Sinatra::Base - Middleware, Libraries, and Modular Apps

Defining your app at the top-level works well for micro-apps but has considerable drawbacks when building reusable components such as Rack middleware, Rails metal, simple libraries with a server component, or even Sinatra extensions. The top-level assumes a micro-app style configuration (e.g., a single application file, ./public and ./views directories, logging, exception detail page, etc.). That's where Sinatra::Base comes into play:

```
require 'sinatra/base'

class MyApp < Sinatra::Base
  set :sessions, true
  set :foo, 'bar'

  get '/' do
    'Hello world!'
  end
end
```

The methods available to Sinatra::Base subclasses are exactly the same as those available via the top-level DSL. Most top-level apps can be converted to Sinatra::Base components with two modifications:

- Your file should require sinatra/base instead of sinatra; otherwise, all of Sinatra's DSL methods are imported into the main namespace.
- Put your app's routes, error handlers, filters, and options in a subclass of Sinatra::Base.

Sinatra::Base is a blank slate. Most options are disabled by default, including the built-in server. See Configuring Settings for details on available options and their behavior. If you want behavior more similar to when you define your app at the top level (also known as Classic style), you can subclass Sinatra::Application.

```
require 'sinatra/base'

class MyApp < Sinatra::Application
  get '/' do
    'Hello world!'
  end
end
```

Modular vs. Classic Style

Contrary to common belief, there is nothing wrong with the classic style. If it suits your application, you do not have to switch to a modular application.

The main disadvantage of using the classic style rather than the modular style is that you will only have one Sinatra application per Ruby process. If you plan to use more than one, switch to the modular style. There is no reason you cannot mix the modular and the classic styles.

If switching from one style to the other, you should be aware of slightly different default settings:

Setting	Classic	Modular	Modular

app_file file loading sinatra file subclassing Sinatra::Base file subclassing Sinatra::Application run $0 == app_file false false logging true false true method_override true false true inline_templates true false true static true false true

Serving a Modular Application

There are two common options for starting a modular app, actively starting with run!:

```
# my_app.rb
require 'sinatra/base'

class MyApp < Sinatra::Base
  # ... app code here ...

  # start the server if ruby file executed directly
  run! if app_file == $0
end
```

Start with:

```
ruby my_app.rb
```

Or with a config.ru file, which allows using any Rack handler:

```
# config.ru (run with rackup)
require './my_app'
run MyApp
```

Run:

```
rackup -p 4567
```

Using a Classic Style Application with a config.ru

Write your app file:

```
# app.rb
require 'sinatra'

get '/' do
  'Hello world!'
end
```

And a corresponding config.ru:

```
require './app'
run Sinatra::Application
```

When to use a config.ru?

A config.ru file is recommended if:

- You want to deploy with a different Rack handler (Passenger, Unicorn, Heroku, ...).
- You want to use more than one subclass of Sinatra::Base.
- You want to use Sinatra only for middleware, and not as an endpoint.

There is no need to switch to a config.ru simply because you switched to the modular style, and you don't have to use the modular style for running with a config.ru.

Using Sinatra as Middleware

Not only is Sinatra able to use other Rack middleware, any Sinatra application can in turn be added in front of any Rack endpoint as middleware itself. This endpoint could be another Sinatra application, or any other Rack-based application (Rails/Ramaze/Camping/...):

```
require 'sinatra/base'

class LoginScreen < Sinatra::Base
  enable :sessions

  get('/login') { haml :login }

  post('/login') do
    if params['name'] == 'admin' && params['password'] == 'admin'
      session['user_name'] = params['name']
    else
      redirect '/login'
    end
  end
end

class MyApp < Sinatra::Base
  # middleware will run before filters
  use LoginScreen

  before do
    unless session['user_name']
      halt "Access denied, please <a href='/login'>login</a>."
    end
  end

  get('/') { "Hello #{session['user_name']}." }
end
```

Dynamic Application Creation

Sometimes you want to create new applications at runtime without having to assign them to a constant. You can do this with Sinatra.new:

```
require 'sinatra/base'
my_app = Sinatra.new { get('/') { "hi" } }
my_app.run!
```

It takes the application to inherit from as an optional argument:

```
# config.ru (run with rackup)
require 'sinatra/base'

controller = Sinatra.new do
  enable :logging
  helpers MyHelpers
end

map('/a') do
  run Sinatra.new(controller) { get('/') { 'a' } }
end

map('/b') do
  run Sinatra.new(controller) { get('/') { 'b' } }
end
```

This is especially useful for testing Sinatra extensions or using Sinatra in your own library.

This also makes using Sinatra as middleware extremely easy:

```
require 'sinatra/base'

use Sinatra do
  get('/') { ... }
end

run RailsProject::Application
```