Configuring Smart API Test Generation

In this section:

- Overview
- About Smart Test Templates
- Manually Creating and Configuring Test Templates
- Training Based on .tst Files
- Applying Smart Test Templates to Existing .tst Files
- Example Smart Test Template
- Configuring Smart API Test Generation for Salesforce
- Test Creation Properties

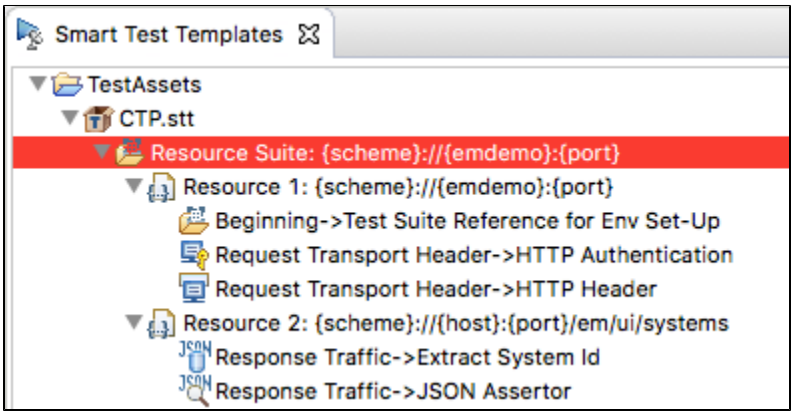
Overview

SOAtest can automatically create API tests for web applications from API traffic captured with Parasoft Recorder. Tests created in this way are referred to as "smart" API tests. The out-of-the-box implementation is designed to help you quickly start testing your APIs, but you can also "teach" SOAtest how you want smart API tests to be generated.

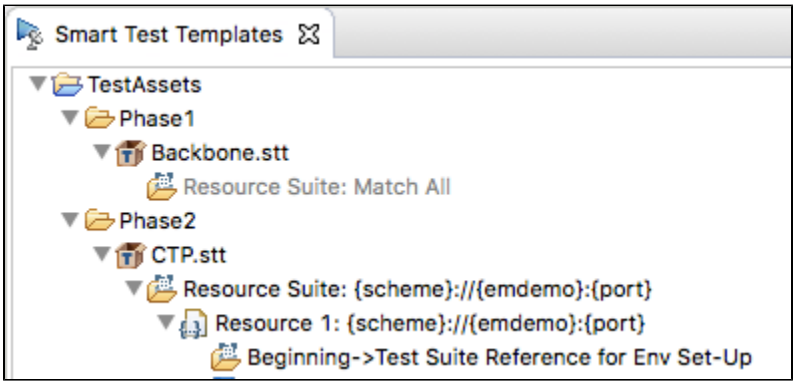
About Smart Test Templates

You can create Smart Test Template (.stt) files to define rules that determine test generation behaviors. These files contain one or more Resource Suites, which identify paths through the application to which your test generation rules will be applied.

Resource Suites contain one or more Resource Templates, which define a scope of API methods and resources to include when the smart test is generated. You can attach tools to Resource Templates and configure them to define specific test generation behaviors. Tools chained to the Resource Template are applied according to the scope defined in the Resource Template.



When the SOAtest creates a new smart API test, it will read and apply the rules specified in the .stt file.

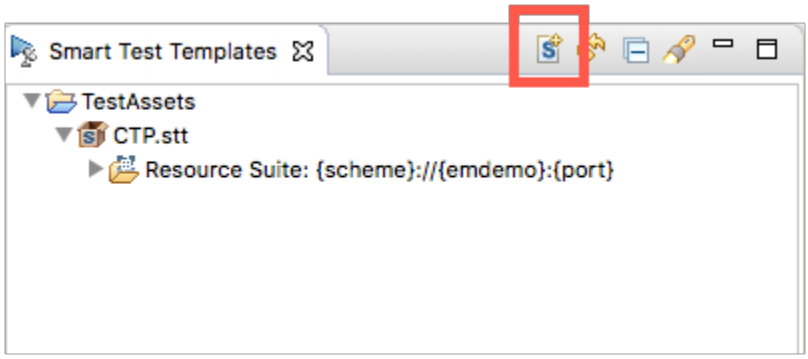


Manually Creating and Configuring Test Templates

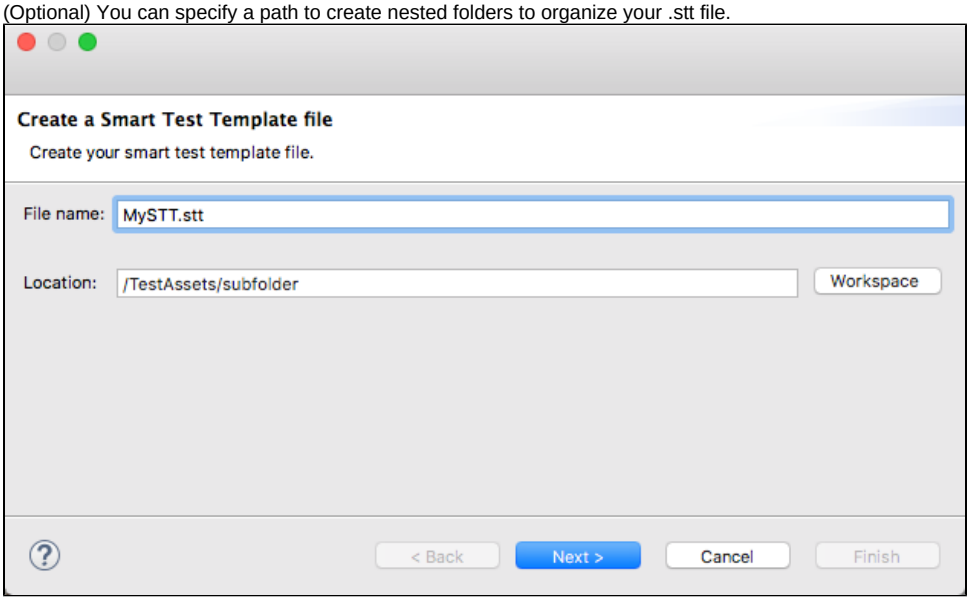
You can create and manage .stt files using the Smart Test Templates View. Additionally, you can add multiple .stt files and organize them into folders.

