# IronRuby on Rails

## Introduction – 4:15

Welcome! I want to make sure to finish on time for Robert Martin’s “What killed Smalltalk could kill Ruby too talk”, so let’s get started. I’m Jimmy Schementi and I work on IronRuby, an open-source implementation of Ruby 1.8.6 for the .NET Framework, at Microsoft … erm, ugh, must have been a projector malfunction. In short, I get paid to work on open-source, and in light of recent events is especially awesome.

## Why the hell are you here

Quick poll: How many are part of .NET shops currently, or have moved from .NET to Rails? Just curious about IronRuby? Implementers? Others: see slide =) seriously though, let me know what you want to get out of this.

## Agenda

Last year, IronRuby was much like a sprout on a tree. Much of the “xylem and phloem” was there, but it has only just begun to stretch towards the sun. Ok, I’m done with this fuckin’ metaphor. Try again.

<>And growing up is hard to do. This was probably IronRuby right after RailsConf 2008. It’s just starting to run Rails, and experience all the stresses of the world around it. <> The good news is that today, IronRuby is growing in, meaningfully interacting with the world around it, and begin to get good at things. Though this kid creeps me out, so bad analogy. Again, point is IronRuby is running real Ruby programs, like Rails and Sinatra, to name a few relevant to RailsConf.

<>However, to work in the real world, where the real world is defined as the .NET framework (which is not the real world, but just an illusion pulled over Microsofty’s eyes), IronRuby needs to stretch out and be good at other things than its parent is (MRI). At a basic form this means accessing the .NET framework just like it was a set of Ruby libraries, but extends much further than that. Today I’ll show IIS running IronRuby-based applications, and other examples of how .NET makes Ruby better (but not at the price of breaking!)

<>And lastly, people are starting to notice us. Community involvement is starting to pick up, and our “real Ruby implementation” status is getting closer and closer.

## Growing up

If you haven’t met IronRuby before, let’s do so.

### .NET Interop

require 'mscorlib'

include System

System.constants.sort

$class\_names = System.constants.sort.grep(/^[A-C]/)

$classes = $class\_names.map { |c| eval(c) }

$classes.first.to\_clr\_type.is\_interface

### COM Interop

def newCOM progId

System::Activator.CreateInstance(System::Type.GetTypeFromProgID(progId))

end

ex = newCOM("Excel.Application")

ex.Visible = true

nb = ex.Workbooks.Add

ws = nb.Worksheets[1]

p ws.Name

10.times do |i|

10.times do |j|

ws.Cells[i + 1, j + 1] = (i + 1) \* (j + 1)

end

end

Ok, let’s cut to the chase: IronRuby running Rails.

**>>>** [**http://localhost:3000**](http://localhost:3000)**, click around on everything. Also switch to script\console and show stuff there too.**

#### activerecord-mssql-adapter.gem

**>>>** [**http://github.com/jschementi/activerecord-mssql-adapter/blob/cd1a04953b5a1c745a6318439b7eac1fc575c26f/mssql\_adapter.rb**](http://github.com/jschementi/activerecord-mssql-adapter/blob/cd1a04953b5a1c745a6318439b7eac1fc575c26f/mssql_adapter.rb)

That database adapter is a IronRuby ActiveRecord adapter for SqlServer; or any database which provides an implementation of the .NET System::Data::SqlClient API, like the .NET version of db4o. Special thanks to Ray Vernagus for helping out on this. He’ll also be abstracting this work to support and ORM, like DataMapper.

We also run Sinatra:

**>>>** [**http://localhost:4567**](http://localhost:4567)**, click around.**

Which means we run “Rack”, since Rails and Sinatra depend on it, but we’ll come back to that.

Let’s see a kinda-real application.

**>>>** [**http://localhost:4568**](http://localhost:4568)

This was written by Chad Fowler, Rich Kilmer, and myself as a demo initially intended for Windows Azure, but useful as a semi-real app IronRuby can showcase.

### Performance

A quick note about the performance just witnessed**. :: TODO ::**

## The Real World

The “Real World” is why you should care about IronRuby, especially if you are already a Windows or .NET shop. None of the benefits of IronRuby would be possibly if you couldn’t deploy your apps on web servers your infrastructure supports.

### ironruby-rack

I mentioned “rack” before; it’s simply a layer between web servers and Ruby frameworks. Since IIS is programmable through .NET code (ASP.NET), and IronRuby runs Rack, we just need glue between IIS and Rack, which Rack calls a “Handler”. That’s exactly what I’ll be demonstrating.

