com.jayway.restassured.specification

Interface RequestSpecification

All Superinterfaces:
RequestSender

All Known Subinterfaces:
FilterableRequestSpecification

All Known Implementing Classes:
RequestSpecificationImpl

public interface **RequestSpecification**
extends RequestSender

Allows you to specify how the request will look like.

Method Summary

RequestSpecification	**and**() Syntactic sugar, e.g.
AuthenticationSpecification	**auth**() A slightly short version of authentication().
AuthenticationSpecification	**authentication**() If you need to specify some credentials when performing a request.
RequestSpecification	**body**(byte[] body) Specify a byte array request body that'll be sent with the request.
RequestSpecification	**body**(Object object) Specify an Object request content that will automatically be serialized to JSON or XML and sent with the request.
RequestSpecification	**body**(Object object, ObjectMapper mapper) Specify an Object request content that will automatically be serialized to JSON or XML and sent with the request using a specific object mapper.
RequestSpecification	**body**(String body) Specify a String request body (such as e.g.
RequestSpecification	**content**(byte[] content) Specify a byte array request content that'll be sent with the request.
RequestSpecification	**content**(Object object) Specify an Object request content that will automatically be serialized to JSON or XML and sent with the request.
RequestSpecification	**content**(Object object, ObjectMapper mapper) Specify an Object request content that will automatically be serialized to JSON or XML and sent with the request using a specific object mapper.
RequestSpecification	**content**(String content) Specify a String request content (such as e.g.
RequestSpecification	**contentType**(groovyx.net.http.ContentType contentType) Specify the content type of the request.
RequestSpecification	**contentType**(String contentType) Specify the content type of the request.
RequestSpecification	**cookie**(Cookie cookie) Specify a Cookie to send with the request.
RequestSpecification	**cookie**(String cookieName) Specify a cookie with no value that'll be sent with the request e.g:
RequestSpecification	**cookie**(String cookieName, Object value, Object... additionalValues) Specify a cookie that'll be sent with the request e.g:
RequestSpecification	**cookies**(Cookies cookies) Specify the cookies that'll be sent with the request as Cookies:
RequestSpecification	**cookies**(Map<String,?> cookies) Specify the cookies that'll be sent with the request as Map e.g:
RequestSpecification	**cookies**(String firstCookieName, Object firstCookieValue, Object... cookieNameValuePairs)

	Specify the cookies that'll be sent with the request.
ResponseSpecification	**expect**() Returns the response specification so that you can setup the expectations on the response.
RequestSpecification	**filter**(Filter filter) Add a filter that will be used in the request.
RequestSpecification	**filters**(List<Filter> filters) Add filters that will be used in the request.
RequestSpecification	**formParam**(String parameterName, List<?> parameterValues) A slightly shorter version of formParameter(String, java.util.List).
RequestSpecification	**formParam**(String parameterName, Object parameterValue, Object... additionalParameterValues) A slightly shorter version of formParameter(String, Object, Object...).
RequestSpecification	**formParameter**(String parameterName, List<?> parameterValues) Specify a multi-value form parameter that'll be sent with the request e.g:
RequestSpecification	**formParameter**(String parameterName, Object parameterValue, Object... additionalParameterValues) Specify a form parameter that'll be sent with the request.
RequestSpecification	**formParameters**(Map<String,?> parametersMap) Specify the form parameters that'll be sent with the request.
RequestSpecification	**formParameters**(String firstParameterName, Object firstParameterValue, Object... parameterNameValuePairs) Specify the form parameters that'll be sent with the request.
RequestSpecification	**formParams**(Map<String,?> parametersMap) A slightly shorter version of formParams(java.util.Map).
RequestSpecification	**formParams**(String firstParameterName, Object firstParameterValue, Object... parameterNameValuePairs) A slightly shorter version of formParameters(String, Object, Object...).
RequestSpecification	**given**() Syntactic sugar, e.g.
RequestSpecification	**header**(Header header) Specify a Header to send with the request.
RequestSpecification	**header**(String headerName, Object headerValue, Object... additionalHeaderValues) Specify a header that'll be sent with the request e.g:
RequestSpecification	**headers**(Headers headers) Specify the headers that'll be sent with the request as Headers, e.g:
RequestSpecification	**headers**(Map<String,?> headers) Specify the headers that'll be sent with the request as Map e.g:
RequestSpecification	**headers**(String firstHeaderName, Object firstHeaderValue, Object... headerNameValuePairs) Specify the headers that'll be sent with the request.
RequestSpecification	**keystore**(String pathToJks, String password) The following documentation is taken from http://groovy.codehaus.org/modules/http-builder/doc/ssl.html:
RequestSpecification	**log**() Log (i.e.
RequestSpecification	**logOnError**() Log (i.e.
RequestSpecification	**multiPart**(File file) Specify a file to upload to the server using multi-part form data uploading.
RequestSpecification	**multiPart**(String controlName, File file) Specify a file to upload to the server using multi-part form data uploading with a specific control name.
RequestSpecification	**multiPart**(String controlName, File file, String mimeType) Specify a file to upload to the server using multi-part form data uploading with a specific control name and mime-type.
RequestSpecification	**multiPart**(String controlName, String contentBody) Specify a string to send to the server using multi-part form data.
RequestSpecification	**multiPart**(String controlName, String fileName, byte[] bytes) Specify a byte-array to upload to the server using multi-part form data.
RequestSpecification	**multiPart**(String controlName, String fileName, byte[] bytes, String mimeType) Specify a byte-array to upload to the server using multi-part form data.
RequestSpecification	**multiPart**(String controlName, String fileName, InputStream stream) Specify an inputstream to upload to the server using multi-part form data.
RequestSpecification	**multiPart**(String controlName, String fileName, InputStream stream, String mimeType) Specify an inputstream to upload to the server using multi-part form data.
RequestSpecification	**multiPart**(String controlName, String contentBody, String mimeType) Specify a string to send to the server using multi-part form data with a specific mime-type.

