LeanDog
AN AGILE SOFTWARE STUDIO

Factory-girl Quick Reference

Defining Factories

You define a factory by simply calling the define method on the FactoryGirl module passing a block. Inside the block you can define individual data factories by calling the factory method like this:

```
FactoryGirl.define do
  factory :user do
    first_name ‘John’
    last_name ‘Doe’
    admin false
  end
end
```

By default the factory defined above will map to a class named User using the name we gave the factory. If you wish to use a different name than the class name you can define it by providing the class name like this:

```
FactoryGirl.define do
  factory :admin, :class => User do
    first_name ‘John’
    last_name ‘Doe’
    admin false
  end
end
```

Using Factories

Return a saved User instance user = FactoryGirl.create(:user)	Returns a User instance that's not saved user = FactoryGirl.build(:user)
Returns a hash of attributes that can be used to build a User instance attrs = FactoryGirl.attributes_for(:user)	Passing a block to any of these methods will yield the return object FactoryGirl.create(:user) do |user| user.posts.create(attributes_for(:post)) end
Override a value in the factory for a User instance user = FactoryGirl.create(:user, :first_name => 'Jared')	Returns a n object with all defined attributes stubbed out stub = FactoryGirl.build_stubbed(:user)

Transient Attributes

There may be times where your code can be DRYed up by passing in transient attributes to factories.

```
factory :user do
  ignore do
    upcased {false }
  end

  name "John Doe"

  after_create do |user, evaluator|
    user.name.upcase! if evaluator.upcased
  end
end

FactoryGirl.create(:user, :upcased => true)
```

Dynamic Attributes

Attributes can be based on the values of other attributes. Here's a simple example

```
factory :user do
  first_name 'Joe'
  last_name 'Blow'
  email { "#{first_name}.#{last_name}@example.com".downcase }
end
```

Unique values in a specific format can be generated using sequences. Sequences are defined by calling sequence in a definition block.

```
FactoryGirl.define do
  sequence :email do |n|
    "person#{n}@example.com"
  end

  factory :user do
    name "Jane Doe"
    email
  end
end
```

It's also possible to define an in-line sequence that is only used in a particular factory.

```
factory :user do
  sequence(:email) { |n| "person#{n}@example.com" }
end
```

Associations

You will more than likely need to setup associations within your factories. If the factory name is the same as the association, the factory name can be left off.

```
factory :post do
  # ...
  author
end
```

You can also specify a different factory or override attributes:

```
factory :post do
  # ...
  association :author, :factory => :user, :last_name => 'Writely'
end
```

Generating data for a has_many relationship is a bit more involved, depending on the amount of flexibility desired.

```
FactoryGirl.define do
  factory :post do
    title "Through the looking Glass"
    user
  end

  factory :user do
    name "John Doe"

    # user_with_posts will create post data after the user has been created
    factory :user_with_posts do
      ignore do
```

```
      posts_count 5
    end

    # the after_create yields two values;  the user instance itself and the evaluator,
    # which stores all values from the factory, including ignored attributes
    after_create do |user, evaluator|
      FactoryGirl.create_list(:post, evaluator.posts_count, :user => user)
    end
  end
 end
end
FactoryGirl.create(:user_with_posts, :posts_count => 15)
```

Aliases

Aliases allow you to use named associations more easily.

```
factory :user, :alias => [:author, :commenter] do
 first_name “John”
 last_name “Doe”
end

factory :post do
 author  # instead of association :author, :factory => :user
 title “How to read a book effectively”
 body “There are five steps involved”
end

factory :comment do
 commenter
 body “Great article!”
end
```

Callbacks

factory_girl makes available three callbacks for injectong some code:

- after_build - called after a factory is built (via FactoryGirl.build)
- after_create - called after a factory is saved (via FactoryGirl.create)
- after_stub - called after a factory is stubbed (via FactoryGirl.build_stubbed)

Building or creating multiple records

Sometimes, you'll want to create or build multiple instances of a factory at once.

```
built_users = FactoryGirl.build_list(:user, 25)
created_users = FactoryGirl.create_list(:user, 25)
```

These methods will build or create a specific amount of factories and return them as an array. To set the attributes of each of the factories, you can pass in a hash as you normally would.

```
twenty_year_olds = FactoryGirl.build_list(:user, 25, :date_of_birth => 20.years.ago)
```

Inheritance

You can easily create multiple factories from the same class without repeating common attributes by nesting factories:

```
factory :post do
  title "A Title"

  factory :approved_post do
    approved true
  end
end

approved_post = FactoryGirl.create(:approved_post)
```