

Peter Armstrong

Foreword by Stuart Eccles

# FLEXIBLE RAILS

FLEX 3 ON RAILS 2

SAMPLE CHAPTER

 MANNING





***Flexible Rails***  
by Peter Armstrong  
Chapter 11

Copyright 2008 Manning Publications

# *brief contents*

---

<b>PART 1</b>	<b>GETTING STARTED .....</b>	<b>1</b>
	1 ■ Why are we here? Where are we going?	3
	2 ■ Hello World	14
	3 ■ Getting started	52
<b>PART 2</b>	<b>BUILDING THE APPLICATION.....</b>	<b>103</b>
	4 ■ Creating the main Flex UI	105
	5 ■ Expanding the Rails code, RESTfully	118
	6 ■ Flex on Rails	186
	7 ■ Validation	261
<b>PART 3</b>	<b>REFACTORING.....</b>	<b>293</b>
	8 ■ Refactoring to Cairngorm	295
	9 ■ Holding state on the client properly	369
<b>PART 4</b>	<b>FINISHING UP.....</b>	<b>419</b>
	10 ■ Finishing the application	421
	11 ■ Refactoring to RubyAMF	468
	12 ■ Rails on AIR (Adobe Integrated Runtime)	512

# *Refactoring to RubyAMF*

---

# *11*

*111000011100011110000011110101000111*

—RubyAMF

RubyAMF is an Open Source, MIT-licensed (with one exception, as explained in a moment) Flash Remoting gateway for Ruby on Rails. It focuses on speed, and it integrates directly with RESTful Rails controllers via `render :amf`. Because of this, RubyAMF can be used as a more efficient wire format for sending information between Flex and Rails. (Of course, any time you see a claim that something is more efficient, you should do your own benchmarks or profiling. As such, I'm not going to do it for you here.)

RubyAMF also features a standalone implementation that doesn't depend on Rails—it's called *RubyAMF* and not *RailsAMF*. But because this book is about Rails in particular and not just Ruby—it's called *Flexible Rails*, not *Flexible Ruby*—the standalone RubyAMF is beyond the scope of the iteration.

In this iteration, we'll start by installing RubyAMF and doing a quick "Hello World" test. We'll then refactor the tasks, projects, locations, and note to be sent between Flex and Rails via `RemoteObject` to RubyAMF instead of using `HTTPService` and XML. (We won't touch the login code: There's no reason to do so, and I want to keep this iteration as small as possible.)

Doing the refactoring will show how RubyAMF integrates with our RESTful controllers and how we can modify our Flex application to use RubyAMF via `RemoteObject` instead of XML via `HTTPService`. We'll also replace the `ServiceUtils.as` class with the standard Cairngorm `Services.mxml` class: Doing this will bring our code more in line with standard Cairngorm, which will be instructive. (Also, because we're focusing on performance, it means we won't create numerous disposable `RemoteObjects`.)

Before we begin, I'll disclose my conflicts of interest up front. In so doing, I also get to stealthily establish the claim that I'm qualified to write about RubyAMF.

## 11.1 **Warning: biased author**

---

In this book, I like to play the role of unbiased (but opinionated) observer of the technologies I'm writing about. In this iteration, however, I'm not impartial. There are currently two AMF technologies for Ruby: RubyAMF and WebORB for Rails. I'm not going to write about WebORB for Rails at all in this book, because I may be biased in favor of RubyAMF. Here's why:

- I've advocated RubyAMF as the most promising AMF implementation for Ruby in various presentations.
- I have commit access on RubyAMF, <http://code.google.com/p/rubyamf/>—*although as of 2007-11-10 I've committed nothing.*

- There is a `flexiblerails` branch of RubyAMF (which as of 2007-11-10 was identical to the 1.5 release) so that readers of this book can follow along with something which is stable.
- A few months ago, Aaron Smith, the creator of RubyAMF, was kind enough to write some code that refactored an older version of `pomodo` to talk to an older version of RubyAMF (the standalone version, not the Rails version). Although this codebase isn't used in this iteration—RubyAMF has changed *a lot* since then—I've learned from it.
- Most important, I've had a number of productive email exchanges with Aaron Smith over the past months, including two email exchanges that constitute contributions to the RubyAMF project:
  - On June 20, 2007, I helped to convince Aaron to change the RubyAMF license from the GPL back to the MIT license (which RubyAMF had previously been licensed under). (Rails is also MIT-licensed, and Ruby uses a license which is similar to the MIT license.)
  - On June 29, 2007, I proposed the `format.amf { render :amf => @task.to_amf }` approach for integration with RESTful controllers and helped to convince Aaron that integration with RESTful controllers was a worthwhile feature. (Aaron subsequently implemented this, and it works well (as you'll soon see.) Note that we don't need to say `to_amf` with Rails 2; we can say `format.amf { render :amf => @task }` instead.)

In my opinion, the use of the MIT license alone will ensure that RubyAMF becomes the dominant AMF implementation for Ruby—especially since Rails also uses the MIT license, which means that RubyAMF and Rails are completely compatible. (As of 2007-11-10, WebORB for Rails was available under two licenses: the GPL and a commercial license.) Furthermore, the RESTful controller integration will help persuade Rails users that RubyAMF can be used as a more efficient wire format for Rails.

### **Note about the RubyAMF 1.5 license**

RubyAMF 1.5 (and the `flexiblerails` branch) is released under a slightly modified MIT license, with one clause added: “There is one exception to the above MIT license. WebORB may not use this code base in any of their releases of WebORB for RoR.”

Nobody ever said Open Source isn't competitive!

**Note about the RubyAMF 1.5 license (continued)**

The interesting thing about this clause is that it counteracts the license inequality of the GPL and the MIT license: Under the terms of the GPL, MIT + GPL code = GPL code. (The GPL is the most viral of the Open Source licenses—GPL'd code can incorporate MIT-licensed code and stay GPL'd, but MIT-licensed code can't incorporate GPL'd code without becoming GPL'd itself. Throughout human history, this type of approach has been an easy way for an idea to spread itself.) Under the GPL, WebORB [GPL] could have copied from RubyAMF [MIT], but RubyAMF could not have copied from WebORB. This clause means that neither project can copy from the other.

I've made contributions (however minor) to RubyAMF, and as such I'm biased in its favor. Although I'll strive to be impartial in this iteration, you can assume that because of my closeness to the project, I'm biased, and adjust your expectations accordingly.

## 11.2 Hello RubyAMF

---

We'll start by getting the flexiblerails branch of RubyAMF, as shown in listing 11.1.

**Listing 11.1 Installing the flexiblerails RubyAMF branch**

```
c:\peter\flexiblerails\current\pomodo>ruby script\plugin install
  ➤ http://rubyamf.googlecode.com/svn/branches/flexiblerails/rubyamf
+ ./CHANGELOG
+ ./LICENSE
+ ./README
+ ./app/actions.rb
+ ./app/amf.rb
+ ./app/configuration.rb
+ ./app/fault_object.rb
+ ./app/filters.rb
+ ./app/rails_gateway.rb
+ ./app/request_store.rb
+ ./exception/exception_handler.rb
+ ./exception/rubyamf_exception.rb
+ ./init.rb
+ ./install.rb
+ ./io/amf_deserializer.rb
+ ./io/amf_serializer.rb
+ ./io/read_write.rb
+ ./rails_installer_files/crossdomain.xml
```

```
+ ./rails_installer_files/rubyamf_config.rb
+ ./rails_installer_files/rubyamf_controller.rb
+ ./rails_installer_files/rubyamf_helper.rb
+ ./util/action_controller.rb
+ ./util/active_record.rb
+ ./util/string.rb
+ ./util/vo_helper.rb

c:\peter\flexiblerails\current\pomodo>
```

Next, because we're new to RubyAMF, we'll do a quick "Hello World" example. After that, I'll show the result of refactoring to RubyAMF all at once, without dwelling on the basics of RubyAMF. These basics are explained extremely well at the following tutorial<sup>1</sup>, written by Bryan Carlson of Trailtracer (<http://trailtracer.com/>): <http://panscendo.com/beginners-tutorial-to-rubyamf-with-restful-rails/>. As such, I'll keep my "Hello World" example extremely brief and explain RubyAMF as I present the result of the refactoring.

We don't need to do any configuration of RubyAMF at this point—the defaults in `config\rubyamf_config.rb` are good enough for "Hello World."

We'll start by creating a new `HelloController` with a `hello` action; see listing 11.2.

**Listing 11.2** `app\controllers\hello_controller.rb`

```
class HelloController < ApplicationController
  skip_before_filter :login_required ①
  def sayhello
    render :amf => "hello world" ②
  end
end
```

It doesn't get much simpler than this: We create a `HelloController` that doesn't require login ① and which has a `sayhello` action which renders the String "hello world" via AMF ②.

