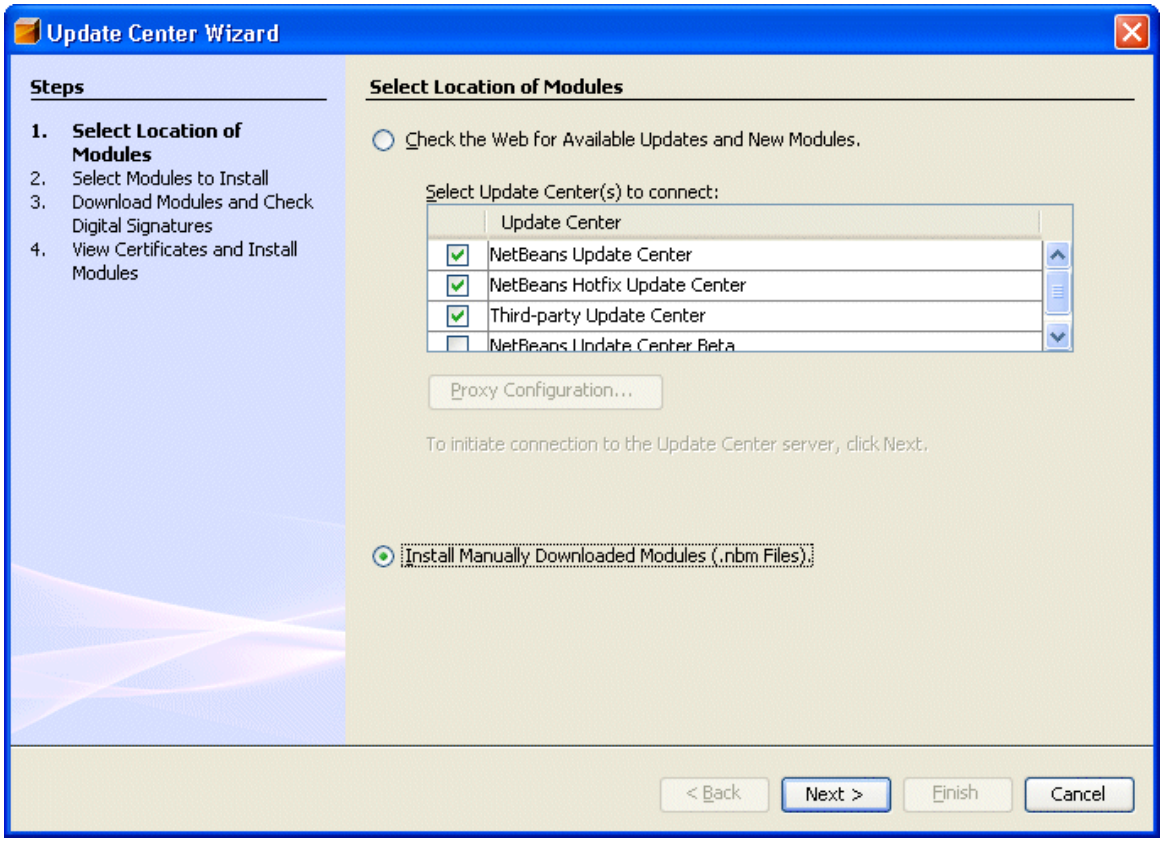
Next: Installing soapUI NetBeans Plugin

1.12.2.1 Installation

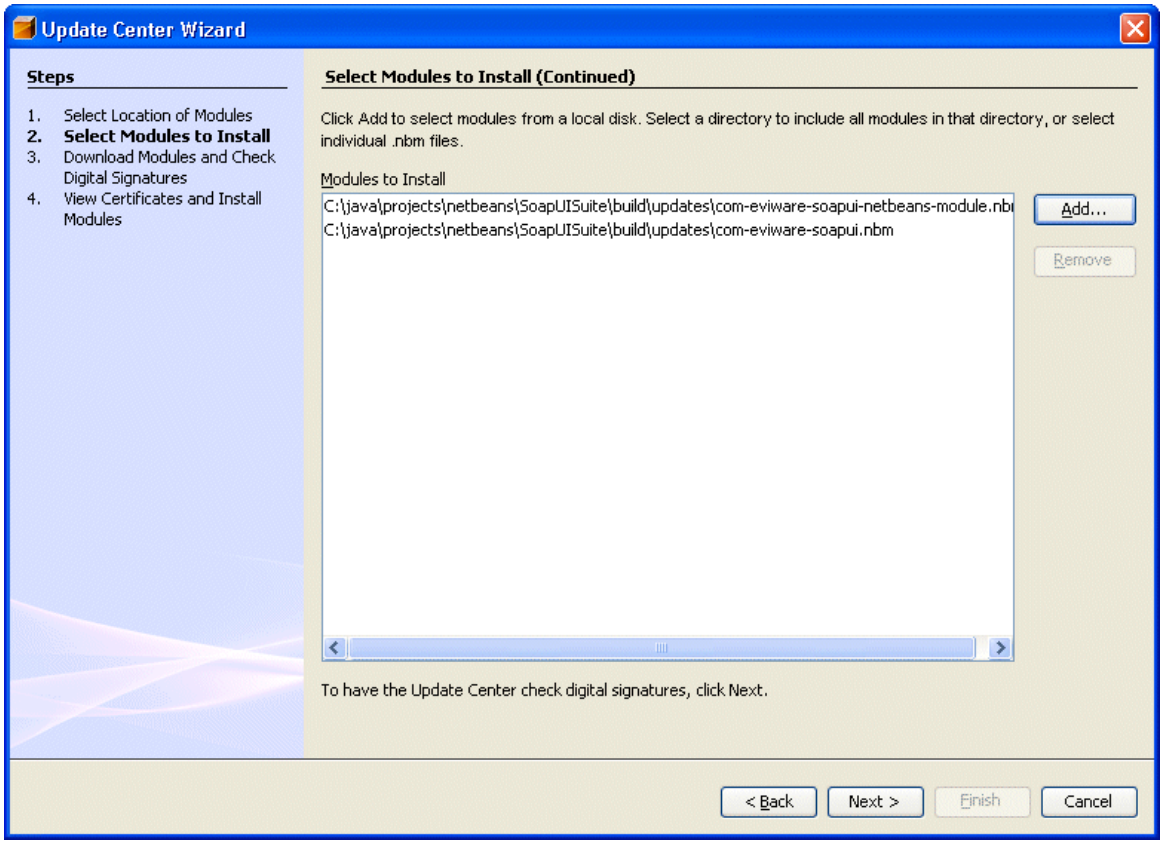
Installing the NetBeans-Plugin

Here comes a walkthrough of how to install the soapUI Plugin in NetBeans 5.5;

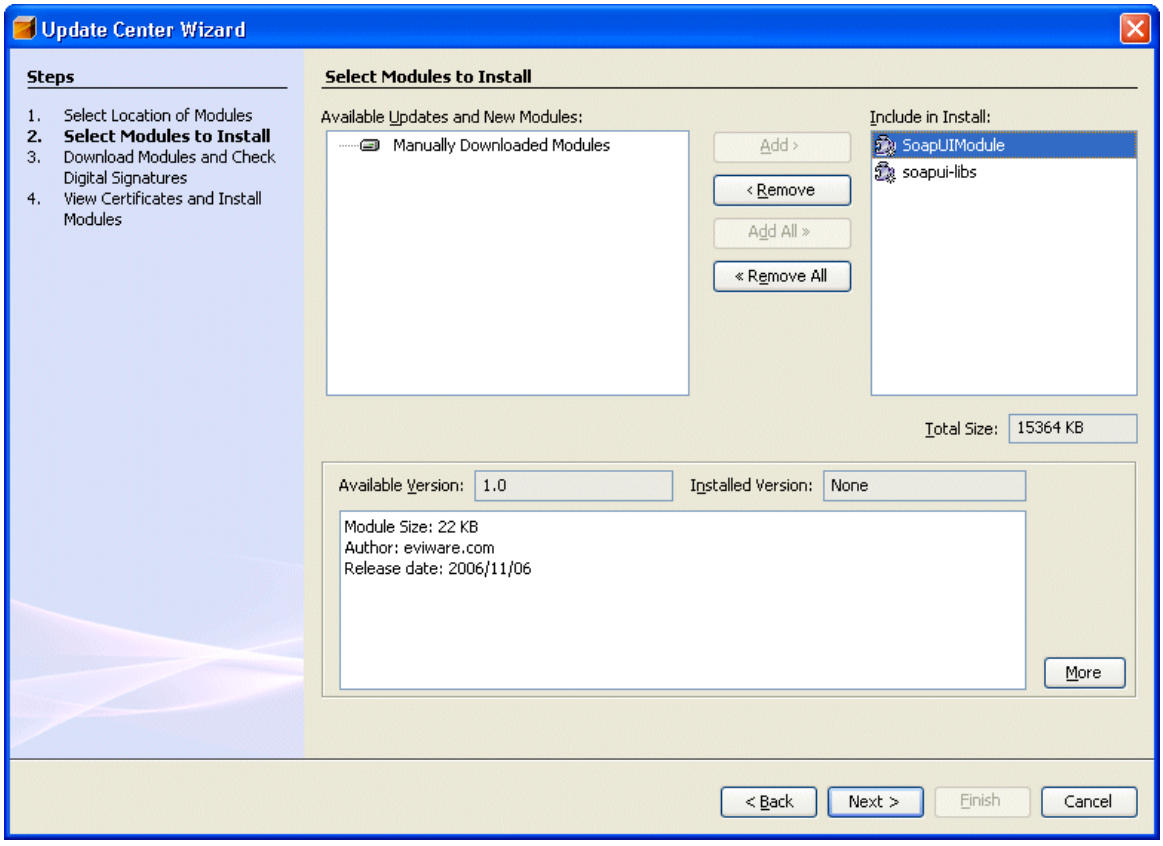
- Start by downloading the zip distribtion from sourceforge and unzipping its 2 files into a temporary folder.
- Start NetBeans and select the "Tools / Update Center" menu item
- In the initial "Select Location Of Modules" step select the "Install Manually Downloaded Modules (.nbm Files)" option:



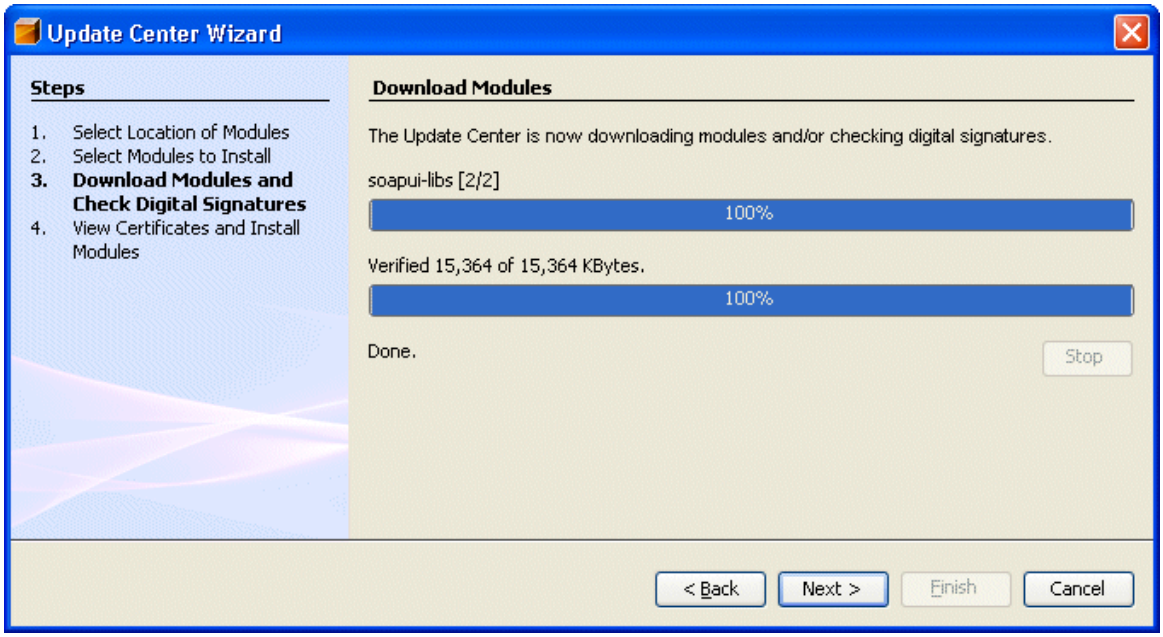
- In the "Select Modules to Install" step select both unzipped *.nbm files with the "Add.." button:



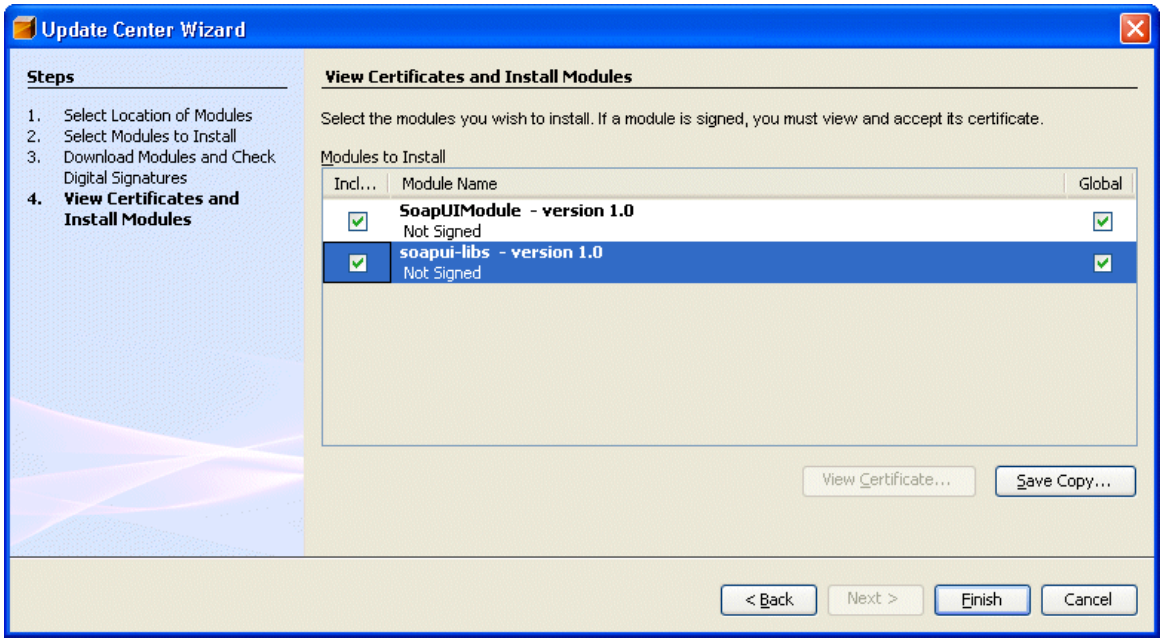
- In the next "Select Location Of Modules" make sure that both modules ("soapui-libs" and "SoapUIModule") are added to the "Include in Install" list:



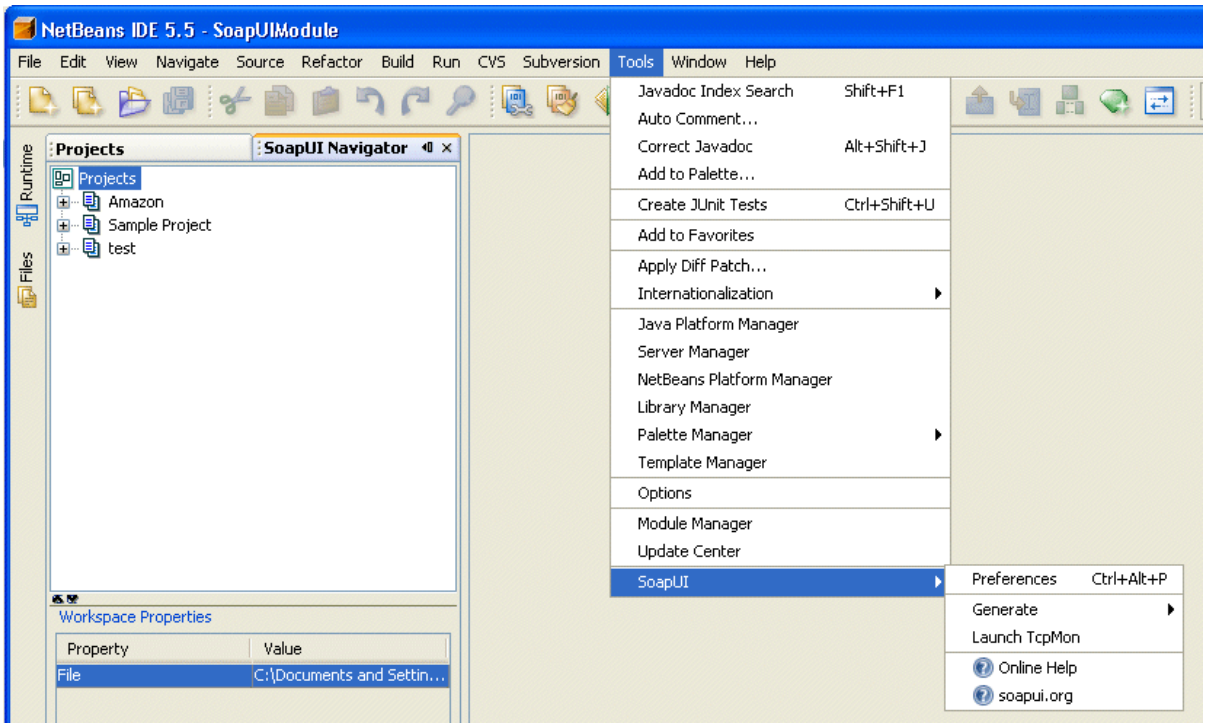
- In the "Download Modules and Check Digtal Signatures" step NetBeans will install the selected modules:



- In the "View Certificates and Install Modeuls" step select both modules (ignore the warning about the unsigned plugins):



After select "Finish", NetBeans will churn a while and if all goes well you should see a new Explorer Tab named "soapUI Navigator" and the Tools menu should now contain a "soapUI" submenu at the bottom:



Next: The Maven Plugins

1.12.3 IntelliJ Plugin

soapUI IntelliJ-Plugin

The soapUI IntelliJ-Plugin provides all soapui-functionality directly from within IntelliJ IDEA allowing one integrated development and testing environment for Web Services in java. Existing soapUI desktop components are made available through two ToolWindows:

- A "soapUI Navigator" ToolWindow at the left contains the Workspace Navigator and Details Tab
- A "soapUI Log" ToolWindow at the bottom contains the same Log Tabs as in soapUI

All soapUI Desktop Panels are made available as editor tabs in IDEA and will be opened/closed as usual. Here are some screenshots:

Currently, the plugin can be seen as a "level-1" integration, ie it is "merely" a packaging of soapUI components in the Intellij environment.

Installation

The plugin is downloadable from sourceforge and through IntelliJs plugin-manager. After installing, you should increase IntelliJs PermGen space as the plugin requires a fair amount of memory; do this by altering the idea.exe.vmoptions file in the idea bin directory to for example:

```
-Xms128m
-Xmx512m
-XX:MaxPermSize=128m
-ea
```

Getting Started

The existing Getting Started applies also to the IntelliJ Plugin, as does the User Guide . The only real difference is that the standalone soapUI Desktop has been replaced by the IntelliJ tabbed interface for editors.

Next: Working with NetBeans

1.12.4 Eclipse Plugin

soapUI eclipse-plugin

The soapUI eclipse plugin provides full soapUI functionality from within eclipse. Apart from "standard" soapUI 1.7 functionality, the eclipse plugin contains a soapUI project nature and also adds a new soapUI perspective which mimics the layout of the standalone soapUI version.The plugin makes use of the SWT_AWT bridge which greatly eases our development effort but may result in graphical glitches with dialogs and some window updates.. but all core functionality should be working ok (let us know otherwise)...

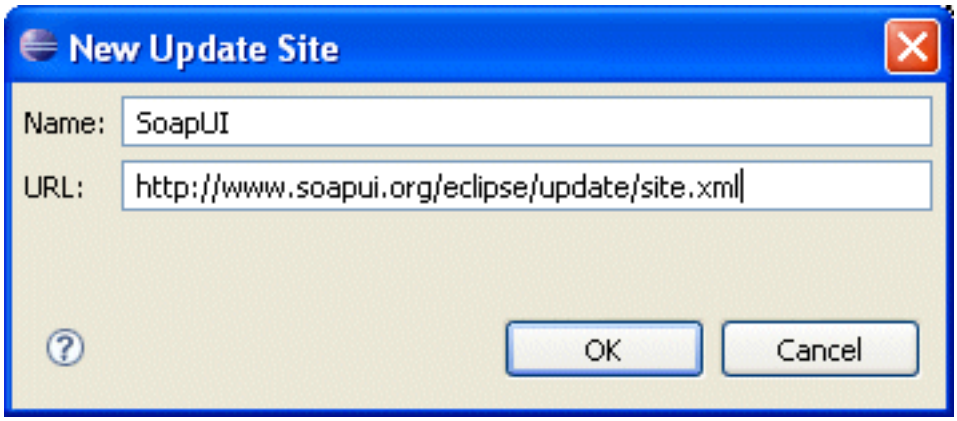
Currently the eclipse-plugin can be downloaded from sourceforge or through the update site located at http://www.soapui.org/eclipse/update (see below)

Update Site

An eclipse update site is now available at http://www.soapui.org/eclipse/update/site.xml, Install the soapui-eclipse-plugin with the following steps:

1. Select "Help"/"Software Updates"/"Find and Install..."
2. Select the "Search for new features to install" option
3. Press the "New Remote Site" button and add the information shown in the image to the right
4. Select Finish and the follow the dialogs to install the soapUI feature

Read the Getting Started document and the User Guide to get going!



The soapUI Perspective

Once installed, open the soapUI Perspective by using the standard Window/Open Perspective/Other.. command and selecting "soapUI" from the list. The perspective will open 2 views;

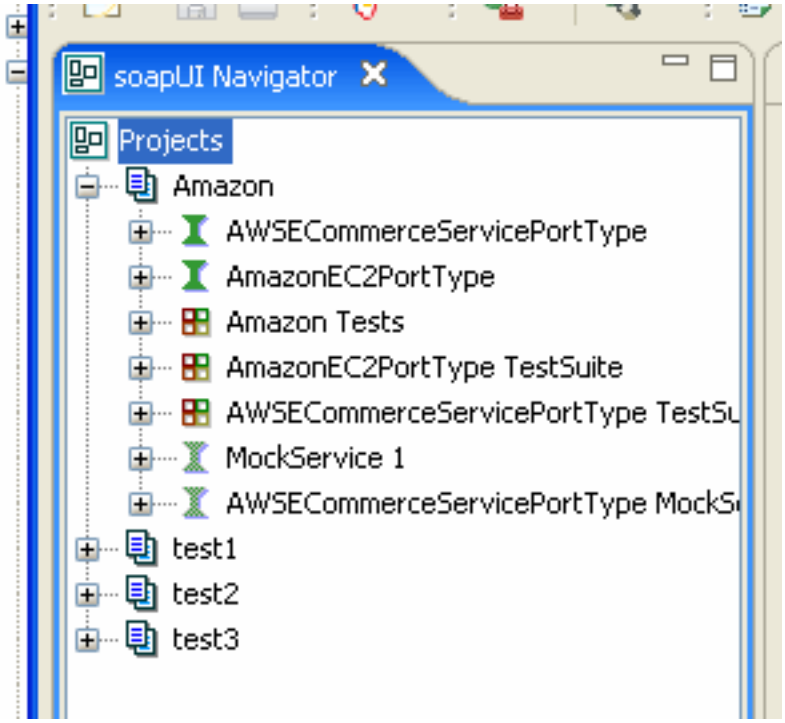
- A "soapUI Navigator" view to the left containing the same Navigator and Details tab as the

standalone soapUI version.

- A "soapUI Logs" view to the bottom containing the same log tabs as the standalone soapUI version.

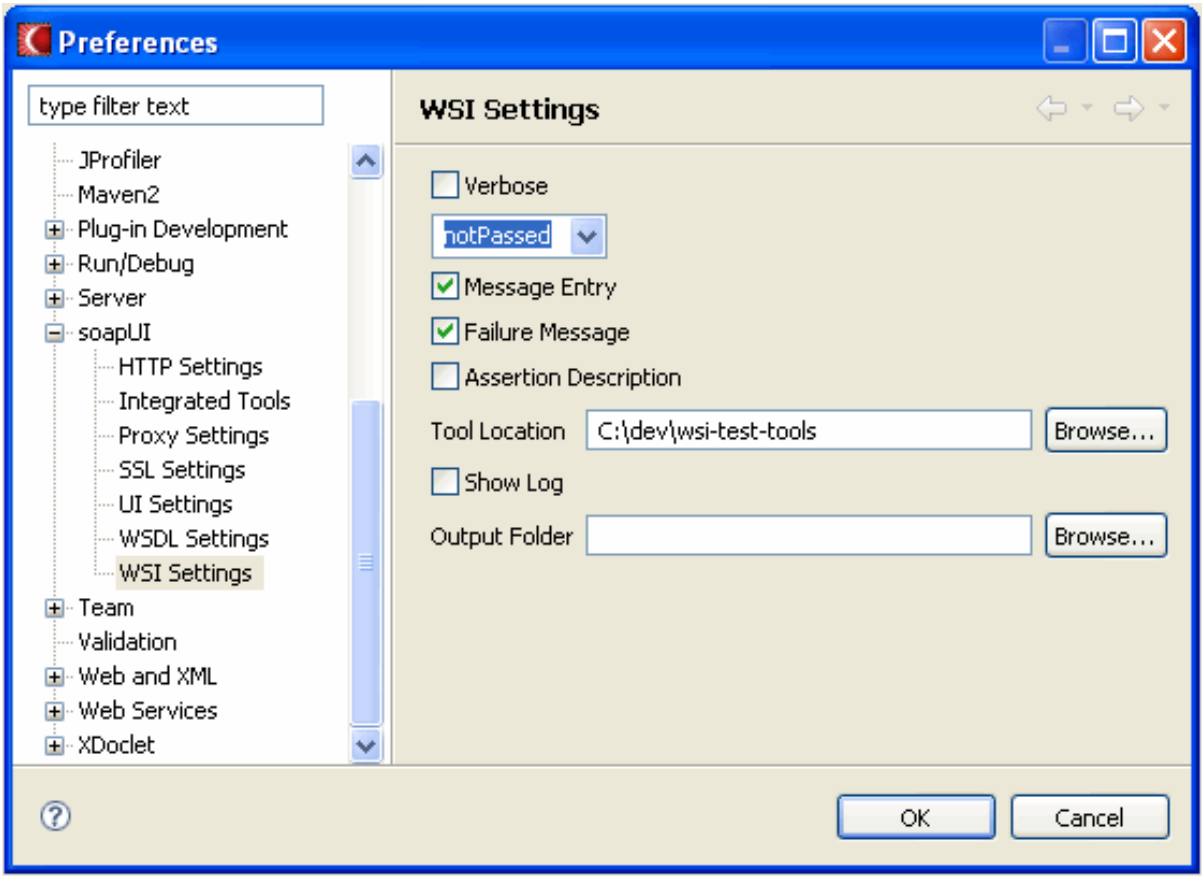
Navigation/actions work the same as in the standalone version, windows are opened as tabs on the eclipse desktop and can be moved/docked around as usual

Of course, the above mentioned views can be added to any perspective using the Window/Show View/Other.. command and selecting either one of them in the soapUI group.



soapUI Preferences

Most soapUI Settings are available from the standard Window/Preferences dialog under the soapUI node;



Next: soapUI Nature

1.12.4.1 soapUI Nature

soapUI Nature

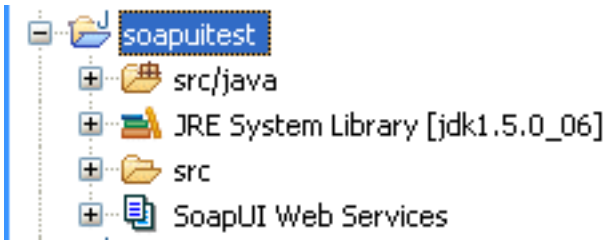
The soapUI Nature allows integrated access to all soapUI-functionality directly from within a java-project. It is not complete but has been released to give an idea of how we envision working with web-services in a eclipse project.