1. Choose **Parasoft> Show View> Smart Test Templates** to open the Smart Test Templates view (if not already open).

2. Click the **Add Template** button and specify a name and location for the .stt file.



3. (Optional) You can specify a path to create nested folders to organize your .stt file.



e

4. Click **Next**.
5. Choose the type of Resource Suite to create. You can create an empty suite or initialize the suite based on a service definition.

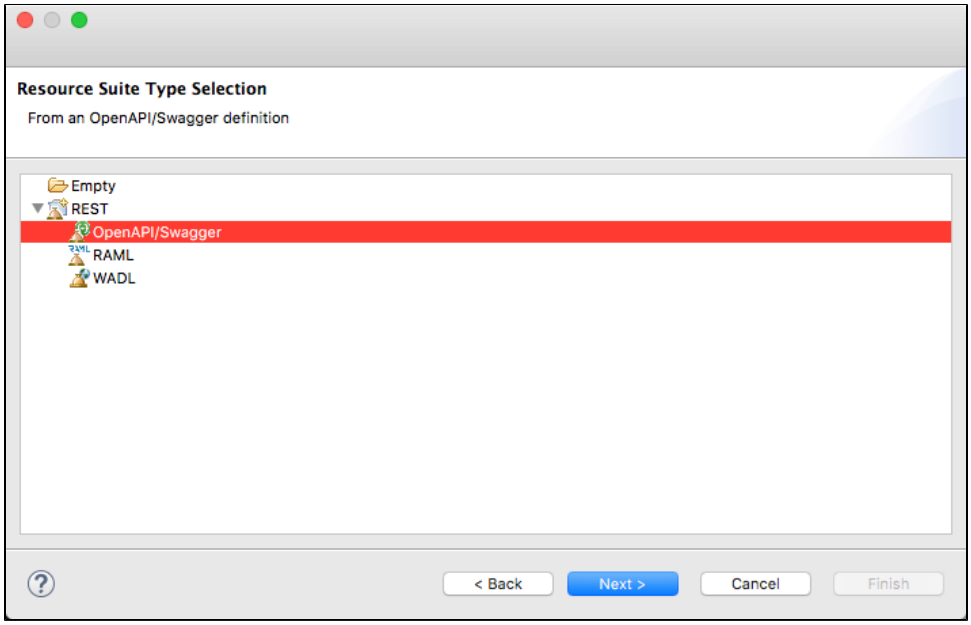
   If you are using a definition file, see the following sections for additional information:

   Creating Tests from an OpenAPI/Swagger Definition

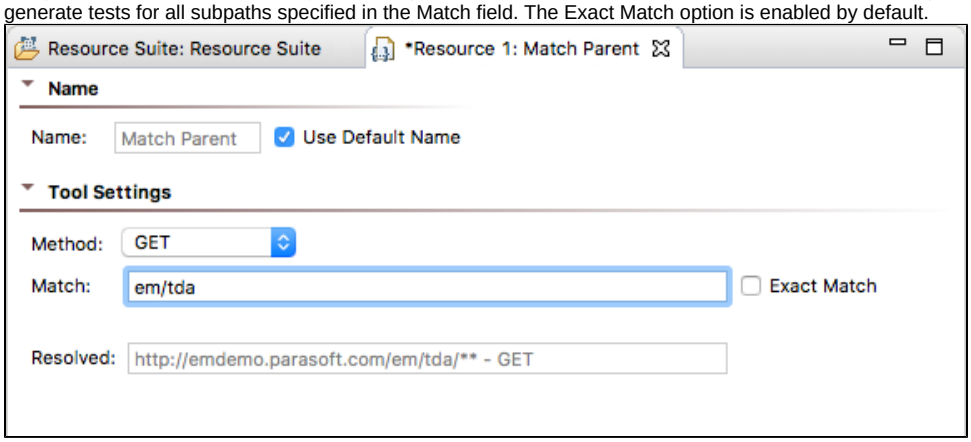
   Creating Tests From a RAML Definition

   Creating Tests From a WADL

6. Click **Next** if you are creating a suite based on a service definition and specify the location of your definition file. Skip this step to create an empty suite.



7. Click **Finish**.
8. Expand the .stt file and double-click the **Resource Suite**.
9. (Optional) You can disable the Use Default Name option and specify a name for the file.
10. Specify a pattern in the Match field to identify the endpoints you want the rules in this suite to apply to (see Defining Smart API Test Generation Scope). The Resolved field renders a preview of the pattern. Nested Resource Suites inherit the Match settings from the parent Resource Suite. You can determine how to structure template files and configure matching patterns accordingly. See Example Smart Test Template for a common use-case.
11. Right-click the Resource Suite and choose **Add New> Resource Template**.
12. (Optional) Disable the Use Default Name option and specify a name for the Resource Template in the Name field.
13. Choose a method from the Method drop-down menu and specify a pattern in the Match field to identify the endpoints you want the rules in the template to apply to (see Defining Smart API Test Generation Scope). The Resolved field renders a preview of the pattern. Resource Templates inherit the Match settings from the parent Resource Suite. You can disable the Exact Match option if you want to allow SOAtest to generate tests for all subpaths specified in the Match field. The Exact Match option is enabled by default.



When SOAtest determines a match in the Resource Template, configurations from all chained tools are applied. You can determine how to structure template files and configure matching patterns accordingly. See Example Smart Test Template for a common use-case.

> **Chaining JSON Assertors**
>
> JSON Assertor tools chained to Resource Templates can be configured to use the values from traffic if present. See Adding JSON Assertions for additional information.

14. Right-click the Resource Template and choose **Add Output...**
15. Choose a tool to include in your generated test and configure the settings. For example, you can chain and configure an HTTP Authentication tool so that all test suites generated with the template automatically authenticate against the application under test (see Enabling Authentication for additional information).
16. Add additional Resource Suites and Resource Templates as necessary. The .stt file is processed in order from first to last. Each Resource Suite is processed according to its hierarchy before the next suite is processed. You can add as many suites and templates and chain as many tools as necessary to train SOAtest.
17. Save your changes.

