DO NOT READ THIS FILE ON GITHUB, GUIDES ARE PUBLISHED ON https://guides.rubyonrails.org.

Ruby on Rails 3.0 Release Notes

Rails 3.0 is ponies and rainbows! It's going to cook you dinner and fold your laundry. You're going to wonder how life was ever possible before it arrived. It's the Best Version of Rails We've Ever Done!

But seriously now, it's really good stuff. There are all the good ideas brought over from when the Merb team joined the party and brought a focus on framework agnosticism, slimmer and faster internals, and a handful of tasty APIs. If you're coming to Rails 3.0 from Merb 1.x, you should recognize lots. If you're coming from Rails 2.x, you're going to love it too.

Even if you don't give a hoot about any of our internal cleanups, Rails 3.0 is going to delight. We have a bunch of new features and improved APIs. It's never been a better time to be a Rails developer. Some of the highlights are:

- Brand new router with an emphasis on RESTful declarations
- New Action Mailer API modeled after Action Controller (now without the agonizing pain of sending multipart messages!)
- New Active Record chainable query language built on top of relational algebra
- Unobtrusive JavaScript helpers with drivers for Prototype, jQuery, and more coming (end of inline JS)
- Explicit dependency management with Bundler

On top of all that, we've tried our best to deprecate the old APIs with nice warnings. That means that you can move your existing application to Rails 3 without immediately rewriting all your old code to the latest best practices.

These release notes cover the major upgrades, but don't include every little bug fix and change. Rails 3.0 consists of almost 4,000 commits by more than 250 authors! If you want to see everything, check out the list of commits in the main Rails repository on GitHub.

To install Rails 3:

```
# Use sudo if your setup requires it
$ gem install rails
```

Upgrading to Rails 3

If you're upgrading an existing application, it's a great idea to have good test coverage before going in. You should also first upgrade to Rails 2.3.5 and make sure your application still runs as expected before attempting to update to Rails 3. Then take heed of the following changes:

Rails 3 requires at least Ruby 1.8.7

Rails 3.0 requires Ruby 1.8.7 or higher. Support for all of the previous Ruby versions has been dropped officially and you should upgrade as early as possible. Rails 3.0 is also compatible with Ruby 1.9.2.

TIP: Note that Ruby 1.8.7 p248 and p249 have marshalling bugs that crash Rails 3.0. Ruby Enterprise Edition have these fixed since release 1.8.7-2010.02 though. On the 1.9 front, Ruby 1.9.1 is not usable because it outright segfaults on Rails 3.0, so if you want to use Rails 3 with 1.9.x jump on 1.9.2 for smooth sailing.

Rails Application object

As part of the groundwork for supporting running multiple Rails applications in the same process, Rails 3 introduces the concept of an Application object. An application object holds all the application specific configurations and is very similar in nature to `config/environment.rb` from the previous versions of Rails.

Each Rails application now must have a corresponding application object. The application object is defined in `config/application.rb`. If you're upgrading an existing application to Rails 3, you must add this file and move the appropriate configurations from `config/environment.rb` to `config/application.rb`.

script/* replaced by script/rails

The new `script/rails` replaces all the scripts that used to be in the `script` directory. You do not run `script/rails` directly though, the `rails` command detects it is being invoked in the root of a Rails application and runs the script for you. Intended usage is:

```
$ rails console                       # instead of script/console
$ rails g scaffold post title:string # instead of script/generate scaffold post
title:string
```

Run `rails --help` for a list of all the options.

Dependencies and config.gem

The `config.gem` method is gone and has been replaced by using `bundler` and a `Gemfile`, see Vendoring Gems below.

Upgrade Process

To help with the upgrade process, a plugin named Rails Upgrade has been created to automate part of it.