RequestSpecification	**param**(String parameterName, List<?> parameterValues) A slightly shorter version of parameter(String, java.util.List) }.
RequestSpecification	**param**(String parameterName, Object parameterValue, Object... additionalParameterValues) A slightly shorter version of parameter(String, Object, Object...).
RequestSpecification	**parameter**(String parameterName, List<?> parameterValues) Specify a multi-value parameter that'll be sent with the request e.g:
RequestSpecification	**parameter**(String parameterName, Object parameterValue, Object... additionalParameterValues) Specify a parameter that'll be sent with the request e.g:
RequestSpecification	**parameters**(Map<String,?> parametersMap) Specify the parameters that'll be sent with the request as Map e.g:
RequestSpecification	**parameters**(String firstParameterName, Object firstParameterValue, Object... parameterNameValuePairs) Specify the parameters that'll be sent with the request.
RequestSpecification	**params**(Map<String,?> parametersMap) A slightly shorter version of parameters(Map).
RequestSpecification	**params**(String firstParameterName, Object firstParameterValue, Object... parameterNameValuePairs) A slightly shorter version of parameters(String, Object, Object...)
RequestSpecification	**pathParam**(String parameterName, Object parameterValue) A slightly shorter version of pathParameter(String, Object).
RequestSpecification	**pathParameter**(String parameterName, Object parameterValue) Specify a path parameter.
RequestSpecification	**pathParameters**(Map<String,?> parameterNameValuePairs) Specify multiple path parameter name-value pairs.
RequestSpecification	**pathParameters**(String firstParameterName, Object firstParameterValue, Object... parameterNameValuePairs) Specify multiple path parameter name-value pairs.
RequestSpecification	**pathParams**(Map<String,?> parameterNameValuePairs) A slightly shorter version of pathParameters(java.util.Map).
RequestSpecification	**pathParams**(String firstParameterName, Object firstParameterValue, Object... parameterNameValuePairs) A slightly shorter version of pathParameters(String, Object, Object...).
RequestSpecification	**port**(int port) Specify the port of the URI.
RequestSpecification	**queryParam**(String parameterName, List<?> parameterValues) A slightly shorter version of queryParameter(String, java.util.List).
RequestSpecification	**queryParam**(String parameterName, Object parameterValue, Object... additionalParameterValues) A slightly shorter version of queryParameter(String, Object, Object...).
RequestSpecification	**queryParameter**(String parameterName, List<?> parameterValues) Specify a multi-value query parameter that'll be sent with the request e.g:
RequestSpecification	**queryParameter**(String parameterName, Object parameterValue, Object... additionalParameterValues) Specify a query parameter that'll be sent with the request.
RequestSpecification	**queryParameters**(Map<String,?> parametersMap) Specify the query parameters that'll be sent with the request.
RequestSpecification	**queryParameters**(String firstParameterName, Object firstParameterValue, Object... parameterNameValuePairs) Specify the query parameters that'll be sent with the request.
RequestSpecification	**queryParams**(Map<String,?> parametersMap) A slightly shorter version of queryParams(java.util.Map).
RequestSpecification	**queryParams**(String firstParameterName, Object firstParameterValue, Object... parameterNameValuePairs) A slightly shorter version of queryParameters(String, Object, Object...).
RequestSpecification	**request**() Syntactic sugar, e.g.
ResponseSpecification	**response**() Returns the response specification so that you can setup the expectations on the response.
RequestSpecification	**spec**(RequestSpecification requestSpecificationToMerge) Add request data from a pre-defined specification.
RequestSpecification	**specification**(RequestSpecification requestSpecificationToMerge) Add request data from a pre-defined specification.
RequestSpecification	**that**() Syntactic sugar, e.g.
ResponseSpecification	**then**() Returns the response specification so that you can setup the expectations on the response.
RequestSpecification	**urlEncodingEnabled**(boolean isEnabled) Specifies if Rest Assured should url encode the URL automatically.
RequestSpecification	**when**() Syntactic sugar, e.g.

RequestSpecification	**with**() Syntactic sugar, e.g.

Methods inherited from interface com.jayway.restassured.specification.RequestSender
delete, delete, get, get, head, head, post, post, put, put

Method Detail

body

RequestSpecification **body**(String body)

Specify a String request body (such as e.g. JSON or XML) that'll be sent with the request. This works for the POST and PUT methods only. Trying to do this for the other http methods will cause an exception to be thrown.

Example of use:

```
given().body("{ \"message\" : \"hello world\"}").then().expect().body(equalTo("hello world")).when().post("/json");
```

This will POST a request containing JSON to "/json" and expect that the response body equals to "hello world".

Note that body(String) and content(String) are the same except for the syntactic difference.

Parameters:
body - The body to send.
Returns:
The request specification

---

body

RequestSpecification **body**(byte[] body)

Specify a byte array request body that'll be sent with the request. This only works for the POST http method. Trying to do this for the other http methods will cause an exception to be thrown.

Example of use:

```
byte[] someBytes = ..
given().body(someBytes).then().expect().body(equalTo("hello world")).when().post("/json");
```

This will POST a request containing `someBytes` to "/json" and expect that the response body equals to "hello world".

Note that body(byte[]) and content(byte[]) are the same except for the syntactic difference.

Parameters:
body - The body to send.
Returns:
The request specification

---

body

RequestSpecification **body**(Object object)

Specify an Object request content that will automatically be serialized to JSON or XML and sent with the request. If the object is a primitive or Number the object will be converted to a String and put in the request body. This works for the POST and PUT methods only. Trying to do this for the other http methods will cause an exception to be thrown.

Example of use:

```
Message message = new Message();
message.setMessage("My beautiful message");

given().
        contentType("application/json").
        body(message).
expect().
        content(equalTo("Response to a beautiful message")).
when().
        post("/beautiful-message");
```

Since the content-type is "application/json" then REST Assured will automatically try to serialize the object using Jackson or

Gson if they are available in the classpath. If any of these frameworks are not in the classpath then an exception is thrown.
If the content-type is "application/xml" then REST Assured will automatically try to serialize the object using JAXB if it's available in the classpath. Otherwise an exception will be thrown.
If no request content-type is specified then REST Assured determine the parser in the following order:

1. Jackson
2. Gson
3. JAXB

Note that body(Object) and content(Object) are the same except for the syntactic difference.

Parameters:
object - The object to serialize and send with the request
Returns:
The request specification

body

RequestSpecification **body**(Object object,
ObjectMapper mapper)

Specify an Object request content that will automatically be serialized to JSON or XML and sent with the request using a specific object mapper. This works for the POST and PUT methods only. Trying to do this for the other http methods will cause an exception to be thrown.

