From Technologies to Solutions

Ruby on Rails
Web Mashup Projects

A step-by-step tutorial to building web mashups

Chang Sau Sheong

PACKT PUBLISHING

Ruby on Rails Web Mashup Projects

A step-by-step tutorial to building web mashups

Chang Sau Sheong

PACKT PUBLISHING

BIRMINGHAM - MUMBAI

Ruby on Rails Web Mashup Projects

A step-by-step tutorial to building web mashups

First published: April 2008

Production Reference: 1160408

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-847193-93-3

`www.packtpub.com`

Cover Image by Raghuram Ashok (`raghuram@iiitb.ac.in`)

Credits

Author
Chang Sau Sheong

Reviewer
Walt Stoneburner

Senior Acquisition Editor
Douglas Paterson

Development Editor
Nikhil Bangera

Technical Editor
James Lumsden

Editorial Team Leader
Mithil Kulkarni

Project Manager
Abhijeet Deobhakta

Project Coordinators
Aboli Mendhe
Lata Basantani

Indexer
Monica Ajmera

Proofreader
Chris Smith

Production Coordinator
Shantanu Zagade

Cover Work
Shantanu Zagade

About the Author

Chang Sau Sheong has more than 12 years experience in software application development and has spent much of his career in Web and Internet-based applications. He has a wide range of experience in banking payment-related as well as Internet-based e-commerce software. Currently he is the Director of Software Development of a 50+ strong software development team in Welcome Real-time, a multi-national payment/loyalty software company based in France and Singapore.

Sau Sheong frequently writes for technical magazines and journals including Java Report, Java World, and Dr. Dobb's Journal. He also contributes to open-source projects in various technologies including smart cards, Ruby, and Java. His interests revolve mainly around technology and software development. He has done programming in Java/Java EE, C, C++, PHP, Python, Perl, Smalltalk, Erlang, Ruby/ Ruby on Rails, various smart card platforms, and also worked on various databases. He has a wide range of experience in banking payment-related as well as Internet-based e-commerce software.

Sau Sheong hails from tropical Malaysia but has spent most of his adult and working life in sunny Singapore, where he shares his spare time enthusiastically writing software and equally playing Nintendo Wii with his wife and son. He has a Bachelor's degree in Computer Engineering, a Master's degree in Commercial Law, and is a certified international arbitrator.

Acknowledgements

Firstly, many thanks to Douglas Peterson, Nikhil Bangera, Walt Stoneburner, and James Lumsden who patiently guided me through my first book. I would also like to thank the following people from the Singapore Ruby Brigade who contributed comments and reviews on this book as well as general support: Peter Bohm, Chew Choon Keat, Herryanto Siatono, Johan Gozali, Jeffrey Lim, Wong Keng Onn, and many others from the 'last Thursday of the month' sessions. Grateful thanks also to Mech, Watt, and Simon for their good support and general cheerleading in Rails and non-Rails related matters. Special thanks to Sebastien Guillaud and all those people in Welcome Real-time who believed that I can run software development center, write a book, and go home in time to tutor my son.

Final thanks to the love of my life, Wooi Ying, who suffered my erratic 'nightlife' huddling in front of my laptop, creating software and writing this book, and also to Kai Wen for just being my son.

About the Reviewer

Walt Stoneburner is a software architect with over 20 years of commercial application development and consulting experience. Fringe passions involve quality assurance, configuration management, and security. If cornered, he may actually admit to liking statistics and authoring documentation as well.

He's easily amused by programming language design, collaborative applications, and ASCII art. Self-described as a closet geek, Walt also evaluates software products and consumer electronics, draws cartoons, produces photography, writes humor pieces, performs sleight of hand, enjoys game design, and can occasionally be found on ham radio.

Walt may be reached directly via email at `wls@wwco.com`. He publishes a tech and humor blog called the Walt-O-Matic at `http://www.wwco.com/~wls/blog/`. Rumors suggest that some of his strange videography may be found on iTunes.

Currently he is employed at Business & Engineering Systems Corporation as a lead engineer developing advanced software solutions for knowledge management.