**>>> Navigate to** [**http://localhost/RackApp**](http://localhost/RackApp)**, then** [**http://localhost/SinatraApp**](http://localhost/SinatraApp)**. Then walk-through it. Focus on hosting API use.**

<> The point of doing this, rather than just setting up Apache and Mongrel on Windows, despite the Windows performance implications, it’s a people issue: if your organization has an IIS knowledgebase, you’ll want to utilize it. Deploying this way requires no retraining, as this solution is purely ASP.NET based; no throwing away of experience. It’s also a great way to sneak Ruby into your organization =)

>>> Show how to deploy the app easily, with a Ruby script.

>>> Performance?

### Pimp.NET

The next couple of demos are purely to Pimp .NET, so you’ve been warned. If I had a pimp I’d be wearing it. The point of all these demos is to show how IronRuby makes the .NET platform look just like Ruby code.

#### DSLs

<> Ruby’s beautiful “Internal DSL” ability is very useful in frameworks which weren’t designed for Ruby. A small number of methods and you’ve got a DSL for creating WPF UIs. Thaibut has taken this a step further and is creating the same DSL for building WinForms, WPF, and Silverlight UIs. Pretty awesome stuff.

#### ActiveRecord and Databinding

I remember the first time I used ActiveRecord outside Rails, and it was awesome. Taking that a step further, using ActiveRecord to “data bind” to WinForms UI elements makes UI creation stupid-simple.

> rc09

> cd ardb

> ir app.rb

#### ASP.NET MVC

Ivan Porto Carrero, Phil Haack, and I have been building a nice Ruby API around the new ASP.NET MVC framework, which is very much influenced by Merb. The building of this API has helped inform the MVC team about places where they were killing extensibility, so the project benefited everyone. And because the whole web stack is compiled code, it’s the fastest Ruby web framework I’ve shown. =P

**:: Show calculator Silverlight app ::**

#### Silverlight

The calculator in the previous app was written in Silverlight, but tested with IronRuby and Bacon in IN THE BROWSER. For people who still love static languages, Ruby can be extremely useful for testing and scripting scenarios.

**:: Show tests ::**

I’ve got a BoF session tonight at 9pm to elaborate a bit more about IronRuby in the browser, so if you’re interested in this please attend.

## Getting Noticed

**::TODO::**

## Backup

### Scenarios

Running actual application is really cool to see, but the real value Rails and the plethora of existing Ruby code gives to newer languages like IronRuby (as well as Rubinius, MacRuby, etc) is the real-world testing that can be done simply. I’ll talk later about IronRuby’s involvement with RubySpec and other testing challenges, but having this massive amount of code to just throw at the implementation is a fool-proof way to test (and fix) the right parts of developing implementations. So IronRuby tracks its ability to run existing code very closely, and I’ll walk you through our status with Rails.

A walk-through can be found on <http://ironruby.net/Documentation/Rails>.

#### irails Foo

Works as of very recently due to Jirapong taking ownership of the OpenSSL library, which fell over since #random\_bytes was not implemented.

#### ir script\generate

This seems to work all the time, which is awesome.

#### irake db:migrate

Both tests out Rake as well as ActiveRecord. This works nicely with one patch to Rake, since we don’t correctly implement Method#to\_proc.

#### ir script\server

We only support WEBrick, since all the other servers have some portion of native code that would need to be ported to .NET (much like how JRuby has a port of Mongrel. In the future we might just port the Java version).

### Rails unit tests (ActionPack and ActiveRecord)

The ultimate measure is of how the unit test suites run

<show unit test results />

### Performance

So, IronRuby can run real Ruby programs, awesome. The “vision” is *finally* coming true. But is it fast? How does it stack up to JRuby and MRI?

#### Startup

IronRuby is pretty horrible at startup. Our ir.exe vs. ruby.exe benchmark shows we are about 10x slower that MRI. (JRuby?). 4x of that diff is mostly due to the cost of starting the CLR and JIT’ing the initial code to run, where ruby.exe just needs to run the native code. IronRuby takes 39 seconds to boot WEBrick+Rails to the point it can serve a request, where ruby.exe takes 6.7 seconds, so we are about 6x slower than Ruby on Rails startup.

#### Throughput

Once IronRuby has started up, the performance gap narrows further. The first request is only 3x slower than MRI, while n requests after that are about 2x slower, the same benchmark result we see when running a tight-loop.

#### Working set

Being a .NET application needing to emit a ton of IL, our working set is always painful. Startup working set for ruby.exe is 26MB, while IronRuby is 99MB. JRuby? Serving a simple hello world request has no working set cost on Ruby, but IronRuby jumps to 113MB. But memory is cheap, right? =P

#### Compare against JRuby/CRuby

**TODO**