Simply install the plugin, then run `rake rails:upgrade:check` to check your app for pieces that need to be updated (with links to information on how to update them). It also offers a task to generate a `Gemfile` based on your current `config.gem` calls and a task to generate a new routes file from your current one. To get the plugin, simply run the following:

```
$ ruby script/plugin install git://github.com/rails/rails_upgrade.git
```

You can see an example of how that works at Rails Upgrade is now an Official Plugin

Aside from Rails Upgrade tool, if you need more help, there are people on IRC and rubyonrails-talk that are probably doing the same thing, possibly hitting the same issues. Be sure to blog your own experiences when upgrading so others can benefit from your knowledge!

Creating a Rails 3.0 application

```
# You should have the 'rails' RubyGem installed
$ rails new myapp
$ cd myapp
```

Vendoring Gems

Rails now uses a `Gemfile` in the application root to determine the gems you require for your application to start. This `Gemfile` is processed by the Bundler which then installs all your dependencies. It can even install all the dependencies locally to your application so that it doesn't depend on the system gems.

More information: - bundler homepage

Living on the Edge

`Bundler` and `Gemfile` makes freezing your Rails application easy as pie with the new dedicated `bundle` command, so `rake freeze` is no longer relevant and has been dropped.

If you want to bundle straight from the Git repository, you can pass the `--edge` flag:

```
$ rails new myapp --edge
```

If you have a local checkout of the Rails repository and want to generate an application using that, you can pass the `--dev` flag:

```
$ ruby /path/to/rails/bin/rails new myapp --dev
```

Rails Architectural Changes

There are six major changes in the architecture of Rails.

Railties Restrung

Railties was updated to provide a consistent plugin API for the entire Rails framework as well as a total rewrite of generators and the Rails bindings, the result is that developers can now hook into any significant stage of the generators and application framework in a consistent, defined manner.

All Rails core components are decoupled

With the merge of Merb and Rails, one of the big jobs was to remove the tight coupling between Rails core components. This has now been achieved, and all Rails core components are now using the same API that you can use for developing plugins. This means any plugin you make, or any core component replacement (like DataMapper or Sequel) can access all the functionality that the Rails core components have access to and extend and enhance at will.

More information: - The Great Decoupling

Active Model Abstraction

Part of decoupling the core components was extracting all ties to Active Record from Action Pack. This has now been completed. All new ORM plugins now just need to implement Active Model interfaces to work seamlessly with Action Pack.

More information: - Make Any Ruby Object Feel Like ActiveRecord

Controller Abstraction

Another big part of decoupling the core components was creating a base superclass that is separated from the notions of HTTP in order to handle rendering of views, etc. This creation of `AbstractController` allowed

`ActionController` and `ActionMailer` to be greatly simplified with common code removed from all these libraries and put into Abstract Controller.

More Information: - Rails Edge Architecture

Arel Integration

Arel (or Active Relation) has been taken on as the underpinnings of Active Record and is now required for Rails. Arel provides an SQL abstraction that simplifies out Active Record and provides the underpinnings for the relation functionality in Active Record.

More information: - Why I wrote Arel

Mail Extraction

Action Mailer ever since its beginnings has had monkey patches, pre parsers and even delivery and receiver agents, all in addition to having TMail vendored in the source tree. Version 3 changes that with all email message related functionality abstracted out to the Mail gem. This again reduces code duplication and helps create definable boundaries between Action Mailer and the email parser.

More information: - New Action Mailer API in Rails 3

Documentation

The documentation in the Rails tree is being updated with all the API changes, additionally, the Rails Edge Guides are being updated one by one to reflect the changes in Rails 3.0. The guides at guides.rubyonrails.org however will continue to contain only the stable version of Rails (at this point, version 2.3.5, until 3.0 is released).

More Information: - Rails Documentation Projects

Internationalization

A large amount of work has been done with I18n support in Rails 3, including the latest I18n gem supplying many speed improvements.

- I18n for any object - I18n behavior can be added to any object by including `ActiveModel::Translation` and `ActiveModel::Validations`. There is also an `errors.messages` fallback for translations.
- Attributes can have default translations.
- Form Submit Tags automatically pull the correct status (Create or Update) depending on the object status, and so pull the correct translation.
- Labels with I18n also now work by just passing the attribute name.