The trick is that this doesn't work from a normal HTTP request—try it in the browser or on the command line with `curl`: You get nothing. We need to call it using AMF—specifically, using `RemoteObject`, which uses AMF. From the *Flex 3*

---

<sup>1</sup> This tutorial used to live at <http://natureandtech.blogspot.com/2007/10/beginners-tutorial-to-rubyamf-with.html>. Since it was the first really good RubyAMF tutorial, you'll see various links to that URL online.

*Developer's Guide*. “RemoteObject components let you access the methods of server-side objects, such as ColdFusion components (CFCs), Java objects, PHP objects, and .NET objects, without configuring the objects as web services. You can use RemoteObject components in MXML or ActionScript.”<sup>2</sup> (Of course, it didn't mention Ruby—maybe the next version will.)

**NOTE** Flex 3 includes two RemoteObject classes: `mx.rpc.remoting.RemoteObject` and `mx.rpc.remoting.mxml.RemoteObject`. The latter is intended for use in MXML; what makes it confusing is that `mx.rpc.remoting.mxml.RemoteObject` extends `mx.rpc.remoting.RemoteObject`.

We'll need to create and use a RemoteObject that uses AMF to talk to RubyAMF. The problem is, we haven't configured AMF on the Flex side yet. Let's fix that now; see listing 11.3.

**NOTE** This file is taken directly (with permission) from Bryan Carlson's tutorial.

#### Listing 11.3 app\flex\services-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
  <services>
    <service id="rubyamf-flashremoting-service" ❶
      class="flex.messaging.services.RemotingService"
      messageTypes="flex.messaging.messages.RemotingMessage">
      <destination id="rubyamf" ❷
        <channels>
          <channel ref="rubyamf"/>
        </channels>
        <properties>
          <source>*</source>
        </properties>
      </destination>
    </service>
  </services>
  <channels>
    <channel-definition id="rubyamf" ❸
      class="mx.messaging.channels.AMFChannel">
      <endpoint uri="http://localhost:3000/rubyamf/gateway" ❹
        class="flex.messaging.endpoints.AMFEndpoint"/>
      </channel-definition>
    </channels>
  </services-config>
```

<sup>2</sup> *Flex 3 Developer's Guide, Beta 2*, p. 85.

Explaining this type of file in depth is best left to reference documentation; all we need to note is that we're creating a service whose id is "rubyamf-flashremoting-service" **1** and whose destination has an id of "rubyamf" **2**. We're also creating a channel whose id is "rubyamf" **3** and whose endpoint URI (the cool-kid way of saying what amounts to a URL) points at the URL of the rubyamf gateway on our local server **4**: `http://localhost:3000/rubyamf/gateway`.

All we need to do now is write the Flex code to test this. To keep things simple, we'll modify Pomodo.mxml to include our "Hello World" code; see listing 11.4.

**Listing 11.4** `app\flex\Pomodo.mxml`

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  ...
  creationCompleteEffect="fadeIn">
<mx:Script>
<![CDATA[
  import mx.core.Container;
  import mx.rpc.events.ResultEvent;
  import mx.rpc.events.FaultEvent;
  import com.pomodo.components.DebugPanel;
  import com.pomodo.control.EventNames;
  import com.pomodo.util.CairngormUtils;
  import com.pomodo.util.DebugMessage;
  import com.pomodo.model.PomodoModelLocator;
  ...
  private function loadFlexibleRails():void {
    CairngormUtils.dispatchEvent(EventNames.LOAD_URL,
      "http://www.flexiblerails.com");
  }

  private function handleHelloResult(e:ResultEvent):void { 1
    Pomodo.debug("hello result:\n" + e.message);
  }

  private function handleFault(e:FaultEvent):void { 2
    Pomodo.debug("FAULT:\n" + e.fault.faultString);
  }
  ]]>
</mx:Script>
  <mx:RemoteObject id="helloRO" 3
    source="HelloController" 4
    destination="rubyamf" 5
    fault="handleFault(event)">
    <mx:method name="sayhello" 6
      result="handleHelloResult(event)"/>
  </mx:RemoteObject>

```

Add  
imports

```

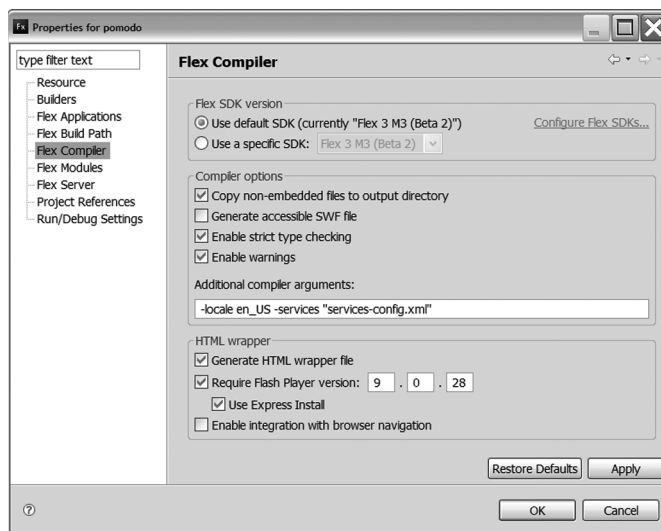
<mx:Fade id="fadeIn" duration="500"/>
...
<mx:HBox backgroundColor="#000000" width="100%" height="30"
  horizontalAlign="center" verticalAlign="middle">
  <mx:LinkButton color="#FFFFFF"
    click="loadFlexibleRails()" label="{MARKETING}"/>
</mx:HBox>
<mx:Button label="hello" click="helloRO.sayhello.send()" /> ⑦
<mx:Spacer height="10"/>
...
</mx:Application>

```

We add a RemoteObject called helloRO ③, which has one mx:method whose name is sayhello ⑥. This method is invoked by clicking the button ⑦ whose label is hello. We add a fault handler method called handleFault ② for the entire RemoteObject and a handleHelloResult ① method for the result of the sayhello method. Both the fault handler and the result handler call Pomodo.debug().

Note that the source of the helloRO RemoteObject is HelloController ④; this is the name of the controller we created in Rails. We also see that the destination is rubyamf ⑤: This refers to the id of the destination we defined in services-config.xml earlier.

Before we run, we need to add the argument -services "services-config.xml" to the compiler arguments so the compiler knows to use our services; see figure 11.1.



**Figure 11.1**  
Adding the services-config.xml to the compiler arguments



Figure 11.2 “Hello World” from RubyAMF

With this done, we restart our server (to pick up the RubyAMF configuration), rebuild the Flex project, and load `http://localhost:3000/Pomodo.html`. We see the pomodo application with a Hello button on it. Drag the debug console into view, and click the Hello button; we see something like figure 11.2.

Note that the body of the `AcknowledgeMessage` is `"hello world"`, which is what we rendered with `render :amf => "hello world"`.

### 11.3 Refactoring to RubyAMF, fast-forwarded

---

The result of this refactoring will be presented all at once. I got it working for tasks with a standalone tester inside pomodo and then ported the changes to projects and locations. However, because tasks, projects, and locations are so similar, it makes sense to present them together. I'll start by showing the `rubyamf_config.rb` file, then the changes to the controllers, and then the various changes to the Flex code. This will include replacing the `ServiceUtils.as` class with the standard Cairngorm `Services.mxml` class: Doing this will bring our code more in line with standard Cairngorm, which will be instructive. We'll also violate the Don't Repeat Yourself (DRY) principle (I'm sorry, Dave) and create a layer of

value object (VO) classes to marshal and unmarshal. This turns out to be simpler, because we don't need to worry about the logic we've built into our model constructors conflicting with the unmarshaling.

**NOTE** Joking aside, principles such as DRY are meant to be applied pragmatically, not absolutely. Dave Thomas would be the first person to agree with this, being a Pragmatic Programmer after all. If you find yourself twisting into contortions to stay true to DRY, ask yourself if the contortions are worth it. (This was one of Zed Shaw's many points in his Rails to Italy keynote.)

Enough talk. Let's get coding!

### 11.3.1 Modifying `rubyamf_config.rb`

We'll start by modifying the `rubyamf_config.rb` file, as shown in listing 11.5. Note that I made a *lot* of changes to make it fit the page width without showing those changes as diffs.