Other book reviews and contributions include *AntiPatterns and Patterns in Software Configuration Management* (ISBN 978-0-471-32929-9, p. xi) and *Exploiting Software: How to Break Code* (ISBN 978-0-201-78695-8, p. xxxiii).

Table of Contents

Preface

A web mashup is a new type of web application that uses data and services from one or more external sources to build entirely new and different web applications. Web mashups usually mash up data and services that are available on the Internet—freely, commercially, or through other partnership agreements. The external sources that a mashup uses are known as mashup APIs.

This book shows you how to write web mashups using Ruby on Rails—the new web application development framework. The book has seven real-world projects—the format of each project is similar, with a statement of the project, discussion of the main protocols involved, an overview of the API, and then complete code for building the project. You will be led methodically through concrete steps to build the mashup, with asides to explain the theory behind the code.

What This Book Covers

The first chapter introduces the concepts of web mashups to the reader and provides a general introduction to the benefits and pitfalls of using web mashups as stand-alone applications or as part of existing web applications.

The first project is a mashup plugin into an existing web application that allows users to find the location of the closest facility from a particular geographic location based on a specified search radius. The location is mapped and displayed on Google Maps.

The second project is another mashup plugin. This plugin allows users to send messages to their own list of recipients, people who are previously unknown to the website, on behalf of the website. The project uses Google Spreadsheets and EditGrid to aggregate the information, and Clickatell and Interfax to send SMS messages and faxes respectively.

The third project describes a mashup plugin that allows you to track the sales ranking and customer reviews of a particular product from Amazon.com. The main API used is the Amazon E-Commerce Service (ECS).

The fourth project shows you how to create a full-fledged Facebook application that allows a user to perform some of the functions and features of a job board. This mashup uses Facebook, Google Maps, Daylife, Technorati and Indeed.com APIs.

The fifth project shows you how to create a full web mashup application that allows users to view information on a location. This is the chapter that uses the most mashup APIs, including Google Maps, FUTEF, WebserviceX, Yahoo! geocoding services, WeatherBug, Kayak, GeoNames, Flickr, and Hostip.info.

The sixth project describes a mashup plugin that allows an online event ticketing application to receive payment through Paypal, send SMS receipts, and add event records in the customer's Google Calendar account. The APIs used are Google Calendar, PayPal, and Clickatell.

The final project shows a complex mashup plugin used for making corporate expense claims. It allows an employee to submit expense claims in Google Docs and Spreadsheets, attaching the claims form and the supporting receipts. His or her manager, also using Google Docs and Spreadsheets, then approves the expense claims and the approved claims are retrieved by the mashup and used to reimburse the employee through PayPal. It uses the PayPal APIs and various Google APIs.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

There are three styles for code. Code words in text are shown as follows: "This will copy the necessary files to your `RAILS_ROOT/vendor/plugins` folder and run the `install.rb` script."

A block of code will be set as follows:

```
class Kiosk < ActiveRecord::Base
  def address
  "#{self.street}, #{self.city}, #{self.state}, #{self.zipcode}"
  end
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be made bold:

```
begin
kiosks.each { |kiosk|
loc = MultiGeocoder.geocode(kiosk.address)
kiosk.lat = loc.lat
```

Any command-line input and output is written as follows:

```
$./script/plugin install svn://rubyforge.org/var/svn/geokit/trunk
```

New terms and **important words** are introduced in a bold-type font. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "clicking the **Next** button moves you to the next screen".

[Important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader Feedback

Feedback from our readers is always welcome. Let us know what you think about this book, what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an email to `feedback@packtpub.com`, making sure to mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or email `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer Support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the Example Code for the Book

Visit `http://www.packtpub.com/files/code/3933_Code.zip` to directly download the example code.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing this you can save other readers from frustration, and help to improve subsequent versions of this book. If you find any errata, report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be added to the list of existing errata. The existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with some aspect of the book, and we will do our best to address it.

1
Introduction to Web Mashups

Web mashups

Welcome to the world of web mashups! A web mashup is a new type of web application that uses data and services from one or more external sources (usually from the Internet) to build entirely new and different web applications. This book shows you how to write web mashups using Ruby on Rails—the new web application development framework.

The idea of taking data and services from various places and making them available in a single application is not new. Data feeds such as RSS and ATOM feeds have been around for a while, making information available for anyone to re-use in another application. Screen scraping was a commonly used older technology that takes content directly from another application's display. Portals where different data and services were aggregated into portlets and displayed on the portal were popular during the dot-com era. What's so different about web mashups?

The answer is that while older data and service aggregation technologies aggregate and integrate in a fashion, a true web mashup creates a completely different and new function out of the existing content and services, driving different purposes and objectives.

The word **mashup** itself comes from the world of hip-hop music, where two or more songs are mixed together to form a new song. Web mashups are primarily web applications (though it is not a strict requirement). Web mashups also usually mash up data and services that are available on the Internet—freely, commercially or through other partnership agreements. The external sources that a mashup uses are known as mashup APIs.

Ruby and Ruby on Rails

Ruby is a dynamic, object-oriented programming language that is highly suitable for integrating various pieces of data and software together:

- Ruby is designed for programmer productivity and can be used to quickly develop maintainable pieces of software.
- Ruby is interpreted in real time, meaning that whatever is coded can be executed immediately without compilation.
- Ruby has a significant number of libraries that can be easily re-used through the gem packaging mechanism.

Ruby on Rails is an open-source full stack web application framework built on Ruby. Ruby on Rails follows two basic guiding principles—Convention over Configuration and Don't Repeat Yourself (DRY).

Convention over Configuration is a programming design that favors following a certain set of programming conventions instead of configuring an application framework. Certain commonly used configurations (by convention and not by rule) are pre-set and the framework just works if you follow those conventions. For example in Ruby on Rails, the convention states that a controller for a model object `Book` will be called `BookController` and all view pages relating to that controller will be kept in a folder called `book`.

DRY is a principle that focuses on reducing information duplication by keeping any piece of knowledge in a system in only one place. For example, in ActiveRecord (a major component of Ruby on Rails), schema information doesn't need to be duplicated in complex XML configuration files but is derived from the database schema itself. If the schema changes, the model changes accordingly, without the need to make changes in other parts of the system.

All this translates into a highly productive development framework in which web applications can be developed, deployed, and maintained easily. This framework, coupled with the fact that it uses Ruby, makes it an excellent platform for developing web mashups.

For more in-depth discussion into Ruby's capabilities I would recommend you look at *Programming Ruby: The Pragmatic Programmer's Guide* by Dave Thomas, Chad Fowler, and Andy Hunt as well as *The Ruby Way, Second Edition: Solutions and Techniques in Ruby Programming* by Hal Fulton.

The recommended reading for Ruby on Rails is *Agile Web Development with Rails, Second Edition* by Dave Thomas and David Heinemeier Hansson.

A note of caution here—this book is written with Rails 1.2.x in mind and the projects and examples in this book follows this version. There is no significant change in the projects though, if you choose to use Rails 2.x instead. As of writing, Rails does not work with Ruby 1.9. If you're a complete beginner with Ruby and Ruby on Rails I would recommend you flip through the books mentioned in the information box opposite before plunging into this one.

Types of web mashups

There are some existing classifications of mashups in various literatures available on this subject though none are authoritative. In many cases, web mashups are categorized according to their functionality; for example, some define data mashups, photo and video mashups, news mashups, and business mashups. However, in this book, I classify web mashups by how they are used in building an application. From this perspective, we can see two broad types of web mashups:

- A fully standalone mashup application.
- An embedded mashup plugin.

A **mashup application** is a web mashup that provides a complete set of functions for the user. This means a mashup application is the entire purpose of the system. For example, a mashup might take data from Flickr, the photo storage and sharing application and mash it up with Google Maps, the online mapping application to display photos that come from a particular geographical area. By themselves, neither Flickr nor Google Maps are able to provide these features. However, this mashup's functionalities only come from combining both APIs; the web application cannot exist without the APIs. The functionality of the mashup is a synergistic product of creative usage of APIs from both sources.

A **mashup plugin**, on the other hand, only provides part of the functionality of an existing web application. For example a leave (time off work) submission and approval application's core functionality is to allow users to submit and approve leave as part of an HR process. A mashup plugin can be embedded into this application to allow an employee to apply optionally for leave through an online calendar and send a text message to the manager to alert him or her. The data from the online calendar is passed to the core application and also the text messaging APIs to enhance the value of the core application. However, the leave submission and approval application can still exist and its core functionality is not reduced without the mashup plugin.

The difference might not be apparent at first glance, but the thinking behind the mashups and their creation can be quite different. Mashup plugins are usually created to supplement an existing application that is probably not a mashup. They are a means of providing more services and data to the user of the application. Mashup applications, on the other hand, are created mashups in the first place and all the functionalities are derived from the mashup APIs they source from.

This has an interesting implication in developing web mashups. While many still regard web mashups as interesting technology toys and probably the latest buzzword alongside AJAX and Web 2.0, this classification of web mashups allow us to see mashups not just as Web 2.0 startup applications but potential value-added services for our existing applications. While mashup applications are an exciting and growing phenomenon on the Internet, mashup plugins will probably provide the most practical way of using mashups immediately within an existing environment.

What can I do with web mashups?

So what is in it for *you?* I assume you are a programmer, either professional or amateur, looking perhaps to extend your repertoire of skills and capabilities to develop and maintain software more easily, better, and faster.

Web mashups represent a new way of developing software and along with any new development techniques come opportunities and risks. Here's an example of what you can do with mashups:

- Create a platform for a new breed of applications
- Provide access to a large set of external data and service sources
- Innovate and create extra value for your existing applications quickly
- Save on development and maintenance
- Leverage on common or widely available external applications and integrate them into your application

As a new breed of applications

Web mashups are a new breed of web applications (Wikipedia defines a mashup as a web application hybrid). While most prominent web mashups use publicly available APIs like Google Maps, Amazon ECS, and so on, this is not the only way to do mashups. Significant innovation can be achieved with further aggregation and hybridization of code and data from publicly available APIs, with private data as well as private applications.

The idea behind web mashups is creativity and innovation in new data and services, not just aggregation of existing ones, which most of the older technologies focus on. In comparison with portals, web mashups differ because portals aggregate and dish out content and applications in discrete packages, whereas mashups integrate the data and services together and serve them out as a single application.

An example of this is that while a portal will happily display a map of your current location, your address book, and today's astrology readings in 3 different portlet windows, a mashup will display the astrology readings of 10 of your friends who are closest to you, in an online map occupying the whole browser space.

This integrated and mashed up approach to programming can provide much insight into the way we program applications.

Access large sets of external sources

There are an increasing number of applications on the Internet providing an amazing variety of data and services as APIs or data sets for mashups. A quick check on the Programmable Web (`http://www.programmableweb.com`), which hosts a directory listing mashup APIs as well as mashup applications, shows up service APIs ranging from social networks to sending snail mail through the Internet. You can also get tons of data from hotel bookings to government spending data.

With the wealth of these external data sources, you can build amazing new applications that bring these data and services into meaningful new services. While mashups are not the only way to consume large sets of external sources of data and services, they are probably the most creative. Buzzword aside, anytime you take data or services from another application, you're already doing a mashup.

Innovate and create extra value for your application

If you have an existing application already, web mashups can allow you to innovate and create new value to your application by grafting new functionality through the external sources. For example, if you run a reservation application, you can alert your user through text messages from an SMS mashup API, add the date of reservation into his or her Google Calendar account through Google Calendar APIs, and show the location of the venue on Yahoo Maps.