Example of use:

```
Message message = new Message();
message.setMessage("My beautiful message");

given().
        body(message, ObjectMapper.GSON).
expect().
        content(equalTo("Response to a beautiful message")).
when().
        post("/beautiful-message");
```

Note that body(Object, ObjectMapper) and content(Object, ObjectMapper) are the same except for the syntactic difference.

Parameters:
object - The object to serialize and send with the request
Returns:
The request specification

content

RequestSpecification **content**(String content)

Specify a String request content (such as e.g. JSON or XML) that'll be sent with the request. This works for the POST and PUT methods only. Trying to do this for the other http methods will cause an exception to be thrown.

Example of use:

```
given().content("{ \"message\" : \"hello world\"}").then().expect().content(equalTo("hello world")).when().post("/json");
```

This will POST a request containing JSON to "/json" and expect that the response content equals to "hello world".

Note that body(String) and content(String) are the same except for the syntactic difference.

Parameters:
content - The content to send.
Returns:
The request specification

content

RequestSpecification **content**(byte[] content)

Specify a byte array request content that'll be sent with the request. This only works for the POST http method. Trying to do this for the other http methods will cause an exception to be thrown.

Example of use:

```
byte[] someBytes = ..
given().content(someBytes).then().expect().content(equalTo("hello world")).when().post("/json");
```

This will POST a request containing someBytes to "/json" and expect that the response content equals to "hello world".

Note that body(byte[]) and content(byte[]) are the same except for the syntactic difference.

Parameters:
content - The content to send.
Returns:
The request specification

---

content

RequestSpecification **content**(Object object)

Specify an Object request content that will automatically be serialized to JSON or XML and sent with the request. If the object is a primitive or Number the object will be converted to a String and put in the request body. This works for the POST and PUT methods only. Trying to do this for the other http methods will cause an exception to be thrown.

Example of use:

```
Message message = new Message();
message.setMessage("My beautiful message");

given().
        contentType("application/json").
        content(message).
expect().
        content(equalTo("Response to a beautiful message")).
when().
        post("/beautiful-message");
```

Since the content-type is "application/json" then REST Assured will automatically try to serialize the object using Jackson or Gson if they are available in the classpath. If any of these frameworks are not in the classpath then an exception is thrown.
If the content-type is "application/xml" then REST Assured will automatically try to serialize the object using JAXB if it's available in the classpath. Otherwise an exception will be thrown.
If no request content-type is specified then REST Assured determine the parser in the following order:

1. Jackson
2. Gson
3. JAXB

Note that body(Object) and content(Object) are the same except for the syntactic difference.

Parameters:
object - The object to serialize and send with the request
Returns:
The request specification

---

content

RequestSpecification **content**(Object object,
ObjectMapper mapper)

Specify an Object request content that will automatically be serialized to JSON or XML and sent with the request using a specific object mapper. This works for the POST and PUT methods only. Trying to do this for the other http methods will cause an exception to be thrown.

Example of use:

```
Message message = new Message();
message.setMessage("My beautiful message");

given().
        content(message, ObjectMapper.GSON).
expect().
        content(equalTo("Response to a beautiful message")).
when().
        post("/beautiful-message");
```

Note that body(Object, ObjectMapper) and content(Object, ObjectMapper) are the same except for the syntactic difference.

Parameters:
object - The object to serialize and send with the request
Returns:
The request specification

---

cookies

RequestSpecification **cookies**(String firstCookieName,
                          Object firstCookieValue,
                          Object... cookieNameValuePairs)

Specify the cookies that'll be sent with the request. This is done by specifying the cookies in name-value pairs, e.g:

```
given().cookies("username", "John", "token", "1234").then().expect().body(equalTo("username, token")).when().get("/cookie");
```

This will send a GET request to "/cookie" with two cookies:

1. username=John
2. token=1234

and expect that the response body is equal to "username, token".

Parameters:
firstCookieName - The name of the first cookie
firstCookieValue - The value of the first cookie
cookieNameValuePairs - Additional cookies in name-value pairs.
Returns:
The request specification

---

cookies

RequestSpecification **cookies**(Map<String,?> cookies)

Specify the cookies that'll be sent with the request as Map e.g:

```
Map<String, String> cookies = new HashMap<String, String>();
cookies.put("username", "John");
cookies.put("token", "1234");
given().cookies(cookies).then().expect().body(equalTo("username, token")).when().get("/cookie");
```

This will send a GET request to "/cookie" with two cookies:

1. username=John
2. token=1234

and expect that the response body is equal to "username, token".

Parameters:
cookies - The Map containing the cookie names and their values to set in the request.
Returns:
The request specification

---

cookies

RequestSpecification **cookies**(Cookies cookies)

Specify the cookies that'll be sent with the request asCookies:

```
Cookie cookie1 = Cookie.Builder("username", "John").setComment("comment 1").build();
Cookie cookie2 = Cookie.Builder("token", 1234).setComment("comment 2").build();
Cookies cookies = new Cookies(cookie1, cookie2);
given().cookies(cookies).then().expect().body(equalTo("username, token")).when().get("/cookie");
```

This will send a GET request to "/cookie" with two cookies:

1. username=John
2. token=1234

and expect that the response body is equal to "username, token".

Parameters:
cookies - The cookies to set in the request.
Returns:
The request specification

---

cookie

RequestSpecification **cookie**(String cookieName,
                         Object value,
                         Object... additionalValues)

Specify a cookie that'll be sent with the request e.g:

```
given().cookie("username", "John").and().expect().body(equalTo("username")).when().get("/cookie");
```

This will set the cookie username=John in the GET request to "/cookie".

You can also specify several cookies like this:

```
given().cookie("username", "John").and().cookie("password", "1234").and().expect().body(equalTo("username")).when().get("/cookie");
```

If you specify additionalValues then the Cookie will be a multi-value cookie. This means that you'll create several cookies with the same name but with different values.

Parameters:
cookieName - The cookie cookieName
value - The cookie value
additionalValues - Additional cookies values. This will actually create two cookies with the same name but with different values.
Returns:
The request specification
See Also:
cookies(String, Object, Object...)

