swiftcore iowa

A web development framework for Ruby

Kirk Haines
wyhaines@gmail.com

History

Originated as a prototype/experiment by Avi Bryant in 2001.

I liked what I saw. It was completely unsuited for production use, though.

Avi had moved on and more or less abandoned the code by early in 2002.

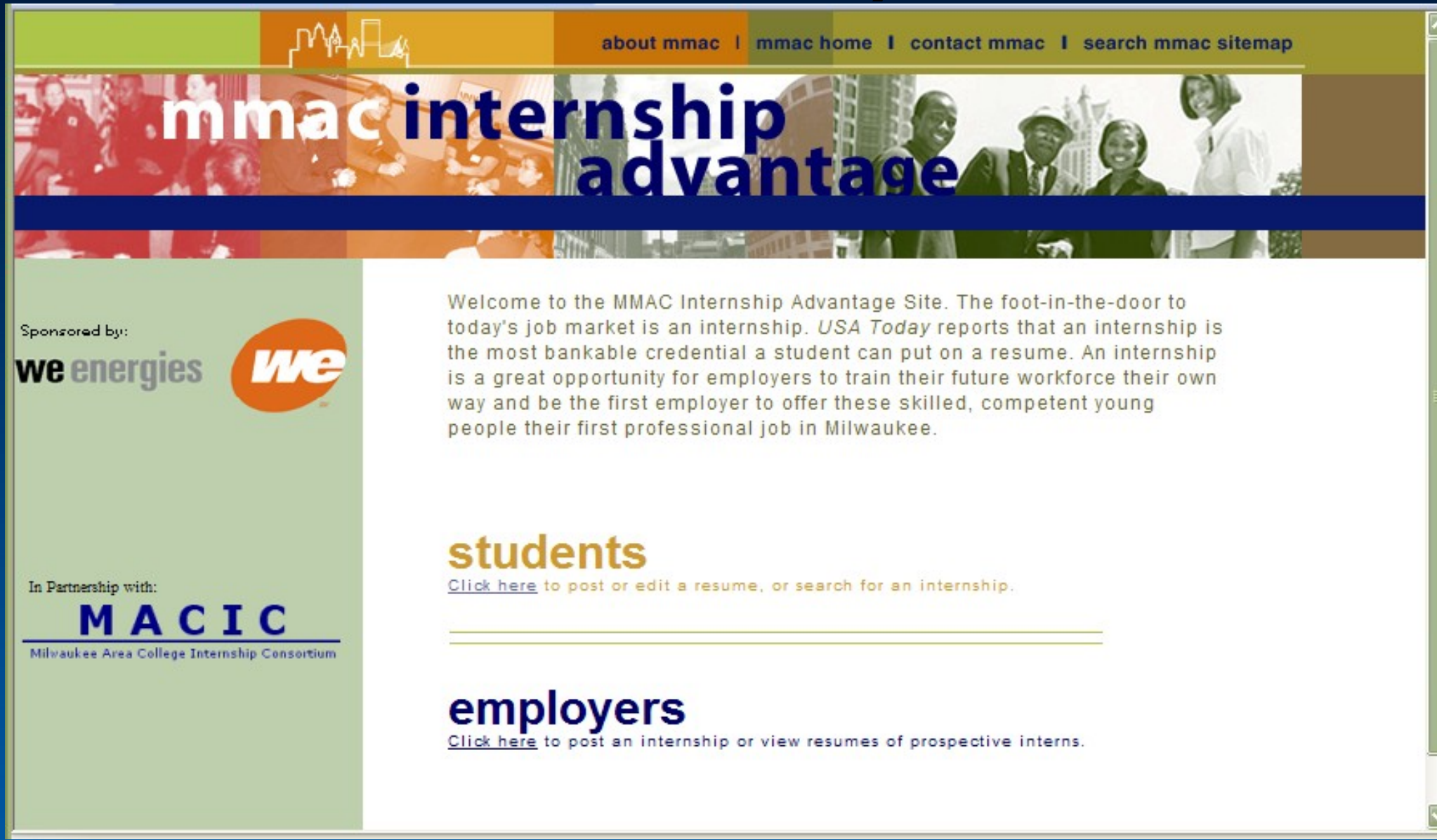
History

Hacked around those issues just enough, and then decided I would use Ruby and IOWA to deliver an application for a customer.

It was successful. The site ran for 2.25 years:

http://web.archive.org/web/20030312010411/www.internshipsmilwaukee.com/intern

History



swiftcore iowa

History

Took ownership of IOWA after successfully using it.

Development over the last almost-five-years has been driven by customer needs and user community requests.

~75 dynamic websites/applications running or in process, personally + numerous apps deployed by the small user community.

swiftcore iowa

History

There have been a handful of public releases, but no real focus on building a community, until now.

On the brink of the 1.0 release; focused on cleanup, examples, and documentation.

That's it for today's history lesson.

Performance

I have heard it said that if one cares about performance, one wouldn't be using Ruby to write web applications anyway; one would be using Python.

I don't have a thorough comparison of benchmarks, but...

Performance

Informal benchmarks show it to be faster than several Ruby frameworks (Rails, Nitro, Camping), and at least one Python framework (Nevow), with a fairly modest memory footprint, at least with the least common denominator benchmark, *Hello World*.

I commonly deploy low/moderate traffic sites on just a single process because it has more than enough capacity to handle high traffic bursts.

swiftcore iowa

Performance

I commonly deploy low/moderate traffic sites on just a single process because it has more than enough capacity to handle high traffic bursts.

I have production web sites with content and navigation being dynamically rendered from db content, running in excess of 1100 requests per second on a single process.

Enough background!
Let's get on to the code!

!

Code and Architecture

What is it? What makes it different from ___?

Component based.

Classes and objects represent pages and building blocks of pages.

It tries to treat web app development a lot like other app development.

I have focused a lot on maximizing performance.

Code and Architecture

Are there any things that it is particularly well suited for?

IOWA, much like Seaside now, was originally aimed squarely at applications that feature a clearly defined entry point.

I've expanded its capabilities so that it is a great tool for the common scenario, the dynamic site with application functions, because that's mostly what I write and mostly what other people write.

Code and Architecture

It is, however, still ideal for its original purpose of developing hardcore applications.

It would be a perfect choice for anything that can benefit from resource persistence over the life of a session. For example, a web based email client that maintains a persistent IMAP connection instead of connecting and disconnecting every time a user performs an action.