Generate a test from an existing traffic file or create a new test using the Parasoft Recorder browser extension and the configurations defined in the .stt file will be applied.

Defining Smart API Test Generation Scope

Patterns you define in Resource Suites and Resource Templates will be matched according to the following rules:

- Curly braces, e.g., `{},` indicate wildcards and can contain any text. For example, you can specify a pattern that translates into scheme, host, and port:

  ```
  {scheme}://{host}:{port}
  ```

- Asterisks (*) also indicate wildcards. A single asterisk and curly braces, e.g., {}, may be used interchangeably:

  ```
  {scheme}://{host}:*/parabank/*/accounts/{id}
  ```

- You can use a double-asterisk to match multiple directories.

  ```
  {scheme}://{host}:*/parabank/**/{id}
  ```

- Excluding scheme, host, and port will match any scheme, host, and port. For example, the following pattern matches the specific path on any host, port, and method:

  ```
  /parabank/services_proxy/bank/accounts/{id}
  ```

- Specifying a negative number or a number greater than 65535 for the port treats the port as a wildcard.
- Resource Templates and nested Resource Suites inherit the Match settings from the parent Resource Suite.
- You can disable the Exact Match option if you want to allow the SOAtest to generate tests for all subpaths specified in the Match field. The Exact Match option is enabled by default in Resource Templates.

Default Scope

By default, SOAtest will create tests for all resources called in the scenario and only apply training tools to resources identified with the Resource Template tools. You can configure SOAtest, however, to include or exclude specific resource URLs from test generation by specifying Ant-style patterns in the `includeURLPatterns` and `excludeURLPatterns` in the tst_configuration.properties file. See Test Creation Properties for additional information.

Enabling Authentication

SOAtest uses the HTTP Authentication tool for configuring authentication credentials for your tests. The HTTP Authentication tool is a special tool that you can only use in .stt files. When SOAtest matches the settings in the Resource Template tool, credentials configured in the HTTP Authentication tool are applied to the generated tests.

1. Right-click a Resource Template tool and choose **Add Output..**.
2. Expand the Request menu and choose **Transport Header**.
3. Choose **HTTP Authentication** from the tools panel and click **Finish**.
4. Configure the authentication settings in the HTTP Authentication tool. The following authentication types are supported:
   - Basic (simple authentication built into the HTTP protocol)
   - NTLM
   - Kerberos
   - Digest

See the following sections for additional information on configuring authentication settings:

SOAtest

- HTTP 1.0
- HTTP 1.1

Virtualize

- HTTP 1.0
- HTTP 1.1

Adding Headers

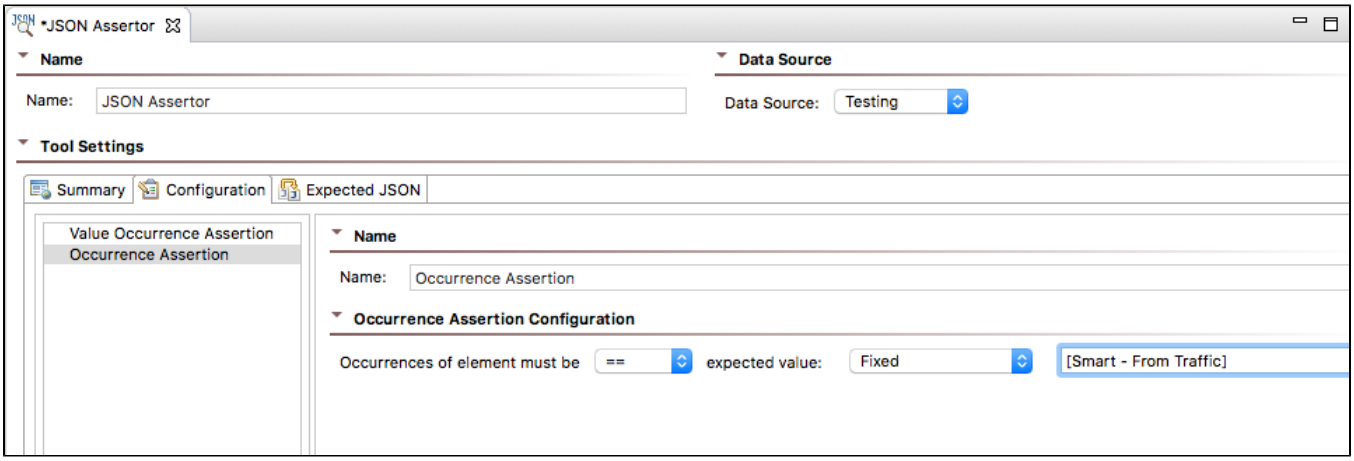
SOAtest uses the HTTP Header tool to add header fields to your tests. The HTTP Header tool is a special tool that you can only use in .stt files. When SOAtest matches the settings in the Resource Template tool, any headers configured in the HTTP Header tool will automatically be added to the generated tests. You also can use this tool to override header values captured during test creation.

1. Right-click a Resource Template tool and choose **Add Output..**.
2. Expand the Request menu and choose **Transport Header**.
3. Choose **HTTP Header** from the tools panel and click **Finish**.
4. Click **Add** in the Tool Settings section to define values.

5. Save your changes.

Adding JSON Assertions

You can configure JSON Assertions chained to Resource Template tools to use values from traffic. Use == or `equals` as the operator and set the value field `[Smart - From Traffic]`.



JSON Assertor Behaviors by Type

There are several types of assertions you can use in your tests (see JSON Assertor for additional information). The following sections describes how SOAtest applies settings for each type of assertion.

Value Assertions

- The Expected Value field for Value Assertions will be set from the recorded traffic if the field is set to `[Smart - From Traffic]`.
- The Element value field and the expected value field for Value Occurrence Assertions, Numeric Assertions, and String Comparison Assertions will be set based on the recorded traffic if the field is set to `[Smart - From Traffic]` and the operation is set to == or `equal`.
- The Expected value field for Value Occurrence Assertions will be set to `1` or `0` if the field is set to `[Smart - From Traffic]` and the operation is set to ==.
- Fields for Regular Expression Assertions, Expression Assertions, and Custom Assertions will not be set from traffic.