---

cookie

RequestSpecification **cookie**(String cookieName)

Specify a cookie with no value that'll be sent with the request e.g:

```
given().cookie("some_cookie").and().expect().body(equalTo("x")).when().get("/cookie");
```

This will set the cookie some_cookie in the GET request to "/cookie".

Parameters:
cookieName - The cookie cookieName
Returns:
The request specification
See Also:
cookies(String, Object, Object...)

---

cookie

RequestSpecification **cookie**(Cookie cookie)

Specify a Cookie to send with the request.

```
Cookie someCookie = new Cookie.Builder("some_cookie", "some_value").setSecured(true).build();
given().cookie(someCookie).and().expect().body(equalTo("x")).when().get("/cookie");
```

This will set the cookie someCookie in the GET request to "/cookie".

Parameters:
cookie - The cookie to add to the request
Returns:
The request specification
See Also:
cookies(com.jayway.restassured.response.Cookies)

---

parameters

RequestSpecification **parameters**(String firstParameterName,
                                  Object firstParameterValue,
                                  Object... parameterNameValuePairs)

Specify the parameters that'll be sent with the request. This is done by specifying the parameters in name-value pairs, e.g:

```
given().parameters("username", "John", "token", "1234").then().expect().body(equalTo("username, token")).when().get("/parameters");
```

This will send a GET request to "/parameters" with two parameters:

1. username=John
2. token=1234

and expect that the response body is equal to "username, token".

Parameters:
firstParameterName - The name of the first parameter
firstParameterValue - The value of the first parameter
parameterNameValuePairs - Additional parameters in name-value pairs.
Returns:
The request specification

---

parameters

RequestSpecification **parameters**(Map<String,?> parametersMap)

Specify the parameters that'll be sent with the request as Map e.g:

```
Map<String, String> parameters = new HashMap<String, String>();
parameters.put("username", "John");
parameters.put("token", "1234");
given().parameters(parameters).then().expect().body(equalTo("username, token")).when().get("/cookie");
```

This will send a GET request to "/cookie" with two parameters:

1. username=John
2. token=1234

and expect that the response body is equal to "username, token".

Parameters:
parametersMap - The Map containing the parameter names and their values to send with the request.
Returns:
The request specification

---

parameter

RequestSpecification **parameter**(String parameterName,
Object parameterValue,
Object... additionalParameterValues)

Specify a parameter that'll be sent with the request e.g:

```
given().parameter("username", "John").and().expect().body(equalTo("username")).when().get("/cookie");
```

This will set the parameter `username=John` in the GET request to "/cookie".

You can also specify several parameters like this:

```
given().parameter("username", "John").and().parameter("password", "1234").and().expect().body(equalTo("username")).when().get("/cookie");
```

Parameters:
parameterName - The parameter key
parameterValue - The parameter value
additionalParameterValues - Additional parameter values if you want to specify multiple values for the same parameter
Returns:
The request specification
See Also:
parameters(String, Object, Object...)

---

parameter

RequestSpecification **parameter**(String parameterName,
List<?> parameterValues)

Specify a multi-value parameter that'll be sent with the request e.g:

```
given().parameter("cars", asList("Volvo", "Saab"))..;
```

This will set the parameter `cars=Volvo` and `cars=Saab`.

Parameters:
parameterName - The parameter key
parameterValues - The parameter values
Returns:
The request specification

params

```
RequestSpecification params(String firstParameterName,
                            Object firstParameterValue,
                            Object... parameterNameValuePairs)
```

A slightly shorter version of parameters(String, Object, Object...)

Parameters:
firstParameterName - The name of the first parameter
firstParameterValue - The value of the first parameter
parameterNameValuePairs - Additional parameters in name-value pairs.
Returns:
The request specification
See Also:
parameters(String, Object, Object...)

params

```
RequestSpecification params(Map<String,?> parametersMap)
```

A slightly shorter version of parameters(Map).

Parameters:
parametersMap - The Map containing the parameter names and their values to send with the request.
Returns:
The request specification
See Also:
parameters(Map)

param

```
RequestSpecification param(String parameterName,
                           Object parameterValue,
                           Object... additionalParameterValues)
```

A slightly shorter version of parameter(String, Object, Object...).

Parameters:
parameterName - The parameter key
parameterValue - The parameter value
additionalParameterValues - Additional parameter values if you want to specify multiple values for the same parameter
Returns:
The request specification
See Also:
parameter(String, Object, Object...)

param

```
RequestSpecification param(String parameterName,
                           List<?> parameterValues)
```

A slightly shorter version of parameter(String, java.util.List) }.

Parameters:
parameterName - The parameter key
parameterValues - The parameter values
Returns:
The request specification

queryParameters

```
RequestSpecification queryParameters(String firstParameterName,
                                     Object firstParameterValue,
                                     Object... parameterNameValuePairs)
```

Specify the query parameters that'll be sent with the request. Note that this method is the same as parameters(String, Object, Object...) for all http methods except for POST where parameters(String, Object, Object...) sets the form parameters and this method sets the query parameters.

Parameters:
firstParameterName - The name of the first parameter
firstParameterValue - The value of the first parameter

parameterNameValuePairs - The value of the first parameter followed by additional parameters in name-value pairs.

Returns:

The request specification

---

queryParameters

RequestSpecification **queryParameters**(Map<String,?> parametersMap)

Specify the query parameters that'll be sent with the request. Note that this method is the same as parameters(Map) for all http methods except for POST where parameters(Map) sets the form parameters and this method sets the query parameters.

Parameters:

parametersMap - The Map containing the parameter names and their values to send with the request.

Returns:

The request specification

---

queryParameter

RequestSpecification **queryParameter**(String parameterName,
Object parameterValue,
Object... additionalParameterValues)

Specify a query parameter that'll be sent with the request. Note that this method is the same as parameter(String, Object, Object...) for all http methods except for POST where parameter(String, Object, Object...) adds a form parameter and this method sets a query parameter.

Parameters:

parameterName - The parameter key

parameterValue - The parameter value

additionalParameterValues - Additional parameter values if you want to specify multiple values for the same parameter

Returns:

The request specification

See Also:

parameter(String, Object, Object...)