More Information: - Rails 3 I18n changes

Railties

With the decoupling of the main Rails frameworks, Railties got a huge overhaul so as to make linking up frameworks, engines, or plugins as painless and extensible as possible:

- Each application now has its own name space, application is started with `YourAppName.boot` for example, makes interacting with other applications a lot easier.

- Anything under `Rails.root/app` is now added to the load path, so you can make `app/observers/user_observer.rb` and Rails will load it without any modifications.
- Rails 3.0 now provides a `Rails.config` object, which provides a central repository of all sorts of Rails wide configuration options.

  Application generation has received extra flags allowing you to skip the installation of test-unit, Active Record, Prototype and Git. Also a new `--dev` flag has been added which sets the application up with the `Gemfile` pointing to your Rails checkout (which is determined by the path to the `rails` binary). See `rails --help` for more info.

Railties generators got a huge amount of attention in Rails 3.0, basically:

- Generators were completely rewritten and are backwards incompatible.
- Rails templates API and generators API were merged (they are the same as the former).
- Generators are no longer loaded from special paths anymore, they are just found in the Ruby load path, so calling `rails generate foo` will look for `generators/foo_generator`.
- New generators provide hooks, so any template engine, ORM, test framework can easily hook in.
- New generators allow you to override the templates by placing a copy at `Rails.root/lib/templates`.
- `Rails::Generators::TestCase` is also supplied so you can create your own generators and test them.

Also, the views generated by Railties generators had some overhaul:

- Views now use `div` tags instead of `p` tags.
- Scaffolds generated now make use of `_form` partials, instead of duplicated code in the edit and new views.
- Scaffold forms now use `f.submit` which returns "Create ModelName" or "Update ModelName" depending on the state of the object passed in.

Finally a couple of enhancements were added to the rake tasks:

- `rake db:forward` was added, allowing you to roll forward your migrations individually or in groups.
- `rake routes CONTROLLER=x` was added allowing you to just view the routes for one controller.

Railties now deprecates:

- `RAILS_ROOT` in favor of `Rails.root`,
- `RAILS_ENV` in favor of `Rails.env`, and
- `RAILS_DEFAULT_LOGGER` in favor of `Rails.logger`.

`PLUGIN/rails/tasks`, and `PLUGIN/tasks` are no longer loaded all tasks now must be in `PLUGIN/lib/tasks`.

More information:

- Discovering Rails 3 generators
- The Rails Module (in Rails 3)

Action Pack

There have been significant internal and external changes in Action Pack.

Abstract Controller

Abstract Controller pulls out the generic parts of Action Controller into a reusable module that any library can use to render templates, render partials, helpers, translations, logging, any part of the request response cycle. This abstraction allowed `ActionMailer::Base` to now just inherit from `AbstractController` and just wrap the Rails DSL onto the Mail gem.

It also provided an opportunity to clean up Action Controller, abstracting out what could to simplify the code.

Note however that Abstract Controller is not a user facing API, you will not run into it in your day to day use of Rails.

More Information: - Rails Edge Architecture

Action Controller

- `application_controller.rb` now has `protect_from_forgery` on by default.
- The `cookie_verifier_secret` has been deprecated and now instead it is assigned through `Rails.application.config.cookie_secret` and moved into its own file: `config/initializers/cookie_verification_secret.rb` .
- The `session_store` was configured in `ActionController::Base.session` , and that is now moved to `Rails.application.config.session_store` . Defaults are set up in `config/initializers/session_store.rb` .
- `cookies.secure` allowing you to set encrypted values in cookies with `cookie.secure[:key] => value` .
- `cookies.permanent` allowing you to set permanent values in the cookie hash `cookie.permanent[:key] => value` that raise exceptions on signed values if verification failures.
- You can now pass `:notice => 'This is a flash message'` or `:alert => 'Something went wrong'` to the `format` call inside a `respond_to` block. The `flash[]` hash still works as previously.
- `respond_with` method has now been added to your controllers simplifying the venerable `format` blocks.
- `ActionController::Responder` added allowing you flexibility in how your responses get generated.

