_Configuring_for_Different_Environments

Understanding SOAtest Environments

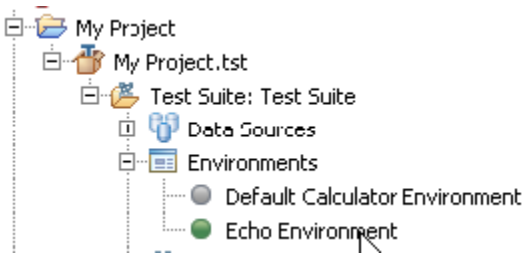
You may want to run the same test suite against other target systems (different testing environments, production environments, etc.). You can use SOAtest Environments to decouple configuration settings from your test data. Once you have configured environments for your test suites, running tests against another systems is as easy as clicking a button.

An environment is a collection of variables that can be referenced in your SOAtest test suite. When running a test, SOAtest will substitute the names of variables with the values assigned to those variables in the active environment. By changing which Environment is active, you can quickly and easily change which values SOAtest uses.

Environments are automatically defined during generation from a WSDL, by recording from a browser, or through another process. In addition, you can manually define one as described below.

Manually Defining an Environment in SOAtest

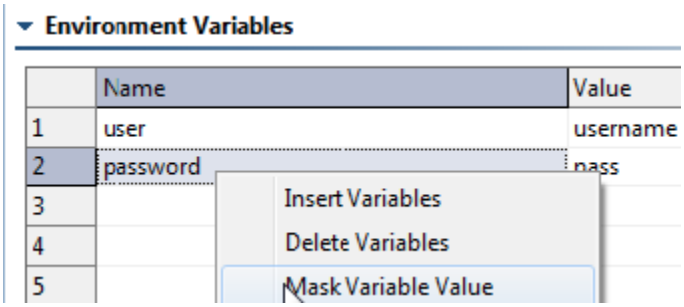
Creating and switching environments is done through the **Environments** branch of the test suite's Test Case Explorer node.



The Environments branch is created by default when a new test suite is created.

To add a new environment:

1. Right-click the **Environments** node, then choose **New Environment**.
2. In the configuration panel that opens on the right, use the available controls to define environment variables. Note that you can mask a value by right-clicking and choosing **Mask Variable Value**.



Using Environment Variables in Tests and Tools

Environment variables can be accessed in test or tool configuration fields using a special syntax. To reference a variable, enclose the variable name in the following character sequence: `${env_name}`.

For example, if you have a variable named HOST, you would reference the variable in a field by typing: `${HOST}`. Variable references may appear anywhere within a field.

In the data source editor, use the `soa_env` prefix to reference environment variables. For example, `${soa_env:Variable}/calc_values.xlsx`

Variable references may appear anywhere within a field. For example let's say your environment contains the variables `HOST = localhost` and `PORT = 8080`, and you have an **Endpoint** field in a SOAP Client containing: "`http://${HOST}:${PORT}/Service`". Upon running the test, the value used for the endpoint will be "`http://localhost:8080/Service`".

You can access Environment Variable values from a Extension Tool/Script through the Extensibility API. ExtensionToolContext now has a method called "g`etEnvironmentVariableValue(String)`" which will lookup and return the current value of an Environment variable. This will allow you to use the value within your scripts.

> **Note**
>
> If your test case or tool requires the character sequence `${}`, you can escape the sequence by adding a backslash. For example, if SOAtest or Virtualize encounters the value "`\${HOST}`" it will use the value "`${HOST}`" and will not try to resolve the variable. Also note that environment variable names are case sensitive.

Changing Environments from the GUI

To change what environment is active:

- Right-click the node representing the environment you want to make active, then choose **Set as Active Environment**.

Changing Environments from the Command Line

In addition to being able to select the active Environment from the GUI, you can also switch the active environment from the command line, using the `-environment` option. See Testing from the Command Line Interface - soatestcli for details.

Overriding Environments Using Test Configurations

To set a Test Configuration to always use a specific environment for test execution (regardless of what environment is active in the Test Case Explorer):

- Set the Test Configuration's **Override default Environment during Test Execution** option in the Execution tab. See Execution Tab Settings: Defining How Tests are Executed for details.

Sample Usage

Let's say you are developing a service on your local machine. Once the code works locally, you commit the code to source control and start a build process targeted to a staging server. You want to create a suite of tests that will test against both your local machine and the staging server with minimal modification. This is the perfect case for using SOAtest Environments.

Begin by creating a localhost environment for your new .tst file. The SOAtest New .tst Wizard will get you started with a few basic environment variables. If your WSDL varies across the two machines, you can decompose the WSDL URI into variables. If the WSDL location is constant across the machines, you can alternatively decompose the endpoint URI.

The next step is to identify other machine-specific settings in your projects and to create environment variables for them. For example, let's say you have a JMS step in your test suite and you need to send the message to different queues depending on whether it's a localhost test or a staging test. First, create a new environment variable, let's call it "`JMS_REPLY_QUEUE`". Next, revisit your SOAP Client, go to the JMS settings, and enter `${JMS_REPLY_QUEUE}` as the value for the **JMSReplyTo** field.

Once you have created your localhost environment, you can copy and paste the existing environment and rename the new environment to "Staging Environment". Then, simply modify the values of each of the variables so that they reflect the settings of your staging server.

Once your environments are setup, targeting your tests against the various environments is simply a matter of selecting the active environment. If you want to test against your local machine, you would set your active environment to be "Localhost Environment". This will run your tests using the values defined in the localhost environment. To test against the staging server, set the "Staging Server" environment to active and its values will be used.

Exporting, Importing, and Referencing Environments

You may find that many configuration settings, such as server names and ports, will be common across multiple projects. Rather than duplicating these settings, you can export environment settings to an external file and import or reference the values in other projects.

Exporting Environments

To export an environment:

1. Right-click the node representing the environment you want to export, then choose **Export Environment**.
2. In the file chooser that opens, specify a location for the exported environments file.

The environments configuration will be written in an XML-based text file. If one Environment is selected, a `*.env` file will be created, containing a single environment. If multiple environments are selected, a `*.envs`, or Environment Set, file will be created containing all of the selected environments.

Importing Environments

When you import environments, you are bringing a copy of the values from the external environment file into your project. Further modification to the XML file will not be reflected in your project.

To import an environment:

1. Right-click the **Environments** node, then choose **Import Environment**.
2. In the file chooser that opens, specify the location of the environments file that you want to import.

Referencing Environments

Referencing environments is the most efficient way to share a single environment configuration across multiple projects. Using environment references, you can easily modify the configurations of multiple projects from a single location.

To reference an environment:

1. Right-click the **Environments** node, then choose **Reference Environment**.
2. In the configuration panel that opens on the right, specify the location of the environments file that you want to reference.

Note that when an environment configuration is referenced, you cannot edit the environment variables in the environment directly. However, your project will always use the values reflected in the referenced `*.env` file. Modifying the `*.env` file will propagate changes to all projects that reference it.