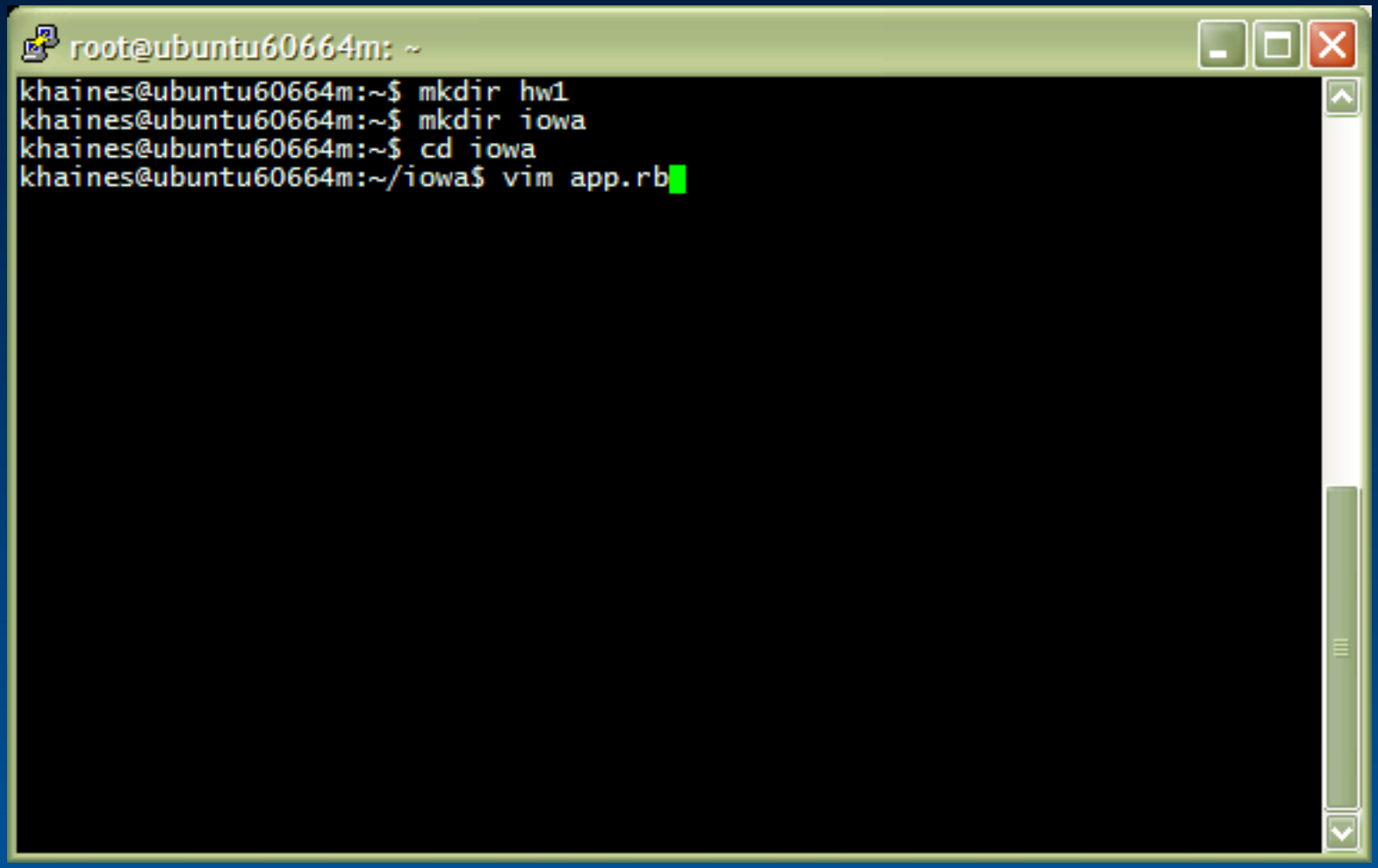
Code and Architecture

The simplest app – *Hello World!*

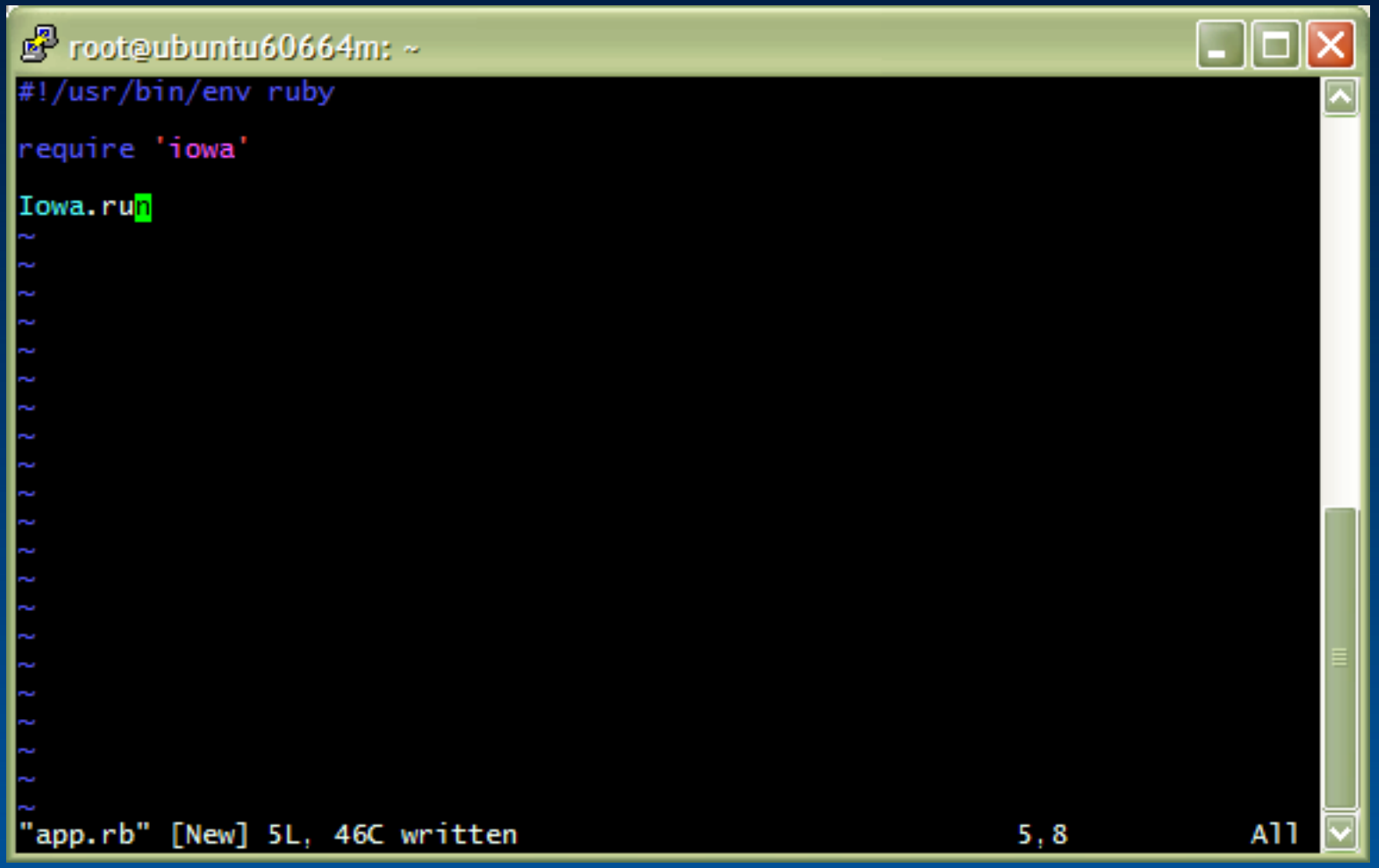
Nothing complicated. Just make a directory for it to live in.