Structure Assertions

- The expected value field for Occurrence Assertions will be set to `1` or `0` if the field is set to `[Smart - From Traffic]` and the operation is set to ==.
- Fields for Has Contents Assertions, Has Children Assertions, and Type assertions will not be set from traffic.

Difference Assertions

- The Base Value field will be set based on the recorded traffic if the field is set to `[Smart - From Traffic]` and if the traffic value validates successfully.
- The Difference Value field for Numeric Assertions will be set to `0` if the field is set to `[Smart - From Traffic]`.
- Difference configuration fields for Date Difference Assertions and DateTime Difference Assertions will not be set from traffic.

Range Assertions

- The Lower Bound and Upper Bound fields will be set based on the recorded traffic if the field is set to `[Smart - From Traffic]` and the value validates successfully. In this case, the same traffic value will be applied to all fields in the assertion.

Additional JSON Assertor Behaviors

By default, SOAtest ignores timestamps, but it can be configured to ignore other parameters to fit your needs. See Test Creation Parameters for additional information.

Training Templates on Assertions from a .tst File

You can apply assertion logic from an existing .tst file to train Smart API Test Generator. See Training Based on .tst Files.

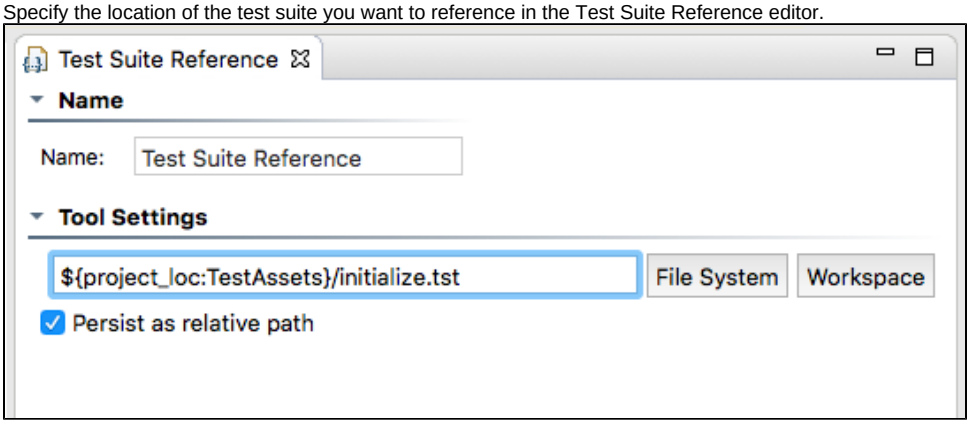
Adding Test Suite References

You can reference other test suites in your .stt file, which enables you to include additional set-up steps that may be necessary for your tests to execute properly. Only one referenced test suite will be added per Resource Template tool. Tests included by reference are added as the first test in the generated suite.

Relative paths are relative to the TestAssets project. To reference tests in another project, an environment variable should be defined and included in the path using the `${project_loc}` variable.

See Reusing and Modularizing Test Suites for End-to-end Testing for additional information about references.

1. Right-click a Resource Template tool and choose **Add Output...**
2. Expand the Suite category and choose **Beginning**.
3. Choose **Test Suite Reference** in the Smart category and click **Finish**.
4. Specify the location of the test suite you want to reference in the Test Suite Reference editor.



   You can click **File System** or **Workspace** to browse for the .tst file you want to reference.
5. Save your changes.

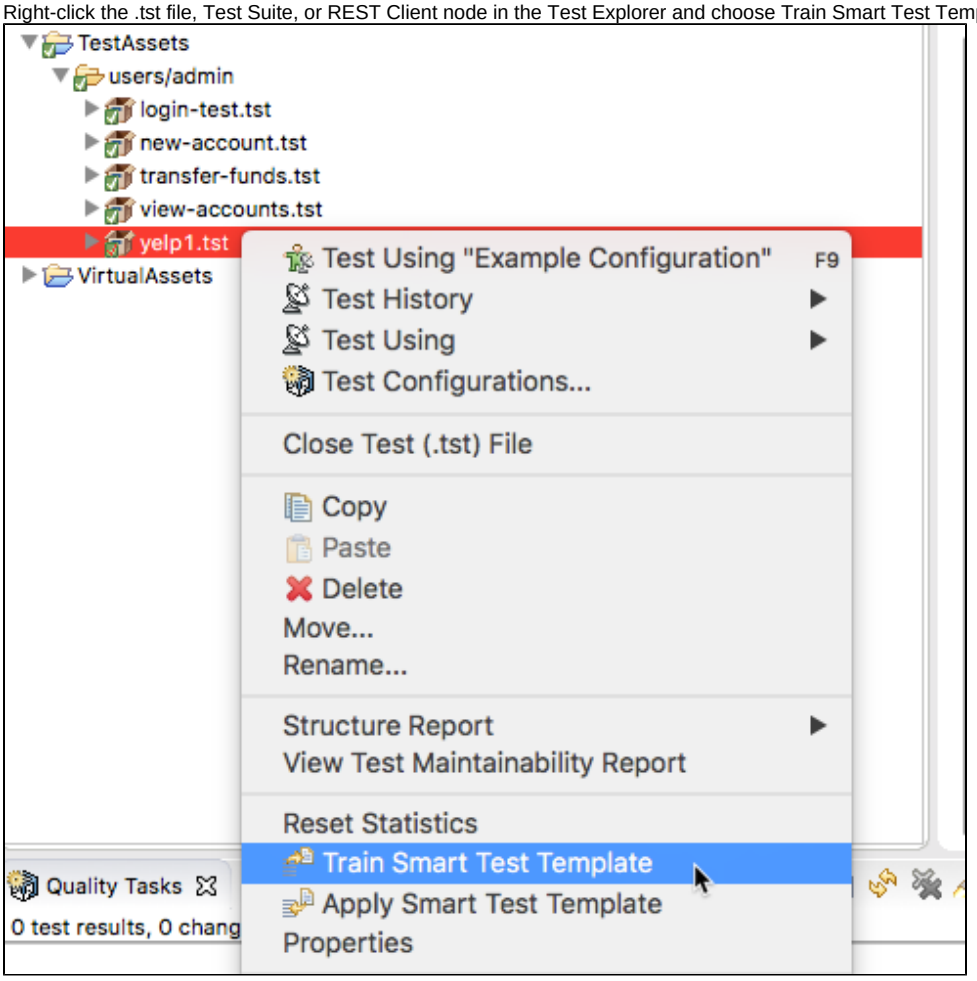
Training Based on .tst Files

You can teach SOAtest to generate tests that include authentication settings and JSON assertion logic from existing REST Clients. SOAtest will create an HTTP Authentication or a JSON Assertor rule under the resource template that most closely matches the path specified in the REST Client. If no match is identified, new .stt files will be created that contain resource templates for each REST Client URL. The .stt file(s) will contain Resource Suites and Resource Templates with their Match settings configured to match the REST Client's endpoint.