---

queryParameter

RequestSpecification **queryParameter**(String parameterName,
List<?> parameterValues)

Specify a multi-value query parameter that'll be sent with the request e.g:

```
given().queryParameter("cars", asList("Volvo", "Saab"))..;
```

This will set the parameter cars=Volvo and cars=Saab.

Note that this method is the same as parameter(String, java.util.List) for all http methods except for POST where parameter(String, java.util.List) adds a form parameter and this method sets a query parameter.

Parameters:

parameterName - The parameter key

parameterValues - The parameter values

Returns:

The request specification

---

queryParams

RequestSpecification **queryParams**(String firstParameterName,
Object firstParameterValue,
Object... parameterNameValuePairs)

A slightly shorter version of queryParameters(String, Object, Object...).

Parameters:

firstParameterName - The name of the first parameter

firstParameterValue - The value of the first parameter

parameterNameValuePairs - The value of the first parameter followed by additional parameters in name-value pairs.

Returns:

The request specification

See Also:

queryParameters(String, Object, Object...)

---

queryParams

RequestSpecification **queryParams**(Map<String,?> parametersMap)

A slightly shorter version of queryParams(java.util.Map).

Parameters:
parametersMap - The Map containing the parameter names and their values to send with the request.
Returns:
The request specification
See Also:
queryParams(java.util.Map)

---

queryParam

RequestSpecification **queryParam**(String parameterName,
Object parameterValue,
Object... additionalParameterValues)

A slightly shorter version of queryParameter(String, Object, Object...).

Parameters:
parameterName - The parameter key
parameterValue - The parameter value
additionalParameterValues - Additional parameter values if you want to specify multiple values for the same parameter
Returns:
The request specification
See Also:
parameter(String, Object, Object...)

---

queryParam

RequestSpecification **queryParam**(String parameterName,
List<?> parameterValues)

A slightly shorter version of queryParameter(String, java.util.List).

Parameters:
parameterName - The parameter key
parameterValues - The parameter values
Returns:
The request specification
See Also:
queryParam(String, java.util.List)

---

formParameters

RequestSpecification **formParameters**(String firstParameterName,
Object firstParameterValue,
Object... parameterNameValuePairs)

Specify the form parameters that'll be sent with the request. Note that this method is the same as parameters(String, Object, Object...) for all http methods except for PUT where parameters(String, Object, Object...) sets the query parameters and this method sets the form parameters.

Parameters:
firstParameterName - The name of the first parameter
firstParameterValue - The value of the first parameter
parameterNameValuePairs - The value of the first parameter followed by additional parameters in name-value pairs.
Returns:
The request specification

---

formParameters

RequestSpecification **formParameters**(Map<String,?> parametersMap)

Specify the form parameters that'll be sent with the request. Note that this method is the same as parameters(Map) for all http methods except for PUT where parameters(Map) sets the query parameters and this method sets the form parameters.

Parameters:
parametersMap - The Map containing the parameter names and their values to send with the request.
Returns:
The request specification

formParameter

```
RequestSpecification formParameter(String parameterName,
                                   Object parameterValue,
                                   Object... additionalParameterValues)
```

Specify a form parameter that'll be sent with the request. Note that this method is the same as parameter(String, Object, Object...) for all http methods except for PUT where parameter(String, Object, Object...) adds a query parameter and this method sets a form parameter.

Parameters:
parameterName - The parameter key
parameterValue - The parameter value
additionalParameterValues - Additional parameter values if you want to specify multiple values for the same parameter
Returns:
The request specification
See Also:
parameter(String, Object, Object...)

formParameter

```
RequestSpecification formParameter(String parameterName,
                                   List<?> parameterValues)
```

Specify a multi-value form parameter that'll be sent with the request e.g:

```
given().formParameter("cars", asList("Volvo", "Saab"))..;
```

This will set the parameter cars=Volvo and cars=Saab.

Note that this method is the same as parameter(String, java.util.List) for all http methods except for PUT where parameter(String, java.util.List) adds a query parameter and this method sets a form parameter.

Parameters:
parameterName - The parameter key
parameterValues - The parameter values
Returns:
The request specification

formParams

```
RequestSpecification formParams(String firstParameterName,
                                Object firstParameterValue,
                                Object... parameterNameValuePairs)
```

A slightly shorter version of formParameters(String, Object, Object...).

Parameters:
firstParameterName - The name of the first parameter
firstParameterValue - The value of the first parameter
parameterNameValuePairs - The value of the first parameter followed by additional parameters in name-value pairs.
Returns:
The request specification
See Also:
formParameters(String, Object, Object...)

formParams

```
RequestSpecification formParams(Map<String,?> parametersMap)
```

A slightly shorter version of formParams(java.util.Map).

Parameters:
parametersMap - The Map containing the parameter names and their values to send with the request.
Returns:
The request specification
See Also:
formParams(java.util.Map)

formParam

```
RequestSpecification formParam(String parameterName,
                    Object parameterValue,
                    Object... additionalParameterValues)
```

A slightly shorter version of formParameter(String, Object, Object...).

Parameters:
: parameterName - The parameter key
: parameterValue - The parameter value
: additionalParameterValues - Additional parameter values if you want to specify multiple values for the same parameter

Returns:
: The request specification

See Also:
: parameter(String, Object, Object...)

---

formParam

```
RequestSpecification formParam(String parameterName,
                    List<?> parameterValues)
```

A slightly shorter version of formParameter(String, java.util.List).

