SOAtest Test Configuration Settings

This topic describes the settings available for SOAtest Test Configurations.

Sections include:

How to Configure and Share Test Configurations

The general procedures related to configuring and sharing Test Configurations are standardized across the Parasoft Test family, and are discussed in Configuring Test Configurations and Rules for Policies.

Scope Tab Settings: Defining What Code is Tested

For source code static analysis only.

During a test, SOAtest will perform the specified action(s) on all code in the selected resource that satisfies the scope criteria for the selected Test Configuration. By default, SOAtest operates on all files in the selected project or asset. However, you can use the Scope tab to configure restrictions such as:

- Test only files or lines added or modified after a given date.
- Test only files or lines added or modified on the local machine.
- Test only files modified by a specific user.
- Test only files that match certain filter criteria

Note that some file filters and line filters are only applicable if you are working with projects that are under supported source control systems.

The Scope tab has the following settings:

- **File Filters:** Restricts SOAtest from testing files that do not meet the specified timestamp and/or author criteria.
  - **Time options:** Restricts SOAtest from testing files that do not meet the specified timestamp criteria. Available time options include:
    - **No time filters:** Does not filter out any files based on their last modification date.
    - **Test only files added or modified since the cutoff date:** Filters out files that were not added or modified since the cutoff date.
    - **Test only files added or modified in the last n days:** Filters out files that were not added or modified in the specified time period.
    - **Test only files modified between working and ____:** Filters out files that were not modified between the working developer branch (in the workspace) and the specified branch (or the default integration stream detected, if that option is enabled). The stream name can be any stream from the parent's hierarchy of developer working streams. The default integration stream is the parent stream of the developer working stream. For example, if you have a stream hierarchy of `[Main] --- [Integration] --- [Developer]`, then `Integration` is the default integration stream for the `Developer` stream. This is currently supported for SVN, AccuRev, and Clear Case.
    - **Test only files added or modified locally:** Filters out files that were not added or modified on the local machine. This feature only applies to files that are under supported source control systems.
  - **Author options:** Restricts SOAtest from testing files that do not meet the specified author criteria. Available author filter options include:
    - **No author filters:** Does not filter out any files based on their author.
    - **Test only files authored by preferred user:** Filters out any files that were not authored by the specified user (i.e., filters out any files that were authored by another user).
  - **Path options:** Configures SOAtest to filter in or out files that match the specified filter criteria. Use the "accept" filter to specify the types of files you want to include. Use the "reject" filter to specify the types of files you want to exclude. For code review test configurations, these filters will be pre-populated with common filtering options.

Filter Tips and Examples

Tips

- Perl-style expressions can be used.
- The following wildcards are supported:
  - * matches 0 or more characters except '/'.
  - ? matches any single character except '/'.
  - ** matches 0 or more characters, including '/'. This allows you to include path elements.
- The following sample elements are added by default to the Code Review configuration:
  - **/bin/**/*.properties is added to the sample list of rejected wildcards.
  - (.*?/(bin|obj)(/x86|/x64){0,1}/(Debug|Release)/.*?\\.(dll|exe|pdb))$ is added to the sample list of rejected regexps.
- Regular expressions can be used to identify specific differences. For instance, if you want to flag only source code changes that add, remove, or modify TODO tags, you would use the Differences regular expression `.*TODO.*`

Examples

A basic file mask might be:

- *.java, *.xml, *.properties
- *.c, *.cpp, *.h, *.cc, *.hpp, makefile, .project, .classpath
- *.c, *.cpp, *.h, *.cc, *.hpp, makefile, *.sln, *.prj, *.res
- *.cs, *.vb, *.sln, *.prj, *.resx

To include every file whose path has a folder named "bank" or "customer", use:

- **/bank/**, **/customer/**

To include every file whose path has a folder with a name that either starts with "bank", includes "customer", or ends with "invoice", use:

- **/bank*/**, **/*customer*/**, **/*invoice/**

To include every .java file that 1) has name that starts with "Test", and 2) is located in a folder named "security" (which is within the src/test directory of any project), use:

/src/test//security/Test*.java

To include every .cs file that 1) is in the ATM solution, 2) is in the ATMLib project, 3) is within the CompanyTests subfolder, 4) has a name that starts with "Test", use:

ATM/ATMLib/CompanyTests/**/Test*.cs

- **Line filters:** Restricts the lines of code that SOAtest operates on. The file filter is applied first, so code that reaches the line filter must have already passed the file filter. Available line filter options include:
  - **Time options:** Restricts SOAtest from testing lines of code that do not meet the specified timestamp criteria. Available time options include:
    - **No time filters:** Does not filter out any lines of code based on their last modification date.
    - **Test only lines added or modified since the cutoff date:** Filters out lines of code that were not added or modified since the cutoff date. This feature only applies to files that are under supported source control systems.
    - **Test only lines added or modified in the last n days:** Filters out lines of code that were not added or modified in the specified time period.
    - **Test only lines added or modified locally:** Filters out lines of code that were not added or modified on the local machine. This feature only applies to files that are under supported source control systems.
  - **Author options:** Restricts SOAtest from testing lines of code that do not meet the specified author criteria. Available author filter options include:
    - **No author filters:** Does not filter out any lines of code based on their author.
    - **Test only files authored by users:** Filters out any files that were not authored by the specified users. For example, you can use this to focus on files that you—or a selected group of teammates—worked on. To specify multiple users, use a comma-separated list (for example: matt, tom, joe).

Configuring Scope and Authorship

Code authorship information and last modified date is determined in the manner set in the Scope and Authorship preferences page; for details about available settings, see Configuring Task Assignment and Code Authorship Settings.

Static Tab Settings: Defining How Static Analysis is Performed

During a test, SOAtest will perform static analysis based on the parameters defined in the Test Configuration used for that test.

The Static tab has the following settings:

- **Enable Static Analysis:** Determines whether SOAtest performs static analysis, which involves checking whether the selected resources follow the rules that are enabled for this Test Configuration.
- **Limit maximum number of tasks reported per rule to:** Determines whether SOAtest limits the number of violations (tasks) reported for each rule, and—if so—the maximum number of violations per rule that should be reported during a single test. For instance, if you want to see no more than five violations of each rule, set this parameter to 5. The default setting is 1,000.
- **Do not apply suppressions:** Determines whether SOAtest applies the specified suppressions. If suppressions are not applied, SOAtest will report all violations found.
- **Rules tree:** Determines which rules are checked during static analysis. Use the rules tree and related controls to indicate which rules and rule categories you want checked during static analysis.
    - To view a description of a rule, right-click the node that represents that rule, then choose **View Rule Documentation** from the shortcut menu.
    - To view a description of a rule category, right-click the node that represents that rule category, then choose **View Category Documentation** from the shortcut menu.
    - To enable or disable all rules in a specific rule category or certain types of rules within a specific rule category, right-click the category node, then choose **Enable Rules>[desired option]** or **Disable Rules> [desired option]**.
    - To search for a rule, click the **Find** button, then use that dialog to search for the rule.
    - To hide the rules that are not enabled, click the **Hide Disabled** button. If you later want all rules displayed, click **Show All**.

Tips

- The number next to each rule ID indicates the rule's severity level. The severity level indicates the chance that a violation of the rule will cause a serious construction defect (a coding construct that causes application problems such as slow performance, security vulnerabilities, and so on). Possible severity levels (listed from most severe to least severe) are:
    - Highest - Level 1
    - High - Level 2
    - Medium - Level 3
    - Low - Level 4
    - Lowest - Level 5
- To learn about the rules that are included with SOAtest, choose **Help> Help Contents**, then open the SOAtest **Static Analysis Rules** book, then browse the available rule description files.
- To generate a printable list of all rules that a given Test Configuration is configured to check:

    a. Open the Test Configurations panel by choosing **Parasoft> Test Configurations**.
    b. Select the Test Configurations category that represents the user-defined Test Configuration you want to modify.
    c. Open the **Static** tab.
    d. Click the **PrintableDocs** button.

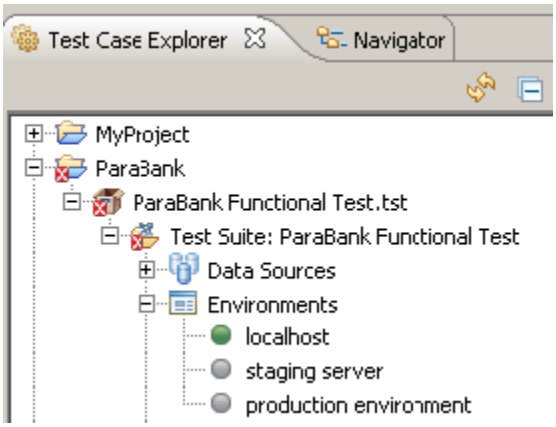
Execution Tab Settings: Defining How Tests are Executed

During a test, SOAtest will execute test cases based on the parameters defined in the selected Test Configuration's Execution tab. For any level of execution, the top-level **Enable Test Execution** option must be enabled.