Then, create the ruby script that will define and start the application.

Code and Architecture



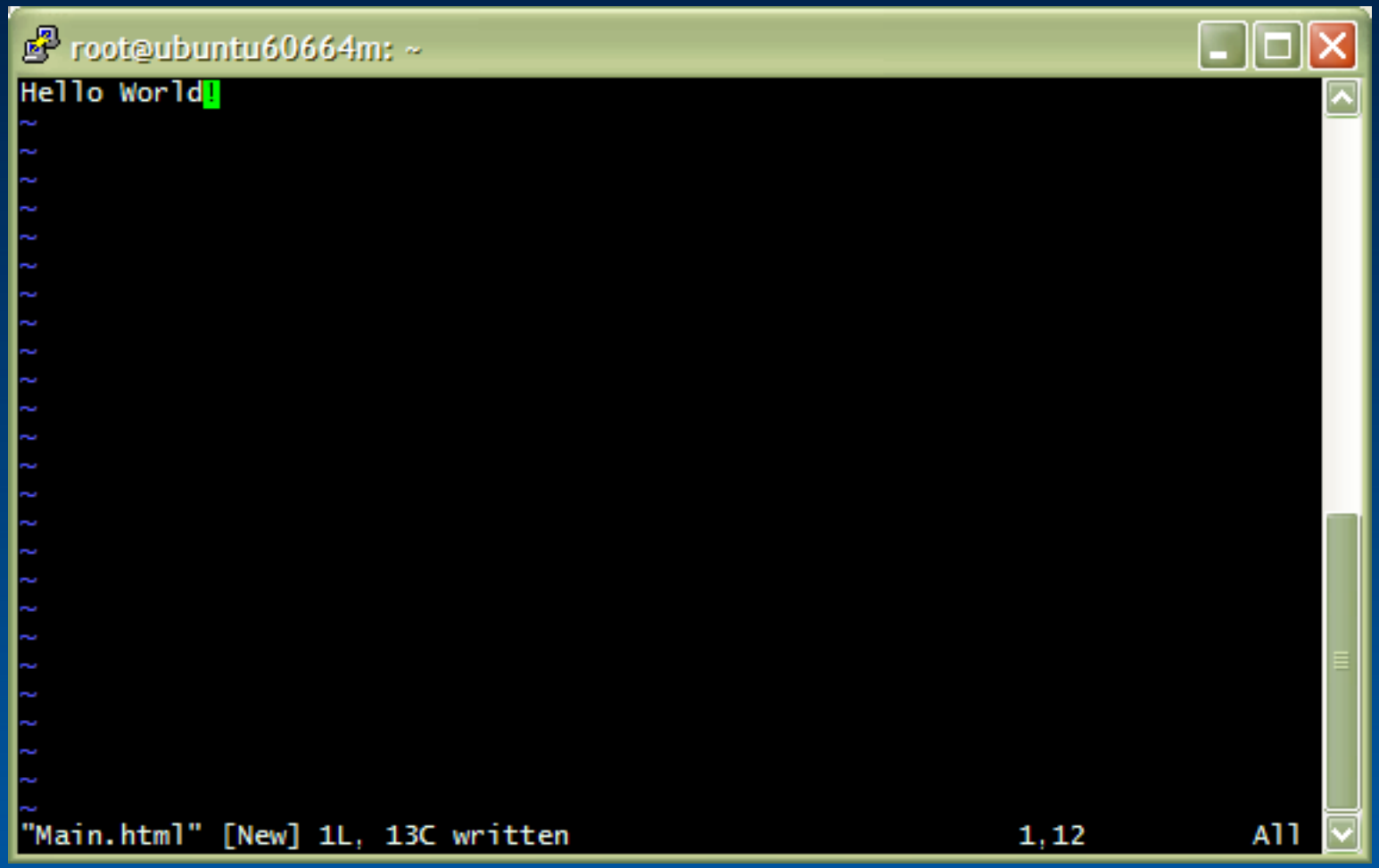
Code and Architecture



swiftcore iowa

