10

Must have tips

for Testing

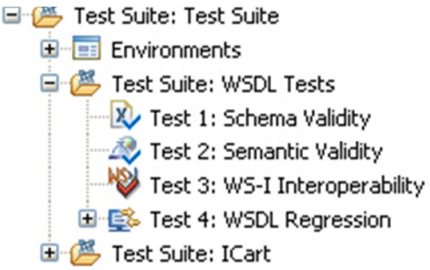
Web Services

Table of Contents

Check the WSDL is correct

Web services are commonly defined within either a WSDL or an XSD file. These files will define the format of the requests and responses. It is critical therefore that tests should be performed on these files to ensure they are correct and conform to company standards. It is suggested the following types of tests are undertaken.

Schema tests - WSDL & XSD files will have schema definitions that the XML must conform to. Failure to comply with these definitions will could mean WS calls have unpredictable results.



Semantic tests – It is common practise to 'include' other files within a WSDL or XSD file. These may define operations common to other applications. We need to test that these 'import' statements can find the required files. Failure to do so will render any functional tests invalid. This is particularly important when your project will be copied across different staging environments before going live. You need to test at every stage that all the relevant files are copied over.

Regression – Has the file defining the service been amended since running the last set of tests? If so, what has changed, and does it affect your functional tests?

WS-I Interoperability – The Web Services Interoperability Organization (WS-I) is an open industry organization chartered to establish Best Practices for Web services interoperability, for selected groups of Web services standards, across platforms, operating systems and programming languages. Compliance to these standards will help ensure easier compatibility with third parties that

may interface to your service either now or in the future. The definition of your web services will need to be checked against these standards, as well as the messaging for all functional tests.

Find out if your company policy is to comply with WSI 1.1, 1.2, or 2.0.

Internal Policies – has your company laid down internal governance policies which the messaging must comply to? An example of this might be a standard naming convention for field names for a house address.

If you are not sure what a WSDL file looks like then example WSDL files can be viewed here:

http://soaptest.parasoft.com/store-01.wsdl

http://soap.amazon.com/schemas2/AmazonWebServices.wsdl

Doing it with SOATest

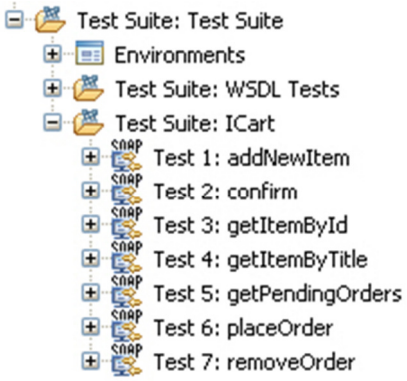
These tests are created automatically within SOAtest, and take seconds to run!

Functional Unit Tests

A web service will comprise of one or many operations. For example a simple service for a book store may define search, order, payment and cancel operations. Each of these operations should initially be tested in isolation, before integrating them into scenario based end to end tests.

Each operation should have the following tests:

- Each field accepts 'good' data, your go right path.
- Check how each field handles negative data. This may be incorrect field length, incorrect format, e.g. String instead of Integer data, etc.
- WS-I Interoperability tests for the go right path.
- The response message must comply with the WSDL/XSD definitions.
- And of course, did you get the response you were expecting.
- Run each test with a range of data to ensure good coverage.



What constitutes a correct response? The ultimate correct response is where the application returns the data you were expecting - the answer to your request, however there is much more to look out for.

Don't forget there may be more than one correct response, e.g. when our search found a book that matches, more than one book or no books.

How is negative data handled? The application should return a specific error message e.g.

> ERROR – Invalid data, please use a String.

We should not get a system generated error message that is thrown by default due to the application not handling our data properly. e.g.