Parameters:
: parameterName - The parameter key
: parameterValues - The parameter values

Returns:
: The request specification

See Also:
: formParam(String, java.util.List)

---

pathParameter

```
RequestSpecification pathParameter(String parameterName,
                                   Object parameterValue)
```

Specify a path parameter. Path parameters are used to improve readability of the request path. E.g. instead of writing:

```
 expect().statusCode(200).when().get("/item/"+myItem.getItemNumber()+"/buy/"+2);
```

you can write:

```
 given().
        pathParameter("itemNumber", myItem.getItemNumber()).
        pathParameter("amount", 2).
 expect().
         statusCode(200).
 when().
       get("/item/{itemNumber}/buy/{amount}");
```

which improves readability and allows the path to be reusable in many tests. Another alternative is to use:

```
 expect().statusCode(200).when().get("/item/{itemNumber}/buy/{amount}", myItem.getItemNumber(), 2);
```

Parameters:
: parameterName - The parameter key
: parameterValue - The parameter value

Returns:
: The request specification

---

pathParameters

```
RequestSpecification pathParameters(String firstParameterName,
                                    Object firstParameterValue,
                                    Object... parameterNameValuePairs)
```

Specify multiple path parameter name-value pairs. Path parameters are used to improve readability of the request path. E.g. instead of writing:

```
 expect().statusCode(200).when().get("/item/"+myItem.getItemNumber()+"/buy/"+2);
```

you can write:

```
 given().
        pathParameters("itemNumber", myItem.getItemNumber(), "amount", 2).
 expect().
         statusCode(200).
 when().
```

```
      get("/item/{itemNumber}/buy/{amount}");
```

which improves readability and allows the path to be reusable in many tests. Another alternative is to use:

```
expect().statusCode(200).when().get("/item/{itemNumber}/buy/{amount}", myItem.getItemNumber(), 2);
```

Parameters:
firstParameterName - The name of the first parameter
firstParameterValue - The value of the first parameter
parameterNameValuePairs - Additional parameters in name-value pairs.
Returns:
The request specification

---

pathParameters

RequestSpecification **pathParameters**(Map<String,?> parameterNameValuePairs)

Specify multiple path parameter name-value pairs. Path parameters are used to improve readability of the request path. E.g. instead of writing:

```
expect().statusCode(200).when().get("/item/"+myItem.getItemNumber()+"/buy/"+2);
```

you can write:

```
Map<String,Object> pathParams = new HashMap<String,Object>();
pathParams.add("itemNumber",myItem.getItemNumber());
pathParams.add("amount",2);

given().
        pathParameters(pathParams).
expect().
         statusCode(200).
when().
      get("/item/{itemNumber}/buy/{amount}");
```

which improves readability and allows the path to be reusable in many tests. Another alternative is to use:

```
expect().statusCode(200).when().get("/item/{itemNumber}/buy/{amount}", myItem.getItemNumber(), 2);
```

Parameters:
parameterNameValuePairs - A map containing the path parameters.
Returns:
The request specification

---

pathParam

RequestSpecification **pathParam**(String parameterName,
                                   Object parameterValue)

A slightly shorter version of pathParameter(String, Object).

Parameters:
parameterName - The parameter key
parameterValue - The parameter value
Returns:
The request specification
See Also:
pathParameter(String, Object)

---

pathParams

RequestSpecification **pathParams**(String firstParameterName,
                                    Object firstParameterValue,
                                    Object... parameterNameValuePairs)

A slightly shorter version of pathParameters(String, Object, Object...).

Parameters:
firstParameterName - The name of the first parameter
firstParameterValue - The value of the first parameter
parameterNameValuePairs - Additional parameters in name-value pairs.
Returns:
The request specification
See Also:

pathParameters(String, Object, Object...)

---

pathParams

RequestSpecification **pathParams**(Map<String,?> parameterNameValuePairs)

A slightly shorter version of pathParameters(java.util.Map).

Parameters:
parameterNameValuePairs - A map containing the path parameters.
Returns:
The request specification
See Also:
pathParameters(java.util.Map)

---

keystore

RequestSpecification **keystore**(String pathToJks,
String password)

The following documentation is taken from http://groovy.codehaus.org/modules/http-builder/doc/ssl.html:

SSL Configuration

SSL should, for the most part, "just work." There are a few situations where it is not completely intuitive. You can follow the example below, or see HttpClient's SSLSocketFactory documentation for more information.

SSLPeerUnverifiedException

If you can't connect to an SSL website, it is likely because the certificate chain is not trusted. This is an Apache HttpClient issue, but explained here for convenience. To correct the untrusted certificate, you need to import a certificate into an SSL truststore. First, export a certificate from the website using your browser. For example, if you go to https://dev.java.net in Firefox, you will probably get a warning in your browser. Choose "Add Exception," "Get Certificate," "View," "Details tab." Choose a certificate in the chain and export it as a PEM file. You can view the details of the exported certificate like so:

```
$ keytool -printcert -file EquifaxSecureGlobaleBusinessCA-1.crt
Owner: CN=Equifax Secure Global eBusiness CA-1, O=Equifax Secure Inc., C=US
Issuer: CN=Equifax Secure Global eBusiness CA-1, O=Equifax Secure Inc., C=US
Serial number: 1
Valid from: Mon Jun 21 00:00:00 EDT 1999 until: Sun Jun 21 00:00:00 EDT 2020
Certificate fingerprints:
MD5:  8F:5D:77:06:27:C4:98:3C:5B:93:78:E7:D7:7D:9B:CC
SHA1: 7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
Signature algorithm name: MD5withRSA
Version: 3
....
```

Now, import that into a Java keystore file:

```
$ keytool -importcert -alias "equifax-ca" -file EquifaxSecureGlobaleBusinessCA-1.crt -keystore truststore.jks -storepass test1234
Owner: CN=Equifax Secure Global eBusiness CA-1, O=Equifax Secure Inc., C=US
Issuer: CN=Equifax Secure Global eBusiness CA-1, O=Equifax Secure Inc., C=US
Serial number: 1
Valid from: Mon Jun 21 00:00:00 EDT 1999 until: Sun Jun 21 00:00:00 EDT 2020
Certificate fingerprints:
MD5:  8F:5D:77:06:27:C4:98:3C:5B:93:78:E7:D7:7D:9B:CC
SHA1: 7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
Signature algorithm name: MD5withRSA
Version: 3
...
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

Now you want to use this truststore in your client:

```
RestAssured.keystore("/truststore.jks", "test1234");
```

or

```
given().keystore("/truststore.jks", "test1234"). ..
```

Parameters:
pathToJks - The path to the JKS
password - The store pass

---

headers

RequestSpecification **headers**(String firstHeaderName,
Object firstHeaderValue,
Object... headerNameValuePairs)

Specify the headers that'll be sent with the request. This is done by specifying the headers in name-value pairs, e.g:

```
given().headers("headerName1", "headerValue1", "headerName2", "headerValue2").then().expect().body(equalTo("something")).when().get("/headers");
```

This will send a GET request to "/headers" with two headers:

1. headerName1=headerValue1
2. headerName2=headerValue2

and expect that the response body is equal to "something".

Parameters:

firstHeaderName - The name of the first header
firstHeaderValue - The value of the first header
headerNameValuePairs - Additional headers in name-value pairs.

Returns:

The request specification

---

headers

RequestSpecification **headers**(Map<String,?> headers)

Specify the headers that'll be sent with the request as Map e.g:

```
Map<String, String> headers = new HashMap<String, String>();
parameters.put("headerName1", "headerValue1");
parameters.put("headerName2", "headerValue2");
given().headers(headers).then().expect().body(equalTo("something")).when().get("/headers");
```

This will send a GET request to "/headers" with two headers:

1. headerName1=headerValue1
2. headerName2=headerValue2

and expect that the response body is equal to "something".

Parameters:

headers - The Map containing the header names and their values to send with the request.

Returns:

The request specification

---

headers

RequestSpecification **headers**(Headers headers)

Specify the headers that'll be sent with the request asHeaders, e.g:

```
Header first = new Header("headerName1", "headerValue1");
Header second = new Header("headerName2", "headerValue2");
Headers headers = new Header(first, second);
given().headers(headers).then().expect().body(equalTo("something")).when().get("/headers");
```

This will send a GET request to "/headers" with two headers:

1. headerName1=headerValue1
2. headerName2=headerValue2

and expect that the response body is equal to "something".

Parameters:

headers - The headers to use in the request

Returns:

The request specification

---

header

RequestSpecification **header**(String headerName,
Object headerValue,
Object... additionalHeaderValues)

Specify a header that'll be sent with the request e.g:

```
given().header("username", "John").and().expect().body(equalTo("something")).when().get("/header");
```

This will set the header username=John in the GET request to "/header".

You can also specify several headers like this:

```
given().header("username", "John").and().header("zipCode", "12345").and().expect().body(equalTo("something")).when().get("/header");
```

If you specify additionalHeaderValues then the Header will be a multi-value header. This means that you'll create several headers with the same name but with different values.

Parameters:
headerName - The header name
headerValue - The header value
additionalHeaderValues - Additional header values. This will actually create two headers with the same name but with different values.
Returns:
The request specification
See Also:
headers(String, Object, Object...)

---

header

RequestSpecification **header**(Header header)

Specify a Header to send with the request.

```
Header someHeader = new Header("some_name", "some_value");
given().header(someHeader).and().expect().body(equalTo("x")).when().get("/header");
```

This will set the header some_name=some_value in the GET request to "/header".

Parameters:
header - The header to add to the request
Returns:
The request specification
See Also:
headers(com.jayway.restassured.response.Headers)

---

contentType

RequestSpecification **contentType**(groovyx.net.http.ContentType contentType)

Specify the content type of the request.

Parameters:
contentType - The content type of the request
Returns:
The request specification
See Also:
ContentType

---

contentType

RequestSpecification **contentType**(String contentType)

Specify the content type of the request.

Parameters:
contentType - The content type of the request
Returns:
The request specification
See Also:
ContentType

---

multiPart

RequestSpecification **multiPart**(File file)

Specify a file to upload to the server using multi-part form data uploading. It will assume that the control name is file and the mime-type is application/octet-stream. If this is not what you want please use an overloaded method.

Parameters:
file - The file to upload
Returns:
The request specification

---

multiPart

```
RequestSpecification multiPart(String controlName,
                               File file)
```

Specify a file to upload to the server using multi-part form data uploading with a specific control name. It will use the mime-type application/octet-stream. If this is not what you want please use an overloaded method.

Parameters:
file - The file to upload
controlName - Defines the control name of the body part. In HTML this is the attribute name of the input tag.
Returns:
The request specification

---

multiPart

```
RequestSpecification multiPart(String controlName,
                               File file,
                               String mimeType)
```

Specify a file to upload to the server using multi-part form data uploading with a specific control name and mime-type.

Parameters:
file - The file to upload
controlName - Defines the control name of the body part. In HTML this is the attribute name of the input tag.
mimeType - The mime-type
Returns:
The request specification

---

multiPart

```
RequestSpecification multiPart(String controlName,
                               String fileName,
                               byte[] bytes)
```

Specify a byte-array to upload to the server using multi-part form data. It will use the mime-type application/octet-stream. If this is not what you want please use an overloaded method.

Parameters:
controlName - Defines the control name of the body part. In HTML this is the attribute name of the input tag.
fileName - The name of the content you're uploading
bytes - The bytes you want to send
Returns:
The request specification

---

multiPart

```
RequestSpecification multiPart(String controlName,
                               String fileName,
                               byte[] bytes,
                               String mimeType)
```

Specify a byte-array to upload to the server using multi-part form data.

Parameters:
controlName - Defines the control name of the body part. In HTML this is the attribute name of the input tag.
fileName - The name of the content you're uploading
bytes - The bytes you want to send
mimeType - The mime-type
Returns:
The request specification

---

multiPart

```
RequestSpecification multiPart(String controlName,
```

String fileName,
InputStream stream)

Specify an inputstream to upload to the server using multi-part form data. It will use the mime-type application/octet-stream. If this is not what you want please use an overloaded method.

Parameters:
controlName - Defines the control name of the body part. In HTML this is the attribute name of the input tag.
fileName - The name of the content you're uploading
stream - The stream you want to send
Returns:
The request specification

---

multiPart

RequestSpecification **multiPart**(String controlName,
String fileName,
InputStream stream,
String mimeType)

Specify an inputstream to upload to the server using multi-part form data.

Parameters:
controlName - Defines the control name of the body part. In HTML this is the attribute name of the input tag.
fileName - The name of the content you're uploading
stream - The stream you want to send
mimeType - The mime-type
Returns:
The request specification

---

multiPart

RequestSpecification **multiPart**(String controlName,
String contentBody)

Specify a string to send to the server using multi-part form data. It will use the mime-type text/plain. If this is not what you want please use an overloaded method.

Parameters:
controlName - Defines the control name of the body part. In HTML this is the attribute name of the input tag.
contentBody - The string to send
Returns:
The request specification

---

multiPart

RequestSpecification **multiPart**(String controlName,
String contentBody,
String mimeType)

Specify a string to send to the server using multi-part form data with a specific mime-type.

Parameters:
controlName - Defines the control name of the body part. In HTML this is the attribute name of the input tag.
contentBody - The string to send
mimeType - The mime-type
Returns:
The request specification

---

authentication

AuthenticationSpecification **authentication**()

If you need to specify some credentials when performing a request.

Returns:
The authentication specification
See Also:
AuthenticationSpecification

---

auth

AuthenticationSpecification **auth**()

A slightly short version of authentication().

Returns:
The authentication specification
See Also:
authentication(), AuthenticationSpecification

---

port

RequestSpecification **port**(int port)

Specify the port of the URI. E.g.