The following example walks through a "top-down" scenario:

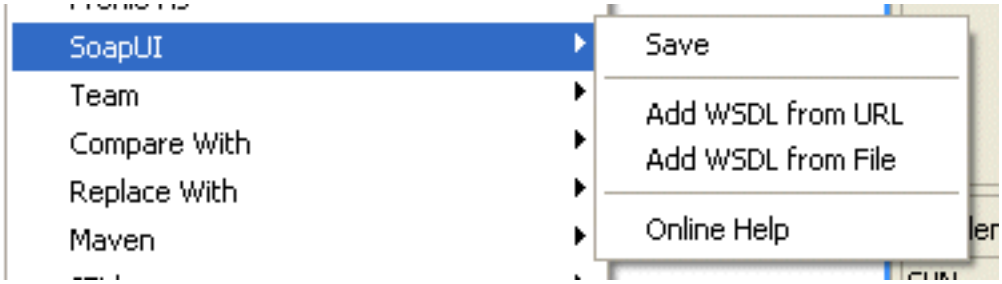
Step 1: Enable soapUI Nature

Start by creating an empty java project and enabling the soapUI Nature from the projects popup menu:

If enabling goes will, you will see a "soapUI Web Services" node in the project:



When shown in the Eclipse Project Explorer View, this node behaves like a standard soapUI project node. Its right-click-menu includes a "soapUI" menu with common project actions:

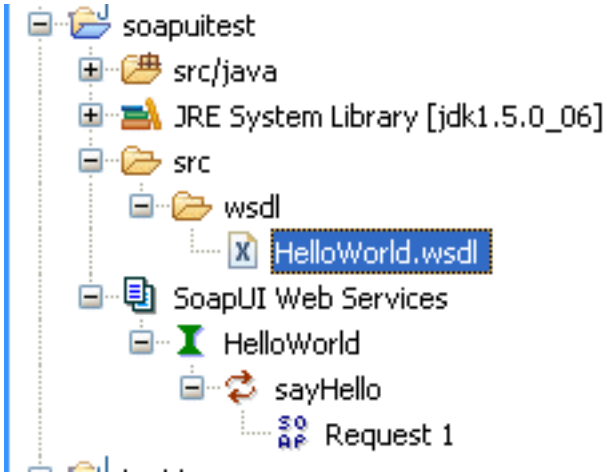


Step 2: Create or Import a WSDL

You can either manually create a WSDL in your project (using for example the formidable Web Tools Project WSDL Editor) or just import one into your project using one of the standard "Add WSDL from ... " actions.

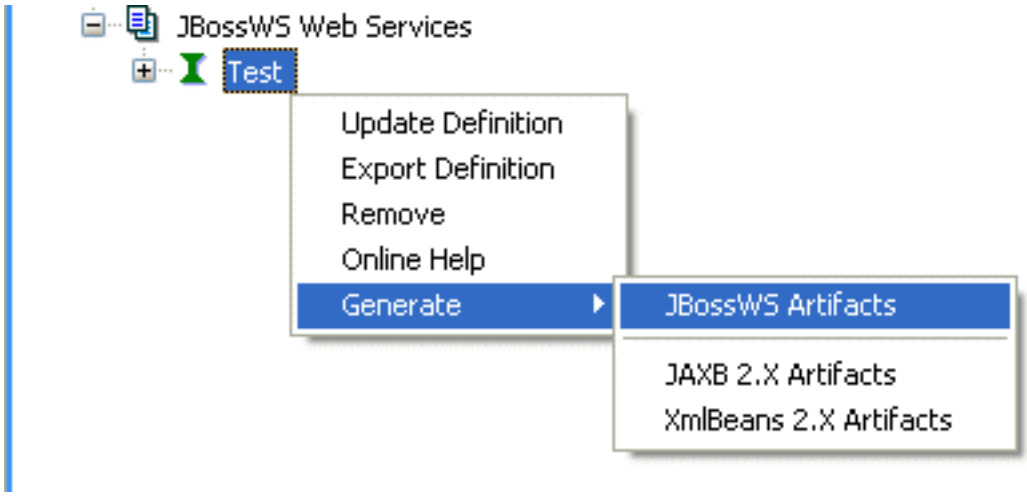
When creating the WSDL in your project, right click on the WSDL file and select "soapUI -> Add to soapUI Project"

Once imported, the WSDL is shown as a "standard" soapUI Interface node under the "soapUI Web Services" node:

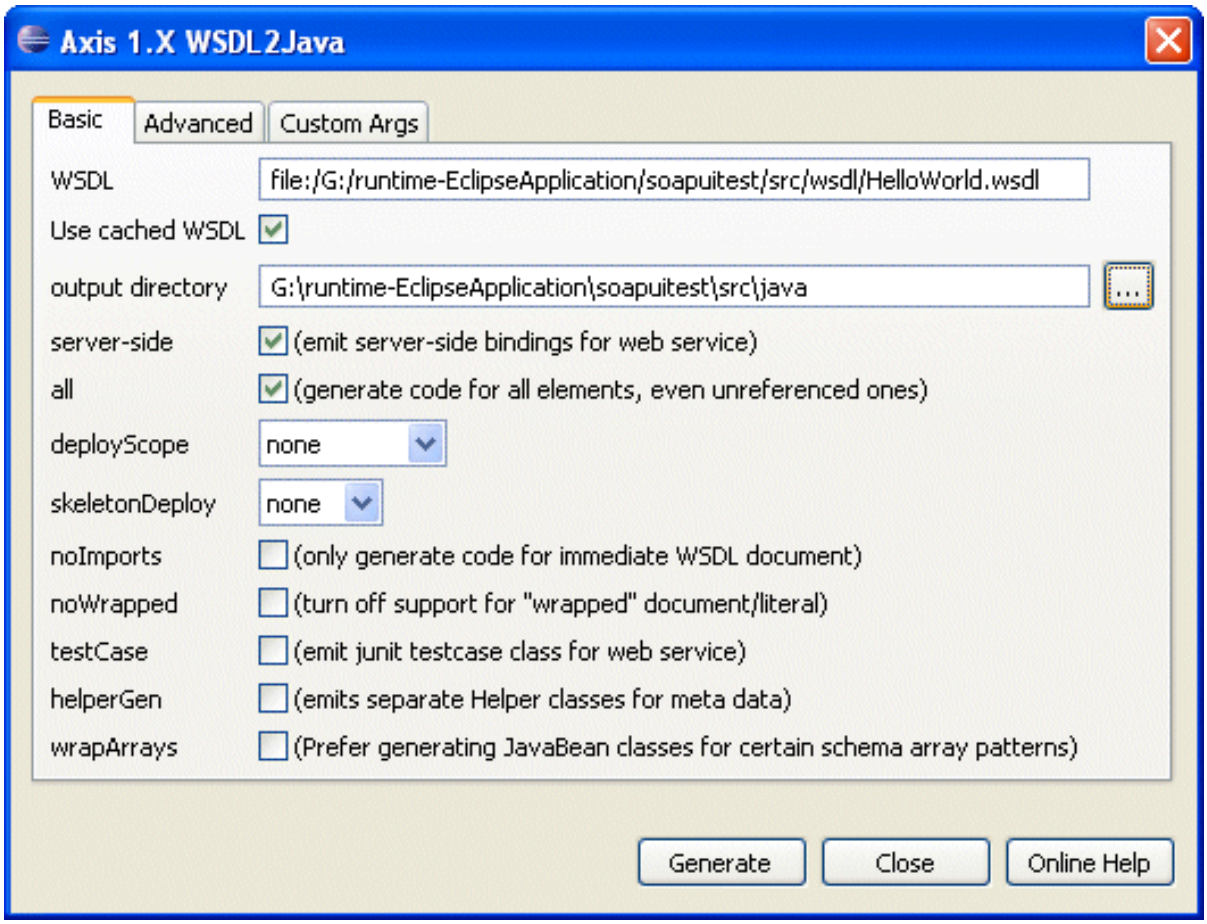


Step 3: Generate Code

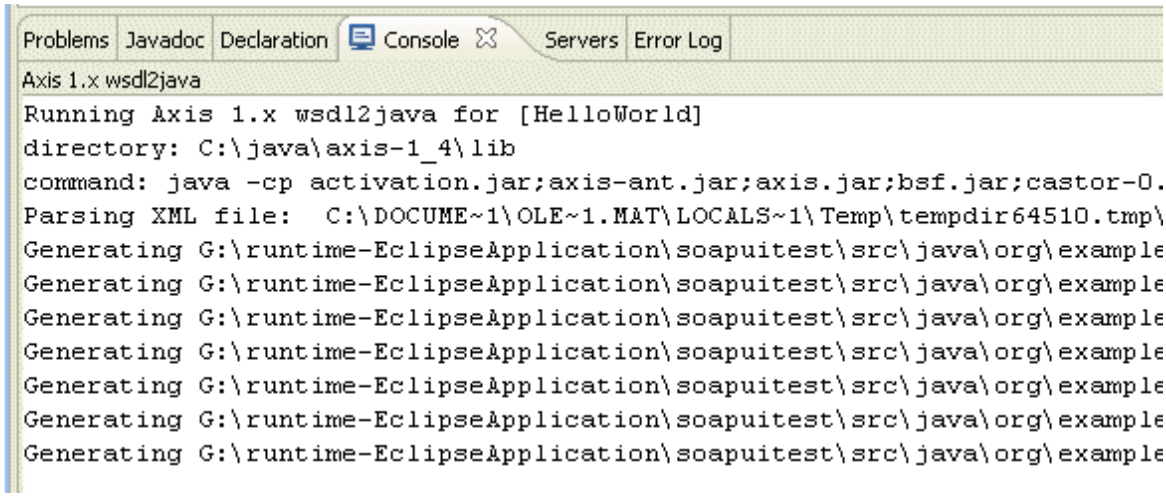
Now its time to generate some code.. select the "Generate -> Axis 1.X Artifacts" menu option from the Interface nodes popup menu which will show the below dialog



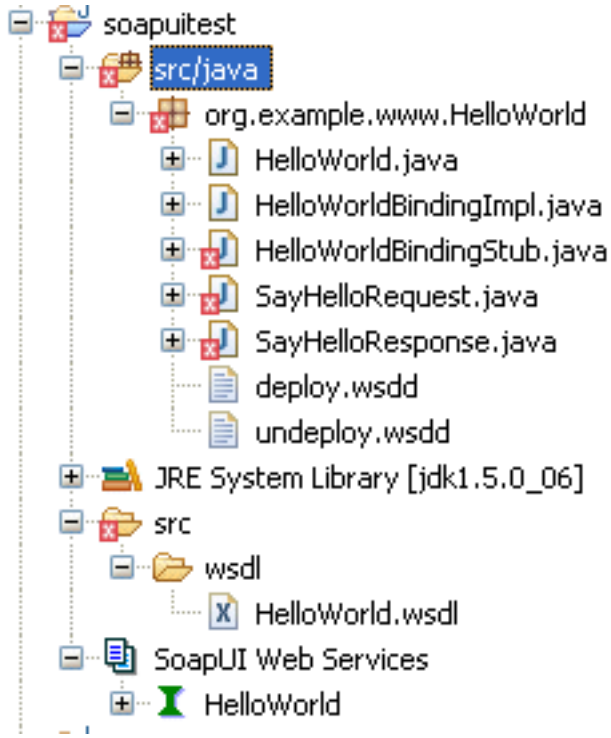
Select the desired options and set the output folder to a java source folder in your project..



Select the "Generate" button which will invoke Axis (as configured under "Preferences -> soapUI -> Integrated Tools") and show the output in the console window:



The generated classes are now visible under the java source node (refresh first!)



(The classes are marked red above since the axis-libraries are not in the projects classpath. These will be added automatically by soapUI in a future version)

Step 4: Implement, Deploy and Test

After implementing and deploying your Web Service to the desired container, you can now start sending webservice requests "as usual".. good luck!

Next: Working with JBoss WS

1.12.5 JBossWS Plugin

JBossWS Plugin

The JBossWS Plugin is a specialized Eclipse plugin for working with the formidable JBossWS Web Service stack together with JBossIDE or standalone. Apart from the standard soapUI Eclipse Plugin features, the JBossWS Plugin currently has initial functionality that supports:

- Publishing POJO's/EJB's as JBossWS Web Services
- Consuming an existing Web Service with JBossWS
- Implementing an existing WSDL contract with JBossWS
- Adding JSR-181 Web Service Annotations to an existing java class

The following improvements are planned to be available for the final JBossIDE 2.0.0 release:

- Improved support for implementing Web Services from an existing WSDL contract
- Improved support for consuming Web Services from en EJB or Servlet
- Improved support for adding Web Service annotations on a method level
- Wizards for security configurations
- Project/File templates for JBossWS projects and related artifacts
- Tighter integration with the Eclipse Web Tools Projects Web Service support
- Integrated Help

Please submit questions and discuss issues on the JBoss Eclipse IDE User Forum , and report bugs/requests/etc in JBossIDE JIRA under the "Web Services Plugin" component!