Save on development and maintenance

Using mashups you can build new functions much faster and save on the development and maintenance effort. For example, if you are the developer of a facilities reservation system you don't want to spend time mastering the development of a text message sending component, which you normally would have to do if you wanted to have that feature.

Besides development, you can also reduce the maintenance of a feature that is outside your core domain. While this is often critical for startups, it is equally important for larger organizations that want to focus on their core domain. In the example given earlier, you don't want to spend time developing and maintaining a text-messaging component—you'll want to leave it to the text-messaging experts to do their job.

Leverage on and integrate common and widely available external applications

Besides saving on effort, instead of doing it yourself—you might want to leverage on common and popular applications to do the work for you. Effort aside, such applications already have a widespread user base that is familiar with the functionality. You can tap these users to extend your own user base and use the features of these applications to give an easily recognizable interface for your users.

For example, if you want online calendaring features, you wouldn't want to redevelop another Google Calendar. Instead, you would mash up Google Calendar APIs into your application and use their interface to provide something more familiar to your users.

Things to watch out for when doing web mashups

With all the exciting talk on mashups, it's important to realize that, as with any new technology and way of programming, the road is usually fraught with dangers. Rightly the map around mashups should have bright neon lights flashing 'Here Be Dragons'. Here are some possible problems (but not all) you might face when developing web mashups:

- Unreliable external APIs
- Commercial dependency on third party data and services
- Losing your users to external source providers

Unreliable external APIs

One of the most common complaints you will encounter as you develop web mashups is that you are highly, if not totally, dependent of the reliability of the mashup APIs you use. The two critical aspects of a web application—availability and response time—are not under your control, especially from sources that are provided freely to you.

Unfortunately at this point in time there is no viable way of resolving this completely. The only way of ensuring full availability and response time that meets your own requirements is to not have external dependencies at all. This is not possible of course, because web mashups are all about using external data and services.

However there are a number of ways to work around this issue:

- Do not use mashups for mission-critical services. If the service is mission critical for you or your user, don't use mashups or at least not those that fail to guarantee certain availability and response time.
- Have an agreement (normally commercial) with your external mashup API provider that provides back-to-back service agreements with your own services. For example, if you promise 98% uptime, make sure you have an agreement with your provider that also agrees to 98% uptime.
- Design your mashup to have graceful error handling. This could range from a user-friendly error page to a caching system for data feeds and even a standby secondary service provider. For example, if you have a mashup that sends text messages to your users, you can do a mashup with more than one provider—if a provider fails you, quickly switch to another.

This issue is generally more difficult to accept in mashup applications because should core functionality of the system be compromised, it is difficult to proceed. In any case, catering and planning for backup or alternatives in case of an external source breakdown is a must for all mashups if you intend to go into production.

Commercial dependency

This problem is related to the first. Besides being dependent on the external APIs for functionality, the larger issue could be that the provider of the external API changes its service partially or completely. This could range from the provider being shut down altogether, to the provider changing its business model or commercial terms and it becoming no longer viable to continue with that provider anymore. Even simpler issues like changes in API parameters and accessibility can potentially cause service outage.

For example, a free service could start to charge a fee (or increase its existing fees) and you can no longer afford to include it in your mashup. Sometimes the service itself is no longer sustainable because of licensing issues or the company behind the service abandons its business model in pursuit of another revenue source.

This problem is more acute in areas where the provider is the only one around. Again, planning for alternatives is important if you intend to go into production because this problem can potentially kill your mashup altogether. Some possible defenses against this risk:

- Avoid using mashups in cases where there is only a single provider.
- Plan for backups and be alert to the happenings of the external providers you're using. Keep an active lookout for API changes as well as news on the company providing your sources. For example, if an online mapping provider is being bought by Google and you use either Google Maps or the online mapping provider's sources you should be wary that either one or both services are likely to change.
- Be aware of the competition available to the providers you're using and design your mashup for easy switching. For example, if you need an online map make sure you're familiar with more than just Google Maps and design your system to be able to switch to another online mapping service easily.