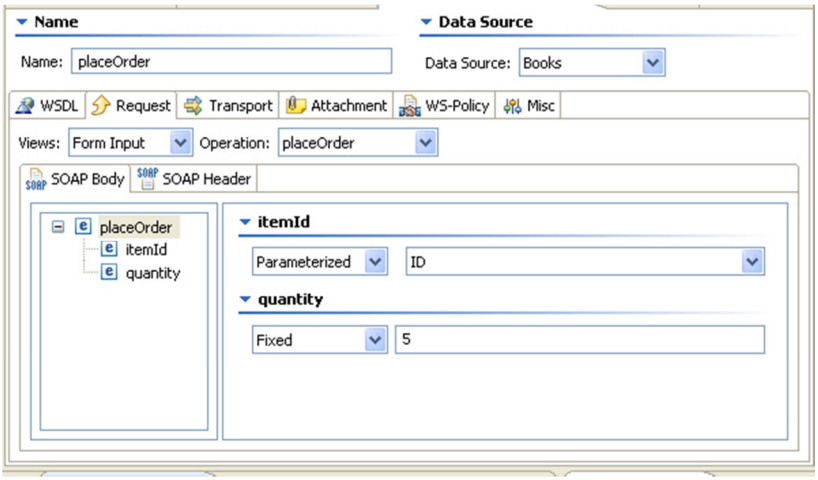
```
Exception in thread "main" java.lang.NumberFormatException: For input string: ""
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
at java.lang.Integer.parseInt(Integer.java:447)
at java.lang.Integer.parseInt(Integer.java:476)
at numconv.main(numconv.java:24)
```

This may potentially provide information to a hacker that could compromise security.

What happens when another component is not present? Take away the database and do you get a clear error message, or an application failure?

Doing it with SOATest

SOAtest can generate functional tests from a WSDL, WADL or XSD files, from traffic logs, by recording actions on a web browser.



Scenario Testing

Now each operation has been tested in isolation we need to ensure they all work together. Taking our bookstore example, the customer may search for a book, retrieve the ID number of the book they are after, and use this to place an order. They then require the order number to make a payment.

Now of course if we are doing this manually then we just need to make a note of the values returned in the 'id' and 'order' number fields, and use these as input into the following request.

E.g. This is part of the response to our search for a book.

```
<i xsi:type="n3:Book">
      <id xsi:type="xsd:int">2</id>
      <title
         xsi:type="xsd:string">Java How to Program (4th Edition)
      </title>
      <quantity_in_stock
        ..
```

And here is part of our next request message placing an order:

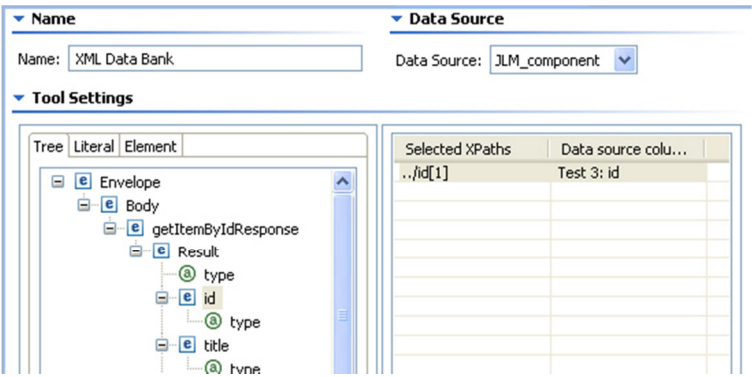
```
<SOAP-ENV:Body>
  <placeOrder xmlns="http://www.parasoft.com/wsdl/store-01/">
       <itemId>2</itemId>
       <quantity>1</quantity>
  </placeOrder>
```

Of course this is a very simple example, and the complexity of some services only lends weight to the argument for automating this transfer of data between tests.

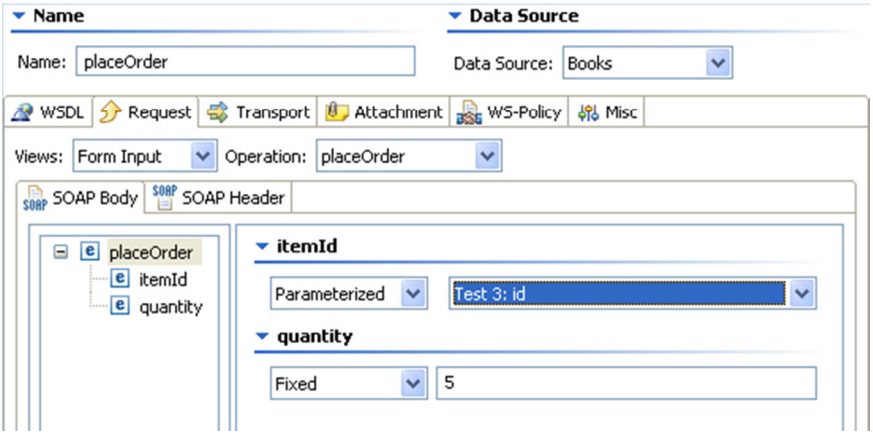
Scenarios may be further complicated by additional validation steps. You may want to check that the scenario just tested did indeed update the database with the relevant details as intended. Perhaps there is a

manual intervention step that requires a ‘non web service’ step to be actioned. An example of this might be a loan approval process that in some circumstances may require a loan to be manually approved on a web page before our scenario can complete.