Latest News

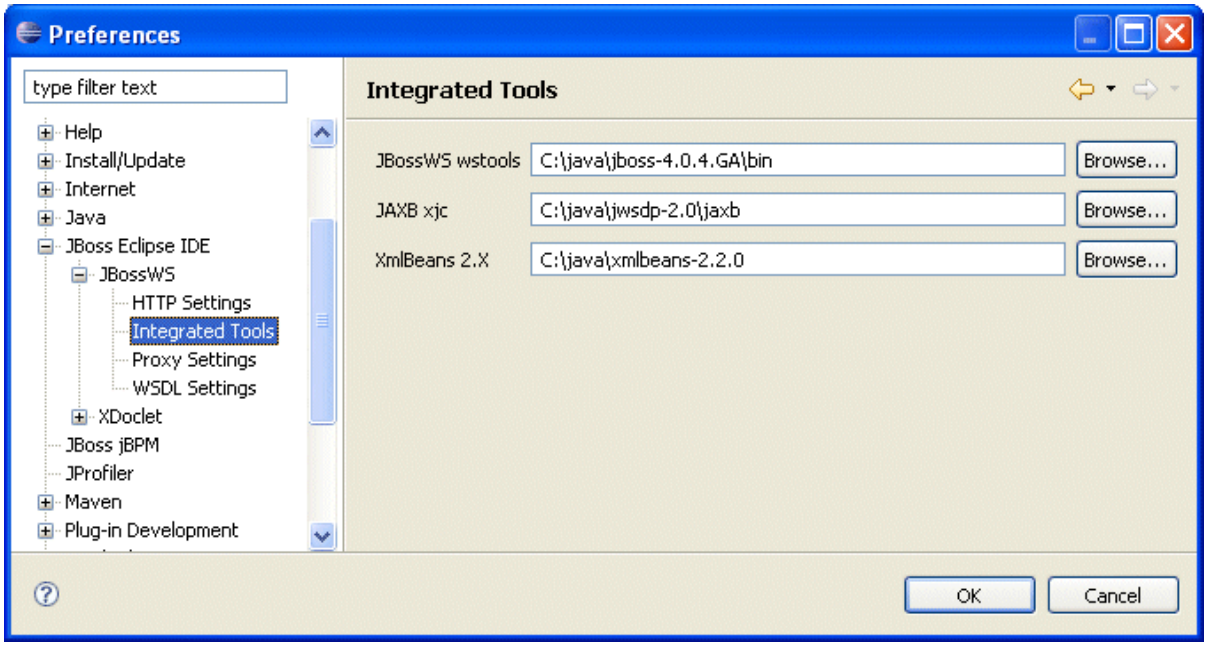
2006-10-12 : JBossIDE 2.0.0 beta2- first "public" release of JBossWS plugin for eclipse / JBossIDE [Read more]

Installation

The plugin is distributed together with JBossIDE 2.0.0 beta2 .

Currently, the plugin requires you to install JBossWS separately (download from http://labs.jboss.com/portal/jbossws/downloads), but it will be bundled in the final jbosside 2.0.0 version

Once you have installed the plugin, start by setting the path to the JBossWS WSTools script in the "JBossIDE / JBossWS / Integrated Tools" preferences page:



(The JAXB/XmlBeans tools are not required for JBossWS support)

Next: Getting Started with JBossWS

1.12.5.1 New in 2.0.0 beta2

New and Noteworthy in 2.0.0 beta2

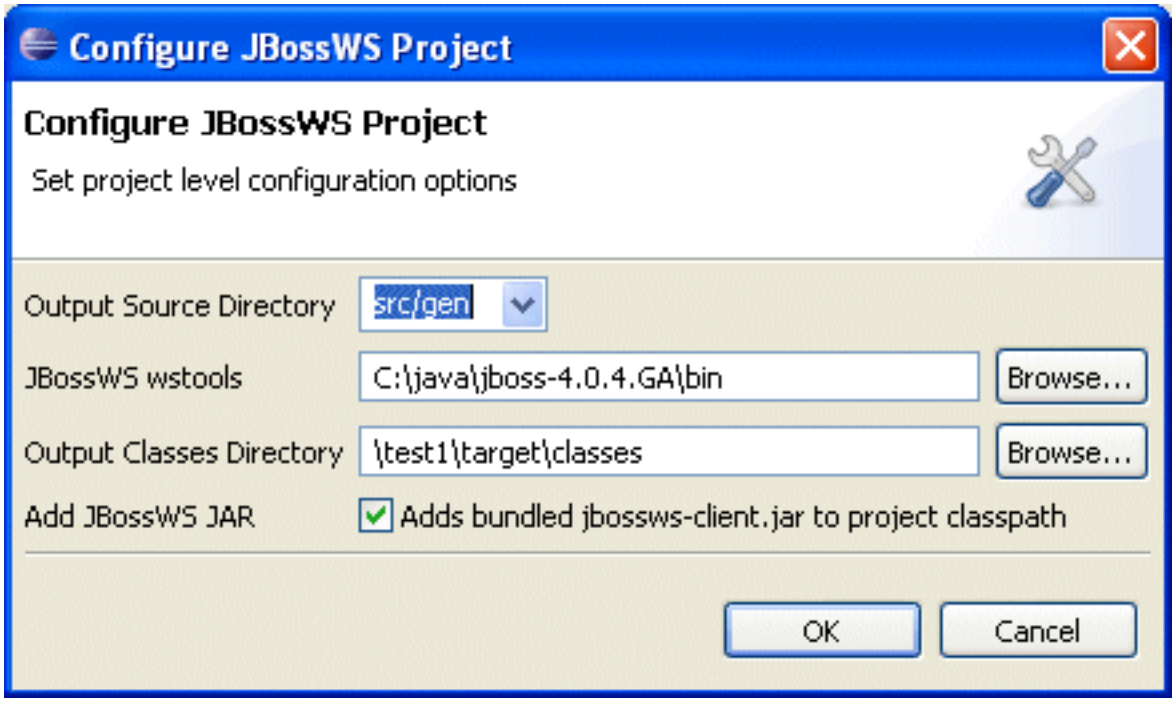
The JBossWS Plugin released together with JBossIDE 2.0.0 beta2 is its first "public" release, including initial site documentation with a number of walk-through-examples for:

- Publishing POJOs as Web Services
- Implementing WSDLs as Web Services
- Annotating EJBs as Web Services

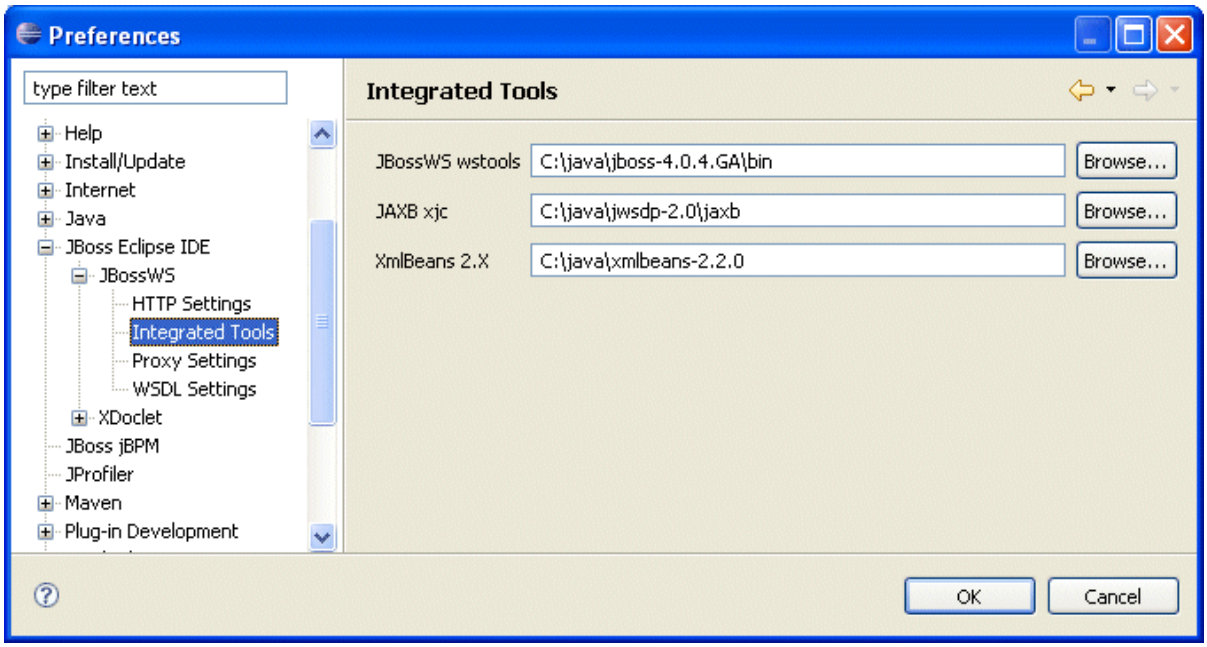
Below is a quick overview of major features:

JBossWS Project Nature

JBossWS-related functionality is made available as a "JBossWS Project Nature" as described in Getting Started

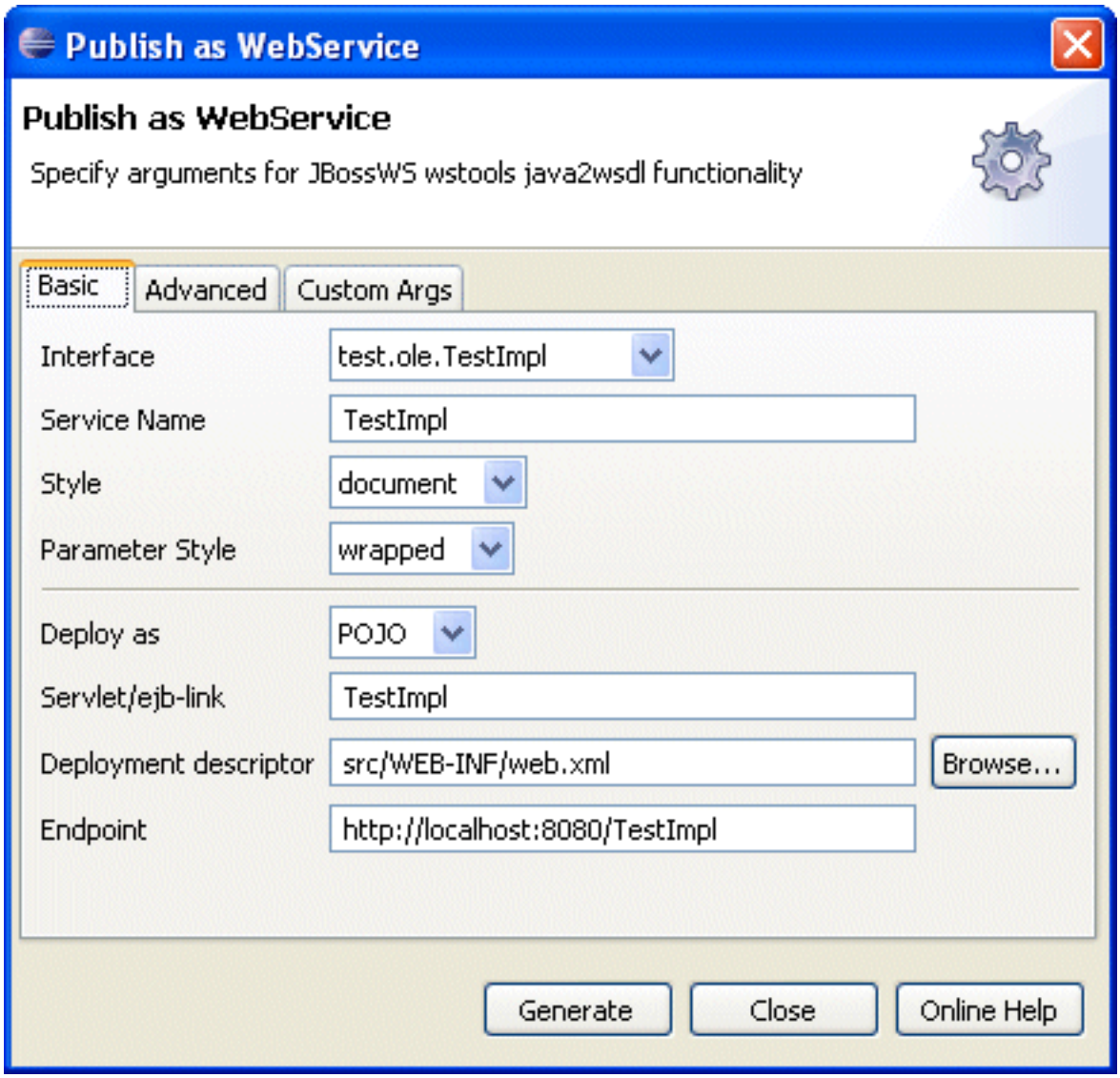


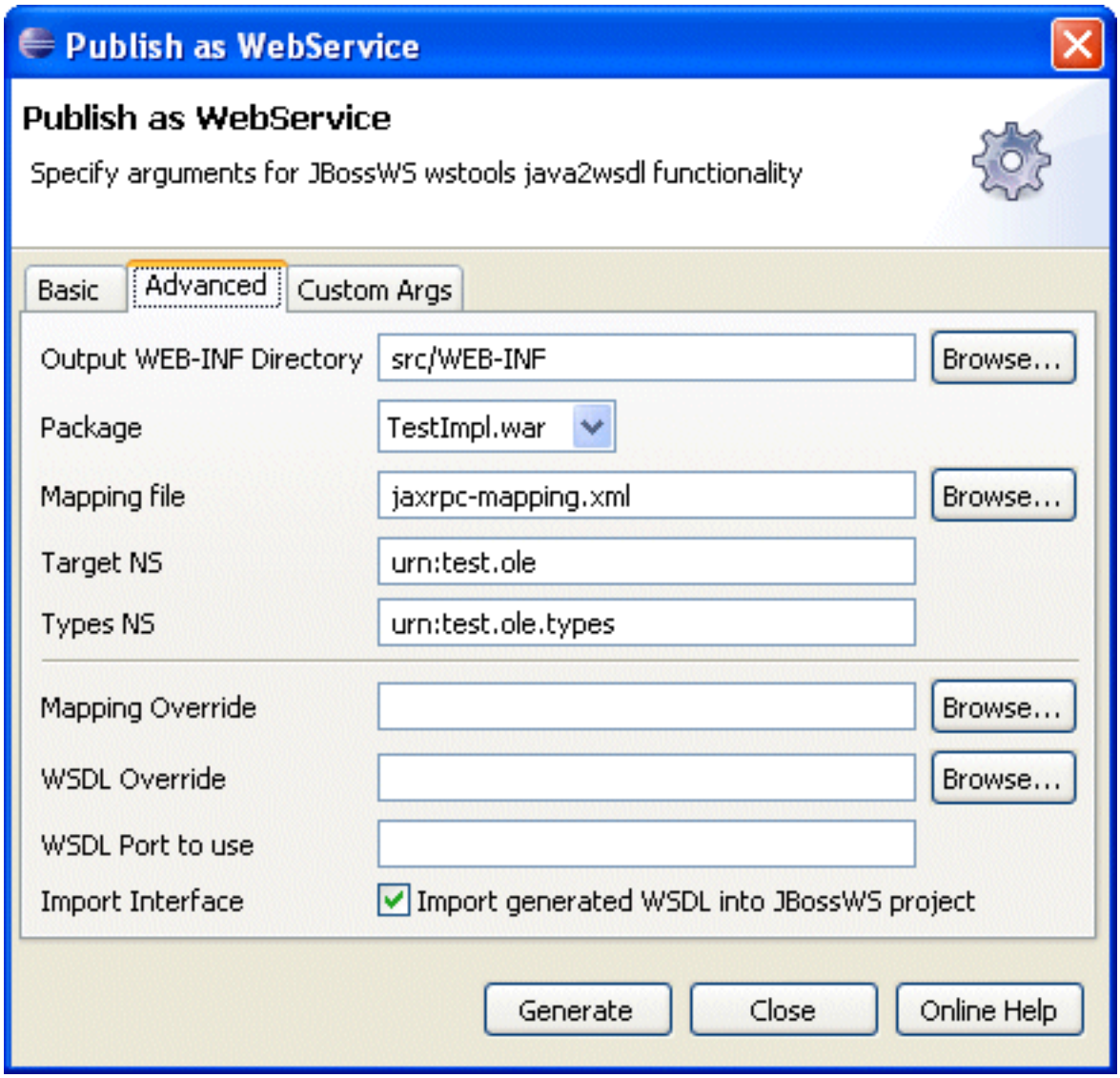
A number of global preferences-pages are available:



Functionality for Publishing Web Services

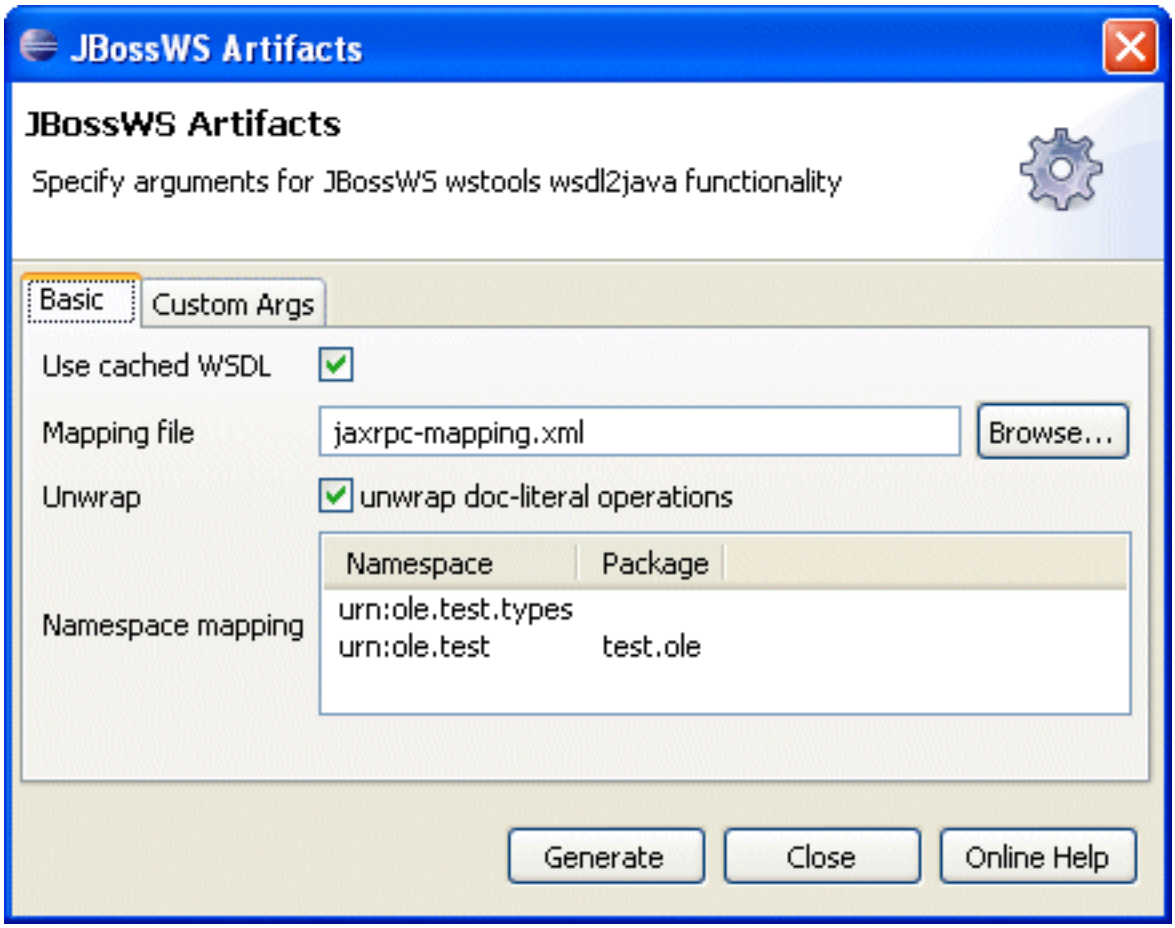
Dialog/functionality for publishing Web Services with JBossWS:





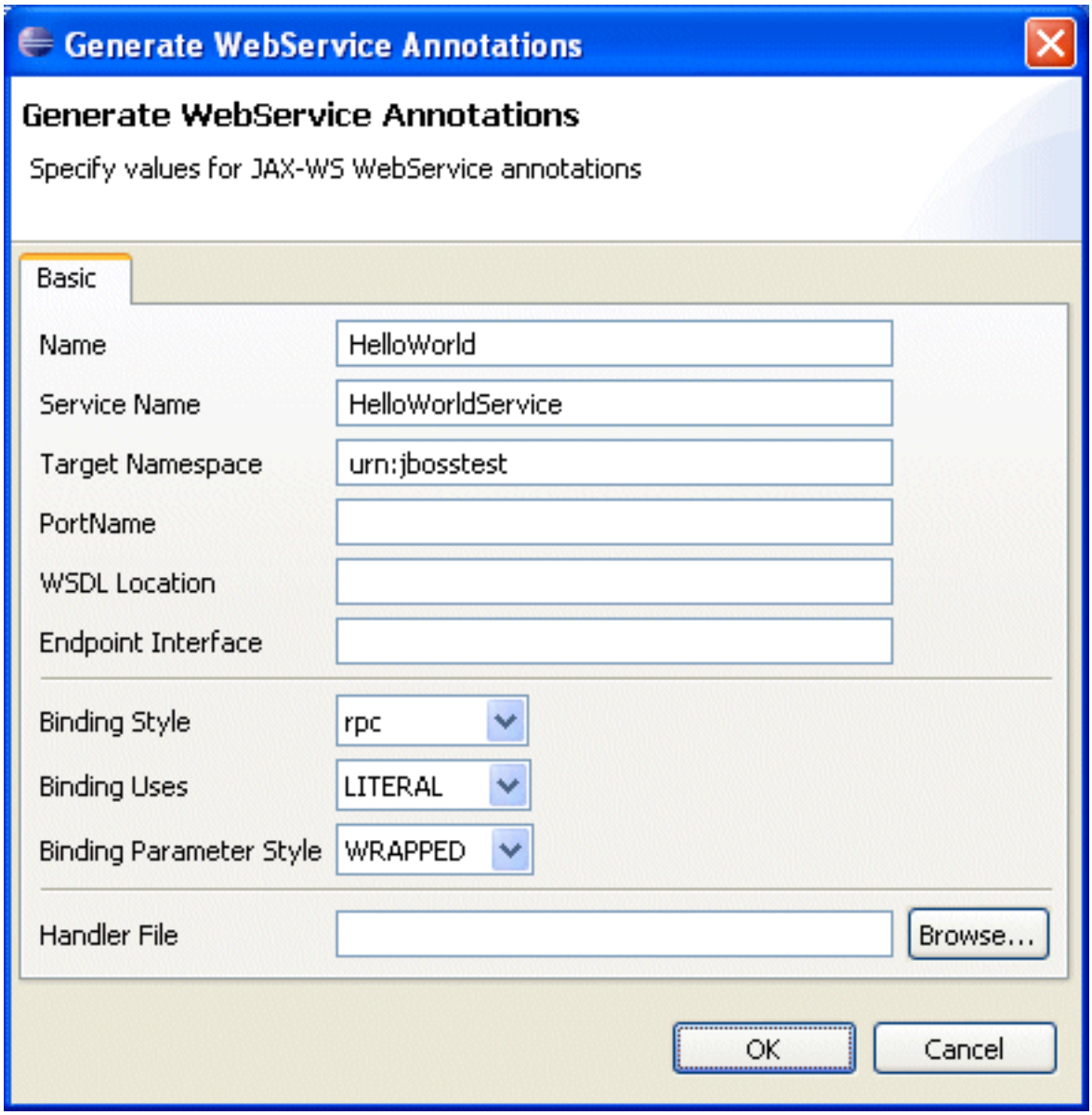
Functionality for Consuming / Implementing Web Services

Dialog/functionality for consuming and implementing Web Services with JBossWS:



Functionality for Web Service Annotations

Dialog/functionality for adding Web Service annotations to existing POJO/EJB3 classes:



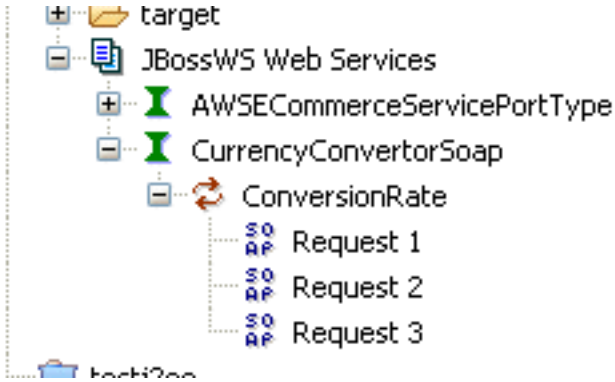
Next: Getting Started with the JBossWS Plugin

1.12.5.2 Getting Started

Getting Started with the JBossWS Plugin

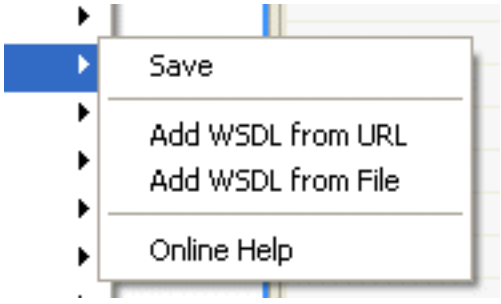
JBossWS-related functionality is made available as a JBossWS-Nature which can be enabled for any type of java-project. This allows you to easily publish, consume and implement webservice from most kinds of projects (web, ejb, jar, etc.). Enabling the nature adds a "JBossWS Web Services" node to the project which corresponds to a soapUI-project.xml file in the file system. In the Eclipse Project Explorer, this node will contain all WebS ervices handled by the project:

- Published Web Services - generated from existing EJB / Pojos as described under Publishing Web Services .
- Imported Web Services - imported from external Web Services either for consuming or implementing from in the project.



Internally, this node corresponds to a soapUI project-node (with some limitations), specific actions available for this node are:

- **Save** - saves changes specific to this node
- **Add WSDL from URL** - imports an external Web Service definition (for example for implementing or for consuming)
- **Add WSDL from File** - imports an existing WebS ervice definition (for example for implementing)
- **Online Help** - shows online help

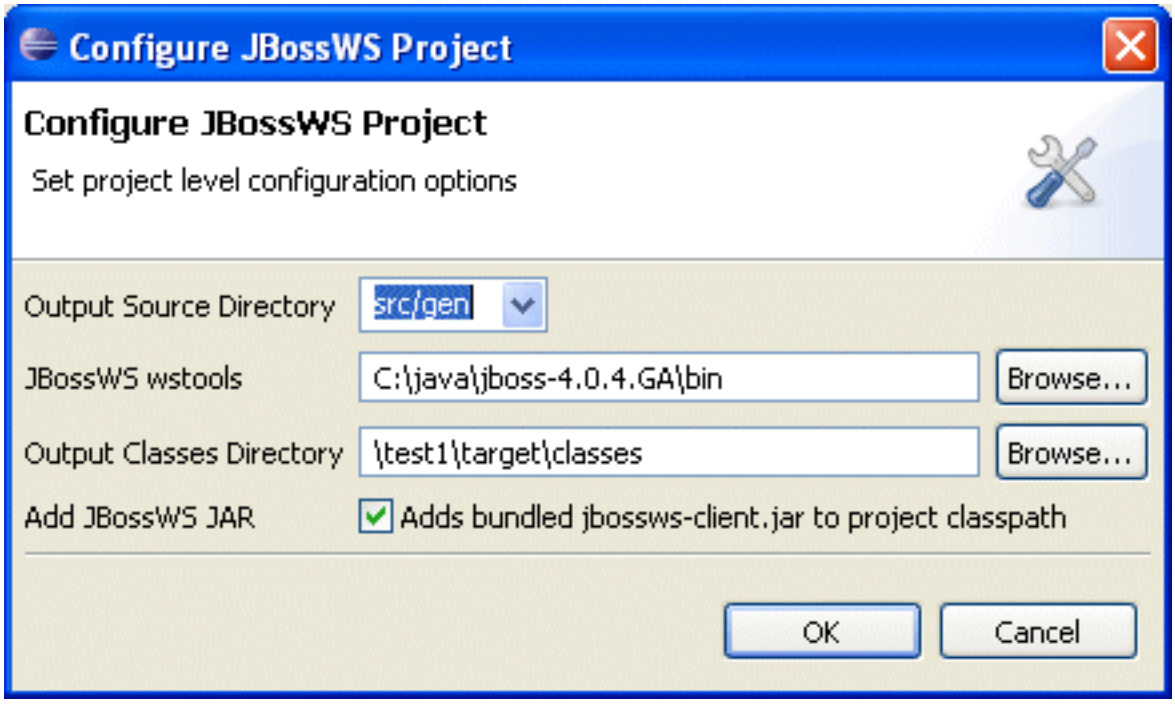


More information on this node and its functionality is availble in the Eclipse User-Guide .

