Mobilizing Your Application with Rhomobile

From Rhomobile

Rhomobile provides a set of components that makes it easy to build mobile interfaces to your enterprise application backends. All of these components are available in current form at Rhomobile repository on github (http://github.com/rhomobile) The two primary relevant tools are Rhodes, a Rails-like Ruby-based microframework for building locally executing mobile applications, and RhoSync, which provides client and server sync components for maintaining current copies of backend app information on your device. This article describes how to use Rhodes and RhoSync to build such mobile apps.

Contents

The Rhodes Microframework

We at Rhomobile are big fans of both Ruby on Rails and Ruby. We've all developed quite a few applications, including mobile web applications, in Ruby. But we believe that the best mobile applications execute natively against local data, as opposed to connecting to remote websites. Rhodes allows you to build such applications easily, writing the app one time in a way such that it will run natively on all major smartphone operating systems: iPhone, Windows Mobile, BlackBerries and Symbian.

Rhodes is essentially a combination of:

- a minimalist Ruby implementation for mobile devices
- a Model-View-Controller "microframework" that consists of a directory structure and file naming convention. Writing to this framework primarily consists of editing a set of ERB templates for creating HTML to display data
- an application generator called RhoGen which generates a basic Create-Read-Update-Delete controller and associated views for specified data objects
- the RhoSync synchronization engine client
- a simple object relational manager (ORM) called Rhom
- a web server which is installed on the mobile device and runs locally

All of Rhodes fits in around 2 megabytes of local storage on the mobile device. In this article we'll describe how to use Rhodes to provide a mobile interface to your backend application. The first step of this is to provide a way to get data of interest available on your mobile device.

RhoSync - the Web Service Sync Engine

RhoSync is a synchronization engine consisting of a client component on the device and a server component that runs on any server capable of running Rails applications. Unlike other sync solutions that have existed in the past, such as the now discontinued IntelliSync, RhoSync has been written to interact with web services versus databases. In this way, it is particularly suited to integrate with Software As A Service (SAAS) applications such as SugarCRM, SalesForce, Siebel and other vendors that provide enterprise applications on a hosted subscription basis.

Like Rhodes, RhoSync is available on Github (http://github.com/rhomobile/rhosync/tree/master) . RhoSync requires that you write a small amount of Ruby code for the query, create, update and delete operations of your particular enterprise backend. The collection of the Ruby code for these operations we refer to as a "source" or "source adapter". The RhoSync server provides a web interface that makes it easy to manage such sources.

Install the RhoSync Server

This section describes how to get RhoSync built from github and get it running on a server. These instructions do assume an operational and current Rails environment. If you don't have at least version 2.2.2 of Ruby on Rails installed, go to the Ruby on Rails site (http://rubyonrails.org) for instructions on installing Rails.

First create a clone of the RhoSync project. [Windows users: Note that in all of the text below on Windows the sudo commands are not necessary, just do "gem install ..."]. You might need to install git if you don't have it.

```
sudo gem install rspec-rails rake
git clone git://github.com/rhomobile/rhosync.git
cd rhosync
```

Install your database driver of choice

```
sudo gem install sqlite3-ruby (-v=1.2.3 if you're in windows)
or
sudo gem install msyql
```

If you don't have the file "log/development.log" then go ahead and create the file:

```
mkdir -p log; touch log/development.log (or create manually if in windows)
```

Install required gems

```
sudo rake gems:install
```

Configure config/database.yml to tell rails how to connect to your database:

```
<edit> config/database.yml
```

Then run the db:bootstrap rake task provided, creating the database and associated tables that RhoSync needs. It will also load the source adapters that we ship with.

```
rake db:bootstrap
```

Then use your Rails server of choice to run the application. If you don't have a Rails server, you can install mongrel with:

```
sudo gem install mongrel
```

Then start your server:

```
script/server
```

Then you can view the Sources from your web browser:

```
http://localhost:3000
```