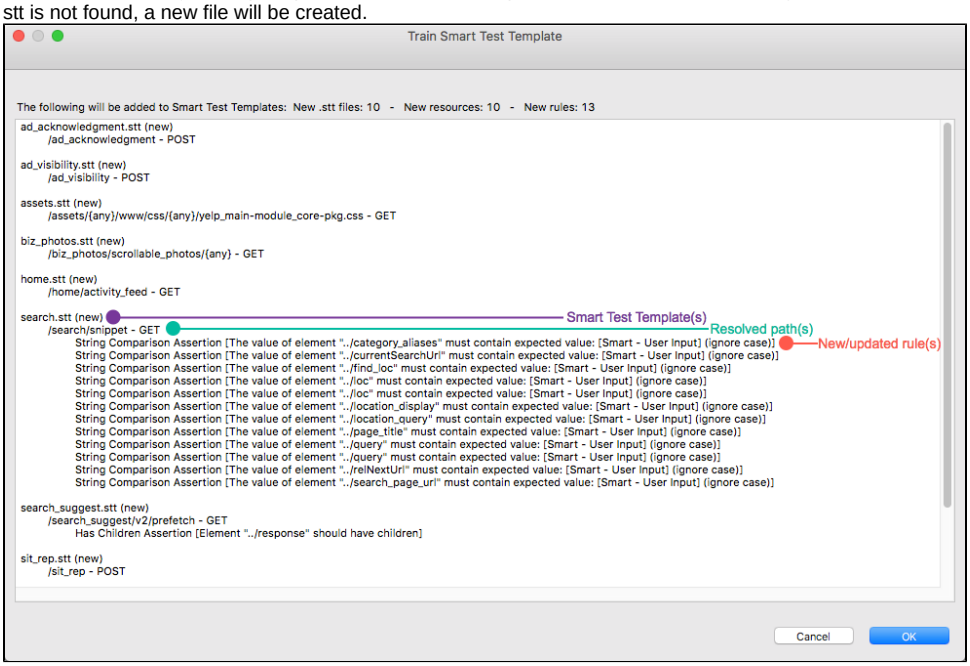
The match pattern for new .stt files is based on the type of rule you are teaching SOAtest. For authentication, the parent suite match is set using the REST client's scheme, host, and port. The resource template contained in the parent suite will be configured to match to the REST client's basePath and all subpaths and method types.

The parent suite match is set using the REST client's basePath when teaching SOAtest assertion rules and the structure of the test suites. Subpaths are configured in a nested suite match and/or a resource template match. The resource template will be configured to match the "Exact" field at the specific method type.

1. Right-click the .tst file, Test Suite, or REST Client node in the Test Explorer and choose Train Smart Test Template.



2. Review the summary when prompted. SOAtest will attempt to match resources in existing .stt files and add new rules accordingly. If a matching .stt is not found, a new file will be created.

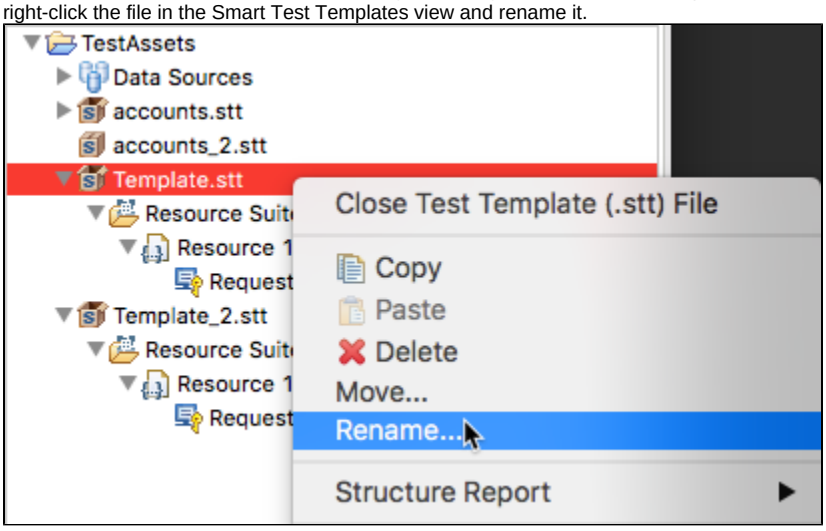


Training Smart API Test Generator with JSON Assertors

Equality assertions (assertions configured with the == or `equals` operator) with fixed values will be created based on traffic. If the value is parameterized with a non-equality assertions, the assertor chained to the resource will be a converted to a fixed value set to `[Smart - User Input]`. For parameterized equality assertions the value will be `[Smart - From Traffic]`.

New assertions will be added to existing assertors in the .stt file.

3. Click **OK** to update or add a new .stt file to the Smart Test Templates view that contains the settings configured in the REST Client. If new .stt files are created, the files will be named according to the original client's URL. Template resources will be grouped by base path segment. You can right-click the file in the Smart Test Templates view and rename it.

