README.rdoc

* README.rdoc
* Last modified: Sun Aug 22 14:30:31 -0700 2010

[FakeWeb](http://fakeweb.rubyforge.org/classes/FakeWeb.html)

[FakeWeb](http://fakeweb.rubyforge.org/classes/FakeWeb.html) is a helper for faking web requests in Ruby. It works at a global level, without modifying code or writing extensive stubs.

Installation

gem install fakeweb

Note: the gem was previously available as FakeWeb (capital letters), but now all versions are simply registered as fakeweb. If you have any old FakeWeb gems lying around, remove them: gem uninstall FakeWeb

Help and discussion

RDocs for the current release are available at [fakeweb.rubyforge.org.](http://fakeweb.rubyforge.org./)

There’s a mailing list for questions and discussion at [groups.google.com/group/fakeweb-users.](http://groups.google.com/group/fakeweb-users.)

The main source repository is [github.com/chrisk/fakeweb.](http://github.com/chrisk/fakeweb.)

Examples

Start by requiring FakeWeb:

require 'fakeweb'

Registering basic string responses

FakeWeb.register\_uri(:get, "http://example.com/test1", :body => "Hello World!")

Net::HTTP.get(URI.parse("http://example.com/test1"))

=> "Hello World!"

Net::HTTP.get(URI.parse("http://example.com/test2"))

=> FakeWeb is bypassed and the response from a real request is returned

You can also call register\_uri with a regular expression, to match more than one URI.

FakeWeb.register\_uri(:get, %r|http://example\.com/|, :body => "Hello World!")

Net::HTTP.get(URI.parse("http://example.com/test3"))

=> "Hello World!"

Replaying a recorded response

page = `curl -is http://www.google.com/`

FakeWeb.register\_uri(:get, "http://www.google.com/", :response => page)

Net::HTTP.get(URI.parse("http://www.google.com/"))

# => Full response, including headers

Adding a custom status to the response

FakeWeb.register\_uri(:get, "http://example.com/", :body => "Nothing to be found 'round here",

:status => ["404", "Not Found"])

Net::HTTP.start("example.com") do |req|

response = req.get("/")

response.code # => "404"

response.message # => "Not Found"

response.body # => "Nothing to be found 'round here"

end

Responding to any HTTP method

FakeWeb.register\_uri(:any, "http://example.com", :body => "response for any HTTP method")

If you use the :any symbol, the URI you specify will be completely stubbed out (regardless of the HTTP method of the request). This can be useful for RPC-style services, where the HTTP method isn’t significant. (Older versions of [FakeWeb](http://fakeweb.rubyforge.org/classes/FakeWeb.html) always behaved like this, and didn’t accept the firstmethod argument above; this syntax is now deprecated.)

Rotating responses

You can optionally call FakeWeb.register\_uri with an array of options hashes; these are used, in order, to respond to repeated requests. Once you run out of responses, further requests always receive the last response. (You can also send a response more than once before rotating, by specifying a :times option for that response.)

FakeWeb.register\_uri(:delete, "http://example.com/posts/1",

[{:body => "Post 1 deleted.", :status => ["200", "OK"]},

{:body => "Post not found", :status => ["404", "Not Found"]}])

Net::HTTP.start("example.com") do |req|

req.delete("/posts/1").body # => "Post 1 deleted"

req.delete("/posts/1").body # => "Post not found"

req.delete("/posts/1").body # => "Post not found"

end

Using HTTP basic authentication

You can fake requests that use basic authentication by adding userinfo strings to your URIs:

FakeWeb.register\_uri(:get, "http://example.com/secret", :body => "Unauthorized", :status => ["401", "Unauthorized"])

FakeWeb.register\_uri(:get, "http://user:pass@example.com/secret", :body => "Authorized")

Net::HTTP.start("example.com") do |http|

req = Net::HTTP::Get.new("/secret")

http.request(req) # => "Unauthorized"

req.basic\_auth("user", "pass")

http.request(req) # => "Authorized"

end

Clearing registered URIs

The [FakeWeb](http://fakeweb.rubyforge.org/classes/FakeWeb.html) registry is a singleton that lasts for the duration of your program, maintaining every fake response you register. If needed, you can clean out the registry and remove all registered URIs:

FakeWeb.clean\_registry

Blocking all real requests

When you’re using [FakeWeb](http://fakeweb.rubyforge.org/classes/FakeWeb.html) to replace *all* of your requests, it’s useful to catch when requests are made for unregistered URIs (unlike the default behavior, which is to pass those requests through to Net::HTTP as usual).

FakeWeb.allow\_net\_connect = false

Net::HTTP.get(URI.parse("http://example.com/"))

=> raises FakeWeb::NetConnectNotAllowedError

FakeWeb.allow\_net\_connect = true

Net::HTTP.get(URI.parse("http://example.com/"))

=> FakeWeb is bypassed and the response from a real request is returned

It’s recommended that you set FakeWeb.allow\_net\_connect = false in the setup for your tests.

**Allowing requests to a specific server**

If you want to prevent your tests from hitting the internet while allowing access to a specific server for integration testing, you can assign a URI or Regexp to be used as a whitelist for outbound requests:

FakeWeb.allow\_net\_connect = %r[^https?://localhost]

Net::HTTP.get(URI.parse("http://localhost/path")) # => allowed

Net::HTTP.get(URI.parse("http://example.com/")) # => raises FakeWeb::NetConnectNotAllowedError

Specifying HTTP response headers

When you register a response using the :body option, you’re only setting the body of the response. If you want to add headers to these responses, simply add the header as an option toregister\_uri:

FakeWeb.register\_uri(:get, "http://example.com/hello.txt", :body => "Hello", :content\_type => "text/plain")

This sets the “Content-Type” header in the response.

Checking the last request

It’s often useful to retrieve the last request made by your code, so you can write tests for its content. [FakeWeb](http://fakeweb.rubyforge.org/classes/FakeWeb.html) keeps track of the last request, whether it was stubbed or not:

Net::HTTP.get(URI.parse("http://example.com"))

FakeWeb.last\_request # => Net::HTTP::Get request object

More info

[FakeWeb](http://fakeweb.rubyforge.org/classes/FakeWeb.html) lets you decouple your test environment from live services without modifying code or writing extensive stubs.

In addition to the conceptual advantage of having idempotent request behaviour, [FakeWeb](http://fakeweb.rubyforge.org/classes/FakeWeb.html) makes tests run faster than if they were made to remote (or even local) web servers. It also makes it possible to run tests without a network connection or in situations where the server is behind a firewall or has host-based access controls.

[FakeWeb](http://fakeweb.rubyforge.org/classes/FakeWeb.html) works with anything based on Net::HTTP—both higher-level wrappers, like OpenURI, as well as a ton of libraries for popular web services.

Known Issues

* Request bodies are ignored, including PUT and POST parameters. If you need different responses for different request bodies, you need to request different URLs, and register different responses for each. (Query strings are fully supported, though.) We’re currently considering how the API should change to add support for request bodies in 1.3.0. Your input would be really helpful: see [groups.google.com/group/fakeweb-users/browse\_thread/thread/44d190a6b12e4273](http://groups.google.com/group/fakeweb-users/browse_thread/thread/44d190a6b12e4273) for a discussion of some different options. Thanks!