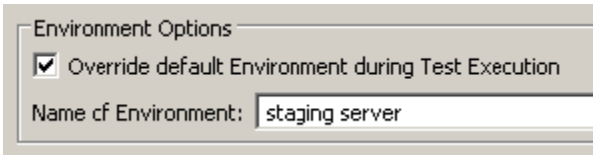
Functional tab

The **Execution> Functional** tab has the following settings:

- **Execute functional tests:** Determines whether any functional tests are run.
- **Enable event logging:** Determines whether SOAtest logs the data needed to provide a detailed chronologist sequence of all the events that occurred between the start and end of the test (for instance, all requests sent, responses received, data source rows used, wait times, navigation tasks, and so on). See Exploring Test Event Details for details.
- **Execute in load test mode:** Determines whether SOAtest executes available tests in load testing mode and alerts you to any outstanding issues that might impact your load testing—for example, incorrectly configured HTTP requests. See Validating Tests for details.
    - **Auto-configure tests in preparation for load testing:** Determines whether SOAtest configures browser-based web scenariosto run in a browser-less load test environment. See Configuring Tests for details.
- **Execute only opened Test Suite (.tst) Files (always false in command-line mode):** Determines whether SOAtest executes Test Suites that are not currently active (i.e., tests that you are not currently working on).
- **Report traffic for all tests:** Determines whether reports contain a "Test Traffic [All Tests]" section, which contains traffic for every test execution—whether or not it was successful. If this is enabled, you can also configure the traffic limit: the amount of traffic that will be stored during a test execution session (not per test). The default is 500 KB.
- **Launch an application:** Allows you to configure a SOAtest test configuration to run an Eclipse launch configuration at the beginning of the execution of the test configuration. For example, assume you want to run a test scenario against a local copy of an application that you start and run within Eclipse. If you want to start your application and run your tests in a single step, you can create a Test Configuration to launch this application as well as execute tests.
- **Override default environment during test execution:** Configures SOAtest to always use the specified environment for tests run with this Test Configuration—regardless of what environment is active in the Test Case Explorer.
  For example, assume you have the following environments:



This is how you set the Test Configuration to always use the "staging server" environment:



- **Use playback engine:** Allows you to override a test's playback engine settings at the time of test execution. By default, Test Configurations are set to play web scenarios using the playback engine specified at the test suite level. This allows you to use a single Test Configuration to execute a mixture of tests configured for Selenium and tests configured for the legacy engine. If you select a specific driver here, it will be used regardless of what engine is configured at the test scenario level. See Using Selenium WebDriver for Legacy Browser Recordings and Using the Legacy Native Driver Instead of Selenium for details.
- **Use browser:** Allows you to override a test's browser playback settings at the time of test execution. See Configuring Browser Playback Options for details.
- **Apply static analysis to:** If a Test Configuration runs both static analysis and test execution (e.g., for performing static analysis on a web scenario), this setting determines whether static analysis is performed on the HTTP responses, or the browser contents.
    - **HTTP Responses** refers to the individual HTTP messages that the browser made in order to construct its data model—the content returned by the server as is (before any browser processing).
    - **Browser-Constructed HTML** refers to the real-time data model that the browser constructed from all of the HTML, JS, CSS, and other files it loaded.

Security tab

The **Execution> Security** tab allows you to configure penetration testing, which is described in Penetration Testing.

Runtime Error Detection tab

The **Execution> Runtime Error Detection** tab allows you to configure runtime error detection, which is described in Performing Runtime Error Detection.

Change Impact tab

The **Execution> Change Impact** tab contains an option (**Perform change impact analysis**) that controls whether the current Test Configuration performs change impact analysis during test execution. See Updating Messages with Change Advisor for details.

API Coverage tab

The **Execution> API Coverage** tab contains options for calculating API Coverage during test execution. See API Coverage for details.

Application Coverage tab

The **Execution> Application Coverage** tab contains options for collecting application coverage data, which provides visibility into the level of code coverage achieved by your SOAtest tests. See Application Coverage for details.