Doing it with SOATest



Results can be easily stored to be reused in subsequent tests.



Regression Testing

Once a test, either a Unit or a Scenario test, has been created, it can and should be added to a daily regression test suite.

Basic Regression tests

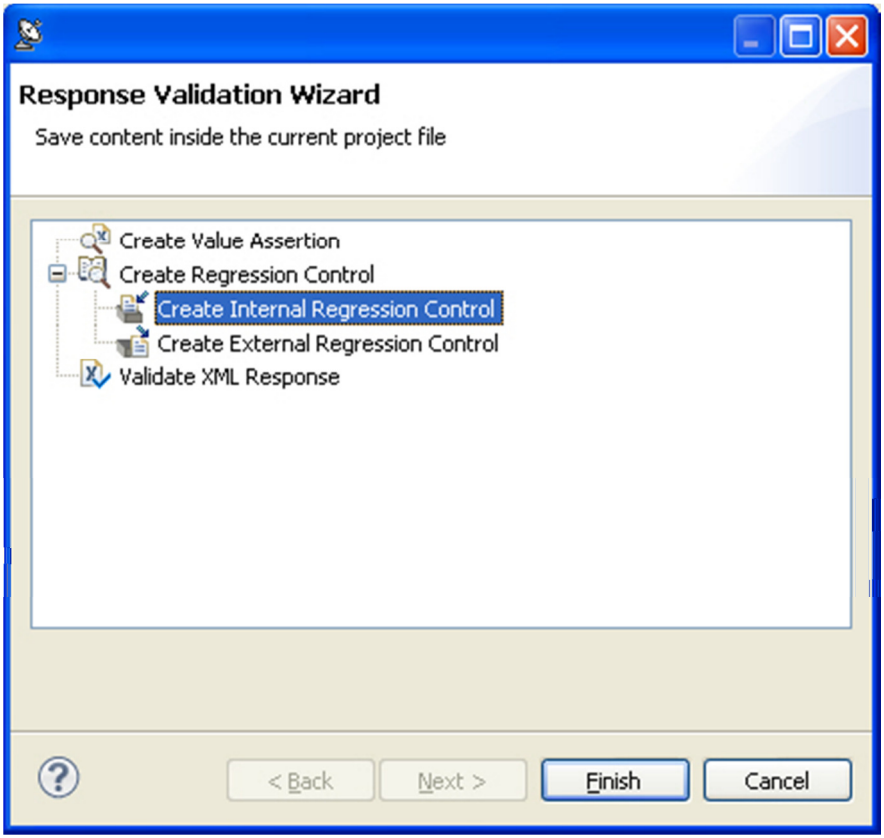
> Check that the XML returned for a given request matches that returned by the previous test. Individual fields can be checked, or the entire XML payload. When checking the entire XML message, it may be necessary to ignore specific fields that are constantly changing but are irrelevant to the success of the test, e.g. timestamps.

Dynamic

> Using a data source that has fields for both the request and response, you can check that the actual response data matches that expected. Regression testing can then be managed from a spreadsheet rather than from within the test tool itself.

Doing it with SOATest

A basic regression test that compares the current and previous responses can be created with a couple of clicks