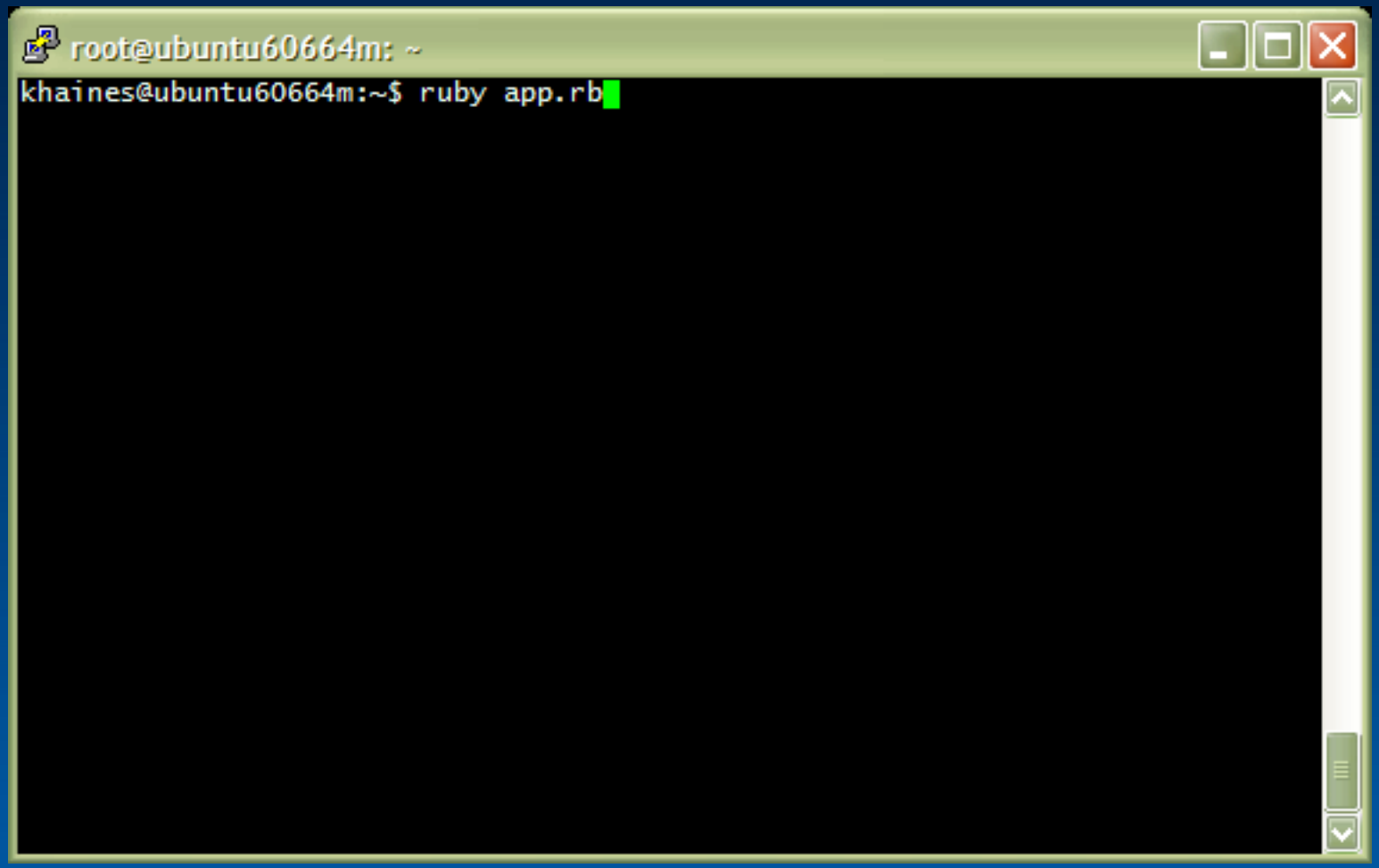
Code and Architecture

Then create a component.



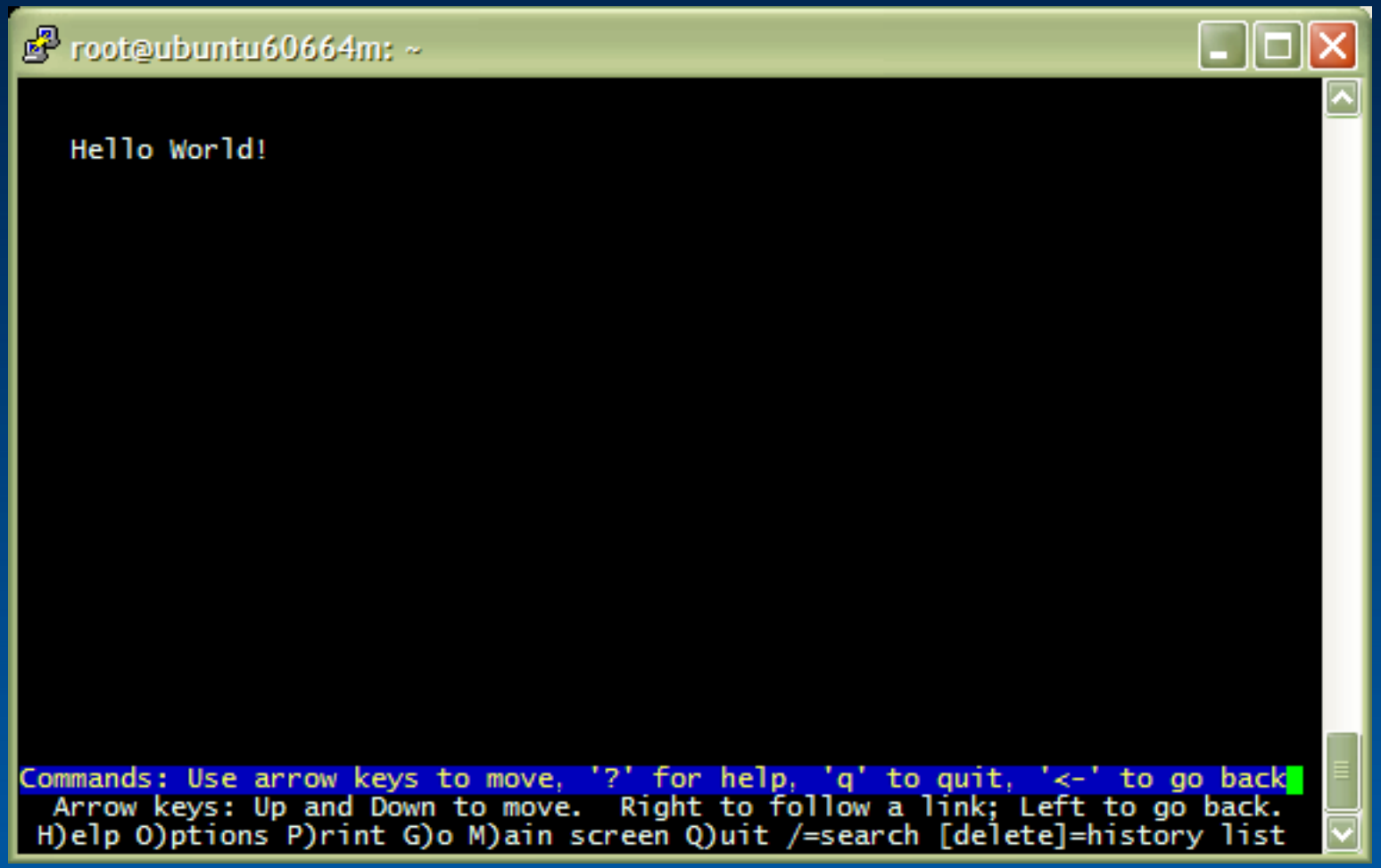
Code and Architecture

Now run it.



