com.jayway.restassured.builder

Class ResponseSpecBuilder

java.lang.Object
└ **com.jayway.restassured.builder.ResponseSpecBuilder**

public class **ResponseSpecBuilder**
extends Object

You can use the builder to construct a response specification. The specification can be used as e.g.

```
ResponseSpecification responseSpec = new ResponseSpecBuilder().expectStatusCode(200).build();
RequestSpecification requestSpec = new RequestSpecBuilder().addParam("parameter1", "value1").build();

given(responseSpec, requestSpec).post("/something");
```

or

```
ResponseSpecification responseSpec = new ResponseSpecBuilder().expectStatusCode(200).build();

expect().
       spec(responseSpec).
       body("x.y.z", equalTo("something")).
when().
      get("/something");
```

Constructor Summary

ResponseSpecBuilder()

Method Summary

ResponseSpecBuilder	**addResponseSpecification**(ResponseSpecification specification) Merge this builder with settings from another specification.
ResponseSpecification	**build**() Build the response specification.
ResponseSpecBuilder	**expectBody**(org.hamcrest.Matcher<?> matcher) Expect that the response content conforms to one or more Hamcrest matchers.
ResponseSpecBuilder	**expectBody**(String key, List<Argument> arguments, org.hamcrest.Matcher<?> matcher) Same as expectBody(String, org.hamcrest.Matcher) expect that you can pass arguments to the key.
ResponseSpecBuilder	**expectBody**(String key, org.hamcrest.Matcher<?> matcher) Expect that the JSON or XML response content conforms to one or more Hamcrest matchers.
ResponseSpecBuilder	**expectContent**(org.hamcrest.Matcher<?> matcher) Expect that the response content conforms to one or more Hamcrest matchers.
ResponseSpecBuilder	**expectContent**(String key, List<Argument> arguments, org.hamcrest.Matcher<?> matcher) Same as expectContent(String, org.hamcrest.Matcher) expect that you can pass arguments to the key.
ResponseSpecBuilder	**expectContent**(String key, org.hamcrest.Matcher<?> matcher) Expect that the JSON or XML response content conforms to one or more Hamcrest matchers.
ResponseSpecBuilder	**expectContentType**(groovyx.net.http.ContentType contentType) Set the response content type to be contentType.
ResponseSpecBuilder	**expectContentType**(String contentType) Set the response content type to be contentType.
ResponseSpecBuilder	**expectCookie**(String cookieName) Expect that a cookie exist in the response, regardless of value (it may have no value at all).
ResponseSpecBuilder	**expectCookie**(String cookieName, org.hamcrest.Matcher<String> expectedValueMatcher) Expect that a response cookie matches the supplied cookie name and hamcrest matcher.
ResponseSpecBuilder	**expectCookie**(String cookieName, String expectedValue) Expect that a response cookie matches the supplied name and value.
ResponseSpecBuilder	**expectCookies**(Map<String,Object> expectedCookies) Expect that response cookies matches those specified in a Map.
ResponseSpecBuilder	**expectHeader**(String headerName, org.hamcrest.Matcher<String> expectedValueMatcher)

	Expect that a response header matches the supplied header name and hamcrest matcher.
ResponseSpecBuilder	**expectHeader**(String headerName, String expectedValue) Expect that a response header matches the supplied name and value.
ResponseSpecBuilder	**expectHeaders**(Map<String,Object> expectedHeaders) Expect that response headers matches those specified in a Map.
ResponseSpecBuilder	**expectStatusCode**(int expectedStatusCode) Expect that the response status code matches an integer.
ResponseSpecBuilder	**expectStatusCode**(org.hamcrest.Matcher<Integer> expectedStatusCode) Expect that the response status code matches the given Hamcrest matcher.
ResponseSpecBuilder	**expectStatusLine**(org.hamcrest.Matcher<String> expectedStatusLine) Expect that the response status line matches the given Hamcrest matcher.
ResponseSpecBuilder	**expectStatusLine**(String expectedStatusLine) Expect that the response status line matches the given String.
ResponseSpecBuilder	**rootPath**(String rootPath) Set the root path of the response body so that you don't need to write the entire path for each expectation.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ResponseSpecBuilder

public **ResponseSpecBuilder**()

Method Detail

expectContent

public ResponseSpecBuilder **expectContent**(org.hamcrest.Matcher<?> matcher)

Expect that the response content conforms to one or more Hamcrest matchers.

Parameters:
matcher - The hamcrest matcher that must response content must match.
Returns:
The builder

---

expectContent

public ResponseSpecBuilder **expectContent**(String key,
org.hamcrest.Matcher<?> matcher)

Expect that the JSON or XML response content conforms to one or more Hamcrest matchers.