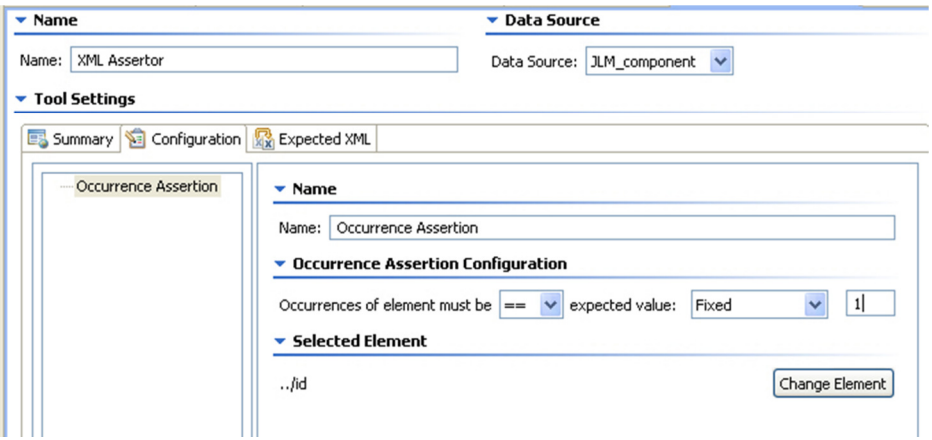


Assertions

Basically an assertion provides an automated way of alerting you to a problem with the response, without having to view the response manually. It is a natural precursor to automating the test for regression purposes.

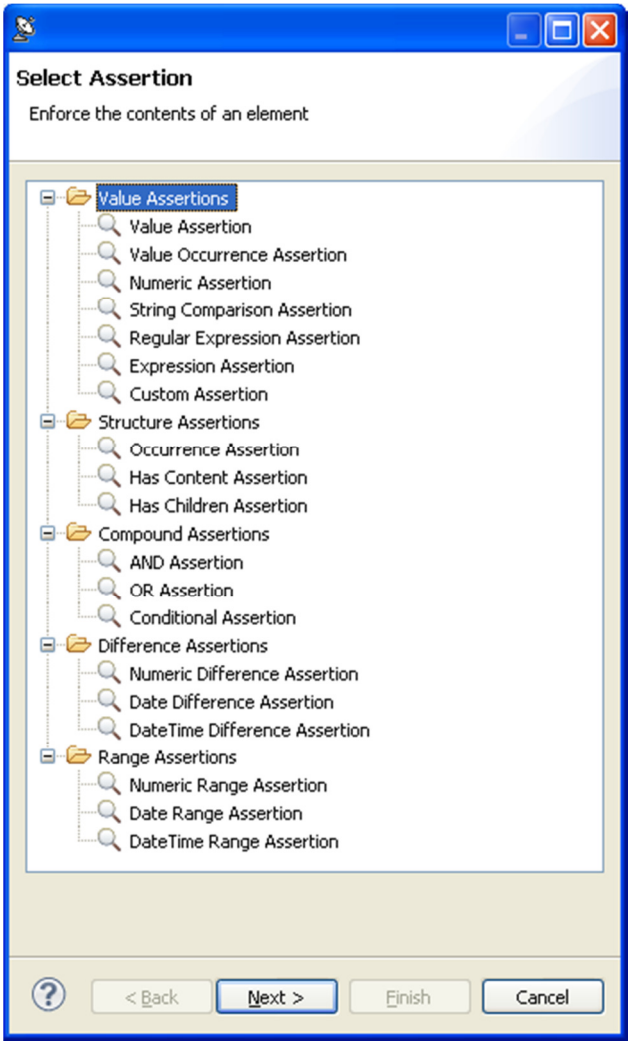
Perhaps you may want to assert something more complex than a simple regression control allows, such as how many instances of a particular element exist, or whether a date or integer field is within a certain range.

Once these assertions are set up, then they will alert you to non-conformity whether the test is run manually or automatically. Some typical assertions can be found opposite.



Doing it with SOATest

SOAtest has many different type of Assertion that can be configured for your test. Additionally Custom assertions can be created using scripting.



Negative Testing

Are you testing the GUI, or the back end application? This is an interesting point to make, as you cannot test both properly at the same time. Negative testing is one very important area where using the browser based GUI to test the BEA cannot work, as you will be testing that the GUI does not allow incorrect field data rather than how the application handles incorrect field data.

We have to bear in mind that at some point in time the web service may be used by multiple different input methods. So whilst our browser based GUI is the only interface at this moment in time, this could change to include mobile phones, thick clients, other applications etc. Therefore we need to test what will happen if we should send an incorrect date format or perhaps a String in an Integer field. The GUI is (or should) be designed to constrain the user, and therefore will constrain the tester, and is the primary reason to use a test tool rather than the GUI for message layer testing. The next user interface may not be as critical in what it accepts as this one!

Another type of negative test is where we deliberately send a request that will cause a failure message to be returned. Obviously we want to check that the 'go wrong' path works as expected as well!

Doing it with SOATest

As with other GUI's, SOAtest will do its best to enforce the schema type of each field. For the purposes of negative testing though, it is possible to turn off this enforcement, allowing the user to deliberately send incorrectly formatted data e.g. a String in an integer field.

