

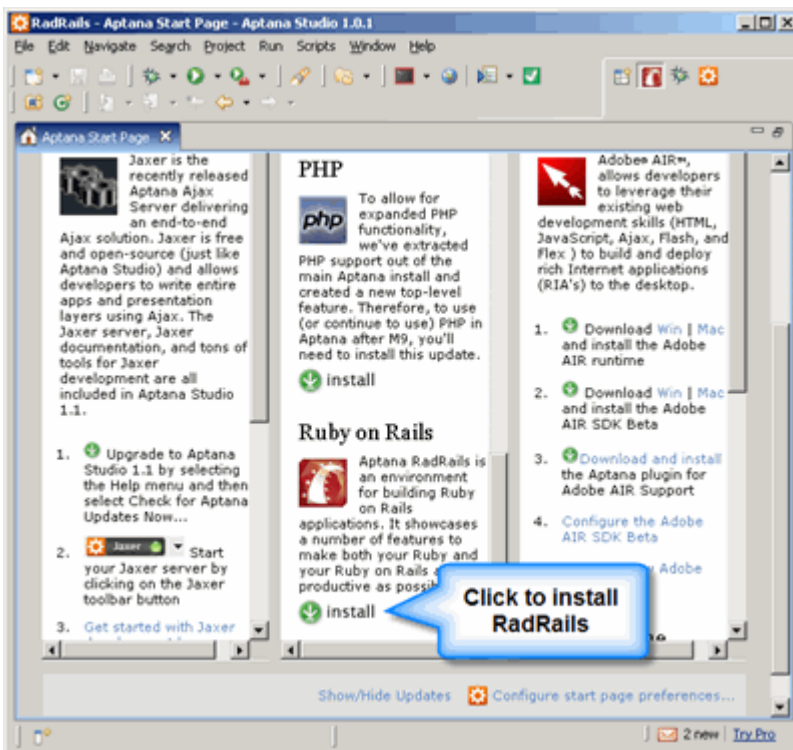
Building RESTful Application in Rails 2.0 Step-by-Step

Rails 2.0 reveals many [changes and improvements](#), but the biggest change involves getting closer to the full web-orientation of REST and HTTP. With a few lines of codes now you can easily create open web applications based on modern web service.

Creating Rails application

This example is a deadly-simple bookstore CMS. All you can do are to get book list, add or delete books. Not much? Yes, but big news is you can do that from anywhere in the world with or **without** a browser.

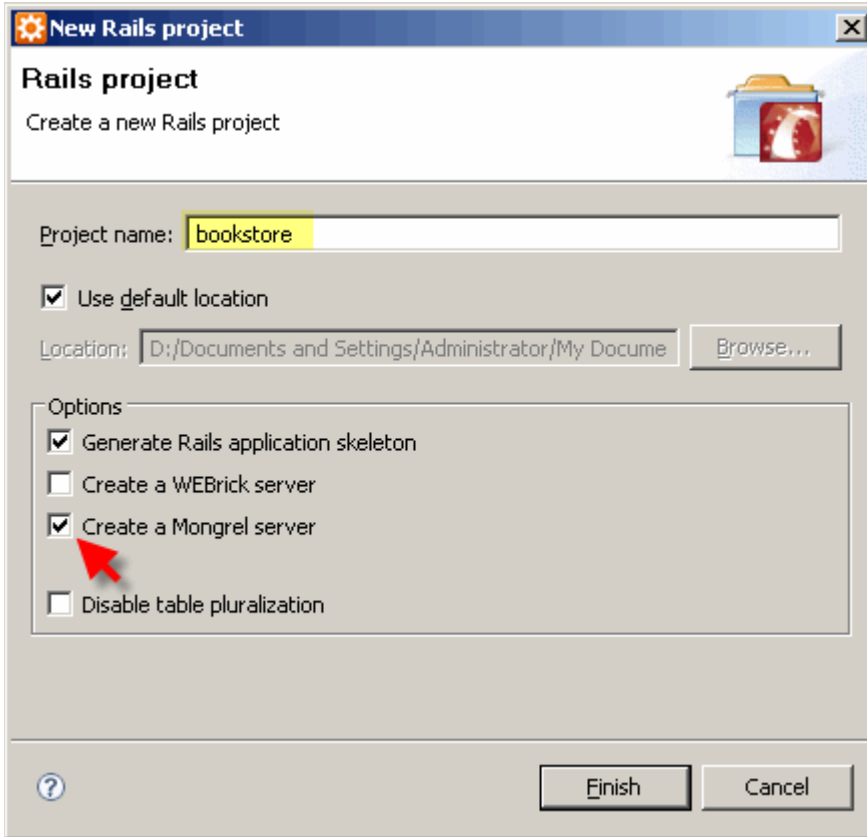
First, I use RadRails, which is now a part of [Aptana Studio](#), to create a Rails 2.0 application (you can learn more about how to add RadRails to Aptana Studio from [this page](#). It should be matter of one click only.)