**Listing 11.5** `config\rubyamf_config.rb`

```
require 'app/configuration'
module RubyAMF
  module Configuration
    #set the service path used in all requests
    # RubyAMF::App::RequestStore.service_path =
    #   File.expand_path(RAILS_ROOT) + '/app/controllers'

    # => CLASS MAPPING CONFIGURATION

    # => Global Property Ignoring
    # By putting attribute names into this array, you opt in to
    # globally ignore these properties on incoming objects.
    # If you want to ignore specific properties on certain
    # objects, use the :ignore_fields property in a
    # Class Mapping definition (see CLASS MAPPING DEFINITIONS)
    ClassMappings.ignore_fields =
      ['created_at', 'created_on', 'updated_at', 'updated_on']

    # => Case Translations
    # Most actionscript uses camelCase instead of snake_case.
    # Set ClassMappings.translate_case to true if want
    # translations to occur.
    # The translations only occur on object properties
    # An incoming property like myProperty gets turned into
    # my_property. An outgoing property like my_property gets
    # turned into myProperty.
```

Ignore  
because  
auto-set  
by Rails

```

ClassMappings.translate_case = true  ← Change case to use proper
                                     coding conventions

# => Force Active Record Ids
# includes the id field for activerecord objects even if you
# don't specify it when using custom attributes. This is
# important for deserialization where ids are needed to keep
# active record association integrity.
ClassMappings.force_active_record_ids = true  ← True because using
                                               ActiveRecord

# => Assume Class Types
# This tells RubyAMF to assume class type transfers. So when
# you register a class Alias from Flash or Flex like this:
# Flash:: fl.net.registerClassAlias('User',User)
# Flex::   [RemoteClass(alias='User')]
# RubyAMF will automagically convert it to a User active
# record without you having to create a class mapping.
# This also works with non active record class mappings. See
# the wiki on the google code page for a downloadable
# example.
ClassMappings.assume_types = false  ← Doing ClassMappings
                                     manually, so false

# => Class Mapping Definitions
# A Class Mapping definition consists of at least these two
# properties:
# :actionscript # The incoming ActionScript class to watch
#               # for
# :ruby         # The Ruby class to turn it into
#
# => Optional value object properties:
# :type         # Used to spectify the type of VO; valid
#               # options are 'active_record', 'custom',
#               # (or don't specify at all)
# :associations # Specify which associations to read on
#               # the active record (only applies to
#               # active records)
# :attributes   # Specifically which attributes to include
#               # in the serialization
# :ignore_fields # An array of field names you want to
#               # ignore on incoming classes
#
# If you are using ActiveRecord VO's you do not need to
# specify a fully qualified class path to the model, you can
# just define the class name; for example:
# ClassMappings.register(:actionscript => 'vo.Person',
#   :ruby => 'Person', :type => 'active_record')
#
# If you are using custom VO's you would need to specify the
# fully qualified class path to the file, for example:
# ClassMappings.register(:actionscript => 'vo.Person',
#   :ruby => 'org.mypackage.Person')

```

...

```

ClassMappings.register(
  :actionscript => 'com.pomodo.vo.TaskVO',
  :ruby => 'Task',
  :type => 'active_record',
  :attributes => ["id", "name", "notes", "next_action",
    "completed", "project_id", "location_id"])
ClassMappings.register(
  :actionscript => 'com.pomodo.vo.ProjectVO',
  :ruby => 'Project',
  :type => 'active_record',
  :attributes => ["id", "name", "notes", "completed"])
ClassMappings.register(
  :actionscript => 'com.pomodo.vo.LocationVO',
  :ruby => 'Location',
  :type => 'active_record',
  :attributes => ["id", "name", "notes"])
ClassMappings.register(
  :actionscript => 'com.pomodo.vo.NoteVO',
  :ruby => 'Note',
  :type => 'active_record',
  :attributes => ["content"])

# => Class Mapping Scope (Advanced Usage)
...
# => Check for Associations
# Enabling this will automagically pick up eager loaded
# association data on objects returned through RubyAMF.
# If this is disabled, you will need to specify any
# associations you DO want picked up in the ClassMapping
ClassMappings.check_for_associations = false
# => NAMED PARAMETER MAPPING CONFIGURATION

#=> Always Put Remoting Parameters into the "params" hash
# If set to true, arguments from Flash/Flex will come in to
# the controllers as params[0], params[1], etc.. This is
# especially useful if you are sending huge objects
# from Flex into Ruby so it doesnt eat up all your output
# window with outputting the params in the controller/action
# header information while in dev mode.
# Even if its set to false, if you specify specific
# ParameterMappings, those will still get entered as the
# param keys you specify. Likewise, you always have access
# to the parameters from rubyamf in your controller by
# calling rubyamf_params[0], rubyamf_params[1], etc
# regardless of if it this is set or not.
ParameterMappings.always_add_to_params = true

# => Return Top Level Hash
# For those scaffolding users out there, who want the
# top-level object to come as a hash so scaffolding works

```

Map TaskVO to/from Task

Map ProjectVO to/from Project

Map LocationVO to/from Location

Map NoteVO to/from Note

1 Don't want associations

Want parameters in params

```

    # out of the box.
    ParameterMappings.scaffolding = false
    ...
  end
end

```

← 2 Currently true is flaky

Basically, this file is the result of a lot of ~~hacking~~ experimentation. By the time you read this, some of the more automatic ways of using this configuration file may work better—this is the way that worked for me. The most important setting is `ParameterMappings.scaffolding = false` ②, which didn't work when true (using Rails 1.99.0 and the `flexiblerails` RubyAMF branch). Also, note that we set `check_for_associations` to false ①, since we're just transferring objects, not object graphs.

All this talk of parameter mapping is making me hungry—for some tables of type conversions. The type conversions that are done by RubyAMF when converting between Flash and Ruby are found at <http://code.google.com/p/rubyamf/wiki/AMFTypeConversions> and are shown in table 11.1.

**Table 11.1** ActionScript 3 to Ruby and Ruby to ActionScript 3 conversions done by RubyAMF

ActionScript 3	...converted to Ruby	...converted to ActionScript 3
undefined	nil	null
null	nil	null
false	false	false
true	true	false
Number	Fixnum	Number
int	Integer	Number
String	String	String
XML	String (cast in your service)	-
-	BeautifulSoup	XML
-	REXML::Doc	XML
Array	Array	Array
MixexArray	Hash	Object
Object	Hash	Object
Custom Class	Ruby Class	Custom Class

See the RubyAMF wiki for details.

### 11.3.2 Modifying the Rails controllers

Next, we need to modify the Rails controllers. We start by deleting `app\controllers\hello_controller.rb`.

Next, we'll modify the `TasksController`; see listing 11.6.

#### Listing 11.6 `app\controllers\tasks_controller.rb`

```
class TasksController < ApplicationController
  # GET /tasks
  # GET /tasks.xml
  def index
    @tasks = current_user.tasks

    respond_to do |format|
      format.html # index.html.erb
      format.xml { render :xml => @tasks }
      format.amf { render :amf => @tasks } ← Render tasks as AMF—it's that easy
    end
  end

  # GET /tasks/1
  # GET /tasks/1.xml
  def show
    @task = current_user.tasks.find(params[:id])
    @task = current_user.tasks.find( ← params[0] instead of in params[:id]
      is_amf ? params[0] : params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.xml { render :xml => @task }
      format.amf { render :amf => @task } ← Render task as AMF
    end
  rescue ActiveRecord::RecordNotFound => e
    prevent_access(e)
  end

  # GET /tasks/new
  # GET /tasks/new.xml
  def new
    @task = Task.new

    respond_to do |format|
      format.html # new.html.erb
      format.xml { render :xml => @task }
    end
  end
end
```

```

# GET /tasks/1/edit
def edit
  @task = current_user.tasks.find(params[:id])
  rescue ActiveRecord::RecordNotFound => e
    prevent_access(e)
  end

# POST /tasks
# POST /tasks.xml
def create
  @task = current_user.tasks.build(params[:task])
  if is_amf
    @task = params[0]
    @task.user_id = current_user.id
    @task.created_at = @task.updated_at = Time.now
  else
    @task = current_user.tasks.build(params[:task])
  end
  respond_to do |format|
    if @task.save
      format.html do
        flash[:notice] = 'Task was successfully created.'
        redirect_to(@task)
      end
      format.xml { render :xml => @task, :status => :created,
        :location => @task }
      format.amf { render :amf => @task }
    else
      format.html { render :action => "new" }
      format.xml { render :xml => @task.errors,
        :status => :unprocessable_entity }
      format.amf { render :amf => @task.errors }
    end
  end
end

# PUT /tasks/1
# PUT /tasks/1.xml
def update
  @task = current_user.tasks.find(params[:id])
  @task.attributes = params[:task]
  @task = current_user.tasks.find(
    is_amf ? params[0].id : params[:id])
  if is_amf
    @task.name = params[0].name
    @task.notes = params[0].notes
    @task.project_id = params[0].project_id
    @task.location_id = params[0].location_id
    @task.next_action = params[0].next_action
    @task.completed = params[0].completed
  else

```

Task in params[0], not Hash in params[:task]

Render task as AMF

Render task errors as AMF

1

```

    @task.attributes = params[:task]
  end
  @task.save_with_gtd_rules!

  respond_to do |format|
    format.html do
      flash[:notice] = 'Task was successfully updated.'
      redirect_to(@task)
    end
    format.xml { render :xml => @task }
    format.amf { render :amf => @task }
  end
rescue ActiveRecord::RecordInvalid
  respond_to do |format|
    format.html { render :action => "edit" }
    format.xml { render :xml => @task.errors.to_xml_full }
    format.amf { render :amf => @task.errors }
  end
rescue ActiveRecord::RecordNotFound => e
  prevent_access(e)
end

# DELETE /tasks/1
# DELETE /tasks/1.xml
def destroy
  @task = current_user.tasks.find(params[:id])
  @task = current_user.tasks.find(
    is_amf ? params[0] : params[:id]
  )
  @task.destroy

  respond_to do |format|
    format.html { redirect_to(tasks_url) }
    format.xml { render :xml => @task }
    format.amf { render :amf => @task }
  end
rescue ActiveRecord::RecordNotFound => e
  prevent_access(e)
end

private
def prevent_access(e)
  logger.info "TasksController#prevent_access: #{e}"
  respond_to do |format|
    format.html { redirect_to(tasks_url) }
    format.xml { render :text => "error" }
    format.amf { render :amf => "error" }
  end
end
end
end

```

2

Render task as AMF

Render task errors as AMF

params[0] instead of in params[:id]

Render task as AMF

Render String "error" as AMF

This code has only a couple of changes. First, there is a new format: `format.amf`. This is set for us by RubyAMF so we can `respond_to` it. Next, note that the parameters we receive from `RemoteObject` method invocations come in as `params[0]`, `params[1]`, and so on. These parameters are converted from `ActionScript` to `Ruby` following the rules shown in table 11.1. Also note that when responding to Flex, we now `render :amf` instead of `render :xml`.