When addressing the second example of a negative test where we are checking that a correct error message comes back, within SOAtest it is possible to accept any response as being correct, even if that is a HTTP error code. Simply apply the regression control, and your test will only accept that error message as being the correct response.

Security Testing

This is not some dark art that has to be left to the specialists. Any response to a test that you make should be checked to see if it contains any information you are not meant to receive. Examples of what to look out for include:

- Application exception messages
- 'Stack Traces' i.e. information related to lines of code that had an error.
- Information related to a database query.
- Error messages that specifically identify which component – userid or password – was incorrect when you tried to log on.

Check that for systems that require a user to log on, can you perform tests with having done so? Do consecutive messages require a valid session ID? What happens if you do not provide this?

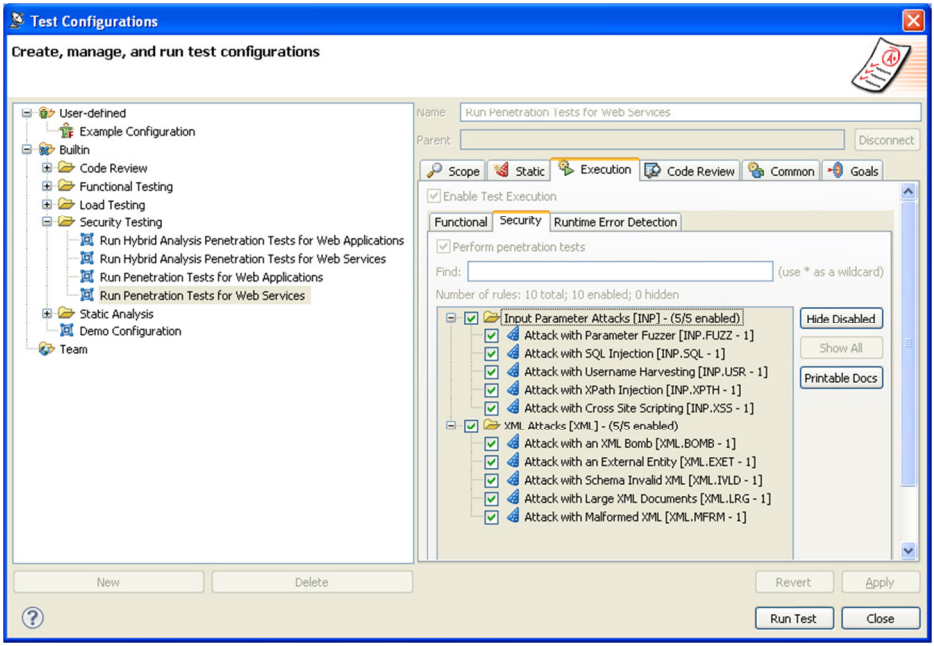
Are any elements of the message encrypted? Provide negative tests that prove that the encryption process is working, i.e. invalid security keys. How does the system handle invalid security keys – do you get a clean error message, or an application crash?

Basic penetration tests should be applied to every field in every message sent. How does the application respond to SQL Injection attacks that attempt to bypass log on steps? Does an XML bomb crash the application? Whilst I would not suggest this is a manual task, there are

tools available to do these tests for you. Again, early detection of these issues is vital to getting your project delivered on time and to budget. The security consultants should only be verifying that the application passes all their tests, they should not be finding anything! If they do find something, then create a test that checks for this failure in the future.

Doing it with SOATest

Security Testing within SOAtest is as simple as running your existing test scenario with the respective test configuration. This can be customised with attacks such as Parameter Fuzzing, SQL Injection, Username Harvesting, XPath Injection, and Cross Site Scripting.



Performance Testing

This always seems to be another of those tasks that ends up on the list to be done once the application is finished and tested - a final tick in the box before ‘go live’. But if this is a true test, are you expecting to find issues? If so, the application or infrastructure will need tweaks to improve performance, leading on to regression testing and therefore delays.

For the purpose of this article I am going to suggest a two stage performance testing system, the first step being to create ‘low volume’ load tests that are there to act as a warning flag. What defines ‘low volume’ will depend on your project, but I would suggest something in the region of 100 Virtual Users. These tests will be there to highlight areas of immediate concern, and can be run by the ordinary test team as opposed to the performance specialists. The second phase is in the realms of the specialist, and is outside the scope of this document.