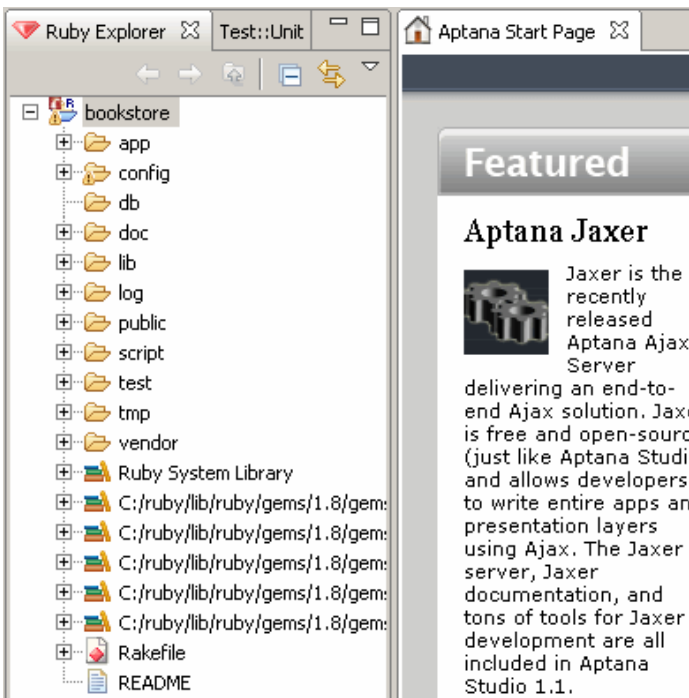
Please note that you have to upgrade to Rails 2.0, if not yet, here are links to the [Ruby Windows installer](#), [Mac OS 10.4 instructions](#).

Just follow step-by-step guide to create a Rails app below.

1. Switch to **RadRails** perspective, select *File -> New -> Rails Project* on main menu. Enter "bookstore" for project name and check "Create a Mongrel server".



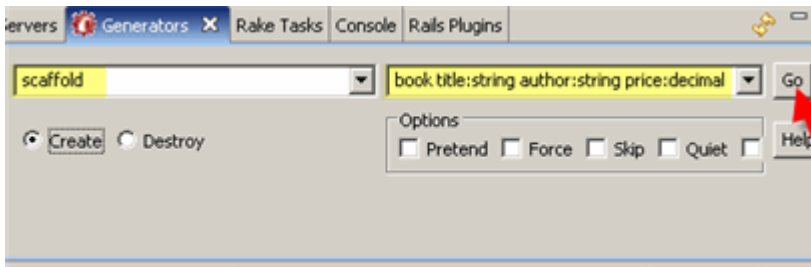
2. Click “**Finish**” button and the skeleton of Rails app is generated (see **Console** view for process details).



3. Open “**Generators**” view, select “**scaffold**” from the command list box (in previous Rails version, you have to use “**scaffold_resource**” command instead), then type in:

```
book title:string author:string price:decimal
```

as arguments. Click “**Go**” button -- this will create a new scaffold for a model called 'book' which has three fields, book title and book author typed as a text string, and a price of decimal type.