```
given().port(8081).and().expect().statusCode(200).when().get("/something");
```

will perform a GET request to http;//localhost:8081/something. It will override the default port of REST assured for this request only.

Note that it's also possible to specify the port like this:

```
expect().statusCode(200).when().get("http://localhost:8081/something");
```

Parameters:
port - The port of URI
Returns:
The request specification

---

spec

RequestSpecification **spec**(RequestSpecification requestSpecificationToMerge)

Add request data from a pre-defined specification. E.g.

```
RequestSpecification requestSpec = new RequestSpecBuilder().addParam("parameter1", "value1").build();

given().
        spec(requestSpec).
        param("parameter2", "value2").
when().
       get("/something");
```

This is useful when you want to reuse an entire specification across multiple requests.

The specification passed to this method is merged with the current specification. Note that the supplied specification can overwrite data in the current specification. The following settings are overwritten:

- Port
- Authentication schemeContent type
- Request body

The following settings are merged:

- Parameters
- Cookies
- Headers

This method is the same as specification(RequestSpecification) but the name is a bit shorter.

Parameters:
requestSpecificationToMerge - The specification to merge with.
Returns:
the request specification

---

specification

RequestSpecification **specification**(RequestSpecification requestSpecificationToMerge)

Add request data from a pre-defined specification. E.g.

```
RequestSpecification requestSpec = new RequestSpecBuilder().addParam("parameter1", "value1").build();

given().
        spec(requestSpec).
        param("parameter2", "value2").
when().
```

```
        get("/something");
```

This is useful when you want to reuse an entire specification across multiple requests.

The specification passed to this method is merged with the current specification. Note that the supplied specification can overwrite data in the current specification. The following settings are overwritten:

- Port
- Authentication schemeContent type
- Request body

The following settings are merged:

- Parameters
- Cookies
- Headers

This method is the same as specification(RequestSpecification) but the name is a bit shorter.

Parameters:
requestSpecificationToMerge - The specification to merge with.
Returns:
the request specification

---

urlEncodingEnabled

RequestSpecification **urlEncodingEnabled**(boolean isEnabled)

Specifies if Rest Assured should url encode the URL automatically. Usually this is a recommended but in some cases e.g. the query parameters are already be encoded before you provide them to Rest Assured then it's useful to disable URL encoding.

Parameters:
isEnabled - Specify whether or not URL encoding should be enabled or disabled.
Returns:
the request specification

---

filter

RequestSpecification **filter**(Filter filter)

Add a filter that will be used in the request.

Parameters:
filter - The filter to add
Returns:
the request specification

---

filters

RequestSpecification **filters**(List<Filter> filters)

Add filters that will be used in the request.

Parameters:
filters - The filters to add
Returns:
the request specification

---

log

RequestSpecification **log**()

Log (i.e. print to system out) the response body to system out. This is mainly useful for debug purposes when writing your tests. A shortcut for:

```
 given().filter(ResponseLoggingFilter.responseLogger()). ..
```

Returns:
the request specification

---

logOnError

RequestSpecification **logOnError**()

Log (i.e. print to system out) the response body to system out if an error occurs. This is mainly useful for debug purposes when writing your tests. A shortcut for:

```
given().filter(ErrorLoggingFilter.errorLogger()). ..
```

Returns:
the request specification

---

response

ResponseSpecification **response**()

Returns the response specification so that you can setup the expectations on the response. E.g.

```
given().param("name", "value").then().response().body(equalTo("something")).when().get("/something");
```

Returns:
the response specification

---

and

RequestSpecification **and**()

Syntactic sugar, e.g.

```
expect().body(containsString("OK")).and().body(containsString("something else")).when().get("/something");
```

is that same as:

```
expect().body(containsString("OK")).body(containsString("something else")).when().get("/something");
```

Returns:
the request specification

---

with

RequestSpecification **with**()

Syntactic sugar, e.g.

```
expect().body(containsString("OK")).and().with().request().parameters("param1", "value1").get("/something");
```

is that same as:

```
expect().body(containsString("OK")).and().request().parameters("param1", "value1").get("/something");
```

Returns:
the request specification

---

then

ResponseSpecification **then**()

Returns the response specification so that you can setup the expectations on the response. E.g.

```
given().param("name", "value").then().body(equalTo("something")).when().get("/something");
```

Returns:
the response specification

---

expect

ResponseSpecification **expect**()

Returns the response specification so that you can setup the expectations on the response. E.g.

```
given().param("name", "value").and().expect().body(equalTo("something")).when().get("/something");
```

Returns:
the response specification

---

when

RequestSpecification **when**()

Syntactic sugar, e.g.

```
expect().body(containsString("OK")).when().get("/something");
```

is that same as:

```
expect().body(containsString("OK")).get("/something");
```

Returns:
the request specification

---

given

RequestSpecification **given**()

Syntactic sugar, e.g.

```
given().param("name1", "value1").and().given().param("name2", "value2").when().get("/something");
```

is that same as:

```
given().param("name1", "value1").and().param("name2", "value2").when().get("/something");
```

Returns:
the request specification

---

that

RequestSpecification **that**()

Syntactic sugar, e.g.

```
expect().that().body(containsString("OK")).when().get("/something");
```

is that same as:

```
expect().body(containsString("OK")).get("/something");
```

Returns:
the request specification

---

request

RequestSpecification **request**()

Syntactic sugar, e.g.

```
given().request().param("name", "John").then().expect().body(containsString("OK")).when().get("/something");
```

is that same as:

```
given().param("name", "John").then().expect().body(containsString("OK")).when().get("/something");
```

Returns:
the request specification