JSON example

Assume that a GET request to "/lotto" returns a JSON response containing:

```
{ "lotto":{
  "lottoId":5,
  "winning-numbers":[2,45,34,23,7,5,3],
  "winners":[{
    "winnerId":23,
    "numbers":[2,45,34,23,3,5]
  },{
    "winnerId":54,
    "numbers":[52,3,12,11,18,22]
  }]
 }}
```

You can verify that the lottoId is equal to 5 like this:

```
ResponseSpecBuilder builder = new ResponseSpecBuilder();
builder.expectContent("lotto.lottoId", equalTo(5));
```

Parameters:

matcher - The hamcrest matcher that the response content must match.

Returns:

The builder

---

expectContent

public ResponseSpecBuilder **expectContent**(String key,
List<Argument> arguments,
org.hamcrest.Matcher<?> matcher)

Same as expectContent(String, org.hamcrest.Matcher) expect that you can pass arguments to the key. This is useful in situations where you have e.g. pre-defined variables that constitutes the key:

```
String someSubPath = "else";
int index = 1;
expect().body("something.%s[%d]", withArgs(someSubPath, index), equalTo("some value")). ..
```

or if you have complex root paths and don't wish to duplicate the path for small variations:

```
expect().
      root("filters.filterConfig[%d].filterConfigGroups.find { it.name == 'Gold' }.includes").
      body("", withArgs(0), hasItem("first")).
      body("", withArgs(1), hasItem("second")).
      ..
```

The key and arguments follows the standard formatting syntax of Java.

Note that withArgs can be statically imported from the com.jayway.restassured.RestAssured class.

Parameters:

key - The body key

matcher - The hamcrest matcher that must response body must match.

Returns:

the response specification

See Also:

expectContent(String, org.hamcrest.Matcher)

---

expectStatusCode

public ResponseSpecBuilder **expectStatusCode**(org.hamcrest.Matcher<Integer> expectedStatusCode)

Expect that the response status code matches the given Hamcrest matcher.

Parameters:

expectedStatusCode - The expected status code matcher.

Returns:

The builder

---

expectStatusCode

public ResponseSpecBuilder **expectStatusCode**(int expectedStatusCode)

Expect that the response status code matches an integer.

Parameters:

expectedStatusCode - The expected status code.

Returns:

The builder

---

expectStatusLine

public ResponseSpecBuilder **expectStatusLine**(org.hamcrest.Matcher<String> expectedStatusLine)

Expect that the response status line matches the given Hamcrest matcher.

Parameters:

expectedStatusLine - The expected status line matcher.

Returns:

The builder

---

expectStatusLine

public ResponseSpecBuilder **expectStatusLine**(String expectedStatusLine)

Expect that the response status line matches the given String.

Parameters:
expectedStatusLine - The expected status line.
Returns:
The builder

---

expectHeaders

public ResponseSpecBuilder **expectHeaders**(Map<String,Object> expectedHeaders)

Expect that response headers matches those specified in a Map.

E.g. expect that the response of the GET request to "/something" contains header headerName1=headerValue1 and headerName2=headerValue2:

```
Map expectedHeaders = new HashMap();
expectedHeaders.put("headerName1", "headerValue1"));
expectedHeaders.put("headerName2", "headerValue2");
```

You can also use Hamcrest matchers:

```
Map expectedHeaders = new HashMap();
expectedHeaders.put("Content-Type", containsString("charset=UTF-8"));
expectedHeaders.put("Content-Length", "160");
```

Parameters:
expectedHeaders - The Map of expected response headers
Returns:
The builder

---

expectHeader

public ResponseSpecBuilder **expectHeader**(String headerName,
org.hamcrest.Matcher<String> expectedValueMatcher)

Expect that a response header matches the supplied header name and hamcrest matcher.

Parameters:
headerName - The name of the expected header
expectedValueMatcher - The Hamcrest matcher that must conform to the value
Returns:
The builder

---

expectHeader

public ResponseSpecBuilder **expectHeader**(String headerName,
String expectedValue)

Expect that a response header matches the supplied name and value.

Parameters:
headerName - The name of the expected header
expectedValue - The value of the expected header
Returns:
The builder

---

expectCookies

public ResponseSpecBuilder **expectCookies**(Map<String,Object> expectedCookies)

Expect that response cookies matches those specified in a Map.

E.g. expect that the response of the GET request to "/something" contains cookies cookieName1=cookieValue1 and cookieName2=cookieValue2:

```
Map expectedCookies = new HashMap();
expectedCookies.put("cookieName1", "cookieValue1"));
expectedCookies.put("cookieName2", "cookieValue2");
```

You can also use Hamcrest matchers:

```
Map expectedCookies = new HashMap();
expectedCookies.put("cookieName1", containsString("Value1"));
expectedCookies.put("cookieName2", "cookieValue2");
```

Parameters:
expectedCookies - A Map of expected response cookies
Returns:
The builder

---

expectCookie

public ResponseSpecBuilder **expectCookie**(String cookieName,
org.hamcrest.Matcher<String> expectedValueMatcher)

Expect that a response cookie matches the supplied cookie name and hamcrest matcher.