Deprecations:

- `filter_parameter_logging` is deprecated in favor of `config.filter_parameters << :password` .

More Information:

- Render Options in Rails 3
- Three reasons to love ActionController::Responder

Action Dispatch

Action Dispatch is new in Rails 3.0 and provides a new, cleaner implementation for routing.

- Big clean up and re-write of the router, the Rails router is now `rack_mount` with a Rails DSL on top, it is a stand alone piece of software.
- Routes defined by each application are now name spaced within your Application module, that is:

```
# Instead of:

ActionController::Routing::Routes.draw do |map|
  map.resources :posts
end
```

```
# You do:

AppName::Application.routes do
  resources :posts
end
```

- Added `match` method to the router, you can also pass any Rack application to the matched route.
- Added `constraints` method to the router, allowing you to guard routers with defined constraints.
- Added `scope` method to the router, allowing you to namespace routes for different languages or different actions, for example:

```
scope 'es' do
  resources :projects, :path_names => { :edit => 'cambiar' }, :path =>
'proyecto'
end

# Gives you the edit action with /es/proyecto/1/cambiar
```

- Added `root` method to the router as a short cut for `match '/', :to => path`.
- You can pass optional segments into the match, for example `match "/:controller(/:action(/:id))(.:format)"`, each parenthesized segment is optional.
- Routes can be expressed via blocks, for example you can call `controller :home { match '/:action' }`.

NOTE. The old style `map` commands still work as before with a backwards compatibility layer, however this will be removed in the 3.1 release.

Deprecations

- The catch all route for non-REST applications (`/:controller/:action/:id`) is now commented out.
- Routes `:path_prefix` no longer exists and `:name_prefix` now automatically adds "_" at the end of the given value.

More Information:

- The Rails 3 Router: Rack it Up
- Revamped Routes in Rails 3
- Generic Actions in Rails 3

Action View

Unobtrusive JavaScript

Major re-write was done in the Action View helpers, implementing Unobtrusive JavaScript (UJS) hooks and removing the old inline AJAX commands. This enables Rails to use any compliant UJS driver to implement the UJS hooks in the helpers.

What this means is that all previous `remote_<method>` helpers have been removed from Rails core and put into the Prototype Legacy Helper. To get UJS hooks into your HTML, you now pass `:remote => true` instead. For

example:

```
form_for @post, :remote => true
```

Produces:

```
<form action="http://host.com" id="create-post" method="post" data-remote="true">
```

Helpers with Blocks

Helpers like `form_for` or `div_for` that insert content from a block use `<%=` now:

```
<%= form_for @post do |f| %>
  ...
<% end %>
```

Your own helpers of that kind are expected to return a string, rather than appending to the output buffer by hand.

Helpers that do something else, like `cache` or `content_for`, are not affected by this change, they need `<%` as before.

Other Changes

- You no longer need to call `h(string)` to escape HTML output, it is on by default in all view templates. If you want the unescaped string, call `raw(string)`.
- Helpers now output HTML5 by default.
- Form label helper now pulls values from I18n with a single value, so `f.label :name` will pull the `:name` translation.
- I18n select label on should now be :en.helpers.select instead of :en.support.select.
- You no longer need to place a minus sign at the end of a Ruby interpolation inside an ERB template to remove the trailing carriage return in the HTML output.
- Added `grouped_collection_select` helper to Action View.
- `content_for?` has been added allowing you to check for the existence of content in a view before rendering.
- passing `:value => nil` to form helpers will set the field's `value` attribute to nil as opposed to using the default value
- passing `:id => nil` to form helpers will cause those fields to be rendered with no `id` attribute
- passing `:alt => nil` to `image_tag` will cause the `img` tag to render with no `alt` attribute

Active Model

Active Model is new in Rails 3.0. It provides an abstraction layer for any ORM libraries to use to interact with Rails by implementing an Active Model interface.

ORM Abstraction and Action Pack Interface

Part of decoupling the core components was extracting all ties to Active Record from Action Pack. This has now been completed. All new ORM plugins now just need to implement Active Model interfaces to work seamlessly with Action Pack.