Enabling JBossWS Support

Enable the nature for a project by selecting "JBossWS / Add JBossWS Nature" from the project menu:

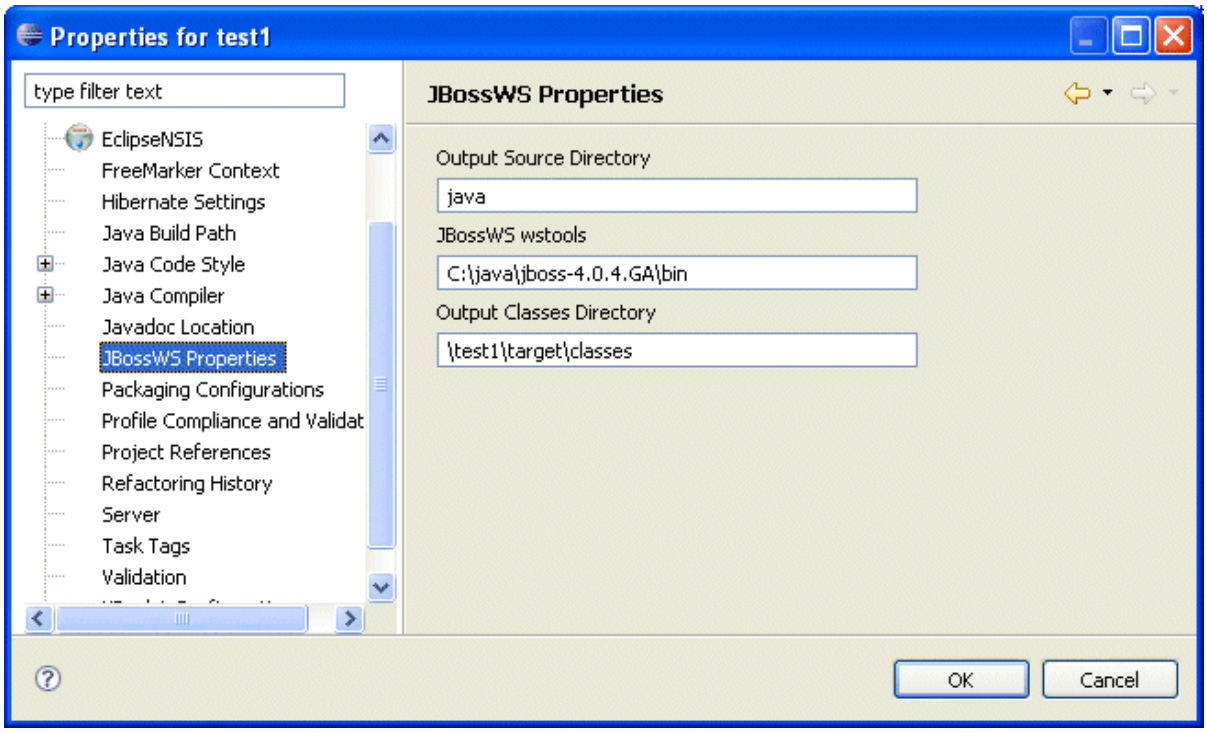
This will prompt with the following dialog:



The prompted options control the basic behaviour of a JBossWS project:

- **Output Source Directory** - the directory into which source code is to be generated when implementing a Web Service
- **JBossWS WSTools** - the path to the installed WSTools script included in the JBossWS distribution. This option defaults to the global setting (as described above), but can be changed on a project basis (for evaluating new versions).
- **Output Classes Directory** - the binary directory for java classes that are to be published as web services.
- **Add JBossWS JAR** - adds the bundled jbossws-client.jar to the projects classpath. Select this option if you intend to use any Web Service specific API's/class (annotations, jax-rpc, etc).

Once the JBossWS Nature has been enabled, these settings are available for later modification from the projects properties:



Once the JBossWS Nature has been enabled, you are now ready to:

- Publish an existing POJO/EJB as a Web Service
- Consume an external Web Service from within your code
- Implement an existing Web Service definition
- Annotate an existing POJO/EJB3 to be published as a JAX-WS Web Service

Next: Publishing Web Services with JBossWS

1.12.5.3 Publishing Web Services

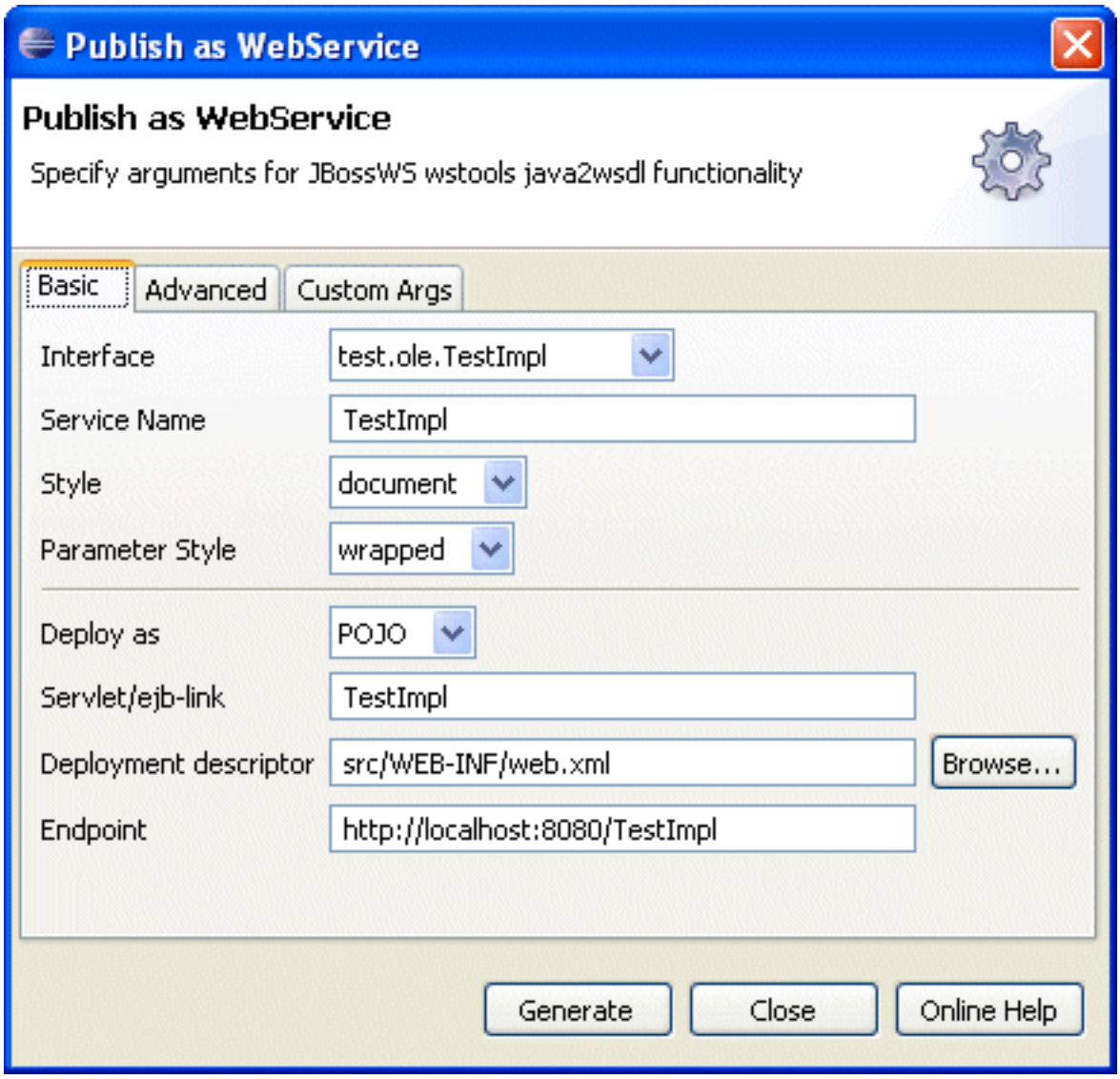
Publishing Web Services

Publishing a Web Service refers to the process of exposing an existing POJO or EJB as a Web Service, either using JSR-109/J2EE based artifacts or using JSR-181 annotations, the JBossWS plugin here acts as a wrapper around the JBossWS WSTools utility and related processes which are described in the JBossWS user guide .

Once the JBossWS Nature has been enabled for a project as described in the Getting Started document, you can right-click on a java-class and select the "JBossWS - Publish as Web Service" option:

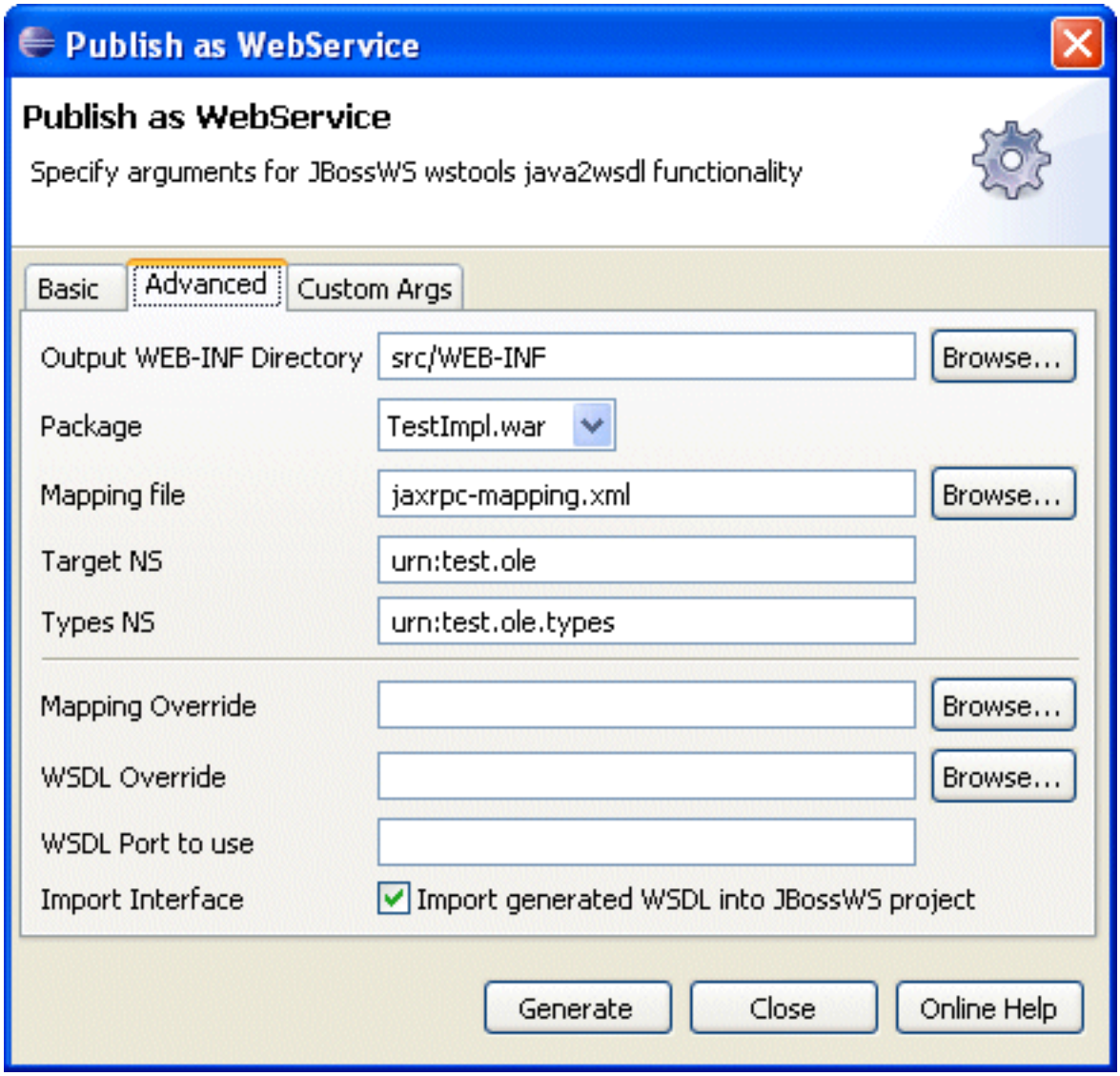
Setting Publish Options

The Basic tab has the following options:



- **Interface** - the Java interface that is to be published, a list containing all interfaces implemented by the selected class will be shown
- **Service Name** - a name for the service to be published
- **Style** - if the service should be document or RPC style
- **Parameter Style** - if the parameters should be wrapped or bare (only for Document style Web Services)
- **Deploy as** - selects if the generated Web Service should be deployed as a POJO (using a servlet) or an EJB.
- **Servlet/EJB-link** - depending on the "Deploy as" setting, this specifies the name of the link for the generated Servlet or EJB
- **Deployment Descriptor** - should point either to a web.xml (for POJO's) or ejb-jar.xml (for EJB's). Generated descriptors will be merged into the specified file
- **Endpoint** - the endpoint under which the Web Service should be accessible. Depends on the deployment and project type (will be automatically suggested/generated in a future version)

The Advanced tab has the following option:



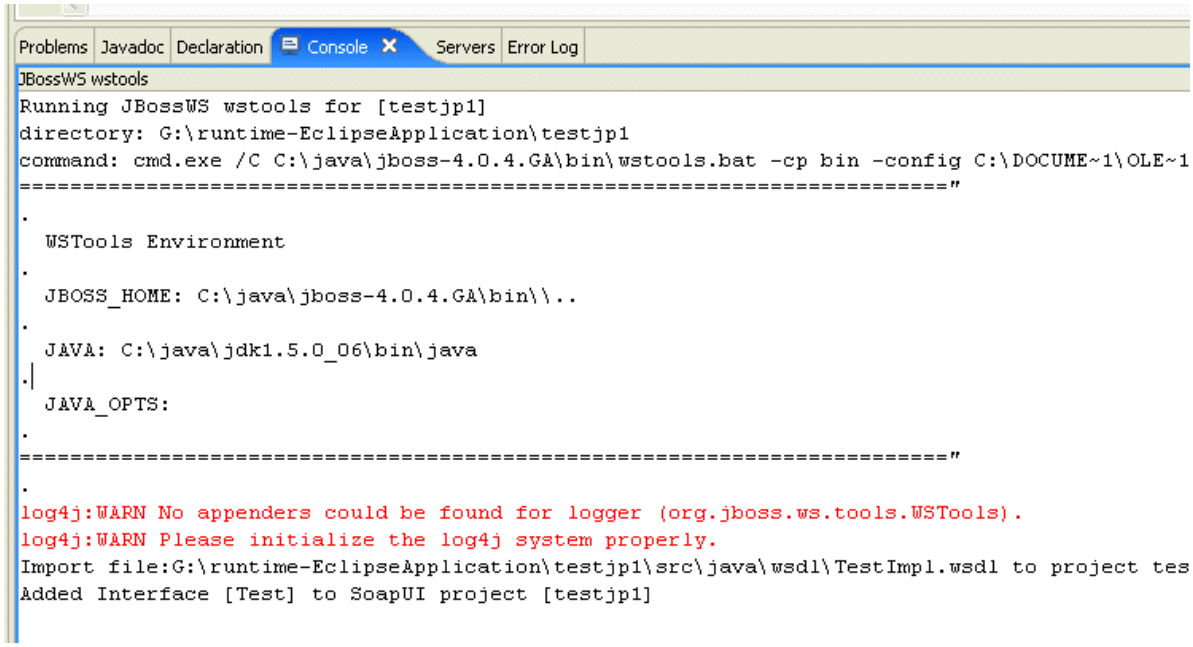
- **Output WEB-INF Directory** - target for generated configuration/mapping files. The JBossWS plugin will merge generated files into existing ones in this directory.
- **Package** - name of a war package to create when publishing as POJO. This will use the packaging features available in jbosside to automatically create a war that can directly be deployed to a jbossws installation
- **Mapping-file** - the name of a JAX-RPC mapping file to create, relative to the output WEB-INF directory. If the file exists, generated mappings will be merged into the existing file.
- **Target NS** - the target namespace for the generated WSDL, defaults to "urn:<package-name>"
- **Types NS** - the namespace for the generated xml-schema types, defaults to "urn:<package-name>.types"
- **Mapping Override** - a previously availabe jaxrpc-mapping file to use instead of the generated one. If specified, the file will be copied into the specified output WEB-INF directory and the generated entry

in webservices.xml will use this file instead.