**NOTE** Yes, the `params[0]`, `params[1]` stuff is unfortunate. I hope a better way will work in the future.

No logic anywhere ensures that the `project_id` and `location_id` refer to projects and locations that belong to the user, so a user who wanted to hack them could cause mischief. This needs to be prevented for both the AMF case where they are assigned explicitly ❶ and the XML over HTTP case where a mass assignment is done ❷. (Exercise for the reader: Fix this.)

Next, we make extremely similar changes to the `ProjectsController`; see listing 11.7.

**Listing 11.7** `app\controllers\projects_controller.rb`

```
class ProjectsController < ApplicationController
  # GET /projects
  # GET /projects.xml
  def index
    @projects = current_user.projects

    respond_to do |format|
      format.html # index.html.erb
      format.xml { render :xml => @projects }
      format.amf { render :amf => @projects } ← Render projects as AMF
    end
  end

  # GET /projects/1
  # GET /projects/1.xml
  def show
    @project = current_user.projects.find(params[:id])
    @project = current_user.projects.find( ← params[0] instead of in params[:id]
      is_amf ? params[0] : params[:id])
    respond_to do |format|
      format.html # show.html.erb
      format.xml { render :xml => @project }
      format.amf { render :amf => @project } ← Render project as AMF
    end
  rescue ActiveRecord::RecordNotFound => e
    prevent_access(e)
  end
end
```

```

end

# GET /projects/new
# GET /projects/new.xml
def new
  @project = Project.new

  respond_to do |format|
    format.html # new.html.erb
    format.xml { render :xml => @project }
  end
end

# GET /projects/1/edit
def edit
  @project = current_user.projects.find(params[:id])
  rescue ActiveRecord::RecordNotFound => e
    prevent_access(e)
  end

# POST /projects
# POST /projects.xml
def create
  @project = current_user.projects.build(params[:project])
  if is_amf
    @project = params[0]
    @project.user_id = current_user.id
    @project.created_at = @project.updated_at = Time.now
  else
    @project = current_user.projects.build(params[:project])
  end

  respond_to do |format|
    if @project.save
      format.html do
        flash[:notice] = 'Project was successfully created.'
        redirect_to(@project)
      end
      format.xml { render :xml => @project,
        :status => :created, :location => @project }
      format.amf { render :amf => @project }
    else
      format.html { render :action => "new" }
      format.xml { render :xml => @project.errors,
        :status => :unprocessable_entity }
      format.amf { render :amf => @project.errors }
    end
  end
end

# PUT /projects/1

```

Project in params[0], not Hash in params[:project]

Render project as AMF

Render project errors as AMF

```

# PUT /projects/1.xml
def update
  @project = current_user.projects.find(params[:id])
  @project.attributes = params[:project]
  @project = current_user.projects.find(
    is_amf ? params[0].id : params[:id])
  if is_amf
    @project.name = params[0].name
    @project.notes = params[0].notes
    @project.completed = params[0].completed
  else
    @project.attributes = params[:project]
  end
  @project.save_with_gtd_rules!

  respond_to do |format|
    format.html do
      flash[:notice] = 'Project was successfully updated.'
      redirect_to(@project)
    end
    format.xml { render :xml => @project }
    format.amf { render :amf => @project }
  end
  rescue ActiveRecord::RecordInvalid
    respond_to do |format|
      format.html { render :action => "edit" }
      format.xml { render :xml => @project.errors.to_xml_full }
      format.amf { render :amf => @project.errors }
    end
  rescue ActiveRecord::RecordNotFound => e
    prevent_access(e)
  end

# DELETE /projects/1
# DELETE /projects/1.xml
def destroy
  @project = current_user.projects.find(params[:id])
  @project = current_user.projects.find(
    is_amf ? params[0] : params[:id])
  @project.destroy

  respond_to do |format|
    format.html { redirect_to(projects_url) }
    format.xml { render :xml => @project }
    format.amf { render :amf => @project }
  end
  rescue ActiveRecord::RecordNotFound => e
    prevent_access(e)
  end

private

```

Project in params[0], not id in params[:id]

Render project as AMF

Render project errors as AMF

params[0] instead of params[:id]

Render project as AMF

```

def prevent_access(e)
  logger.info "ProjectsController#prevent_access: #{e}"
  respond_to do |format|
    format.html { redirect_to(projects_url) }
    format.xml { render :text => "error" }
    format.amf { render :amf => "error" }
  end
end
end
end

```

Render String "error" as AMF

We also make similar changes to the LocationsController; see listing 11.8.

#### Listing 11.8 app\controllers\locations\_controller.rb

```

class LocationsController < ApplicationController
  # GET /locations
  # GET /locations.xml
  def index
    @locations = current_user.locations

    respond_to do |format|
      format.html # index.html.erb
      format.xml { render :xml => @locations }
      format.amf { render :amf => @locations }
    end
  end

  # GET /locations/1
  # GET /locations/1.xml
  def show
    @location = current_user.locations.find(params[:id])
    @location = current_user.locations.find(
      is_amf ? params[0] : params[:id])
  end

  respond_to do |format|
    format.html # show.html.erb
    format.xml { render :xml => @location }
    format.amf { render :amf => @location }
  end
  rescue ActiveRecord::RecordNotFound => e
    prevent_access(e)
  end

  # GET /locations/new
  # GET /locations/new.xml
  def new
    @location = Location.new

    respond_to do |format|

```

Render locations as AMF

params[0] instead of params[:id]

Render location as AMF

```

        format.html # new.html.erb
        format.xml { render :xml => @location }
      end
    end

    # GET /locations/1/edit
    def edit
      @location = current_user.locations.find(params[:id])
      rescue ActiveRecord::RecordNotFound => e
        prevent_access(e)
      end

    # POST /locations
    # POST /locations.xml
    def create
      @location = current_user.locations.build(params[:location])
      if is_amf
        @location = params[0]
        @location.user_id = current_user.id
        @location.created_at = @location.updated_at = Time.now
      else
        @location = current_user.locations.build(
          params[:location])
      end

      respond_to do |format|
        if @location.save
          format.html do
            flash[:notice] = 'Location was successfully created.'
            redirect_to(@location)
          end
          format.xml { render :xml => @location,
            :status => :created, :location => @location }
          format.amf { render :amf => @location }
        else
          format.html { render :action => "new" }
          format.xml { render :xml => @location.errors,
            :status => :unprocessable_entity }
          format.amf { render :amf => @location.errors }
        end
      end
    end

    # PUT /locations/1
    # PUT /locations/1.xml
    def update
      @location = current_user.locations.find(params[:id])
      @location = current_user.locations.find(
        is_amf ? params[0].id : params[:id])
      if is_amf
        @project.name = params[0].name
      end
    end
  end
end

```

Location in params[0], not Hash in params[:location]

Render location as AMF

Render location errors as AMF

Location in params[0], not id in params[:id]

```

    @project.notes = params[0].notes
    @project.completed = params[0].completed
  else
    @location.attributes = params[:location]
  end
  @location.save! ❶

  respond_to do |format|
    if @location.update_attributes(params[:location])
      flash[:notice] = 'Location was successfully updated.'
      format.html { redirect_to(@location) }
      format.xml { render :xml => @location }
    else
      format.html { render :action => "edit" }
      format.xml { render :xml => @location.errors,
        +:status => :unprocessable_entity }
    end
    format.html do
      flash[:notice] = 'Location was successfully updated.' ❷
      redirect_to(@location)
    end
    format.xml { render :xml => @location }
    format.amf { render :amf => @location }
  end
  rescue ActiveRecord::RecordInvalid
    respond_to do |format|
      format.html { render :action => "edit" }
      format.xml { render :xml => @location.errors.to_xml_full }
      format.amf { render :amf => @project.errors }
    end
  rescue ActiveRecord::RecordNotFound => e
    prevent_access(e)
  end

  # DELETE /locations/1
  # DELETE /locations/1.xml
  def destroy
    @location = current_user.locations.find(params[:id])
    @location = current_user.locations.find(
      is_amf ? params[0] : params[:id]
    )
    @location.destroy

    respond_to do |format|
      format.html { redirect_to(locations_url) }
      format.xml { render :xml => @location }
      format.amf { render :amf => @location }
    end
  rescue ActiveRecord::RecordNotFound => e
    prevent_access(e)
  end
end

```

Render location as AMF

Render location errors as AMF

params[0] instead of params[:id]

Render location as AMF

```

private
def prevent_access(e)
  logger.info "LocationsController#prevent_access: #{e}"
  respond_to do |format|
    format.html { redirect_to(locations_url) }
    format.xml { render :text => "error" }
    format.amf { render :amf => "error" } ← Render String
                                             "error" as AMF
  end
end
end
end

```

This is more of the same, except that we're now using `@location.save!` ❶ and only building the `flash[:notice]` in the case of rendering HTML ❷.