See “**Console**” view for what have been created.

```
Servers Generators Rake Tasks Console X Rails Plugins
<terminated> C:\ruby\bin\rubyw.exe (Jan 24, 2008 4:35:37 PM)
exists app/models/
exists app/controllers/
exists app/helpers/
create app/views/books
exists app/views/layouts/
exists test/functional/
exists test/unit/
create app/views/books/index.html.erb
create app/views/books/show.html.erb
create app/views/books/new.html.erb
create app/views/books/edit.html.erb
create app/views/layouts/books.html.erb
create public/stylesheets/scaffold.css
dependency model
exists app/models/
exists test/unit/
exists test/fixtures/
create app/models/book.rb
create test/unit/book_test.rb
create test/fixtures/books.yml
create db/migrate
create db/migrate/001_create_books.rb
create app/controllers/books_controller.rb
create test/functional/books_controller_test.rb
create app/helpers/books_helper.rb
route map.resources :books
```

Now, believe it or not, the bookstore application is almost completed, the only thing we need to do it to update the database so that it has the schema for the book records.

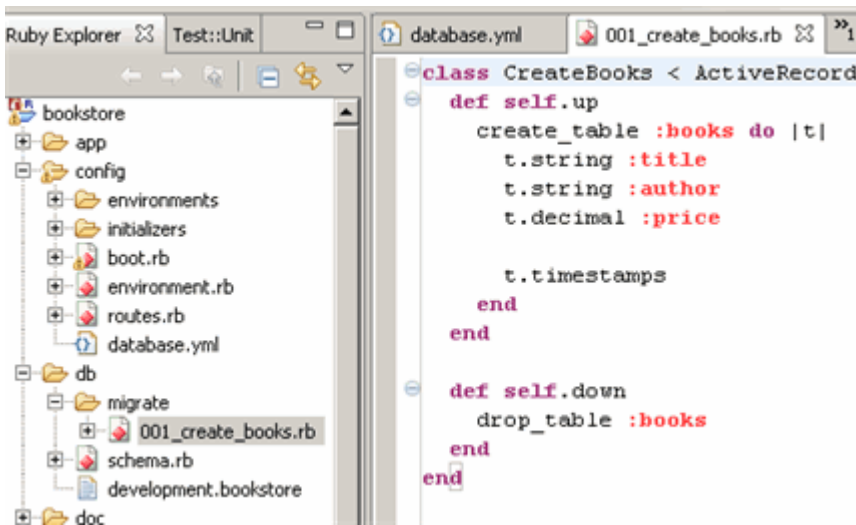
4. Change database names defined in “*database.yml*” file under “**config**” folder to more meaningful names.

```
001_create_books.rb  database.yml X »1
# SQLite version 3.x
#   gem install sqlite3-ruby (not neces
development:
  adapter: sqlite3
  database: db/development.bookstore
  timeout: 5000

# Warning: The database defined as 'tes
# re-generated from your development da
# Do not set this db to the same as dev
test:
  adapter: sqlite3
  database: db/test.bookstore
  timeout: 5000

production:
  adapter: sqlite3
  database: db/production.bookstore
  timeout: 5000
```

5. Navigate to “**db/migrate**” folder and open file “*xxx_create_books.rb*”. You’ll see how “books” table is defined.

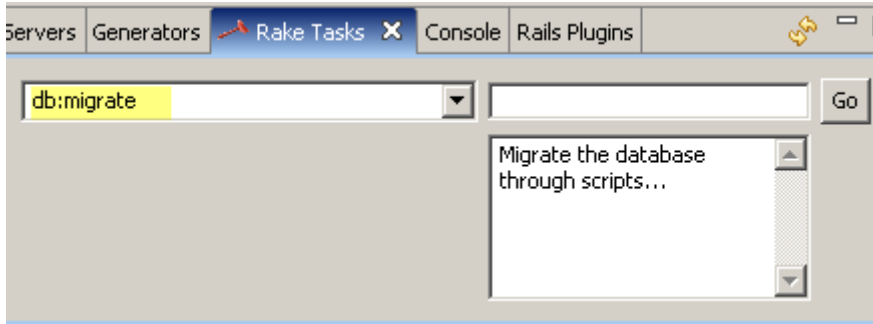


```
class CreateBooks < ActiveRecord
  def self.up
    create_table :books do |t|
      t.string :title
      t.string :author
      t.decimal :price

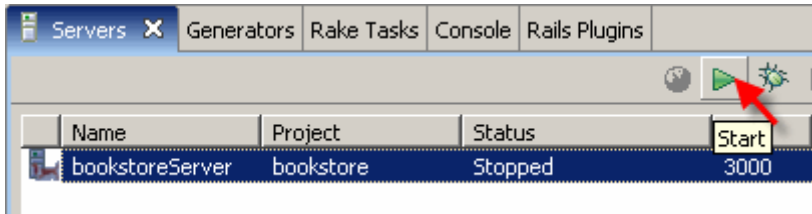
      t.timestamps
    end
  end

  def self.down
    drop_table :books
  end
end
```