- **WSDL Override** - a previously availabe WSDL file to use instead of the generated one. If specified, the file will be copies into the specified output WEB-INF directory and the generated entry in webservices.xml will use this file instead.
- **WSDL Port to Use** - if overriding the WSDL, you need to specify which actual port in your WSDL that should be regarded as the one in use. This will be set in the webservices.xml file.
- **Import Interface** - controls if the generated WSDL is to be imported/updated into the underlying JBossWS project node for testing, etc.. Uncheck this option if you for example implementing a WSDL that has already been imported

Generating

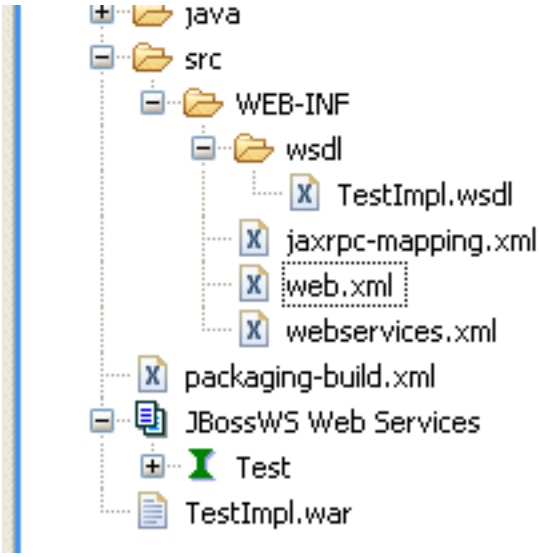
Once all options have been set as required, select the "Generate" button at the bottom of the dialog which will invoke the WSTools utility as configured. The output will be shown in the console view:



The last 2 rows in the above output show that the generated WSDL has been imported into the associated soapUI project; in the Project Explorer you can now expand the "JBossWS Web Services" node and browse the imported WSDLs operations or requests as "normally".

The generated mapping/configuration files are visible under the specified output directory. The plugin will in all cases try to merge created files with existing ones.

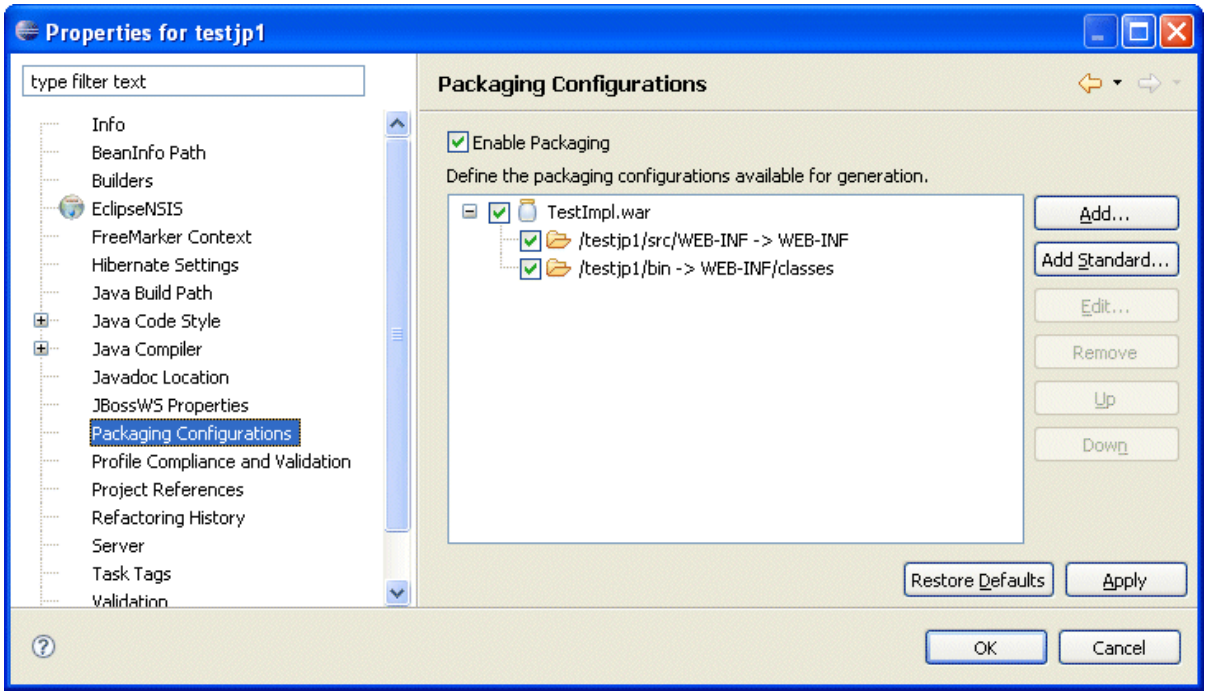
When running the above procedure (for example after adding a new method to the class that is being published), the WSDL will be updated and the corresponding soapUI project will be updated accordingly if the "Import Interface" option has been selected.



Deployment

When running the plugin under JBossIDE and specifying a deployment Package as described above, the jbossws plugin will create a WAR file in the project root containing the current projects classes and WEB-INF directory. Deploy the file by right-clicking and select "Run As / Run on Server" and then selecting a configured JBoss server instance.

The package can be seen and modified under "Project Properties / Packaging Configurations".



Next: Consuming Web Services with JBossWS

1.12.5.4 Consuming a Web Service

Consuming Web Services

Consuming an existing Web Service boils down to creating client side artifacts and then creating instances for invoking the remote service, the process consists of 2 steps:

1. Adding the external WSDL to the "JBossWS Web Services" node
2. Generating a client side classes for invoking the service

Adding the WSDL

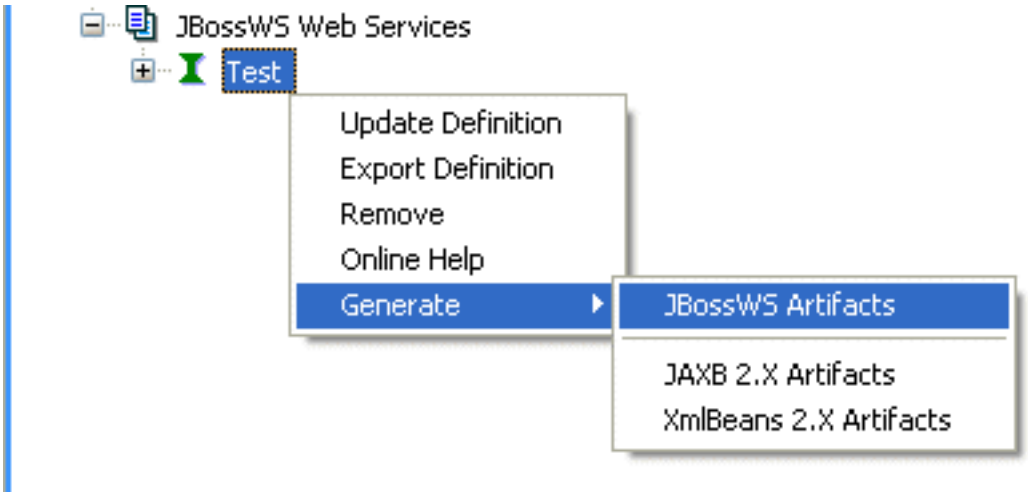
Add the WSDL to the JBossWS Web Services node by selecting the "JBossWS - Add WSDL from URL" option:

This will prompt for a WSDL and import it accordingly into the project

If the WSDL resides locally in the file system it can be added using the corresponding "JBossWS - Add WSDL from File" option. If it is part of the current project, it can be added by selecting the "JBossWS - Add to JBossWS Project" option from the WSDL's right-button menu:

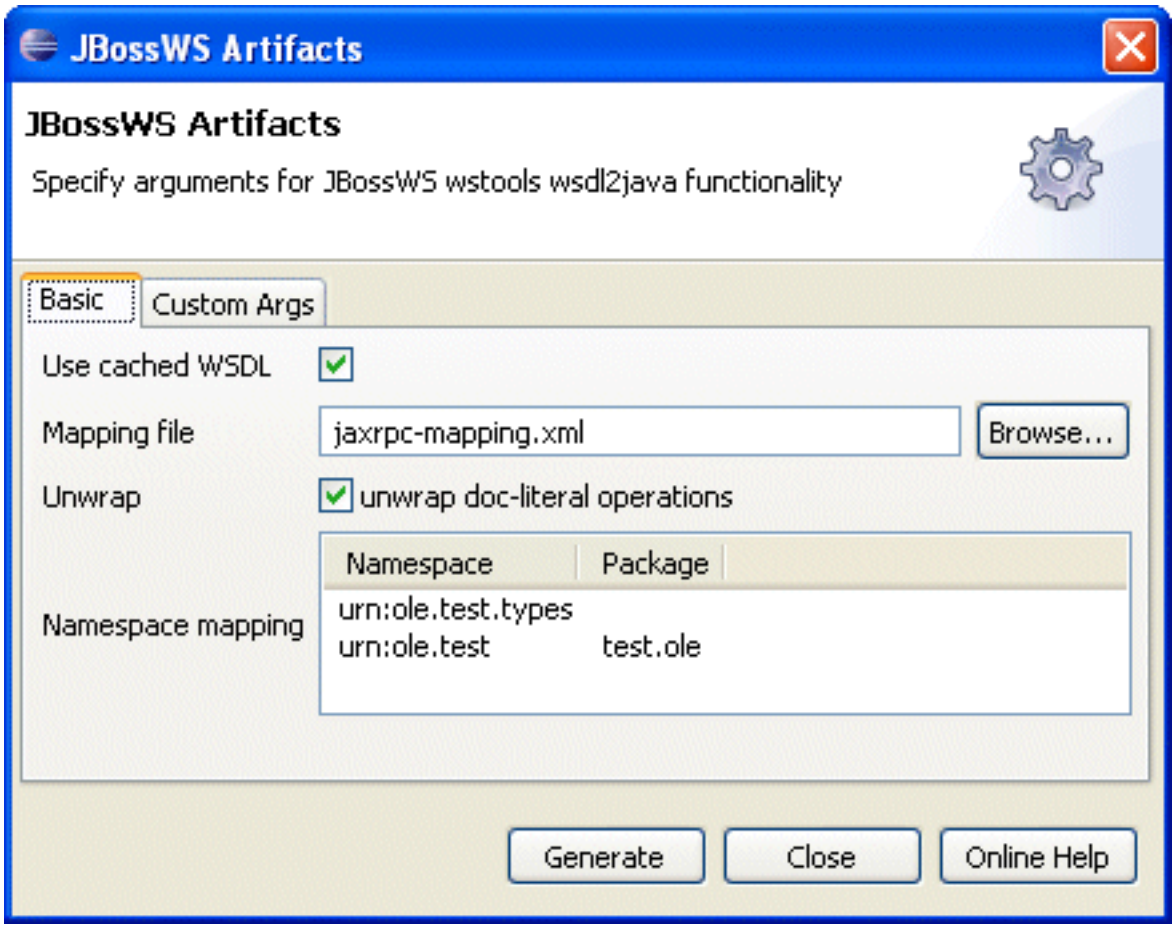
Generating artifacts

Once the WSDL has been imported, select the "Generate - JBossWS artifacts" option from its right-button menu:

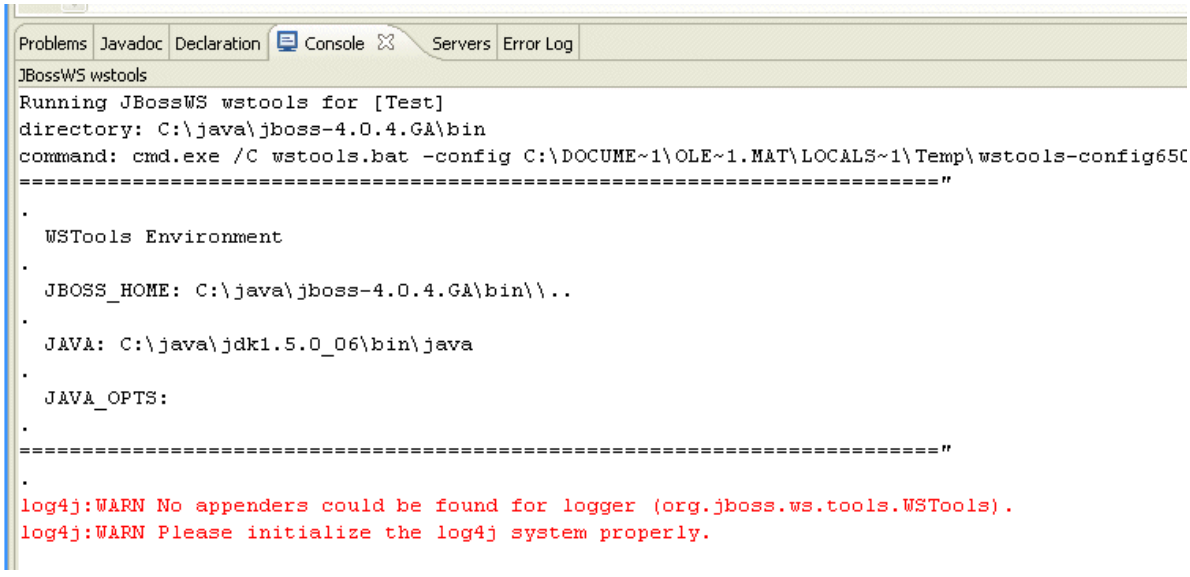


This will open a dialog with the following options:

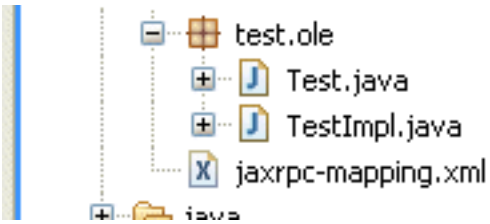
- **Use Cached WSDL** - the plugin caches the WSDL internally when it is added to the project (this can be switched off in the global preferences..). Uncheck this option if you want to generate from the original WSDL location.
- **Mapping file** - the name of a JAX-RPC mapping file to generate for the WSDL (required!)
- **Unwrap** - unwraps doc-literal operations
- **Namespace Mapping** - a table containing all namespaces defined in the imported WSDL, you must specify a destination package for esach namespace.