Next, the `NotesController`; see listing 11.9.

#### Listing 11.9 `app\controllers\notes_controller.rb`

```

class NotesController < ApplicationController
  # GET /users/1/note
  # GET /users/1/note.xml
  def show
    if current_user.id != params[:user_id].to_i ❶
      prevent_access
    else
      @note = current_user.note
      respond_to do |format|
        format.xml { render :xml => @note.to_xml }
      end
    end
    if is_amf ❷
      render :amf => current_user.note
    else
      if current_user.id != params[:user_id].to_i
        prevent_access
      else
        @note = current_user.note
        respond_to do |format|
          format.xml { render :xml => @note }
        end
      end
    end
  end
end

# PUT /users/1/note
# PUT /users/1/note.xml
def update
  if current_user.id != params[:user_id].to_i ❸
    prevent_access
  end
end
end

```

```

else
  @note = current_user.note
  respond_to do |format|
    if @note.update_attributes(params[:note])
      format.xml { render :xml => @note.to_xml }
    else
      format.xml { render :xml => @note.errors.to_xml_full }
    end
  end
end
end
if is_amf ❹
  @note = current_user.note
  @note.content = params[0].content
else
  if current_user.id != params[:user_id].to_i
    prevent_access
  else
    @note = current_user.note
    @note.attributes = params[:note]
  end
end
end
@note.save! ❺

respond_to do |format|
  format.xml { render :xml => @note }
  format.amf { render :amf => @note }
end
rescue ActiveRecord::RecordInvalid
  respond_to do |format|
    format.xml { render :xml => @note.errors.to_xml_full }
    format.amf { render :amf => @note.errors }
  end
end

private
def prevent_access
  respond_to do |format|
    format.xml { render :text => "error" }
    format.amf { render :amf => "error" } ❻
  end
end
end
end

```

This is fairly simple: We start by moving the old code ❶ of the show method inside the else case of an is\_amf test ❷. If the request is\_amf, we render the current user's note. In the update method, we refactor the code ❸ to set the properties of the @note, doing the right thing depending on whether the request is\_amf ❹.

We then call the `save!` method **5** and respond appropriately. Finally, we add support for AMF to the `prevent_access` method **6**.

### 11.3.3 Creating `Services.mxml` and modifying `Pomodo.mxml`

Next, we'll create a new file called `Services.mxml`. (If this approach reminds you too much of Java, you'll understand why I created `ServiceUtils` earlier when we were using `HTTPService`.) We'll use this file to define our `RemoteObjects`, one for each Rails controller we want to talk to; see listing 11.10.

**Listing 11.10** `app\flex\com\pomodo\business\Services.mxml`

```
<?xml version="1.0" encoding="utf-8"?>
<cairngorm:ServiceLocator 1
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:cairngorm="http://www.adobe.com/2006/cairngorm">
  <mx:RemoteObject id="taskRO" 2
    source="TasksController" 3
    destination="rubyamf"> 4
    <mx:method name="index"/> 5
    <mx:method name="create"/>
    <mx:method name="update"/>
    <mx:method name="destroy"/>
  </mx:RemoteObject>
  <mx:RemoteObject id="projectRO" 6
    source="ProjectsController"
    destination="rubyamf">
    <mx:method name="index"/>
    <mx:method name="create"/>
    <mx:method name="update"/>
    <mx:method name="destroy"/>
  </mx:RemoteObject>
  <mx:RemoteObject id="locationRO" 7
    source="LocationsController"
    destination="rubyamf">
    <mx:method name="index"/>
    <mx:method name="create"/>
    <mx:method name="update"/>
    <mx:method name="destroy"/>
  </mx:RemoteObject>
  <mx:RemoteObject id="noteRO" 8
    source="NotesController"
    destination="rubyamf">
    <mx:method name="show"/>
    <mx:method name="update"/>
  </mx:RemoteObject>
</cairngorm:ServiceLocator>
```

Services.mxml is a Cairngorm ServiceLocator **1**. If you've created a project for the Cairngorm source code, the source file is `com.adobe.cairngorm.business.ServiceLocator`. The ServiceLocator is a Singleton that manages HTTPServices, WebServices, and RemoteObjects for you. The code is fairly simple, so you should read it if you plan to use it.

We create four RemoteObjects: a taskRO **2** whose source is the TaskController **3**, a projectRO **6**, a locationRO **7**, and a noteRO **8**. All of them have a destination of "rubyamf" (for example, **4**), which is the destination id we defined in `services-config.xml` earlier. Finally, each RemoteObject defines each of its methods. Note that the taskRO has no show method **5**, and neither do the projectRO and locationRO. We're not creating one because we don't use it. (I did use it when I was doing my standalone experiments with RubyAMF and tasks, so I'm still showing the code in the controllers because I hope it's useful. Strictly speaking, I should delete that code too because I'm not using it.)

Next, we modify `Pomodo.mxml`; see listing 11.11.

#### Listing 11.11 `app\flex\Pomodo.mxml`

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:pom="com.pomodo.components.*"
  xmlns:business="com.pomodo.business.*"
  xmlns:control="com.pomodo.control.*"
  layout="vertical"
  backgroundGradientColors="[#ffffff, #c0c0c0]"
  horizontalAlign="center"
  verticalAlign="top"
  paddingLeft="0"
  paddingRight="0"
  paddingTop="0"
  paddingBottom="0"
  width="100%"
  height="100%"
  creationCompleteEffect="fadeIn">
<mx:Script>
<![CDATA[
  import mx.core.Container;
  import mx.rpc.events.ResultEvent;
  import mx.rpc.events.FaultEvent;
  import com.pomodo.components.DebugPanel;
  import com.pomodo.control.EventNames;
  import com.pomodo.util.CairngormUtils;
  import com.pomodo.util.DebugMessage;
  import com.pomodo.model.PomodoModelLocator;
  ...
```

← Create new XML namespace for business services

← Remove obsolete imports

```

private function loadFlexibleRails():void {
    CairngormUtils.dispatchEvent(EventNames.LOAD_URL,
        "http://www.flexiblerails.com");
}

private function handleHelloResult(e:ResultEvent):void {
    Pomodo.debug("hello result:\n" + e.message);
}

private function handleFault(e:FaultEvent):void {
    Pomodo.debug("FAULT:\n" + e.fault.faultString);
}
}}>
</mx:Script>
<mx:RemoteObject id="helloRO"
    source="HelloController"
    destination="rubyamf"
    fault="handleFault(event)">
<mx:method name="sayhello"
    result="handleHelloResult(event)"/>
</mx:RemoteObject>
<mx:Fade id="fadeIn" duration="500"/>
<mx:WipeUp id="wipeUp" duration="500"/>

<business:Services id="services" /> ❶

<control:PomodoController id="controller" />

<mx:HBox backgroundColor="#000000" width="100%" height="30"
    horizontalAlign="center" verticalAlign="middle">
    <mx:LinkButton color="#FFFFFF"
        click="loadFlexibleRails()" label="{MARKETING}"/>
</mx:HBox>
<del><mx:Button label="hello" click="helloRO.sayhello.send()"/></del>
<mx:Spacer height="10"/>

...
</mx:Application>

```

Remove "hello world" code

Remove "hello world" code

Remove "hello world" code

Because we created a new XML namespace called `business`, we get to create our business services by saying `business:Services` ❶ to create the `ServiceLocator`. This is kind of cute, so it's the Cairngorm convention.

### 11.3.4 Creating the value objects

Next, we need to create the value objects (VOs), which are extremely boring classes with a bunch of public bindable variables. Note that we're not going to send object graphs: A `ProjectVO` won't contain its `TaskVOs`. I'm doing this to keep

things simple and as close as possible to how I was using XML over HTTPService. (This provides the most direct comparison and lets us reuse the most code.)

First, we'll create the TaskVO; see listing 11.12.

**Listing 11.12** app\flex\com\pomodo\vo\TaskVO.as