If you want use to the sample application and sources that we ship with, just register an account called "admin" with your password of choice. You will then see the apps "owned" by "admin": SugarCRM and Siebel. Clicking on SugarCRM will show you the sources that have been defined for SugarCRM: Accounts, Cases and Employees. From the RhoSync screen you can also see the data there, which is using our sample SugarCRM instance. You will need to change the URL for the SugarCRM WSDL, and the login and password however to point to your own SugarCRM instance. For example the WSDL could be "http://yourinstance.sugarondemand.com/soap.php?wsdl". The sections below describe how to set up source adapters for other backends.

Defining RhoSync Source Adapters

Once RhoSync is installed we're ready to build a RhoSync source to integrate with our backend application. To define a RhoSync source you just need to identify a handful of operations to interact with your backend data source. These are:

- login - how to authenticate for the backend

- query - retrieve or read's objects from the backend source
- sync - take apart these objects and put them into a "property bag" (a set of object-attribute-value rows)
- create - creates objects in the backend
- update - updates objects on the backend
- delete - deletes objects on the backend
- logout - how to shutdown your session with your application backend. It doesn't always exist for every backend

There are two ways of creating code for source adapters. For simpler web service interfaces (such as SugarCRM's shown below) its usually easy enough to just edit the code in the provided web page form. For more complex code you may want to just provide RhoSync with a full-fledged class. To do this, just provide the name of the Source Adapter Class in the form for a source adapter.

Generating A Source Adapter Class

To do this, you need the RhoGen application generator. You can get this by getting the Rhodes gem from RubyForge

```
sudo gem install rhodes
```

If you're still running rubygems < 1.3 (simple check with "gem -v"), you'll need to upgrade to the latest rubygems as well

```
sudo gem update --system
```

Then run the source generator with the name of the source adapter class. For example (pick a name for your source adapter to replace SugarAccounts below):

```
rhogen source SugarAccounts
```

You'll see a file similar to the following one below. From there you'll fill in the login, query, create, update, delete and logoff methods with your own code. Note that you don't need to use the source generator. You can just create a Ruby file and place it into your lib directory. The class name must match that identified on the Source Adapter form.

```
class SugarAccounts < SourceAdapter
  def initialize(source)
    super(source)
  end
  def login
  end
  def query
  end
  def sync
    super
  end
  def create(name_value_list)
  end
  def update(name_value_list)
  end
  def delete(name_value_list)
  end
  def logoff
  end
 end
```

The description below shows what such code might look like. If you don't want to write an entire class, just leave Source Adapter Class empty and fill in the code inline on the form.

A RhoSync Login

The following is login code sample source adapter for SugarCRM accounts. This code can be placed inside a Source Adapter Class shown above (specifically inside the login method). Or it can be placed inside the Login part of the Source form if you're not using a source adapter class. It uses the SOAP web service interfaces exposed by SugarCRM.

```
u=@source.login
pwd=Digest::MD5.hexdigest(@source.password)
ua={'user_name' => u,'password' => pwd}
ss=client.login(ua,nil)
if ss.error.number.to_i != 0
 puts 'failed to login - #{ss.error.description}'
else
 session_id = ss['id']
 uid = client.get_user_id(session_id)
end
```

Notice that you access parameters that have been set for the source as attributes of the @source variable. For example the login and password are available in @source.login and @source.password. Also note that the "client" variable contains the SOAP driver that methods are exposed on.

Also note that subsequent operations will use the session_id variable. You can use whatever variable name you wish for the session but we suggest following this convention.