Code and Architecture

To see it yourself:
http://iowa.swiftcore.org/demos/hw1/



swiftcore iowa

Code and Architecture

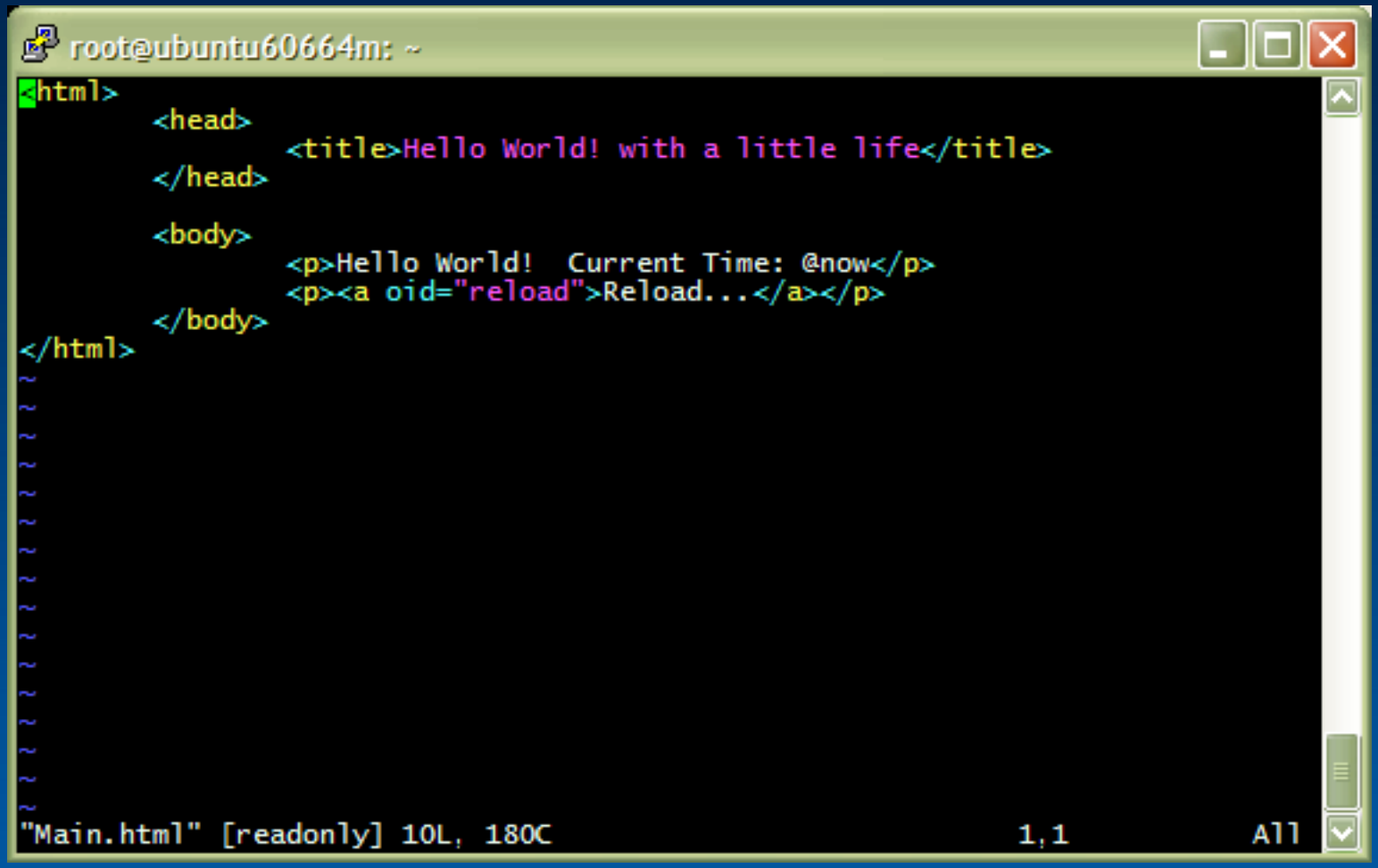
The application defaults to starting on 127.0.0.1 port 2001.

That's not so exciting, though. There's nothing dynamic happening with a static template.

So let's look at a new example that is a little bit more dynamic.