E.g. cookieName1=cookieValue1

Parameters:
cookieName - The name of the expected cookie
expectedValueMatcher - The Hamcrest matcher that must conform to the value
Returns:
The builder

---

expectCookie

public ResponseSpecBuilder **expectCookie**(String cookieName,
String expectedValue)

Expect that a response cookie matches the supplied name and value.

Parameters:
cookieName - The name of the expected cookie
expectedValue - The value of the expected cookie
Returns:
The builder

---

expectCookie

public ResponseSpecBuilder **expectCookie**(String cookieName)

Expect that a cookie exist in the response, regardless of value (it may have no value at all).

Parameters:
cookieName - the cookie to validate that it exists
Returns:
the response specification

---

rootPath

public ResponseSpecBuilder **rootPath**(String rootPath)

Set the root path of the response body so that you don't need to write the entire path for each expectation. E.g. instead of writing:

```
expect().
        body("x.y.firstName", is(..)).
        body("x.y.lastName", is(..)).
        body("x.y.age", is(..)).
        body("x.y.gender", is(..)).
when().
        get(..);
```

you can use a root path and do:

```
expect().
        rootPath("x.y").
        body("firstName", is(..)).
        body("lastName", is(..)).
        body("age", is(..)).
        body("gender", is(..)).
when().
        get(..);
```

Parameters:
rootPath - The root path to use.

---

expectContentType

public ResponseSpecBuilder **expectContentType**(groovyx.net.http.ContentType contentType)

Set the response content type to be contentType.

Note that this will affect the way the response is decoded. E,g. if you can't use JSON/XML matching (see e.g. expectBody(String, org.hamcrest.Matcher)) if you specify a content-type of "text/plain". If you don't specify the response content type REST Assured will automatically try to figure out which content type to use.

Parameters:
contentType - The content type of the response.
Returns:
The builder

---

expectContentType

public ResponseSpecBuilder **expectContentType**(String contentType)

Set the response content type to be contentType.

Note that this will affect the way the response is decoded. E,g. if you can't use JSON/XML matching (see e.g. expectBody(String, org.hamcrest.Matcher)) if you specify a content-type of "text/plain". If you don't specify the response content type REST Assured will automatically try to figure out which content type to use.

Parameters:
contentType - The content type of the response.
Returns:
The builder

---

expectBody

public ResponseSpecBuilder **expectBody**(org.hamcrest.Matcher<?> matcher)

Expect that the response content conforms to one or more Hamcrest matchers.

Parameters:
matcher - The hamcrest matcher that must response content must match.
Returns:
The builder

---

expectBody

public ResponseSpecBuilder **expectBody**(String key,
org.hamcrest.Matcher<?> matcher)

Expect that the JSON or XML response content conforms to one or more Hamcrest matchers.

JSON example

Assume that a GET request to "/lotto" returns a JSON response containing:

```
{ "lotto":{
  "lottoId":5,
  "winning-numbers":[2,45,34,23,7,5,3],
  "winners":[{
    "winnerId":23,
    "numbers":[2,45,34,23,3,5]
  },{
    "winnerId":54,
    "numbers":[52,3,12,11,18,22]
  }]
 }}
```

You can verify that the lottoId is equal to 5 like this:

```
ResponseSpecBuilder builder = new ResponseSpecBuilder();
builder.expectBody("lotto.lottoId", equalTo(5));
```

Parameters:

matcher - The hamcrest matcher that the response content must match.

Returns:

The builder

---

expectBody

public ResponseSpecBuilder **expectBody**(String key,
List<Argument> arguments,
org.hamcrest.Matcher<?> matcher)

Same as expectBody(String, org.hamcrest.Matcher) expect that you can pass arguments to the key. This is useful in situations where you have e.g. pre-defined variables that constitutes the key:

```
String someSubPath = "else";
int index = 1;
expect().body("something.%s[%d]", withArgs(someSubPath, index), equalTo("some value")). ..
```

or if you have complex root paths and don't wish to duplicate the path for small variations:

```
expect().
      root("filters.filterConfig[%d].filterConfigGroups.find { it.name == 'Gold' }.includes").
      body("", withArgs(0), hasItem("first")).
      body("", withArgs(1), hasItem("second")).
      ..
```

The key and arguments follows the standard formatting syntax of Java.

Note that withArgs can be statically imported from the com.jayway.restassured.RestAssured class.

Parameters:

key - The body key

matcher - The hamcrest matcher that must response body must match.

Returns:

the response specification

See Also:

expectBody(String, org.hamcrest.Matcher)

---

addResponseSpecification

public ResponseSpecBuilder **addResponseSpecification**(ResponseSpecification specification)

Merge this builder with settings from another specification. Note that the supplied specification can overwrite data in the current specification. The following settings are overwritten:

- Content type
- Root pathStatus code
- Status line

The following settings are merged:

- Response body expectations
- Cookies
- Headers

Parameters:

specification - The specification the add.

Returns:

The builder

---

build

public ResponseSpecification **build**()

Build the response specification.

Returns:

The assembled response specification