```
package com.pomodo.vo {
    [RemoteClass(alias='com.pomodo.vo.TaskVO')] ❶
    [Bindable] ❷
    public class TaskVO {
        public var id:int;
        public var name:String;
        public var notes:String;
        public var projectId:int;
        public var locationId:int;
        public var nextAction:Boolean;
        public var completed:Boolean;
    }
}
```

The RemoteClass annotation ❶ defines the name that this class is known to RubyAMF by (as specified in the config\rubyamf\_config.rb file earlier. Note also that we annotate the entire class ❷ as [Bindable], to spare ourselves from annotating every field.

Next, we create the ProjectVO, LocationVO, and NoteVO classes; see listings 11.13, 11.14, and 11.15.

**Listing 11.13** app\flex\com\pomodo\vo\ProjectVO.as

```
package com.pomodo.vo {
    [RemoteClass(alias='com.pomodo.vo.ProjectVO')]
    [Bindable]
    public class ProjectVO {
        public var id:int;
        public var name:String;
        public var notes:String;
        public var completed:Boolean;
    }
}
```

**Listing 11.14** app\flex\com\pomodo\vo\LocationVO.as

```
package com.pomodo.vo {
    [RemoteClass(alias='com.pomodo.vo.LocationVO')]
    [Bindable]
```

```

    public class LocationVO {
        public var id:int;
        public var name:String;
        public var notes:String;
    }
}

```

**Listing 11.15** app\flex\com\pomodo\vo\NoteVO.as

```

package com.pomodo.vo {
    [RemoteClass(alias='com.pomodo.vo.NoteVO')]
    [Bindable]
    public class NoteVO {
        public var content:String;
    }
}

```

That was easy.

### 11.3.5 Modifying the model objects to produce value objects

Now, we need to modify the model objects so they have the ability to convert themselves to and from the value objects. We start with the Task class; see listing 11.16.

**Listing 11.16** app\flex\com\pomodo\model\Task.as

```

package com.pomodo.model {
    import com.pomodo.util.XMLUtils;
    import com.pomodo.vo.TaskVO;
    public class Task {
    ...
        public function Task(
            name:String = "",
            notes:String = "",
            project:Project = null,
            location:Location = null,
            nextAction:Boolean = false,
            completed:Boolean = false,
            id:int = UNSAVED_ID)
        {
            this.name = name;
            this.notes = notes;
            if (project == null) {
                project = Project.NONE;
            }
            project.addTask(this);
            if (location == null) {

```

**Add import**

**Use separate VO class approach to avoid issues here**

```

        location = Location.NONE;
    }
    location.addTask(this);
    this.nextAction = nextAction;
    this.completed = completed;
    this.id = id;
}

public function toVO():TaskVO {
    var taskVO:TaskVO = new TaskVO();
    taskVO.id = id;
    taskVO.name = name;
    taskVO.projectId = project.id;
    taskVO.locationId = location.id;
    taskVO.nextAction = nextAction;
    taskVO.completed = completed;
    taskVO.notes = notes;
    return taskVO;
}

public static function fromVO(taskVO:TaskVO):Task {
    var model:PomodoModelLocator =
        PomodoModelLocator.getInstance();
    return new Task(
        taskVO.name,
        taskVO.notes,
        model.getProject(taskVO.projectId),
        model.getLocation(taskVO.locationId),
        taskVO.nextAction,
        taskVO.completed,
        taskVO.id);
}

...
public function toUpdateObject():Object {
...

```

Convert Task to TaskVO

Create new Task from TaskVO (method is static)

Next, the Project class; see listing 11.17.

#### Listing 11.17 app\flex\com\pomodo\model\Project.as

```

package com.pomodo.model {
    import mx.collections.ArrayCollection;
    import com.pomodo.util.XMLUtils;
    import com.pomodo.vo.ProjectVO;

    public class Project {
    ...
        public function removeTask(task:Task):void {
    ...

```

Add import

```

    }

    public function toVO():ProjectVO {
        var projectVO:ProjectVO = new ProjectVO();
        projectVO.id = id;
        projectVO.name = name;
        projectVO.notes = notes;
        projectVO.completed = completed;
        return projectVO;
    }

    public static function fromVO(projectVO:ProjectVO):
    Project {
        return new Project(
            projectVO.name,
            projectVO.notes,
            projectVO.completed,
            projectVO.id);
    }

    public function toUpdateObject():Object {
    ...

```

Convert Project to ProjectVO

Create new Project from ProjectVO (method is static)

Now the Location class; see listing 11.18.

**Listing 11.18** app\flex\com\pomodo\model\Location.as

```

package com.pomodo.model {
    import mx.collections.ArrayCollection;
    import com.pomodo.vo.LocationVO;

    public class Location {
    ...
        public function removeTask(task:Task):void {
    ...
        }

        public function toVO():LocationVO {
            var locationVO:LocationVO = new LocationVO();
            locationVO.id = id;
            locationVO.name = name;
            locationVO.notes = notes;
            return locationVO;
        }

        public static function fromVO(locationVO:LocationVO):
        Location {
            return new Location(
                locationVO.name,

```

Add import

Convert Location to LocationVO

Create new Location from LocationVO (method is static)

```

        locationVO.notes,
        locationVO.id);
    }

    public function toUpdateObject():Object {
...

```

Finally, the Note class; see listing 11.19.

**Listing 11.19** app\flex\com\pomodo\model\Note.as

```

package com.pomodo.model {
    import com.pomodo.vo.NoteVO;
}

public class Note {
    public function Note(content:String = "") {
        this.content = content;
    }

    [Bindable]
    public var content:String;

    public function toVO():NoteVO {
        var noteVO:NoteVO = new NoteVO();
        noteVO.content = content;
        return noteVO;
    }

    public static function fromVO(noteVO:NoteVO):Note {
        return new Note(noteVO.content);
    }

    public function toUpdateObject():Object {
        var obj:Object = new Object();
        obj["note[content]"] = content;
        return obj;
    }

    public static function fromXML(note:XML):Note {
        return new Note(note.content);
    }
}

```

**Add import**

**Convert Note to NoteVO**

**Create new Note from Note VO (method is static)**

Now that we've created VO classes and our model classes know how to convert themselves to and from these VO classes, we're almost done. All we need to do is modify the business delegates to send the new VOs over the new RemoteObjects,

make a small change to the PomodoModelLocator, and then modify the commands that use the business delegates.

First, we'll modify the business delegates.

### 11.3.6 Modifying the business delegates

This is where all the setup work we did starts paying off. We start with the TaskDelegate; see listing 11.20.

**Listing 11.20** app\flex\com\pomodo\business\TaskDelegate.as

```

package com.pomodo.business {
    import com.adobe.cairngorm.business.ServiceLocator;
    import com.pomodo.model.Task;
    import mx.rpc.IResponder;
    import com.pomodo.model.Task;
    import com.pomodo.util.ServiceUtils;
    import mx.rpc.remoting.RemoteObject;

    public class TaskDelegate {
        private var _responder:IResponder;

        private var _taskRO:RemoteObject; ❶

        public function TaskDelegate(responder:IResponder) {
            _responder = responder;
            _taskRO =
                ServiceLocator.getInstance().getRemoteObject(
                    "taskRO"); ❷
        }

        public function listTasks():void {
            ServiceUtils.send("/tasks.xml", _responder);
            var call:Object = _taskRO.index.send(); ❸
            call.addResponder(_responder);
        }

        public function createTask(task:Task):void {
            ServiceUtils.send("/tasks.xml", _responder, "POST",
                task.toXML(), true);
            var call:Object = _taskRO.create.send(task.toVO()); ❹
            call.addResponder(_responder);
        }

        public function updateTask(task:Task):void {
            ServiceUtils.send(
                "/tasks/" + task.id + ".xml", _responder, "PUT",
                task.toUpdateObject(), false);
            var call:Object = _taskRO.update.send(task.toVO()); ❺
        }
    }
}

```

Add  
import

```

        call.addResponder(_responder);
    }

    public function destroyTask(task:Task):void {
        ServiceUtils.send(
            "/tasks/" + task.id + ".xml",
            _responder,
            "DELETE");
        var call:Object = _taskRO.destroy.send(task.id); ⑥
        call.addResponder(_responder);
    }
}
}

```

We create a `_taskRO` RemoteObject variable ① to store a reference to the shared "taskRO" RemoteObject gotten from the ServiceLocator ②. We then do remote method calls on its index ③, create ④, update ⑤, and destroy ⑥ methods, passing either nothing ③, the `task.toVO()` ④⑤, or the `task.id` ⑥. Because our Task class can make a TaskVO, we can keep this class fairly thin.

Next, we make essentially the same changes to the other business delegates. First, the ProjectDelegate; see listing 11.21.

**Listing 11.21** `app\flex\com\pomodo\business\ProjectDelegate.as`

```

package com.pomodo.business {
    import com.adobe.cairngorm.business.ServiceLocator;
    import com.pomodo.model.Project;
    import mx.rpc.IResponder;
    import com.pomodo.model.Project;
    import com.pomodo.util.ServiceUtils;
    import mx.rpc.remoting.RemoteObject;
}

public class ProjectDelegate {
    private var _responder:IResponder;

    private var _projectRO:RemoteObject; ①

    public function ProjectDelegate(responder:IResponder) {
        _responder = responder;
        _projectRO =
            ServiceLocator.getInstance().getRemoteObject(
                "projectRO"); ②
    }

    public function listProjects():void {
        ServiceUtils.send("/projects.xml", _responder);
        var call:Object = _projectRO.index.send(); ③
    }
}

```