More Information: - Make Any Ruby Object Feel Like ActiveRecord

Validations

Validations have been moved from Active Record into Active Model, providing an interface to validations that works across ORM libraries in Rails 3.

- There is now a `validates :attribute, options_hash` shortcut method that allows you to pass options for all the validates class methods, you can pass more than one option to a validate method.
- The `validates` method has the following options:
  - `:acceptance => Boolean`.
  - `:confirmation => Boolean`.
  - `:exclusion => { :in => Enumerable }`.
  - `:inclusion => { :in => Enumerable }`.
  - `:format => { :with => Regexp, :on => :create }`.
  - `:length => { :maximum => Fixnum }`.
  - `:numericality => Boolean`.
  - `:presence => Boolean`.
  - `:uniqueness => Boolean`.

NOTE: All the Rails version 2.3 style validation methods are still supported in Rails 3.0, the new validates method is designed as an additional aid in your model validations, not a replacement for the existing API.

You can also pass in a validator object, which you can then reuse between objects that use Active Model:

```
class TitleValidator < ActiveModel::EachValidator
  Titles = ['Mr.', 'Mrs.', 'Dr.']
  def validate_each(record, attribute, value)
    unless Titles.include?(value)
      record.errors[attribute] << 'must be a valid title'
    end
  end
end
```

```
class Person
  include ActiveModel::Validations
  attr_accessor :title
  validates :title, :presence => true, :title => true
end

# Or for Active Record

class Person < ActiveRecord::Base
  validates :title, :presence => true, :title => true
end
```

There's also support for introspection:

```
User.validators
User.validators_on(:login)
```

More Information:

- Sexy Validation in Rails 3
- Rails 3 Validations Explained

Active Record

Active Record received a lot of attention in Rails 3.0, including abstraction into Active Model, a full update to the Query interface using Arel, validation updates, and many enhancements and fixes. All of the Rails 2.x API will be usable through a compatibility layer that will be supported until version 3.1.

Query Interface

Active Record, through the use of Arel, now returns relations on its core methods. The existing API in Rails 2.3.x is still supported and will not be deprecated until Rails 3.1 and not removed until Rails 3.2, however, the new API provides the following new methods that all return relations allowing them to be chained together:

- `where` - provides conditions on the relation, what gets returned.
- `select` - choose what attributes of the models you wish to have returned from the database.
- `group` - groups the relation on the attribute supplied.
- `having` - provides an expression limiting group relations (GROUP BY constraint).
- `joins` - joins the relation to another table.
- `clause` - provides an expression limiting join relations (JOIN constraint).
- `includes` - includes other relations pre-loaded.
- `order` - orders the relation based on the expression supplied.
- `limit` - limits the relation to the number of records specified.
- `lock` - locks the records returned from the table.
- `readonly` - returns an read only copy of the data.
- `from` - provides a way to select relationships from more than one table.
- `scope` - (previously `named_scope`) return relations and can be chained together with the other relation methods.
- `with_scope` - and `with_exclusive_scope` now also return relations and so can be chained.
- `default_scope` - also works with relations.

More Information:

- Active Record Query Interface
- Let your SQL Growl in Rails 3

Enhancements

- Added `:destroyed?` to Active Record objects.
- Added `:inverse_of` to Active Record associations allowing you to pull the instance of an already loaded association without hitting the database.

Patches and Deprecations

Additionally, many fixes in the Active Record branch:

- SQLite 2 support has been dropped in favor of SQLite 3.
- MySQL support for column order.
- PostgreSQL adapter has had its `TIME ZONE` support fixed so it no longer inserts incorrect values.
- Support multiple schemas in table names for PostgreSQL.

- PostgreSQL support for the XML data type column.
- `table_name` is now cached.
- A large amount of work done on the Oracle adapter as well with many bug fixes.