Available "Builtin" Variables

- client - the SOAP driver created from the WSDL file at @source.url. Not necessary if you are using a REST API
- @source.url - the URL where the backend is exposed. Typically used if you are calling a REST API
- @source.login - the userid to login with
- @source.password - the password to login with

A RhoSync Query

The following is sample query code from the SugarCRM Accounts source. Just one line does the trick. The one that ship is the db/migrate/sources.yml file has more comments and intermediate variables for understandability but does exactly the same thing.

```
result = client.get_entry_list(session_id,'Account','','',0,['name','industry'],10000,0)
```

A RhoSync Sync

The sync code takes apart the query results and puts its into the object_values table. For example, this handles the results coming back from SugarCRM web service calls.

```
 result.entry_list.each do |x|
  x.name_value_list.each do |y|
    o=ObjectValue.new
    o.source_id=@source.id
    o.object=x['id']
    o.attrib=y.name
    o.value=y.value
    o.update_type='query'
    o.save
  end
```

See the Sugar adapter in the db/migrate/sources.yml. This can be loaded into your RhoSync server with the "Load Sources" operation but is automatically if you run rake db:migrate. It has examples of other operations (such as Create, Update, Delete and Logff). Once you've edited the code to your satisfaction, click on Update Source Adapter Code to save it. Assuming your code is correct, you should then see the resulting object/attribute/value triples coming back from the backend in your browser.

Creating Objects with RhoSync

You can assume that you will get an array of hashes of attribute value pairs with entitled "name_value_list". So for example it might be:

```
[{"name"=>"id","value"=>"1"},{"name"=>"name","value"=>"Acme Widgets"}]
```

Your code for create (or edit or delete) needs to use this populated array to do it work. For SugarCRM the create code is:

```
result=client.set_entry(session_id,'Cases',name_value_list)
```

Again, client was created automatically from the WSDL at the address specified in @source.url. Thats a bit too straightforward though, because SugarCRM expects an array of hashes formulated just like we received in our source adapter For Siebel Field Service the code is a bit more complex.

```
  ServiceRequestWS_ServiceRequestInsert_Input.new
  s=ServiceRequest.new
  s.serviceRequestId=""
  s.accountName=name_value_list[0]["value"] # account name
  s.description=name_value_list[1]["value"] # description
  s.priority=name_value_list[2]["value"] # priority
  s.subject=name_value_list[3]["value"] # subject
  input.listOfServiceRequest=[s]
  output=obj.serviceRequestInsert(input)
```

This particular example used methods generated the WSDL2Ruby utility which will be in "defaultDriver.rb".

Testing Your Rhosync Adapter

You can use the web interface to test that you are in fact retrieving data from your backend. Click on Show Records from the source adapter console to view the "object-attribute-values" coming from your backend data source. You can also edit, create and delect objects from this same web interface.

Creating users and apps

In order to use your source adapter, you'll need to setup an application and user for your rhosync system. Please see Configure the SugarCRM Sample App for your own SugarCRM Instance tutorial for more information.

Building Adapters for REST Backends

Building a RhoSync adapter for a REST backend is very similar to doing it for SOAP backends. Its all a matter of writing the right Ruby code. The following example source adapter methods, from our LightHouse app, demonstrate how to interact with backends that expose REST interfaces. For example, below is an example from our Lighthouse sample app of how to query for Lighthouse tickets

```
def query
  uri = URI.parse(@source.url+"/tickets.xml")
  req = Net::HTTP::Get.new(uri.path, 'Accept' => 'application/xml')
  req.basic_auth @source.login, @source.password
  response = Net::HTTP.start(uri.host,uri.port) do |http|
    http.request(req)
  end
  xml_data = XmlSimple.xml_in(response.body);
  @result = xml_data["ticket"]
end
```

Building Your Rhodes Application

With a RhoSync source operational to generate synced data, you are now ready to write a Rhodes application to allow interaction with that data from your mobile device. Writing a Rhodes app consists of the following steps:

- checking out Rhodes
- generating Rhodes controllers for your data models
- editing the views for your controller actions
- testing your application

Checkout Rhodes

The first step in using Rhodes is to make a clone of Rhodes itself so that you can work on your app in the \apps subdirectory of Rhodes. Rhodes is located on Github in the Rhomobile repository (http://github.com/rhomobile) as the Rhodes project (http://github.com/rhomobile/rhodes/tree/master) . To work with it first create a clone of the public URL:

```
git clone git://github.com/rhomobile/rhodes.git rhodes
```

Notice that we are using the public clone URL. We would prefer that you don’t check your apps back in to Rhodes (and you won’t be able to anyway). So this is what makes sense for working on apps versus contributing to Rhodes itself. You will then do your work in \apps directory as described below.