Code and Architecture

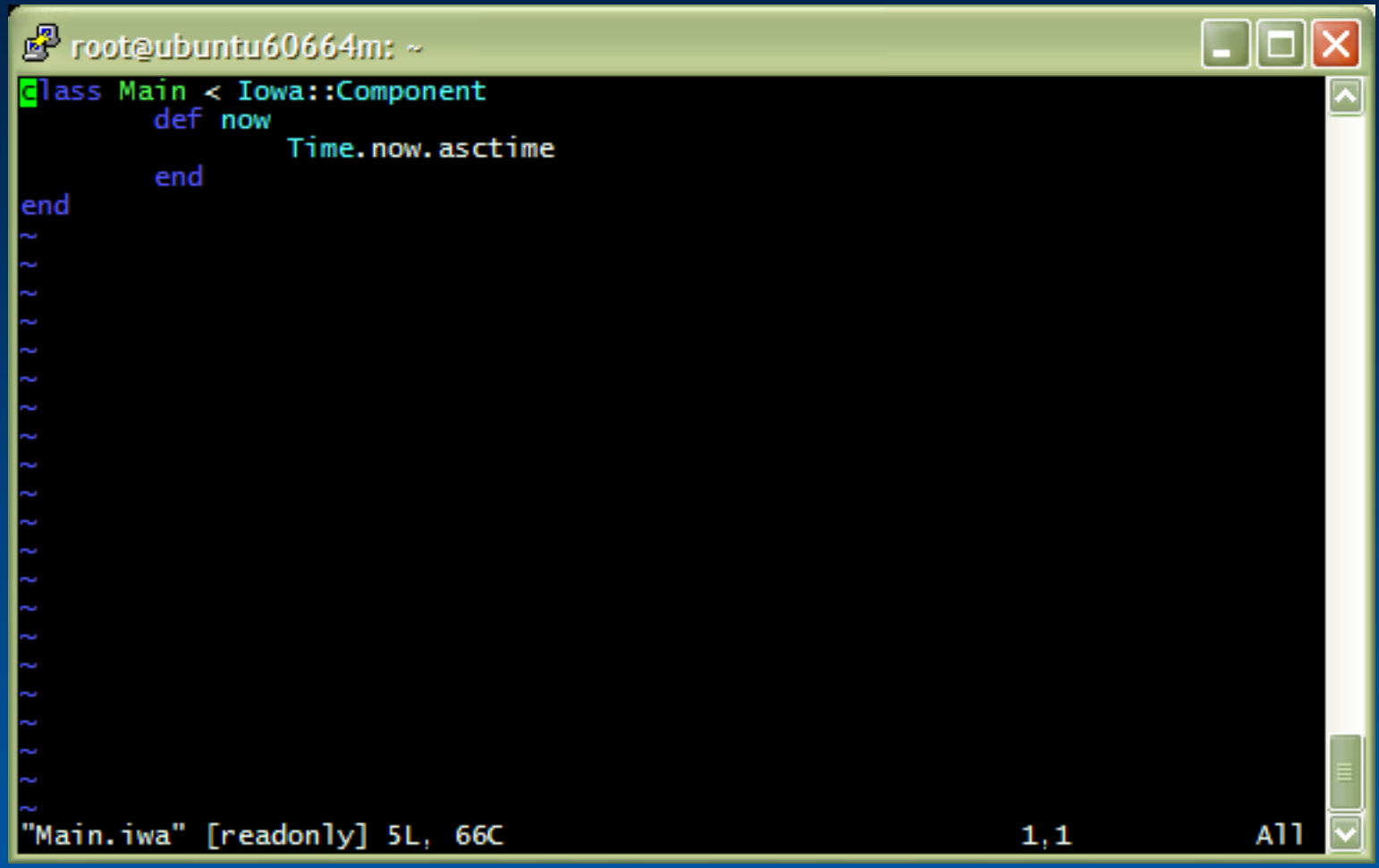
A new Main.html



swiftcore iowa

Code and Architecture

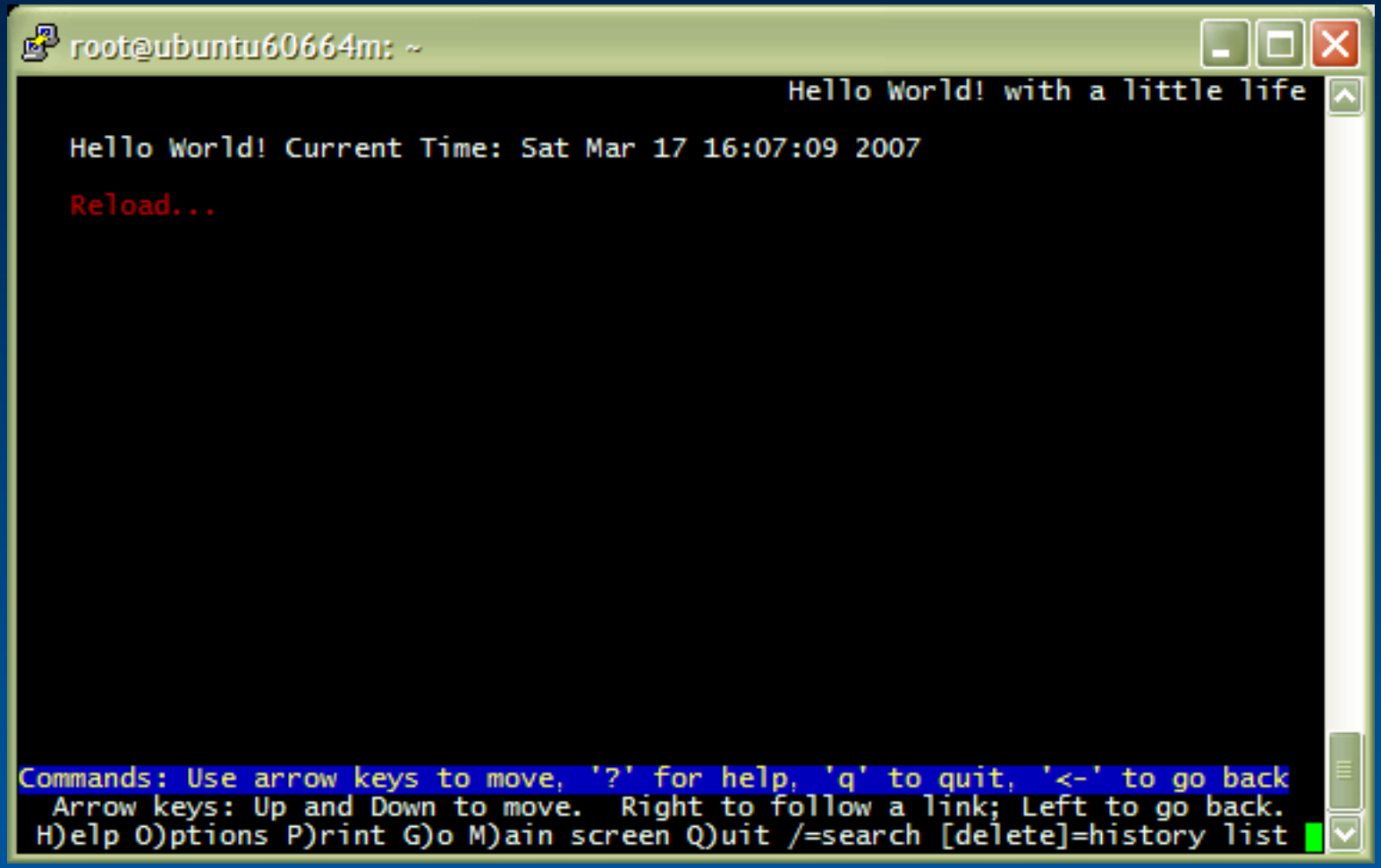
A new file, Main.iwa



swiftcore iowa

Code and Architecture

http://iowa.swiftcore.org/demos/hw2/



swiftcore iowa

Code and Architecture

Main.iwa looked like a regular Ruby class declaration.

That's because it is.

Components are Ruby classes that inherit from Iowa::Component, and the view is an associated template that is parsed into a tree of objects.

Code and Architecture

The default dispatch mechanism is very simple. If the request isn't associated with a specific session, dispatch it to Main.

Dispatching is pluggable, so one can choose a dispatcher that is best for the application. Two are provided, currently, and one may easily write a custom dispatcher.

Code and Architecture

<dynamicstring oid=”now” />

What's that oid attribute?

The oid tells the template parser that this tag is a dynamic tag.

The value of the attribute, in a dynamicstring, indicates the method to call. The method's return value is inserted into the template output.

Code and Architecture

Reload...

This is a dynamic link. The value of the oid in a link is the name of the method that will be invokes when this link it clicked on.

reload() is a method all components inherit from Iowa::Component.