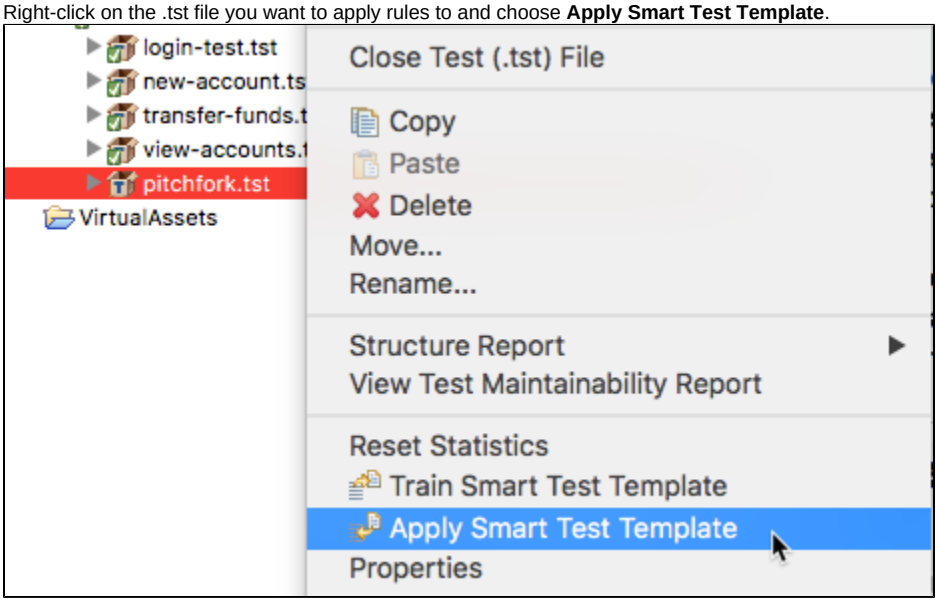


4. Double-click the Resource Suite and the Resource Template and configure their Match settings to define the test generation scope. See Defining Smart API Test Generation Scope.
5. You can manually add additional resources to finish configuring the .stt file. See Manually Creating and Configuring Test Templates.

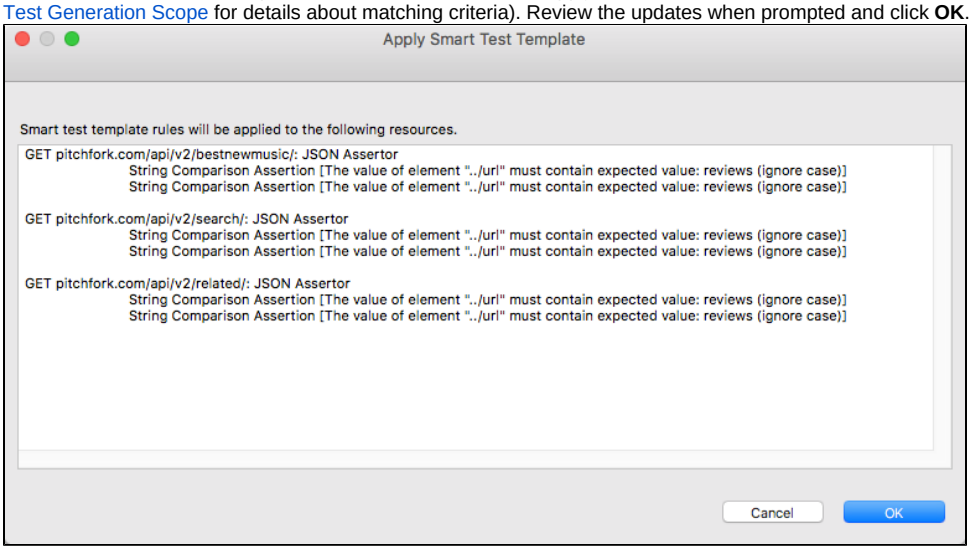
Applying Smart Test Templates to Existing .tst Files

You can update existing tests to use the rules defined in a .stt file. You would commonly use the functionality if you already have REST Clients and want to quickly apply a standard set of assertions, authentication settings, and headers from a template.

1. Right-click on the .tst file you want to apply rules to and choose **Apply Smart Test Template**.

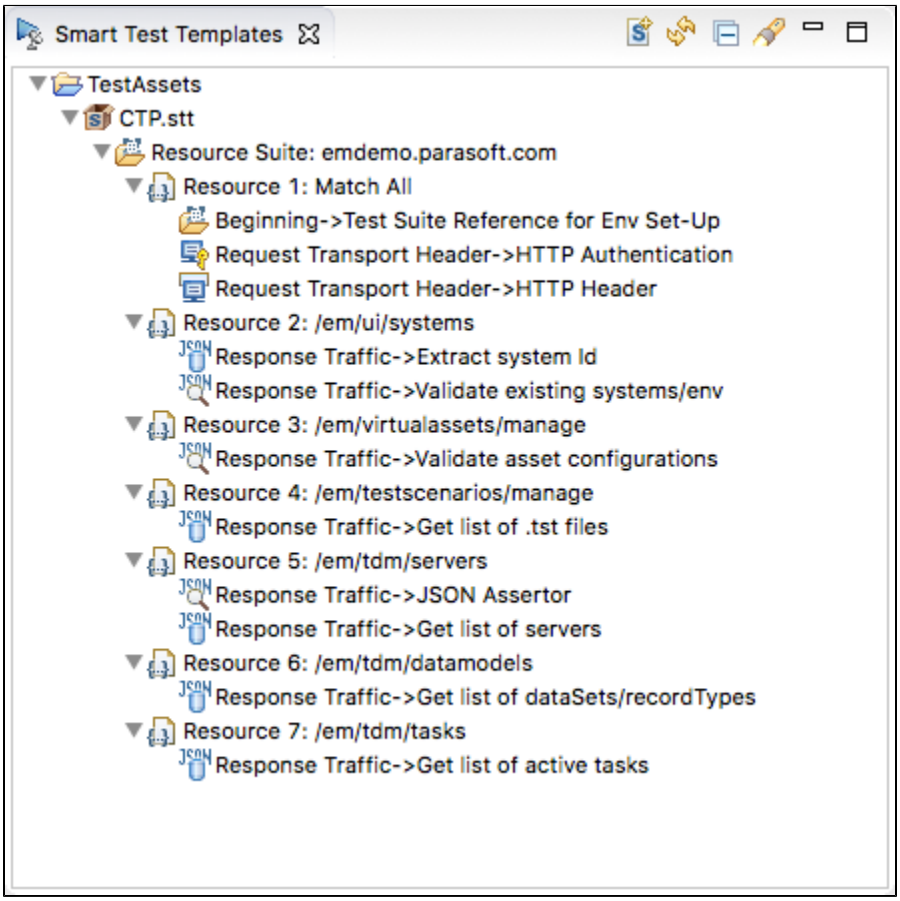


2. SOAtest will match resource paths to REST Client URLs and apply rules defined in the .stt to clients with matching URLs (see Defining Smart API Test Generation Scope for details about matching criteria). Review the updates when prompted and click **OK**.