```

        call.addResponder(_responder);
    }

    public function createProject(project:Project):void {
        ServiceUtils.send("/projects.xml",_responder,
            "POST", project.toXML(), true);
        var call:Object = _projectRO.create.send(    ④
            project.toVO());
        call.addResponder(_responder);
    }

    public function updateProject(project:Project):void {
        ServiceUtils.send(
            "/projects/" + project.id + ".xml",
            _responder, "PUT", project.toUpdateObject(),
            false);
        var call:Object = _projectRO.update.send(    ⑤
            project.toVO());
        call.addResponder(_responder);
    }

    public function destroyProject(project:Project):void {
        ServiceUtils.send(
            "/projects/" + project.id + ".xml",
            _responder, "DELETE");
        var call:Object =
            _projectRO.destroy.send(project.id);    ⑥
        call.addResponder(_responder);
    }
}
}
}

```

We create a `_projectRO` RemoteObject variable ① to store a reference to the shared "projectRO" RemoteObject gotten from the ServiceLocator ②. We then do remote method calls on its index ③, create ④, update ⑤, and destroy ⑥ methods, passing either nothing ③, the `project.toVO()` ④⑤, or the `project.id` ⑥. Again, because our Project class can make a ProjectVO, we can keep this class fairly lightweight.

Next, LocationDelegate, which is more of the same; see listing 11.22.

**Listing 11.22** `app\flex\com\pomodo\business\LocationDelegate.as`

```

package com.pomodo.business {
    import com.adobe.cairngorm.business.ServiceLocator;
    import com.pomodo.model.Location;
    import mx.rpc.IResponder;
    import com.pomodo.model.Location;
    import com.pomodo.util.ServiceUtils;

```

```

import mx.rpc.remoting.RemoteObject;

public class LocationDelegate {
    private var _responder:IResponder;

    private var _locationRO:RemoteObject;

    public function LocationDelegate(responder:IResponder) {
        _responder = responder;
        _locationRO =
            ServiceLocator.getInstance().getRemoteObject(
                "locationRO");
    }

    public function listLocations():void {
ServiceUtils.send("/locations.xml",_responder);
        var call:Object = _locationRO.index.send();
        call.addResponder(_responder);
    }

    public function createLocation(location:Location):void {
ServiceUtils.send("/locations.xml",_responder,
"POST", location.toXML(), true);
        var call:Object = _locationRO.create.send(
            location.toVO());
        call.addResponder(_responder);
    }

    public function updateLocation(location:Location):void {
ServiceUtils.send(
"/locations/" + location.id + ".xml",
_responder, "PUT", location.toUpdateObject(),
false);
        var call:Object = _locationRO.update.send(
            location.toVO());
        call.addResponder(_responder);
    }

    public function destroyLocation(location:Location):
    void {
ServiceUtils.send(
"/locations/" + location.id + ".xml",
_responder, "DELETE");
        var call:Object =
            _locationRO.destroy.send(location.id);
        call.addResponder(_responder);
    }
}
}

```

---

Finally, NoteDelegate, which is also more of the same; see listing 11.23.

**Listing 11.23** app\flex\com\pomodo\business\NoteDelegate.as

```

package com.pomodo.business {
    import com.adobe.cairngorm.business.ServiceLocator;
    import com.pomodo.model.Note;
    import com.pomodo.model.PomodoModelLocator;
    import mx.rpc.IResponder;
    import com.pomodo.model.Note;
    import com.pomodo.model.User;
    import com.pomodo.model.PomodoModelLocator;
    import com.pomodo.util.ServiceUtils;
    import mx.rpc.remoting.RemoteObject;

    public class NoteDelegate {
        private var _responder:IResponder;

        private var _noteRO:RemoteObject;

        public function NoteDelegate(responder:IResponder) {
            _responder = responder;
            _noteRO =
                ServiceLocator.getInstance().getRemoteObject(
                    "noteRO");
        }

        public function showNote():void {
            var model:PomodoModelLocator =
                PomodoModelLocator.getInstance();
            ServiceUtils.send(
                "/users/" + model.user.id + "/note.xml",
                _responder);
            var call:Object = _noteRO.show.send();
            call.addResponder(_responder);
        }

        public function updateNote():void {
            var model:PomodoModelLocator =
                PomodoModelLocator.getInstance();
            ServiceUtils.send(
                "/users/" + model.user.id + "/note.xml",
                _responder,
                "PUT",
                model.note.toUpdateObject(),
                false);
            var call:Object = _noteRO.update.send(
                model.note.toVO());
            call.addResponder(_responder);
        }
    }
}

```

That was easy.

At this point, we'd expect to go modify the commands that use these business delegates. We'll do this, but first we need to modify the PomodoModelLocator.

### 11.3.7 Modifying the PomodoModelLocator

The PomodoModelLocator has methods called setTasks, setProjects, and setLocations that take an XMLList. But we're getting an Array of VOs back from RubyAMF, so we need similar methods to handle these. We don't want a lot of duplication, so we'll refactor out the common functionality as we go. (I don't want to delete the XMLList-using code, because it's nice to have both approaches at our disposal, and because I'm trying to provide as much working code as will fit in a book.)

This is what we'll do. There is a fair bit of new code, but it's all straightforward; see listing 11.24.

**Listing 11.24** app\flex\com\pomodo\model\PomodoModelLocator.as

```
package com.pomodo.model {
    import com.adobe.cairngorm.model.IModelLocator;
    import com.pomodo.control.EventNames;
    import com.pomodo.util.CairngormUtils;
    import com.pomodo.validators.ServerErrors;
    import com.pomodo.vo.TaskVO;
    import com.pomodo.vo.ProjectVO;
    import com.pomodo.vo.LocationVO;

    import mx.collections.ArrayCollection;
    import mx.collections.ListCollectionView;

    [Bindable]
    public class PomodoModelLocator implements IModelLocator {
    ...
        public function removeTask(task:Task):void {
            for (var i:int = 0; i < tasks.length; i++) {
                var ithTask:Task = Task(tasks.getItemAt(i));
                if (ithTask.id == task.id) {
                    ithTask.project.removeTask(ithTask);
                    ithTask.location.removeTask(ithTask);
                    tasks.removeItemAt(i);
                    break;
                }
            }
        }
    }

public function setTasks(list:XMLList):void {

```

**Add imports**

**Rename old setTasks method to setTasksFromList**

```

...
}
public function setTasksFromVOs(taskVOs:Array):void {
    var tasksArray:Array = [];
    for each (var item:TaskVO in taskVOs) {
        tasksArray.push(Task.fromVO(item));
    }
    tasks = new ArrayCollection(tasksArray);
}

public function setTasksFromList(list:XMLList):void {
    var tasksArray:Array = [];
    for each (var item:XML in list) {
        tasksArray.push(Task.fromXML(item));
    }
    tasks = new ArrayCollection(tasksArray);
}

public function setProjects(list:XMLList):void {
...
}
public function setProjectsFromVOs(projectVOs:Array):
void {
    var projectsArray:Array = [];
    for each (var item:ProjectVO in projectVOs) {
        projectsArray.push(Project.fromVO(item));
    }
    setProjects(projectsArray);
}

public function setProjectsFromList(list:XMLList):void {
    var projectsArray:Array = [];
    for each (var item:XML in list) {
        projectsArray.push(Project.fromXML(item));
    }
    setProjects(projectsArray);
}

public function setProjects(projectsArray:Array):void {
    projectIDMap = {};
    projectIDMap[0] = Project.NONE;
    for each (var project:Project in projectsArray) {
        projectIDMap[project.id] = project;
    }
    projects = new ArrayCollection(projectsArray);
    var projectsAndNoneArray:Array =
        projectsArray.slice(0);
    projectsAndNoneArray.splice(0, 0, Project.NONE);
    projectsAndNone =
        new ArrayCollection(projectsAndNoneArray);
}

```

Create new setTasksFromVOs method for AMF  
 Rename setTasks method to setTasksFromList  
 Rename old setProjects method to setProjectsFromList  
 Create new setProjectsFromVOs method for AMF  
 Rename setProjects method to setProjectsFromList  
 Common functionality factored out into method