Performance tests can be set up on a unit level just the same as a functional test can. You do not need to wait until all the pieces are in place to allow you to do a performance test end-to-end. Once you have a working functional test, create a load test for it. Record the metrics for this test so you can see if further development degrades the performance of this function. Forewarned is forearmed as they say, if you can show a degradation of this service over time, the development team will more easily be able to tie that degradation to the work they have done and isolate if this was acceptable or not.

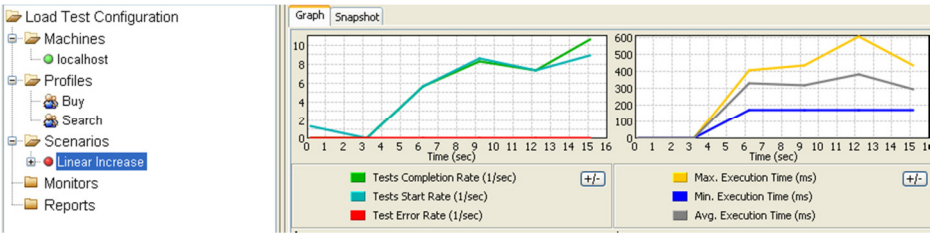
As multiple items of functionality become available and you start on the scenario tests, then add these to your performance tests too. And when you have multiple scenarios, we can start to create a more complex load testing environment where multiple scenarios are run at the same time, in the ratios that they might occur in a real world situation. This is of course where we start to border on the 2nd phase of load testing.

Doing it with SOATest

SOAtest Server edition comes with the associated license for Parasoft LoadTest. This will allow you to take existing SOAtest test suites and run them in a performance situation.

Tests may be distributed built up into 'Virtual User' scenarios that mimic real life expected behaviour, and then run simultaneously in rations reflecting the live environment.

The usual reports are generated at the end, including 'nth Percentile, Quality of Service, Metrics etc.



No re-scripting is required, saving time and money. Load testing can be done against web service, web pages and Java applications.

Virtualisation

Have you ever experienced any of the following?

- You can only test ‘out of hours’ as testing affects live systems
- Testing has to be undertaken on live systems due to the complex nature, or financial cost of running, a test system
- You want to test scenario A, but your colleague is testing scenario B, and you have to wait until they have finished to prevent contamination of the results

If so, then you need to provide Virtualised versions of your test/live environment. These will replicate the original, but at a fraction of the cost, and allow you to test systems independently of live systems, live data and even your colleagues.

Real system behaviour is captured—using monitors to record live transaction details on the system under test; by analyzing transaction logs; or by modelling behaviour from a simple interface.

The virtualized asset's behaviour can be fine-tuned, including performance, data source usage, and conditional response criteria. The environment is then provisioned for simplified uniform access across teams & business partners.

The virtualized asset can now be called for unit, functional and performance tests—both automated and manual. It can be leveraged by any test environment, including Parasoft Test, HP Quality Center suite, IBM Rational Quality Management suite, Oracle ATS, and more.

Doing it with SOATest

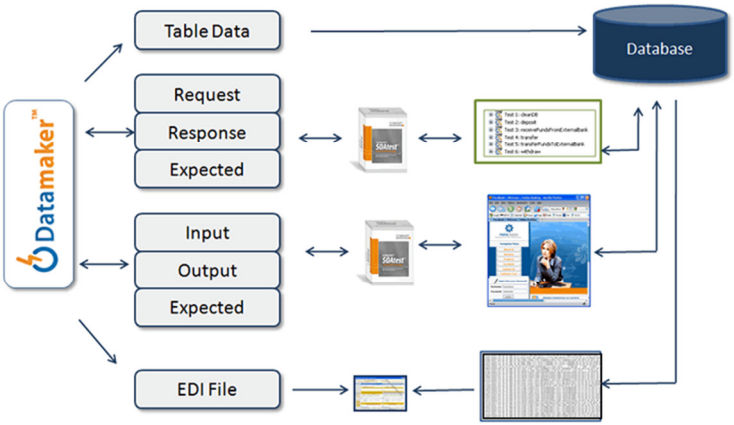
- Rapidly create virtual assets by recording existing application behaviour then playing it back with dynamic values and context awareness
- Easily update virtualized application behavior models through an intuitive graphical interface
- Easily manage virtualization data with zero learning curve
- Perform load tests without impacting existing systems or operations
- Scale virtualized assets to support large-scale, high-throughput load and performance tests
- Model real, complex interdependent environment scenarios
- Rapidly model application behavior, even for dependencies that do not yet exist
- Visually model various message formats such as XML, JSON, and various legacy, financial, healthcare, and other domain-specific formats
- Easily configure various error and failure conditions that are difficult to reproduce or replicate with real systems
- Host virtual assets in a cloud or virtualize applications hosted in a cloud
- Easily manage interdependent system connections
- Automate activities that would otherwise require significant time and resources from operations and infrastructure specialists
- Automate workflows that involve multiple stakeholders, facilitating collaboration between various system owners, administrators, developers, and testers
- Complement existing hardware virtualization infrastructure