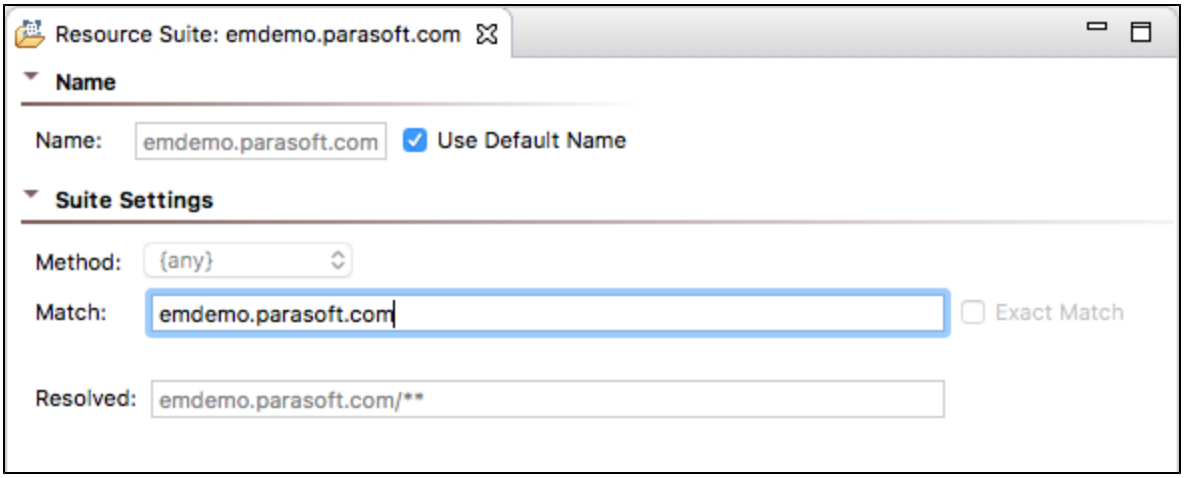


Example Smart Test Template

The structure of your test templates will depend on how you want to test your application. The following example .stt represents one way that an application under test (CTP).



The resource suite is configured to match any scheme, port, and path at host "emdemo." As a result, any tests generated for emdemo will include tools references in the Test Reference Suite. Additionally, tests will be configured with authentication and header settings configured in the HTTP Authentication and HTTP Header tools, respectively.



Each Resource Template points to a specific path under emdemo. Any tests generated for the paths specified in each Resource Template will include configurations from the chained tools. For example, "Resource 3" is configured to match the "/em/virtualassets/manage" path. As a result, tests that contain this path will include a JSON Assertor configured to validate asset configurations that are returned in the response.

Configuring Smart API Test Generation for Salesforce

The Salesforce application architecture requires Smart API test generation to implement a specific configuration to properly generate tests.

Open the tst_creation.properties file and configure the following property that enables Smart API test generation for Salesforce:

```
customHandlerClass.1=com.parasoft.webtool.testcreator.handlers.salesforce.AuraConfig
```

We also recommend configuring the following properties:

```
includeContentTypes=application/json, application/x-www-form-urlencoded, text/html, text/plain
disableDiffCreation=true
disableDiffParameterization=true
includeURLPatterns=**.force.com,**.salesforce.com
```

The patterns specified in the following setting depends on your Salesforce server configuration:

```
excludeURLPatterns=*/jslibrary/,/file-asset/,/auraFW/resources/,/auraCmpDef?,*/LayoutMeta?,*/cometd/,
/_nc_external/system/,/apex/,/l/**
```

The following settings are optional, but they may be helpful for reducing noise:

```
requestPayloadParameterizationExcludeNames.1=^pageSize$
requestPayloadParameterizationExcludeNames.2=^max[A-Za-z]+
requestPayloadParameterizationExcludeNames.3=^actionsRequestId$
requestPayloadParameterizationExcludeNames.4=^request$
requestPayloadParameterizationExcludeNames.5=^numRecordsToShow$
requestQueryStringParameterizationExcludeNames.1=^r$
```

The other properties in the tst_configuration.properties file can be configured to use the default settings or custom values you may have configured.

Test Creation Properties

You can define additional test creation properties in the tst_creation.properties configuration file. The tst_creation.properties file is located in the SOAtest workspace under the TestAssets folder. By default, all web proxies that connect to the SOAtest server will use the settings configured in this file.

The existing tst_creation.properties file is preserved during updates. If the existing file is moved or deleted, all default settings, as well as new settings added from the latest update, will be applied.

includeContentTypes	Defines a comma-separated list of content types to include during traffic processing.

	Default is `application/json,application/x-www-form-urlencoded`
excludeContentTypes	Defines a comma-separated list of content types to exclude during traffic processing. Default is empty.
disableDiffCreation	Enables/disables diff creation. See Diff for additional information. Default is `false`.
disableDiffParameterization	Enables/disables diff parameterization. Parameterization enables you to use data banked values in the diff. If diff parameterization is disabled (setting this property to `true`), only static values will be available. Default is `false`.
disableEnvironmentCreation	Enables/disables the creation of an environment and environment variables. Default is `false` .
disableDataBankCreation	Enables/disables data bank creation. See Data Exchange Tools for additional information. Default is `false`.
disableAssertorCreation	Enables/disables the creation of JSON Assertor tools. Default is `false`. See JSON Assertor for additional information.
customHandlerClass.*<number>*	See Configuring Smart API Test Generation for Salesforce.

diffToolIgnoreNames.*<number>*	Defines a regex matching element names that should be ignored when creating diffs. Default is `(?i)^(time|date|url|href).*`
	
	You can specify additional name patterns by adding properties and appending them with `.<number>`.
	