6. Switch to “**Rake Tasks**” view, select “*db:migrate*” from command list and click “**Go**” button.



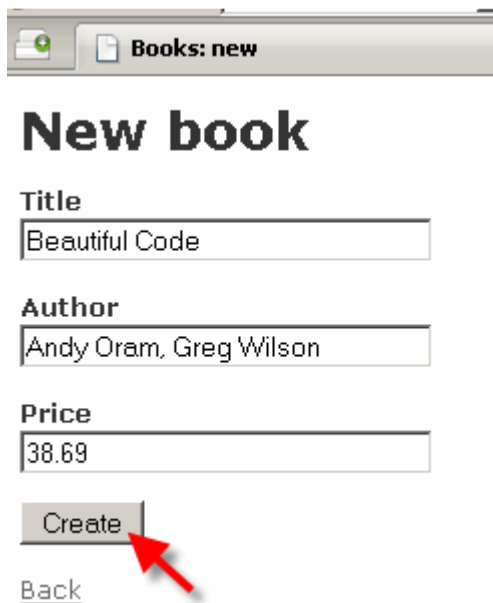
7. Wow, that is! The bookstore application is ready to run now. Switch to “**Server**” view, “**bookstoreServer**” should be there with default configuration listening at port 3000. Just select it and click “**Start**” button to run.



8. Open your favorite browser, type <http://localhost:3000/books/> in address bar. The book list is waiting for your first entry.



9. Click “*New book*” link and input a book title, author and price for it.



Books: new

New book

Title
Beautiful Code

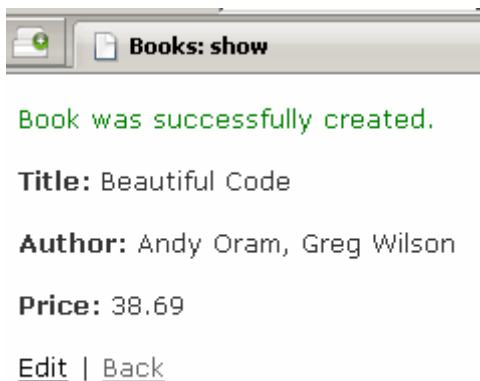
Author
Andy Oram, Greg Wilson

Price
38.69

Create

[Back](#)

Then select “**Create**” button. Now, the book is created and stored to database as desired.



Books: show

Book was successfully created.

Title: Beautiful Code

Author: Andy Oram, Greg Wilson

Price: 38.69

[Edit](#) | [Back](#)

10. Enter info for other books and back to book list view.

Listing books

Title	Author	Price
Beautiful Code	Andy Oram, Greg Wilson	38.69 Show Edit Destroy
Learning Python (Help for Programmers)	Mark Lutz, David Ascher, Frank Willison	24.0 Show Edit Destroy

[New book](#)

Up till now, anyone who is familiar to Rails may find nothing really new in this tutorial. However, interesting part is in the next section which makes use of RESTful web services offered freely along the way.