Test Data

High quality and reliable test data is a must in today's test and development environments. In order to successfully deploy new applications, IT organizations must go about their testing using data which is representative, both functionally and technically, of their live systems. This is incredibly tricky if done manually

Test data is obviously an important part of any serious testing. How to generate meaningful test data can also give serious headaches. Questions to consider include:

- There is a need to create as much data as will allow you to cover as many combinations of paths through your application as possible, both positive and negative call flows.
- You should be able to version control your data, such that you can roll the systems back to a consistent state every time.
- Does your data include the expected response data so that you can compare the response for validation purposes?
- Are you forced to use Live data? If so, are you breaking data security rules by doing so? If this is the only way to test a system consider masking sensitive data so that testers do not get a complete picture of a customer record.



Doing it with SOATest

Test data is consumed by SOAtest, however SOAtest cannot generate it for you. To do this effectively Parasoft Recommend SOA Data Pro by Grid-Tools. This is used by Corporations globally to efficiently generate their test data.

http://www.grid-tools.com/products/SOA_data_pro.php

SOA Data Pro may be integrated to SOAtest so that the data can be used directly for parameterising message layer tests such as SOA, HTML, JMS etc.

Independently of SOA Data Pro, test data can be held in a medium of your choice, including Excel, CSV files and Databases. Once defined as a SOAtest Datasource, the data columns are available to use within Parameterised tests.

Supposing you have a large complex message with potentially hundreds of fields! SOAtest will allow you to create a sample CSV data source with columns that match your fields, and will then map that data source automatically to those fields for you.

The test data can include both the data for the request, and also the expected response, allowing you to check the actual response against your data.

About Parasoft® SOAtest™

Parasoft SOAtest automates web application testing, message/protocol testing, cloud testing and security testing. Parasoft SOAtest and Parasoft Load Test (packaged together) ensure secure, reliable, compliant business processes and seamlessly integrate with Parasoft language products (e.g., Parasoft Jtest) to help teams prevent and detect application-layer defects from the start of the SDLC. Moreover, Parasoft SOAtest integrates with Parasoft Virtualize to provide comprehensive access to traditionally difficult or expensive to access development and test environments.

Parasoft SOAtest provides is an integrated solution for:

End-to-end testing: To continuously validate all critical aspects of complex transactions, which may extend beyond the message layer through a web interface, ESBs, databases, and everything in between.

Environment management: To reduce the complexity of testing in today's heterogeneous environments–with limited visibility/control of distributed components or vendor-specific technologies.

Quality governance: To continuously measure how each service conforms to the often dynamic expectations defined by both your own organization and your partners.

Process visibility and control: To establish a sustainable workflow that helps the entire team efficiently develop, share, and manage the evolution of quality assets throughout the lifecycle.

Parasoft Load Test allows you to load test your SOAtest tests to verify functionality and performance under load. Support is also provided for load testing non-Parasoft components such as JUnits or lightweight socket-based components, and for detecting concurrency issues.

Parasoft's customers, including 58% of the Fortune 500, rely on SOAtest and Load Test for:

- Ensuring the reliability, security, and compliance of SOA and web applications.
- Reducing the time and effort required to construct and maintain automated tests
- Automatically and continuously validating complex business scenarios
- Facilitating testing in incomplete and/or evolving environments
- Validating performance and functionality expectations under load
- Rapidly diagnosing problems directly from the test environment

Contact Details

Parasoft UK Ltd
The Lansdowne Building,
2 Lansdowne Road,
Croydon.
CR9 2ER.

+44 (0) 208 263 6005
sales@parasoft-uk.com

For a Free evaluation please visit the SOAtest home page

http://www.parasoft.com/jsp/products/soatest.jsp

and click on the 

button.