Code and Architecture

You know, one does a lot of inserting of dynamic content into templates, and that <dynamicstring> thing is kind of ugly....

And what about form handling?

Code and Architecture

A new Main.html

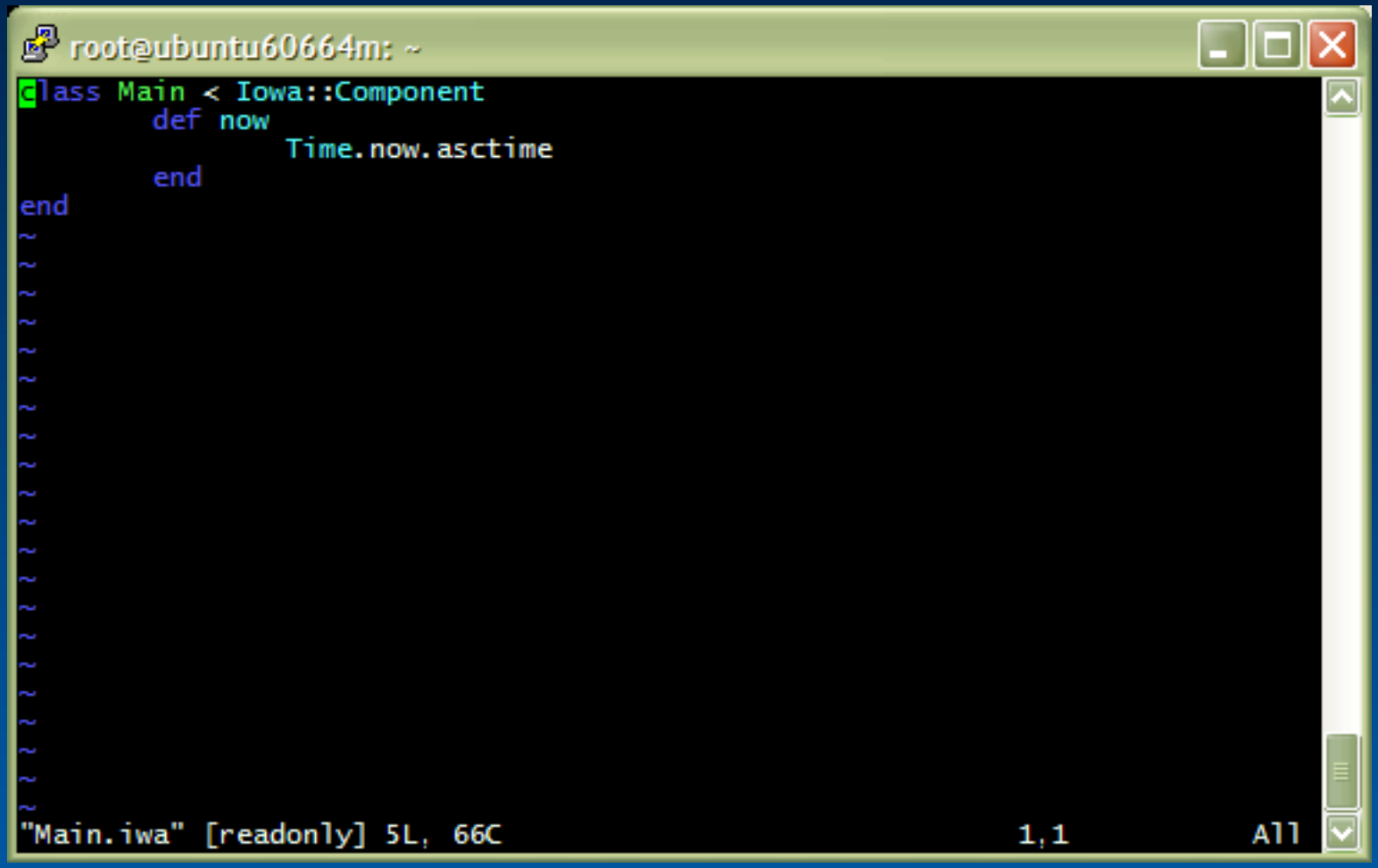
root@ubuntu60664m: ~

```
<html>
  <head>
    <title>Hello World! with a little life</title>
  </head>

  <body>
    <p>Hello representative of the World!
    The current time is: @now</p>
    <p>Tell me a little about yourself.</p>
    <p>
      <form oid="get_name">
        <input type="submit" oid="get_name" value="What is your name?">
        <input type="text" oid="your_name" size="20" maxlength="50"><br>
      </form>
    </p>
  </body>
</html>
~
~
~
~
~
~
~
~
~
"Main.html" 17L, 399C written                                1,1           All
```