Utilizing RESTful web services

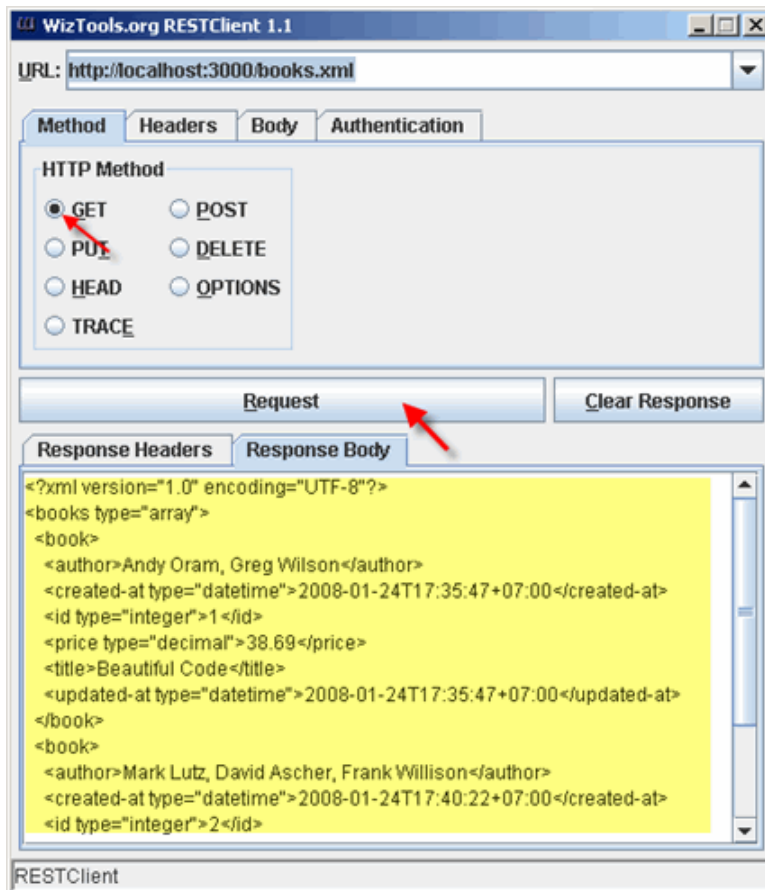
Well, to test how RESTful Web APIs are supported by Rails 2.0 we'll need a client tool. For those who want to use command line only, [cURL](#) is highly recommended (see [Duane Johnson's tutorial](#) for how to use it in testing.) However, I want all processes are visual, so I use [Rest-Client](#) -- a Java desktop tool made for testing RESTful web services. Now, just start Rest-Client by issuing command (you must have Java 5 or above):

```
java -jar {path-to-rest-client-jar-file}
```

The Rest-Client tool should start and be ready for using in our tests.

1. Get book list

Type <http://localhost:3000/books.xml> to URL textbox, select “GET” as request method, and click “Request” button.



See XML version of book list returned in “**Response Body**” tab.

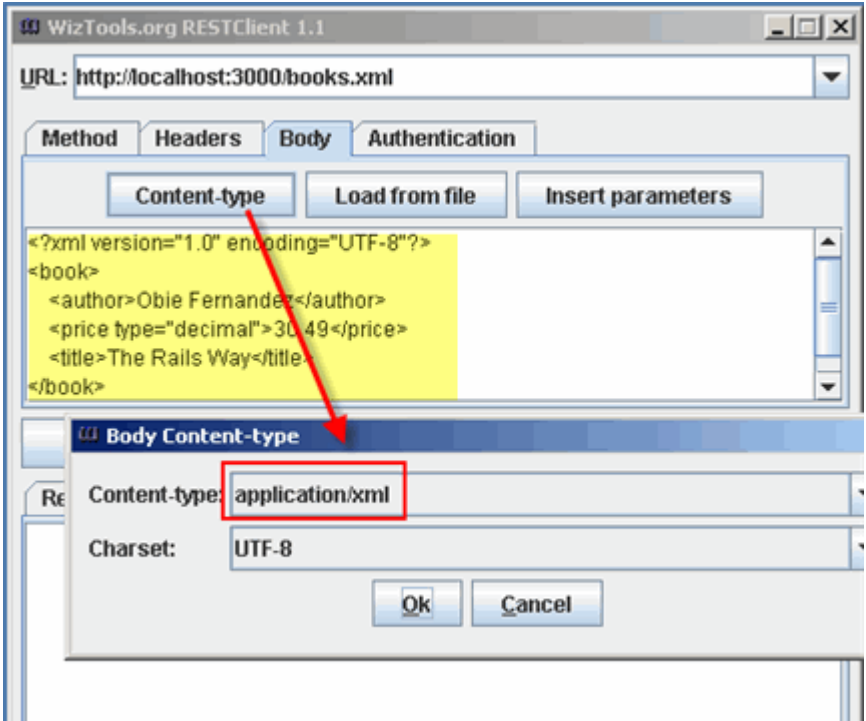
```
<?xml version="1.0" encoding="UTF-8"?>
<books type="array">
  <book>
    <author>Andy Oram, Greg Wilson</author>
    <created-at type="datetime">2008-01-24T17:35:47+07:00</created-at>
    <id type="integer">1</id>
    <price type="decimal">38.69</price>
    <title>Beautiful Code</title>
    <updated-at type="datetime">2008-01-24T17:35:47+07:00</updated-at>
  </book>
  <book>
    <author>Mark Lutz, David Ascher, Frank Willison</author>
    <created-at type="datetime">2008-01-24T17:40:22+07:00</created-at>
    <id type="integer">2</id>
    <price type="decimal">24.0</price>
    <title>Learning Python (Help for Programmers)</title>
    <updated-at type="datetime">2008-01-24T17:40:22+07:00</updated-at>
  </book>
</books>
```