Once all options have been set as required, select the "Generate" button at the bottom of the dialog which will invoke the WSTools utility as configured. The output will be shown in the console view:



The generated files will be placed in the folder specified under project properties. Currently, the JBossWS plugin lacks support for automatically adding service-refs to compontents that are to consume the service, this must be performed manually as described in the JBossWS User Guide



Next: Implementing a Web Service

1.12.5.5 Implementing a Web Service

Implementing Web Services

Implementing or consuming a Web Service refers to the process of starting with an existing WSDL contract and then creating the java classes that either fulfill or consume this contract. The WSDL can either be remotely available over http or a local file in the current project. In either case, the process consists of 4 steps:

1. Adding the WSDL to the "JBossWS Web Services" node
2. Generating a server-side interfaces classes for implementating the contract
3. Implementing the generated interfaces
4. Publishing the implemented interface as a Web Service, optionally using the original WSDL and mapping file

Step 1 and 2 above are the same as described under Consuming Web Services , please refer to that document for these. Be sure to generate the required JAX-RPC mapping file as it is reused in the publish process.

Implementing the interface

When generating artifacts as described under Consuming Web Services , a java Interface will be generated corresponding to the imported interface. For example when generating for the freely available CurrencyConverter available at http://www.webservicex.net/CurrencyConvertor.asmx?WSDL, the following interface is generated for the services SOAP Binding:

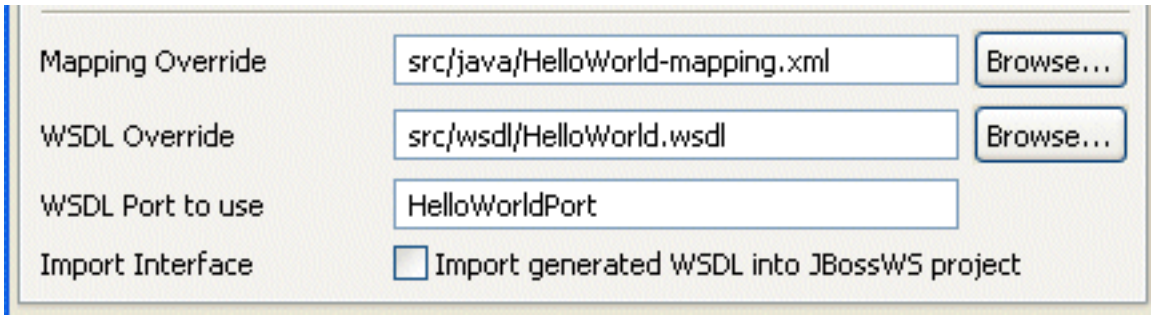
```
/*
 * JBossWS WS-Tools Generated Source
 *
 * Generation Date: Sat Sep 30 15:27:28 CEST 2006
 *
 * This generated source code represents a derivative work of the input to
 * the generator that produced it. Consult the input for the copyright and
 * terms of use that apply to this source code.
 */
package test;

public interface CurrencyConvertorSoap extends java.rmi.Remote {

        public test.ConversionRateResponse conversionRate(
                        test.ConversionRate conversionRate) throws
java.rmi.RemoteException;
}
```

Now implement this interface in a java class:

```
package test.impl;

import java.rmi.RemoteException;

import test.ConversionRate;
import test.ConversionRateResponse;
import test.CurrencyConvertorSoap;

public class CurrencyConverterSoapImpl implements CurrencyConvertorSoap {

        public ConversionRateResponse conversionRate(ConversionRate conversionRate)
                        throws RemoteException {
                return new ConversionRateResponse(0.0);
        }
}
```

Publishing the implementation

Once implemented, publish the Web Service as described in the Publish document. Since you have already generated a mapping file during the "consume" process and the WSDL is publically available, you should specify to reuse the created mapping file and uncheck the option to import the WSDL into the JBossWS Project:



If you have the WSDL locally, you can select to use it instead and must also specify the wsdl port to use.

Next: Generating Web Service Annotations

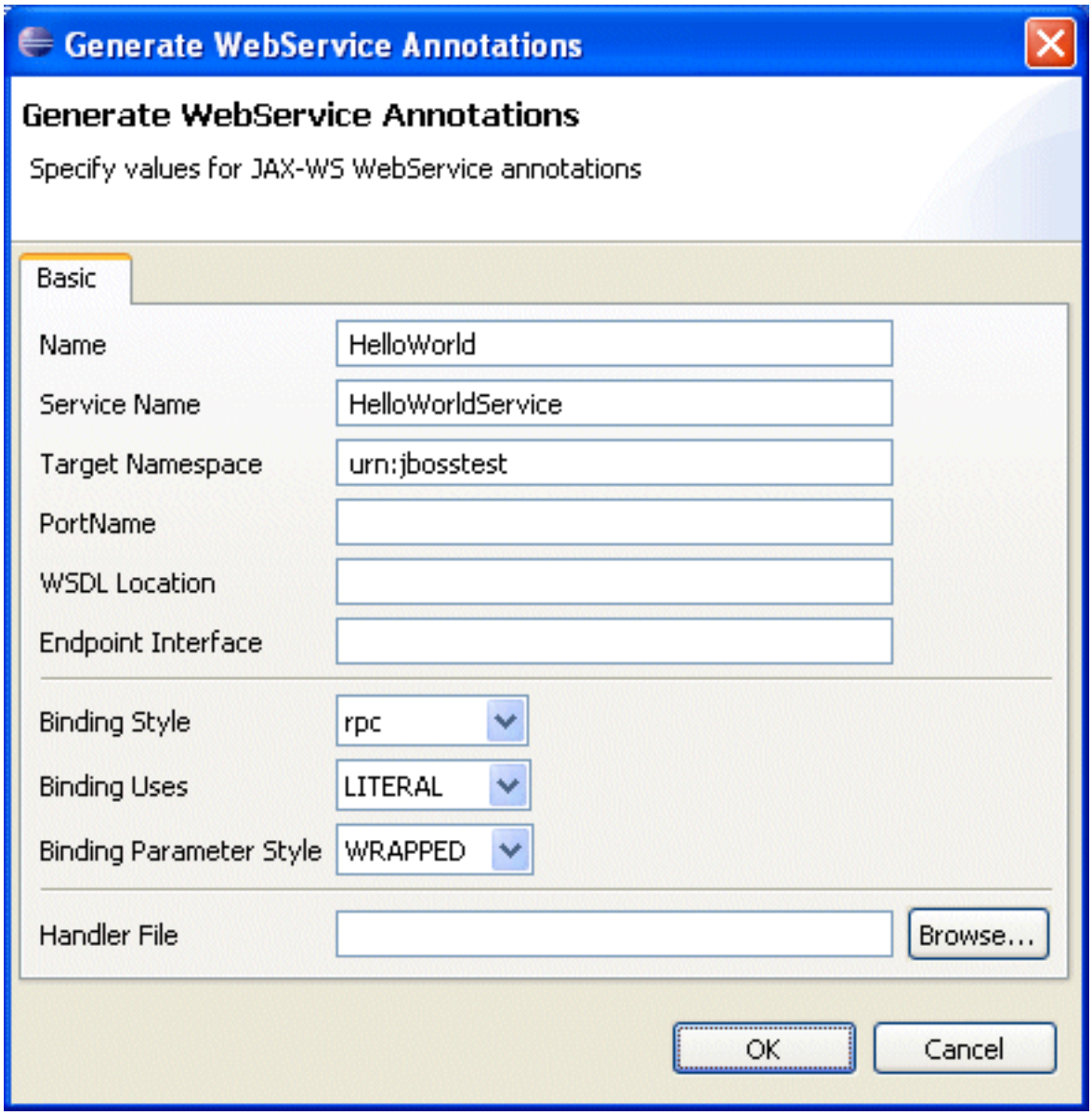
1.12.5.6 Web Service Annotations

Web Service Annotations

The JBossWS plugin contains functionality for easily adding JSR-181 Web Service annotations to existing java code.

Start by enabling the JBossWS nature and by selecting the "JBossWS - Add Web Service Annotations" option from the desired java classes right-button menu:

This will open the following dialog:



The contained options map directly to the service/binding level annotations available in the JSR-181 specification. Once the desired options have been set, select the Generate option which will add the corresponding annotations to the underlying java file, for example:

```
package ole.test;

import java.util.Arrays;

@javax.jws.soap.SOAPBinding(parameterStyle =
javax.jws.soap.SOAPBinding.ParameterStyle.BARE)
@javax.jws.WebService(name="TestImpl", targetNamespace="urn:ole.test",
serviceName="TestImplService")
class TestImpl implements Test
{
        @javax.jws.WebMethod()
        public String toUpperCase(String str)
        {
```

```
                return str.toUpperCase();
        }

        @javax.jws.WebMethod()
        public String toLowerCase(String str)
        {
                return str.toLowerCase();
        }
}
```

Currently, the JBossWS plugin does not support configuring annotations on a method level, standard @WebMethod annotations are added to all methods.

Deployment

Deploy the annotated class in a standard jar archive using the JBossIDE packaging, standard Eclipse/WTP packaging features, or any other desired procedure (ANT, Maven, etc..).

Next: Working with JBoss WS

1.13 Keyboard shortcuts

Keyboard Shortcuts

soapUI supports a number of keyboard shortcuts in order to make your life easier.

Global Actions

The following shortcuts are available for most items in soapUI:

Keyboard Shortcut	Functionality
F1	Shows online help.
Delete	Deletes the selected item.
F2	Renames the selected item.
F9	Copies/Clones the selected item.
Enter	Show the associated editor.

Main Menu Actions

The following shortcuts are available for main-menu actions;

Keyboard Shortcut	Functionality
Ctrl N	Creates a new soapUI project in the workspace.
Ctrl I	Imports an existing project into the workspace.
Ctrl Tab	Select next. Selects next window in the soapUI desktop.
Ctrl Alt P	Preferences.
Ctrl Q	Saves the entire workspace and exits soapUI.
Ctrl Alt Q	Exits soapUI without saving anything.

Desktop Actions

The following shortcuts are availabel for working with the soapUI desktop;

Keyboard Shortcut	Functionality
Ctrl F4	Close Current. Closes the active window in the soapUI desktop
Ctrl Alt O	Close Others. Closes all windows but the active window in the soapUI desktop.
Ctrl Alt L	Close All. Closes all windows in the soapUI desktop.
Ctrl Tab	Select next. Selects next window in the soapUI desktop.

Project Actions

The following actions are available when an Project node is selected in the Navigator

Keyboard Shortcut	Functionality
Ctrl U	Adds a new WSDL from an URL
Ctrl F	Adds a new WSDL from a local File
Ctrl T	Creates a new TestSuite
Ctrl S	Saves the project
Ctrl Shift S	Prompts to save the project to a new file

Interface Actions

The following actions are available when an Interface node is selected in the Navigator

Keyboard Shortcut	Functionality
Ctrl E	Shows the Interface Endpoints dialog.
Ctrl W	Validate the interface with the WS-I validation tools
Ctrl P	Saves the entire WSDL and included/imported files to a local directory
F5	Reloads the definition for this interface and its operations

Operation Actions

The following actions are available when an Operation node is selected in the Navigator

Keyboard Shortcut	Functionality
Ctrl N	Creates a new request for the Operation.

Request Actions

The following actions are available when an Request node is selected in the Navigator

Keyboard Shortcut	Functionality
Ctrl Alt A	Adds the request to a TestCase

Working with Requests and Responses

The following shortcuts are general shortcuts when working in the Request Editor;

Keyboard Shortcut	Functionality
Alt Enter	Submits the request to the specified endpoint.
Alt X	Cancels a running submit.
Alt left/right arrow	Move between element values
Alt O	Changes the orientation of the request/response splitter
Ctrl Alt Tab	Shift focus between request/response area.
Alt V	Validate the contained messages' XML and against its operations message definition.
Alt W (in the response editor)	Validate the current request/response messages against the WS-I Basic Profile (requires a response).
Ctrl D	Delete line. Deletes the current Line in the XML Editor.
F3	Shows the Find/Replace Dialog.
Alt F	Pretty-Prints the current XML
Ctrl S	Prompts to save the contents of the editor to a file.

TestSuite Actions

The following actions are available when a TestSuite node is selected in the Navigator

Keyboard Shortcut	Functionality
Ctrl N	Creates a new TestCase.

TestCase Actions

The following actions are available when a TestCase node is selected in the Navigator

Keyboard Shortcut	Functionality
Ctrl N	Creates a new LoadTest.
Ctrl Shift O	Shows the LoadTest Options.

Working with TestCases

The following shortcuts are general shortcuts when working in the TestCase Editor;

Keyboard Shortcut	Functionality
Ctrl Up/Down	Move a TestCase up or down in the TestCase Editor.

Working with Groovy Scripts

The following shortcuts are general shortcuts when working in the Groovy Editor;

Keyboard Shortcut	Functionality
Ctrl Enter	Executes the current groovy script.

Back to the Overview