Generating Rhodes Application and Resources

With a RhoSync source working to retrieve data locally, now we're ready to start writing our Rhodes app. The first step is to generate the application and base files. This is done with the RhoGen (Rhodes Generator) utility, which is available by installing the Rhodes gem (as described above in the RhoSync section).

First we want to generate an application called AccountApp with the following:

```
cd rhodes/apps
rhogen app AccountApp
```

This will generate an application directory called AccountApp with the following files:

- application.rb
- index.html

Generally you will want to have the Rhodes runner start up with your new app. To do this edit the config.rb file in the apps directory and set the start path to your app:

```
 require 'rho'
 Rho::RhoConfig.start_path = '/AccountApp'
```

Alternatively you can edit the top level sample apps index page (index.erb in the apps directory) as follows:

```
  <h4>Sample Apps</h4>
  <a href="AccountApp">SugarCRM Accounts</a><br/>
```

Next, we want to generate a model for our application, let's call it "Account" and give it a name and industry attribute (first moving to the generated directory):

```
cd AccountApp
rhogen model Account "http://rhosyncdev.rhohub.com/sources/1" 1 name,industry
```

This will generate the following files in a directory called "Account":

- controller.rb - contains the basic CRUD actions: index, new, create, edit, update and delete.
- config.rb - provides the URL to connecting with the sync engine
- index.erb - the HTML template to display the list of objects
- new.erb - the template to supply values to create a new object
- edit.erb - the template to edit an object
- show.erb - displays the selected object

The source_url and source_id (parameters 3 and 4) refer to the rhosync source show url and id for the source, respectively.

Once this is generated you will probably want to go back and edit the index.html file for the app to have a link to the generated model. For example:

```
  <li><a href="Account">Accounts</a></li>
```

Editing Rhodes Views

You can edit the generated ERB files to optimize the HTML as you see fit. For example, below is the edit.erb file for editing an individual account.

```
<form title="Edit Account" class="panel" id="account_edit_form" method="POST"  action="<%=
        <fieldset>
              <input type="hidden" name="id" value="<%=@account.object%>"/>
              <div class="row">
                     <label>Name: </label>
                     <input type="text" name="account[name]" value="<%=@account.name%>"
              </div>
              <div class="row">
                     <label>Industry: </label>
                     <select name="account[industry]">
                            <%@industries.each do |i|%>
                                   <option value=<%=i%>
                                   <%if @account.industry==i%>
                                   selected<%end%>><%=i%>
                                   </option>
                            <%end%>
                     </select>
              </div>
        </fieldset>
        <input type="submit" value="Update"/>
        <p align="center">
              <a href="<%=url_for('delete', @account.object)%>">
                      <font color="#990000">Delete</font>
              </a>
        </p>
 </form>
```

For example, you could create additional instance variables (@ variables) in controller.rb and modify the edit.erb to display them.

Testing Your Application on Devices

After you've built your controllers and templates, the next step to using is to test and execute the application on your device or simulator. The process for this will vary for each mobile device operating system. See Building Rhodes on Supported Platforms.

Questions

Check the Frequently Asked Questions (http://rhomobile.com/documentation/faq) and then write to us (mailto:info@rhomobile.com) or join the Rhomobile Google Group (http://groups.google.com/group/rhomobile) .

Retrieved from "http://wiki.rhomobile.com/index.php/Mobilizing_Your_Application_with_Rhomobile"

- This page was last modified on 19 February 2009, at 23:09.