2. Add a new book:

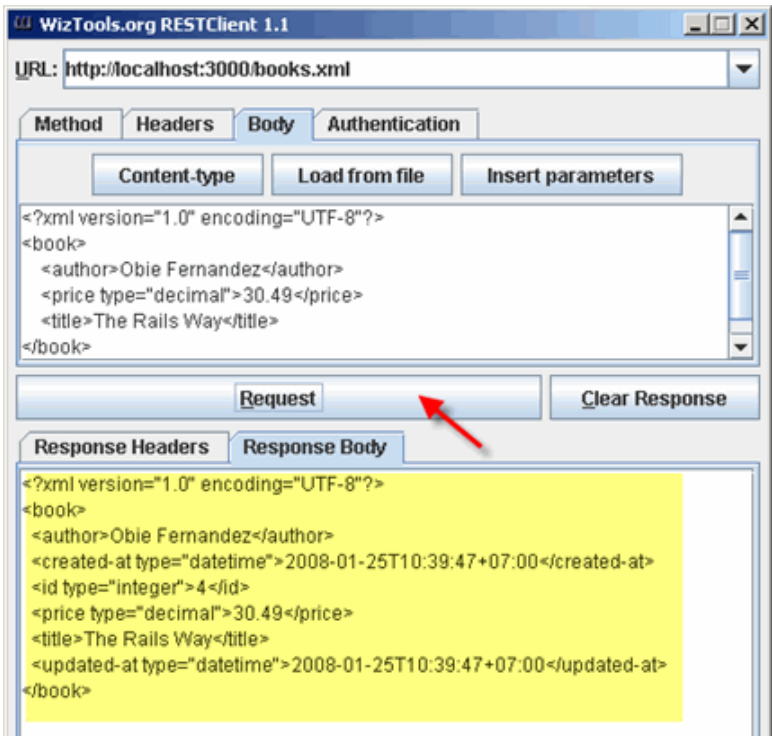
To add a new book, we have to prepare a XML text containing appropriate info about the book. Something should look like this for “The Rails Way”:

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <author>Obie Fernandez</author>
  <price type="decimal">30.49</price>
  <title>The Rails Way</title>
</book>
```

On Rest-Client tool, select “**POST**” request method (instead of “**GET**” in previous step), activate “**Body**” tab. Set request “Content-Type” to “**application/xml**” and paste XML into request body text region.



Click “Request” button. A new book created as returned XML in “Response Body” region shown.



And you can see new book added to the book list in browser too.

Listing books

Title	Author	Price	
Beautiful Code	Andy Oram, Greg Wilson	38.69	Show Edit Destroy
Learning Python (Help for Programmers)	Mark Lutz, David Ascher, Frank Willison	24.0	Show Edit Destroy
The Rails Way	Obie Fernandez	30.49	Show Edit Destroy

[New book](#)

3. Update book info

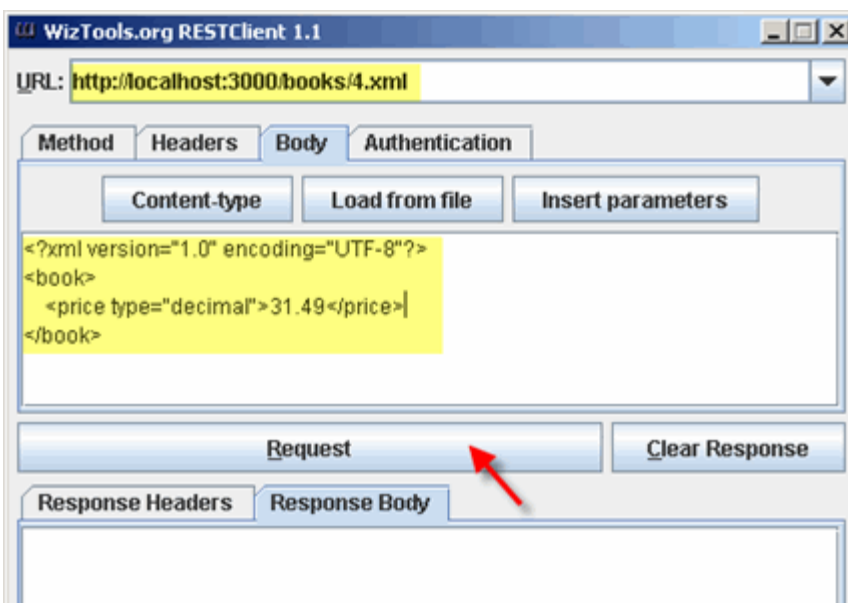
Let say we want to update price of newly-added book “*The Rail Way*” to \$31.49, prepare another XML with information to be updated:

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <price type="decimal">31.49</price>
</book>
```