```

    _gotProjects = true;
    listTasksIfMapsPresent();
}

public function setLocations(list:XMLList):void {
...
}
public function setLocationsFromVOs(locationVOs:Array):
void {
    var locationsArray:Array = [];
    for each (var item:LocationVO in locationVOs) {
        locationsArray.push(Location.fromVO(item));
    }
    setLocations(locationsArray);
}

public function setLocationsFromList(list:XMLList):
void {
    var locationsArray:Array = [];
    for each (var item:XML in list) {
        locationsArray.push(Location.fromXML(item));
    }
    setLocations(locationsArray);
}

public function setLocations(locationsArray:Array):
void {
    locationIDMap = {};
    locationIDMap[0] = Location.NONE;
    for each (var location:Location in locationsArray) {
        locationIDMap[location.id] = location;
    }
    locations = new ArrayCollection(locationsArray);
    var locationsAndNoneArray:Array =
        locationsArray.slice(0);
    locationsAndNoneArray.splice(0, 0, Location.NONE);
    locationsAndNone =
        new ArrayCollection(locationsAndNoneArray);
    _gotLocations = true;
    listTasksIfMapsPresent();
}

public function getProject(projectID:int):Project {
    if (projectIDMap == null) return null;
    return projectIDMap[projectID];
}
...

```

**Rename old setLocations method to setLocationsFromList**

**Create new setLocationsFromVOs method for AMF**

**Rename setLocations method to setLocationsFromList**

**Common functionality factored out into method**

All that is left is modifying the commands!

### 11.3.8 Modifying the commands

Again, this is straightforward. We'll start with the `DestroyTaskCommand`; see listing 11.25.

**Listing 11.25** `app\flex\com\pomodo\command\DestroyTaskCommand.as`

```

package com.pomodo.command {
...
    import com.pomodo.model.Task;
    import com.pomodo.vo.TaskVO;
    import com.pomodo.util.CairngormUtils;
...
    public class DestroyTaskCommand implements ICommand,
        IResponder {
...
        public function result(event:Object):void {
            var resultEvent:ResultEvent = ResultEvent(event);
            var model:PomodoModelLocator =
                PomodoModelLocator.getInstance();
            if (event.result == "error") {
                Alert.show(
                    "The task was not successfully deleted.",
                    "Error");
            } else {
                model.removeTask(
                    Task.fromXML(XML(event.result)),
                    Task.fromVO(TaskVO(event.result)));
            }
        }
...
    }
}

```

**Add import**

**Create new Task from TaskVO, not XML**

Next, the `ListTasksCommand`; see listing 11.26.

**Listing 11.26** `app\flex\com\pomodo\command>ListTasksCommand.as`

```

package com.pomodo.command {
...
    public class ListTasksCommand implements ICommand,
        IResponder {
...
        public function result(event:Object):void {
            var model:PomodoModelLocator =
                PomodoModelLocator.getInstance();
            model.setTasks(
                XMLList(event.result.children())),
            model.setTasksFromVOS(event.result);
        }
...
    }
}

```

**Set Tasks from VOs, not XML**

```
    }  
    ...  
  }
```

Now, the ListProjectsCommand; see listing 11.27.

**Listing 11.27** app\flex\com\pomodo\command>ListProjectsCommand.as

```
package com.pomodo.command {  
    ...  
    public class ListProjectsCommand implements ICommand,  
        IResponder {  
    ...  
        public function result(event:Object):void {  
            var model:PomodoModelLocator =  
                PomodoModelLocator.getInstance();  
            model.setProjects(  
                XMLList(event.result.children()));  
            model.setProjectsFromVOs(event.result);  
        }  
    ...  
    }
```

← Set Projects from  
VOs, not XML

Next, the ListLocationsCommand; see listing 11.28.

**Listing 11.28** app\flex\com\pomodo\command>ListLocationsCommand.as

```
package com.pomodo.command {  
    ...  
    public class ListLocationsCommand implements ICommand,  
        IResponder {  
    ...  
        public function result(event:Object):void {  
            var model:PomodoModelLocator =  
                PomodoModelLocator.getInstance();  
            model.setLocations(  
                XMLList(event.result.children()));  
            model.setLocationsFromVOs(event.result);  
        }  
    ...  
    }
```

← Set Locations from  
VOs, not XML

Now, the ShowNoteCommand; see listing 11.29.

**Listing 11.29** app\flex\com\pomodo\command\ShowNoteCommand.as

```

package com.pomodo.command {
...
    public class ShowNoteCommand implements ICommand,
        IResponder {
...
        public function result(event:Object):void {
            var model:PomodoModelLocator =
                PomodoModelLocator.getInstance();
            model.note = Note.fromXML(event.result);
            model.note = Note.fromVO(event.result);
        }
...
}

```

← **Set Note from the VO, not XML**

Finally, the UpdateTaskCommand; see listing 11.30.

**Listing 11.30** app\flex\com\pomodo\command\UpdateTaskCommand.as

```

package com.pomodo.command {
    import com.adobe.cairngorm.commands.ICommand;
    import com.adobe.cairngorm.control.CairngormEvent;
    import com.pomodo.business.TaskDelegate;
    import com.pomodo.control.EventNames;
    import com.pomodo.model.PomodoModelLocator;
    import com.pomodo.model.Task;
    import com.pomodo.util.CairngormUtils;
    import com.pomodo.vo.TaskVO;
...
    public class UpdateTaskCommand implements ICommand,
        IResponder {
...
        public function result(event:Object):void {
            var resultEvent:ResultEvent = ResultEvent(event);
            var model:PomodoModelLocator =
                PomodoModelLocator.getInstance();
            model.updateTask(Task.fromXML(XML(event.result)));
            model.updateTask(Task.fromVO(TaskVO(event.result)));
            CairngormUtils.dispatchEvent(
                EventNames.LIST_PROJECTS);
        }
...
}

```

← **Add import**

← **Update Task by creating Task from VO, not XML**

Did I forget UpdateProjectCommand, UpdateLocationCommand, and UpdateNoteCommand? No—they're unchanged.

That's it! Run `newdb.bat` to recreate the `ludwig` user that we deleted at the end of iteration 10, then restart your server, rebuild, reload, and log in as `ludwig`. Everything works as before. Finally, run the tests again and confirm that everything still works.

The code at this point is saved as the `iteration11` folder.

## 11.4 Summary

---

Now that we have a proper object model on the client side, we're no longer tightly coupled to the transport mechanism. This let us refactor the code to using `RubyAMF` and keep the object model essentially unchanged (except for adding the ability to convert to/from value objects).

`RubyAMF` is a promising and fast-moving project. The fact that it's essentially MIT-licensed and that it plays so nicely with Rails means that I expect it to become the dominant AMF implementation for Ruby. It has a lot of momentum behind it at the moment.

Using `RubyAMF` isn't an all-or-nothing proposition: You can use it in performance-critical parts of your application, if you want to minimize the amount of code you're writing. For example, in this iteration I didn't feel like revisiting the login code, so I left it using XML and `HTTPService`. Besides demonstrating how lazy I'm getting, this also demonstrated that `RubyAMF` and `HTTPService` can be used side by side.

In the next and final iteration, we'll revisit the `pomodo` application one last time, extending it to becoming an Adobe AIR application.

# FLEXIBLE RAILS

Peter Armstrong

Foreword by Stuart Eccles, Made by Many, Ltd.

**F**lex on Rails may sound like an odd coupling but, in fact, the two are a great fit. Adobe's once expensive and proprietary (now free and open-source) rich user interface platform (Flex) used on top of the upstart, open-source, Ruby-driven web development environment (Rails) is like a Ferrari automobile—beautiful on the outside, fast on the inside, and incredible to drive. Flex offers amazing new UI options to go with Rails' famously-short development cycles.

**Flexible Rails** teaches Flex 3 on Rails 2 techniques in an unusually effective way: its chapters take an application through eleven iterations, each increasingly “real world.” You'll learn how Flex and Rails integrate with HTTPService and XML and see how Rails' RESTful controller design supports both Flex and HTML clients. Along the way, you'll learn to use the Cairngorm MVC framework in large Flex applications and work with the RubyAMF Flash Remoting gateway.

Written with ongoing feedback from over one thousand Early Access readers, **Flexible Rails** is practical, code-focused, and readable. It is accessible to web or desktop UI developers who have “hello world” level exposure to Rails and little or no knowledge of Flex.

## What's Inside

- All content based on Flex 3 and Rails 2
- A complete, professional-quality (and MIT-licensed) Flex on Rails application
- Rails on AIR, the Adobe Integrated Runtime

**Peter Armstrong** is a Vancouver-based RIA developer who uses Flex and Rails daily. He is a frequent public speaker and was a key part of the team that won the 2006 Adobe MAX Award for RIA/Web Development.

For more information, code samples, and to purchase an ebook visit [www.manning.com/FlexibleRails](http://www.manning.com/FlexibleRails)



“... a first-class learning experience that is tough to quantify but easy to qualify: Don't miss it.”

—Louis F. Springer  
Sun Microsystems

“*Flexible Rails* changes the face of web applications development.”

—Robert Dempsey  
Atlantic Dominion Solutions

“Well done! ... a thorough job of blending two of the best technologies!”

—Christopher Bailey  
Bring Light

“Flex 3 on Rails 2 is much more than the sum of its parts—and this book is the key.”

—Arne Pfeilsticker  
Pfeilsticker Software  
+ Services GmbH

“A really nice book that covers all about Flex and Rails working together.”

—Paulo Fernando Larini  
Benner Health Systems