As well as the following deprecations:

- `named_scope` in an Active Record class is deprecated and has been renamed to just `scope`.
- In `scope` methods, you should move to using the relation methods, instead of a `:conditions => {}` finder method, for example `scope :since, lambda {|time| where("created_at > ?", time)}`.
- `save(false)` is deprecated, in favor of `save(:validate => false)`.
- I18n error messages for Active Record should be changed from :en.activerecord.errors.template to `:en.errors.template`.
- `model.errors.on` is deprecated in favor of `model.errors[]`
- validates_presence_of => validates... :presence => true
- `ActiveRecord::Base.colorize_logging` and `config.active_record.colorize_logging` are deprecated in favor of `Rails::LogSubscriber.colorize_logging` or `config.colorize_logging`

NOTE: While an implementation of State Machine has been in Active Record edge for some months now, it has been removed from the Rails 3.0 release.

Active Resource

Active Resource was also extracted out to Active Model allowing you to use Active Resource objects with Action Pack seamlessly.

- Added validations through Active Model.
- Added observing hooks.
- HTTP proxy support.
- Added support for digest authentication.
- Moved model naming into Active Model.
- Changed Active Resource attributes to a Hash with indifferent access.
- Added `first`, `last` and `all` aliases for equivalent find scopes.
- `find_every` now does not return a `ResourceNotFound` error if nothing returned.
- Added `save!` which raises `ResourceInvalid` unless the object is `valid?`.
- `update_attribute` and `update_attributes` added to Active Resource models.
- Added `exists?`.
- Renamed `SchemaDefinition` to `Schema` and `define_schema` to `schema`.
- Use the `format` of Active Resources rather than the `content-type` of remote errors to load errors.
- Use `instance_eval` for schema block.
- Fix `ActiveResource::ConnectionError#to_s` when `@response` does not respond to #code or #message, handles Ruby 1.9 compatibility.
- Add support for errors in JSON format.
- Ensure `load` works with numeric arrays.
- Recognizes a 410 response from remote resource as the resource has been deleted.
- Add ability to set SSL options on Active Resource connections.
- Setting connection timeout also affects `Net::HTTP` `open_timeout`.

Deprecations:

- `save(false)` is deprecated, in favor of `save(:validate => false)`.
- Ruby 1.9.2: `URI.parse` and `.decode` are deprecated and are no longer used in the library.

Active Support

A large effort was made in Active Support to make it cherry pickable, that is, you no longer have to require the entire Active Support library to get pieces of it. This allows the various core components of Rails to run slimmer.

These are the main changes in Active Support:

- Large clean up of the library removing unused methods throughout.
- Active Support no longer provides vendored versions of TZInfo, Memcache Client and Builder. These are all included as dependencies and installed via the `bundle install` command.
- Safe buffers are implemented in `ActiveSupport::SafeBuffer`.
- Added `Array.uniq_by` and `Array.uniq_by!`.
- Removed `Array#rand` and backported `Array#sample` from Ruby 1.9.
- Fixed bug on `TimeZone.seconds_to_utc_offset` returning wrong value.
- Added `ActiveSupport::Notifications` middleware.
- `ActiveSupport.use_standard_json_time_format` now defaults to true.
- `ActiveSupport.escape_html_entities_in_json` now defaults to false.
- `Integer#multiple_of?` accepts zero as an argument, returns false unless the receiver is zero.
- `string.chars` has been renamed to `string.mb_chars`.
- `ActiveSupport::OrderedHash` now can de-serialize through YAML.
- Added SAX-based parser for XmlMini, using LibXML and Nokogiri.
- Added `Object#presence` that returns the object if it's `#present?` otherwise returns `nil`.
- Added `String#exclude?` core extension that returns the inverse of `#include?`.
- Added `to_i` to `DateTime` in `ActiveSupport` so `to_yaml` works correctly on models with `DateTime` attributes.
- Added `Enumerable#exclude?` to bring parity to `Enumerable#include?` and avoid if `!x.include?`.
- Switch to on-by-default XSS escaping for rails.
- Support deep-merging in `ActiveSupport::HashWithIndifferentAccess`.
- `Enumerable#sum` now works will all enumerables, even if they don't respond to `:size`.
- `inspect` on a zero length duration returns '0 seconds' instead of empty string.
- Add `element` and `collection` to `ModelName`.
- `String#to_time` and `String#to_datetime` handle fractional seconds.
- Added support to new callbacks for around filter object that respond to `:before` and `:after` used in before and after callbacks.
- The `ActiveSupport::OrderedHash#to_a` method returns an ordered set of arrays. Matches Ruby 1.9's `Hash#to_a`.
- `MissingSourceFile` exists as a constant but it is now just equal to `LoadError`.
- Added `Class#class_attribute`, to be able to declare a class-level attribute whose value is inheritable and overwritable by subclasses.
- Finally removed `DeprecatedCallbacks` in `ActiveRecord::Associations`.
- `Object#metaclass` is now `Kernel#singleton_class` to match Ruby.