Select “**PUT**” request method, paste the above XML to request body text region and change URL to:

<http://localhost:3000/books/4.xml>

where **4** is the ID of new book as you may notice. Click “**Request**” button to send it to Rails web API.



Now, you can see the update via browser.

Listing books

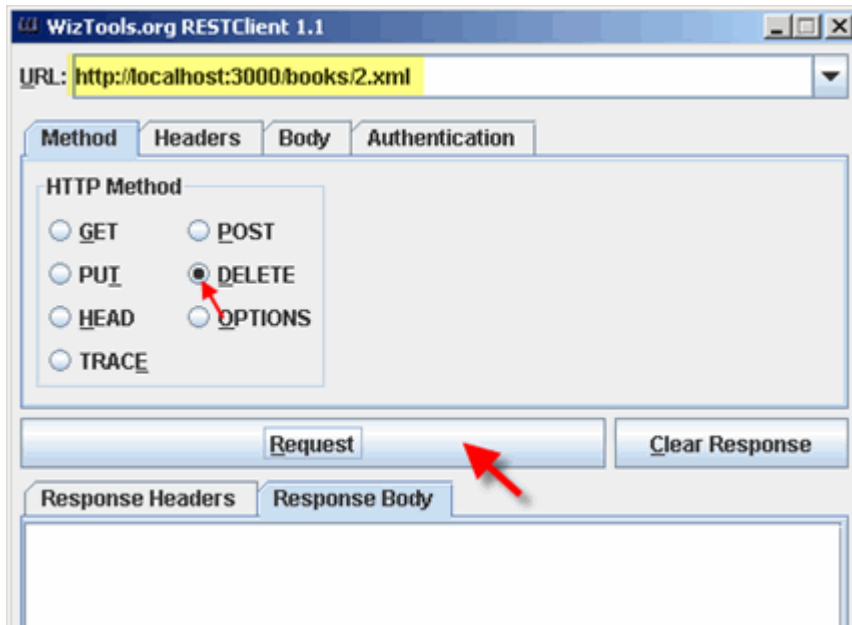
Title	Author	Price	
Beautiful Code	Andy Oram, Greg Wilson	38.69	Show Edit Destroy
Learning Python (Help for Programmers)	Mark Lutz, David Ascher, Frank Willison	24.0	Show Edit Destroy
The Rails Way	Obie Fernandez	31.49	Show Edit Destroy

[New book](#)

4. Delete a book

Our last mission is to delete a book using web service. It's a simple task, just select "DELETE" request method and send to book XML URL, for example:

<http://localhost:3000/books/2.xml>



"*Learning Python*" book is gone:

Listing books

Title	Author	Price	
Beautiful Code	Andy Oram, Greg Wilson	38.69	Show Edit Destroy
The Rails Way	Obie Fernandez	31.49	Show Edit Destroy

[New book](#)

Conclusion

Though this appears a long tutorial but actually it would take less than 10 minutes to create and test the application/web services. I'm quite sure that you can develop the same capability in Java or .Net environments but it would not be that simple. Notify me if this inspires you to create the next Amazon.com (2.0) ☺.

By Jimmy Vu © 2008 – JustTalkAboutWeb.com