Common Tab Settings: Defining Common Options that Affect Multiple Analysis Types

The Test Configuration's Common tab controls test settings for actions that affect multiple analysis types.

The Common tab has the following settings:

- **Override Session Tag:** Assigns the specified session tag to results from test runs performed using the current Test Configuration. This overrides the session tag specified in **Preferences> Parasoft> Reports**. This value is used for uploading summary results to Team Server. The tag is an identifier of the module checked during the analysis process. Reports for different modules should be marked with different tags. The same variables that are valid for Parasoft Test Preferences options can be used here.
- **Before Testing> Refresh projects:** Determines whether projects are refreshed before they are tested. When a project is refreshed, SOAtest checks whether external tools have changed the project in the local file system, and then applies any detected changes. Note that when you test from the command line, projects are always refreshed before testing.
- **Before Testing> Update projects from source control:** Determines whether projects are updated from source control (if you using a supported source control system) before they are tested.
- **Build:** Determines if and when whether projects are built before they are tested. Note that this settings applies to GUI tests, not command-line tests. Available options include:
    - **Full (rebuild all files):** Specifies that all project files should always be rebuilt.
    - **Incremental (build files changed since last build):** Specifies that only the project files that have changed since the previous build should be rebuilt.
    - **Stop testing on build errors:** Specifies that testing should stop when build errors are reported.
- **After Testing> Commit added/modified files to source control if no tasks were reported:** Allows you to combine your testing and your source control checkins into a single step. For example, you would enable this if you want to run static analysis on files, then have SOAtest automatically check in the modified files if no static analysis tasks are reported. In the context of functional testing, it tells SOAtest that if you run modified tests—and they pass—it should check the modified tests into source control.

Code Review Tab Settings: Defining Code Review Options

This tab contains settings for automating preparation, notification, and tracking the peer review process, which can be used to evaluate critical SDLC artifacts (source files, tests, etc.) in the context of the organization's defined quality policies.

For details on configuring Code Review (including details on Code Review tab options), see Code Review.

Goals Tab Settings: Defining Error Reporting and Resolution Targets

The team manager can specify a reporting limit (such as "Do not report more than 25 static analysis tasks per developer per day") and/or a quality goal (such as "All static analysis violations should be fixed in 2 months"). SOAtest will then use the specified criteria to select a subset of testing tasks for each developer to perform each day. These goals are specified in the Goals tab. Progress towards these goals can then be monitored in reports.

Alternatively, you can set global team goals—goals that may span across multiple Test Configurations and even across Parasoft Test products—as described in Configuring Task Goals. This requires Team Server and a Parasoft Test Automation edition license. If goals are set globally, the Goals tab in the Test Configuration panel will be disabled.

The Goals tab has the following settings:

Static tab

- **Perform all tasks:** Specifies that you want SOAtest to report all static analysis tasks it recommends, and the team should perform all static analysis tasks immediately.
- **Don't perform tasks:** Specifies that you want SOAtest to report all static analysis tasks it recommends, but the team is not required to perform all static analysis tasks immediately. This is useful, for instance, if you want to see all recommended static analysis tasks, but you want the team to focus on fixing test failures before addressing static analysis violations.
- **No more than n tasks per developer by date:** Specifies that you want each developer to have only n static analysis tasks by the specified date.
- **Max tasks to recommend:** Limits the number of static analysis tasks reported for each developer on any test run. The tasks shown are selected randomly so that different tasks are shown after each run. For example, if you set the parameter to 50, the first task shown after each run is selected at random, and the following 49 tasks shown are the ones that would follow that first task in a complete report.

Execution tab

- **Perform all tasks:** Specifies that you want SOAtest to report all functional testing tasks, and the team should perform the specified tasks immediately.
- **Don't perform tasks:** Specifies that you want SOAtest to report all functional testing tasks, but the team is not required to perform the specified tasks immediately. This is useful, for instance, if you want to see a list of all necessary functional testing tasks, but you want the team to focus on fixing static analysis tasks before addressing functional test failures.
- **No more than n tasks per developer by date:** Specifies that you want each developer to have only n functional testing tasks by the specified date.
- **Max tasks to recommend:** Limits the number of functional testing tasks reported for each developer on any test run. The tasks shown are selected randomly so that different tasks are shown after each run. For example, if you set the parameter to 50, the first task shown after each run is selected at random, and the following 49 tasks shown are the ones that would follow that first task in a complete report.