The following methods have been removed because they are now available in Ruby 1.8.7 and 1.9.

- `Integer#even?` and `Integer#odd?`
- `String#each_char`

- `String#start_with?` and `String#end_with?` (3rd person aliases still kept)
- `String#bytesize`
- `Object#tap`
- `Symbol#to_proc`
- `Object#instance_variable_defined?`
- `Enumerable#none?`

The security patch for REXML remains in Active Support because early patch-levels of Ruby 1.8.7 still need it. Active Support knows whether it has to apply it or not.

The following methods have been removed because they are no longer used in the framework:

- `Kernel#daemonize`
- `Object#remove_subclasses_of` `Object#extend_with_included_modules_from`, `Object#extended_by`
- `Class#remove_class`
- `Regexp#number_of_captures`, `Regexp.unoptionalize`, `Regexp.optionalize`, `Regexp#number_of_captures`

Action Mailer

Action Mailer has been given a new API with TMail being replaced out with the new Mail as the email library. Action Mailer itself has been given an almost complete re-write with pretty much every line of code touched. The result is that Action Mailer now simply inherits from Abstract Controller and wraps the Mail gem in a Rails DSL. This reduces the amount of code and duplication of other libraries in Action Mailer considerably.

- All mailers are now in `app/mailers` by default.
- Can now send email using new API with three methods: `attachments`, `headers` and `mail`.
- Action Mailer now has native support for inline attachments using the `attachments.inline` method.
- Action Mailer emailing methods now return `Mail::Message` objects, which can then be sent the `deliver` message to send itself.
- All delivery methods are now abstracted out to the Mail gem.
- The mail delivery method can accept a hash of all valid mail header fields with their value pair.
- The `mail` delivery method acts in a similar way to Action Controller's `respond_to`, and you can explicitly or implicitly render templates. Action Mailer will turn the email into a multipart email as needed.
- You can pass a proc to the `format.mime_type` calls within the mail block and explicitly render specific types of text, or add layouts or different templates. The `render` call inside the proc is from Abstract Controller and supports the same options.
- What were mailer unit tests have been moved to functional tests.
- Action Mailer now delegates all auto encoding of header fields and bodies to Mail Gem
- Action Mailer will auto encode email bodies and headers for you

Deprecations:

- `:charset`, `:content_type`, `:mime_version`, `:implicit_parts_order` are all deprecated in favor of `ActionMailer.default :key => value` style declarations.
- Mailer dynamic `create_method_name` and `deliver_method_name` are deprecated, just call `method_name` which now returns a `Mail::Message` object.
- `ActionMailer.deliver(message)` is deprecated, just call `message.deliver`.
- `template_root` is deprecated, pass options to a render call inside a proc from the `format.mime_type` method inside the `mail` generation block

- The `body` method to define instance variables is deprecated (`body {:ivar => value}`), just declare instance variables in the method directly and they will be available in the view.
- Mailers being in `app/models` is deprecated, use `app/mailers` instead.

More Information:

- New Action Mailer API in Rails 3
- New Mail Gem for Ruby

Credits

See the full list of contributors to Rails for the many people who spent many hours making Rails 3. Kudos to all of them.

Rails 3.0 Release Notes were compiled by Mikel Lindsaar.