Losing your users

Another problem might not be related to reliability or availability at all. If your mashup becomes commercially interesting, it is sometimes quite easy for your external source API provider to extend their existing functionality to include yours and you to be left with an 800-pound gorilla in your backyard. A related risk is for your users to decide that if they are already using the external provider anyway, they might as well switch over to it completely and bypass your mashup altogether.

Again both risks are more likely for mashup applications since a mashup application's main value is in the creative combination of the external sources. Mashup plugins are less likely to encounter this because your main application already has functionality that should be different from external APIs (or else you might want to ask yourself why you're doing it!).

The main defense against such risks is to continually innovate and possibly include more APIs. A mashup application that combines two APIs creatively is more likely to be made irrelevant than a mashup application that combines three, four, five or more APIs and uses them in a creative way that none of your external providers can match by themselves. Remember that your main advantage in creating a mashup application is that you are able to be the best of breed by combining the best aspects and features of various providers to create a unique service for your users.

At the end of the day, although there are significant risks in creating web mashups, there are always ways of mitigating them. Ultimately it's up to you to decide if the risks are worth taking compared to the services you're providing in your mashup.

How this book works

The approach I use in this book is pragmatic and direct and tends to be hands-on. If you're not a practicing programmer, you might want to dust off your programming books and read them again!

There are seven mashup projects in this book, with one chapter per project. Each chapter has the following sections:

- What does it do?
- Domain background
- Requirements overview
- Design
- Mashup APIs on the menu
- What we will be doing
- Summary

Each chapter explains the technical (and some domain) aspects of what the project does in increasing levels of difficulty and complexity. The first few chapters will be on simpler mashups and we gradually move on to more complex ones. Also, as you progress with the chapters there are more assumptions made on your abilities to understand how mashups work. For example, the first few chapters describe how you can get accounts in the various mashup APIs but subsequent ones dispense with this altogether, assuming that you can navigate your way to registering for your own account.

The chapters tend to have less theory and more discussion on background technology while focusing more on a step-by-step guide in building the project in the chapter. The chapters also tend to be standalone though there are occasionally some references back to earlier chapters for some background technology; so feel free to explore them in any sequence you like.

The following explains each subsequent chapter's structure, section by section.

What does it do?

This section gives a brief summary of the mashup's functions and objectives. This is normally just a paragraph or two.

Domain background

What follows after the summary of objectives is a description of the domain background of the mashup's functions. For example, in Chapter 5 we will be discussing how we can create a job board mashup. The domain background section of the chapter gives a simple introduction to job boards, what they are and what they do.

Requirements overview

This section provides an overview of the requirements of the mashup we will be creating in the chapter. It lists the requirements and objectives to meet in building the project in the chapter.

Design

This section describes how we will be building the project, the rationale behind the design, and how we approach the creation of the mashup.

Mashup APIs on the menu

This is a major section in the chapter, and it describes the list of mashup APIs that we will be using to create the mashup. In the first few chapters we will also describe how we register for the mashup APIs at the external sources. Later chapters will dispense with this. All of the APIs are either freely available or have trial accounts where you have limited access to the APIs.

This section also describes the various libraries we will be using to build the mashups. This includes open-source libraries as well as libraries included in the standard Ruby distribution. Whenever necessary there will be some discussion on the theory behind the libraries and how they work.

What we will be doing

This section is the bulk of the chapter and goes through the step-by-step creation of the project. All necessary code is shown and major steps are described in detail. Each major step is explained in its own sub-section. In some cases additional information is given to explain why certain aspects of the code are written that way.

This section starts with the creation of a Ruby on Rails project and ends with a final completion of the whole project. Most projects are coded in a straightforward manner without much optimization. In some cases I will even go some way out to code the project in a way that a more advanced Rubyist might find not following the 'Ruby Way'. This is intentional as the purpose of the book is to focus on web mashups and not on Ruby or Ruby on Rails, and the code should normally be very readable by any programmer, even those less familiar with Ruby.

Summary

The chapter is finally wrapped up in a short summary that describes what we have done in the chapter.

Ready?

So much for the brief introduction to web mashups! While some of the caveats in this chapter sound scary, ultimately web mashups are a brave new world altogether, and an exciting one for programmers.

It's time to jump into the projects, so let's begin and have fun!