	For example:
	
	`diffToolIgnoreNames.1=<name_pattern_1>`
	
	`diffToolIgnoreNames.2=<name_pattern_3>`
	
	`diffToolIgnoreNames.3=<name_pattern_3>`

diffToolIgnoreValues.*<number>*	Defines a regex matching values that should be ignored when creating diffs. Default is to ignore timestamps:
	
	`[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}([.][0-9]{1,3})?(([+-][0-9]{2}:[0-9]{2})|Z)?`
	
	You can specify additional value patterns by adding properties and appending them with `.<number>`.
	
	For example:
	
	`diffToolIgnoreValues.1=<value_pattern_1>`
	
	`diffToolIgnoreValues.2=<value_pattern_3>`
	
	`diffToolIgnoreValues.3=<value_pattern_3>`

assertorToolIgnoreQueryParameterNames.*<number>*	Defines a regex matching query parameter names that should be ignored when creating JSON Assertor tools. Default is to ignore query names starting with "maxResultsSize": `(?i)^(maxResultSize).*` The property is not case sensitive. You can specify additional patterns by adding properties and appending them with `.<number>`.
assertorToolIgnoreQueryParameterValues.*<number>*	Defines a regex matching query parameters values that should be ignored when creating JSON Assertor tools. Ignore query parameters when creating assertions based on parameter value pattern Default is to ignore timestamps: `[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}([.][0-9]{1,3})?(([+-][0-9]{2}:[0-9]{2})|Z)?` You can specify additional patterns by adding properties and appending them with `.<number>`.
assertorToolIgnoreFieldNames.*<number>*	

	Defines a regex matching values in the response payload that should be ignored when creating JSON Assertor tools. Default value is to ignore values in response payload that starts with time, date, url, href, SessionId, or transactionId. Default is to ignore timestamps: `(?i)^(time|date|url|href|SessionId|transactionId).*` The property is not case sensitive. You can specify additional patterns by adding properties and appending them with `.<number>`.

assertorToolIgnoreFieldValues.*<number>*	Defines a regex matching values that should be ignored when creating JSON Assertor tools. Default is to ignore timestamps:
	
	`[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}([.][0-9]{1,3})?(([+-][0-9]{2}:[0-9]{2})|Z)?`
	
	You can specify additional value patterns by adding properties and appending them with `.<number>`.
	
	For example:
	
	`assertorToolIgnoreFieldValues.1=<value_pattern_1>`
	
	`assertorToolIgnoreFieldValues.2=<value_pattern_3>`
	
	`assertorToolIgnoreFieldValues.3=<value_pattern_3>`

includeURLPatterns	Defines a comma-separated list of resource URL patterns to include for test generation. You can define case-sensitive patterns using Ant-style syntax. Default is to include all URLs. In the following example, all URLs in the parasoft.com domain will be included: `includeURLPatterns=*.parasoft.com`
excludeURLPatterns	Defines a comma-separated list of resource URL patterns to exclude from test generation. You can define case-sensitive patterns using Ant-style syntax. Default is to include all URLs. In the following example, all URLs at port 8443 in the parasoft.com domain will be excluded: `excludeURLPatterns=*.parasoft.com:8443`
requestPayloadParameterizationExcludeNames.*<number>*	Defines a regex that matches field names in a request payload that should be excluded from parameterization. The following example excludes date and time field names from parameterization: `requestPayloadParameterizationExcludeNames.1=(?i)^(time|date).*`
requestQueryStringParameterizationExcludeNames.*<number>*	Defines a regex that matches field names in a request queries that should be excluded from parameterization. The following example excludes date and time field names from parameterization: `requestPayloadParameterizationExcludeNames.1=(?i)^(time|date).*`
useServerSettings	Enables/disables using the settings from the tst_creation.properties file on the SOAtest server, instead of the settings configured in the local tst_creation.properties file. Default is `true`.

Example tst_creation.properties File

```
includeContentTypes=application/json, application/x-www-form-urlencoded
excludeContentTypes=image/png,font/ttf,text/css,application/javascript
disableDiffCreation=false
disableDiffParameterization=false
disableEnvironmentCreation=false
disableDataBankCreation=false
disableAssertionCreation=true
# Ignore values that start with "time", "date", "url" or "href" in diff tool, case insensitive
diffToolIgnoreNames.1=(?i)^(time|date|url|href).*
# Ignore values that end with "time", "date", "url" or "href" in diff tool, case insensitive
diffToolIgnoreNames.2=(?i).*(time|date|url|href)$
```

```
# Ignore values like a timestamp in diff tool, e.g. 2018-03-21T07:00:00.000Z
diffToolIgnoreValues.1=[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}([.][0-9]{1,3})?(([+-][0-9]{2}:[0-9]
{2})|Z)?
# Ignore query parameters when creating assertions based on parameter name pattern
assertorToolIgnoreQueryParameterNames.1=(?i)^(maxResultSize).*
# Ignore query parameters when creating assertions based on parameter value pattern
assertorToolIgnoreQueryParameterValues.1=[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}([.][0-9]{1,3})?
(([+-][0-9]{2}:[0-9]{2})|Z)?
# Ignore fields in payload when creating assertions based on field name pattern
assertorToolIgnoreFieldNames.1=(?i)^(time|date|url|href|SessionId|transactionId).*
# Ignore fields in payload when creating assertions based on field value pattern
assertorToolIgnoreFieldValues.1=[0-9]\{4}-[0-9]\{2}-[0-9]\{2}T[0-9]\{2}:[0-9]\{2}:[0-9]\{2}([.][0-9]\{1,3})?
(([+-][0-9]\{2}:[0-9]\{2})|Z)?
# Comma separated list of URL patterns where pattern matching is the same as Apache Ant. URLs can be case
sensitive.
includeURLPatterns=http://{host}:8080/path/**/resources/*
excludeURLPatterns=https://{host}:8443/path/**/resources/*

# Exclude from being parameterized field names in request payload
requestPayloadParameterizationExcludeNames.1=(?i)^(time|date).*
```