swiftcore iowa

Code and Architecture

A new Main.iwa



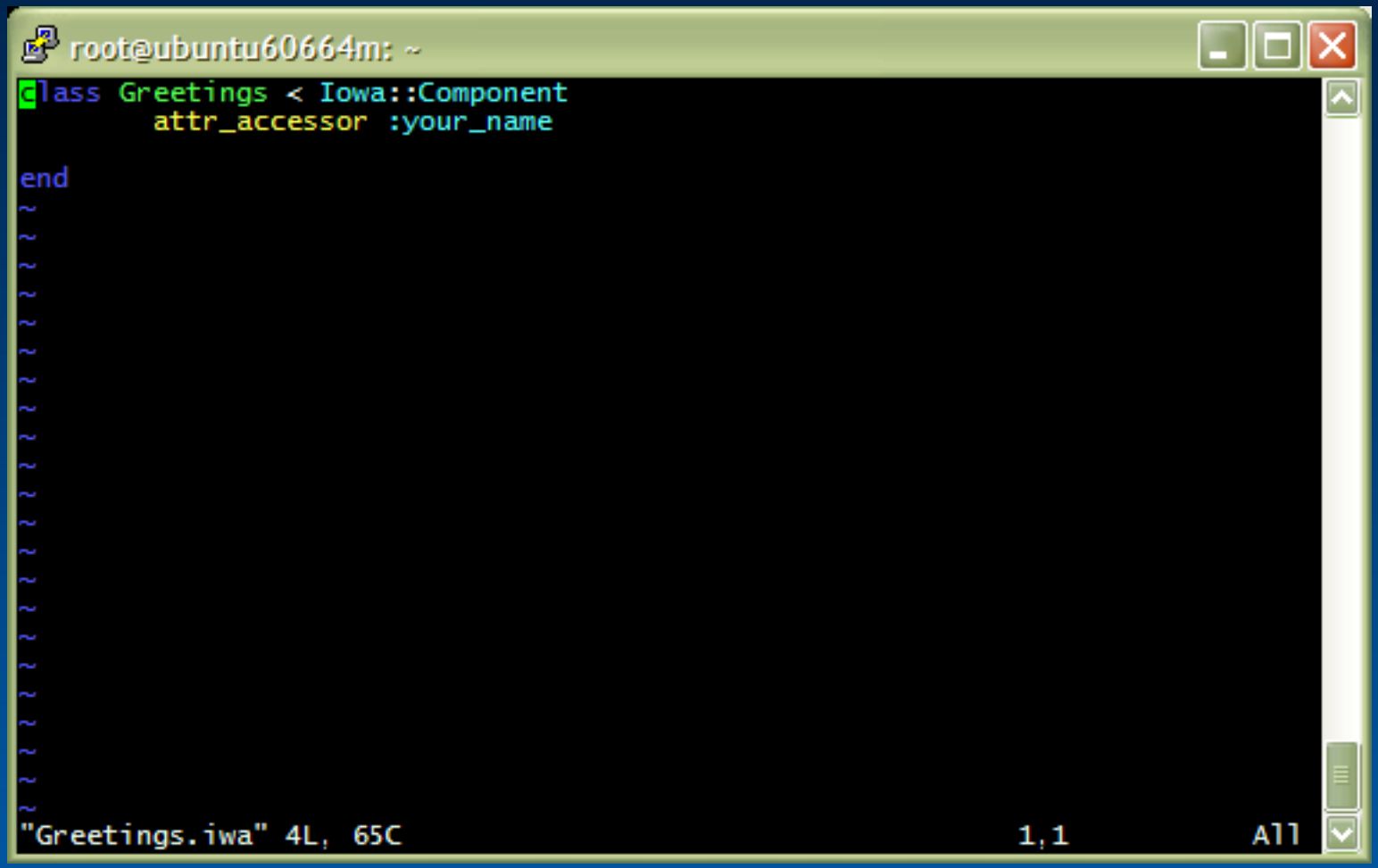
swiftcore iowa

Code and Architecture

Greetings.html for a new component

Code and Architecture

Greetings.iwa



swiftcore iowa

Code and Architecture

<form oid="get_name">

Indicates that the form is a dynamic, IOWA managed form.

get_name is the action; the method that will be invoked by default when the form is submitted.

Code and Architecture

```
<input type="text" oid="your_name" size="20" maxlength="50">
```

your_name is the accessor that will provide any default value for the field, and that will receive the input field's value.

Accessor here means attr_accessor style. i.e.

your_name() and your_name=(val)

Code and Architecture

```
<input type="submit" oid="get_name" value="What is your name?">
```

get_name as with the <form> tag, is the action. In the case of a submit tag, that method is called when the submit button is pressed. The method need not be the same as the <form> tag's method.

Code and Architecture

This is a difference from Rails and some other frameworks.

In Rails, multiple submit buttons on a form all trigger the same form action, so there must be a secondary dispatch done manually in that action if different buttons are to trigger different methods. IOWA permits multiple buttons that dispatch with no intermediate step to different methods.

Code and Architecture

```
new_page = page_named(:Greetings)
```

Since pages are components are objects, creating a new page means creating a new object.

Doing that manually is a hassle because components need a bunch of initialization data. So, page_named (or component_named) does that for you, returning a new component.

Code and Architecture

new_page.your_name = @your_name

Since components are regular Ruby objects, normal Ruby object things like accessors can be used to pass information from one to another.

IOWA also provides per_request and per_session scratch spaces:
session.context[:your_name] = @your_name
session[:your_name] = @your name

Code and Architecture

yield new_page

If an action method yields a component, that component will be called on to generate the response that goes back to the browser.

If an action method doesn't yield anything, the current component will be responsible for generating the response. This is how control is passed from one component to another.

Code and Architecture

Ok, what about the session handling? Just how are form field values getting into the Ruby code, and how is data from the Ruby code getting back out to the browsers without me having to worry about the mechanics of it?

As mentioned earlier, IOWA has implicit sessions.

IOWA constructs urls and form field names so that they encode their relevant context.

Code and Architecture

Look at the form action URL and field names.

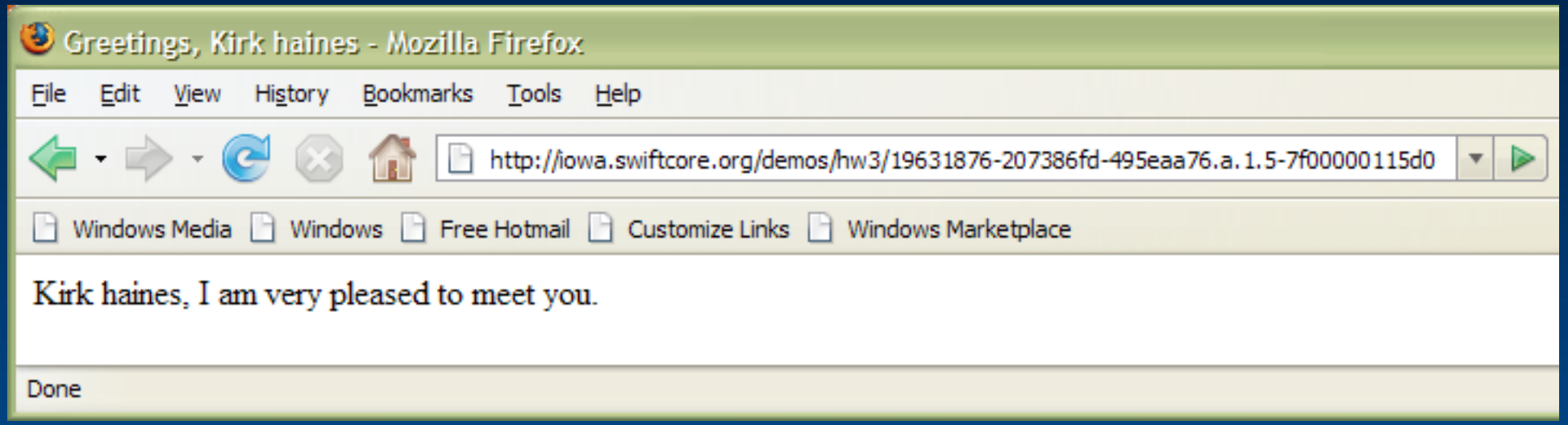
Source of: http://iowa.swiftcore.org/demos/hw3/ - Mozilla Firefox

File Edit View Help

```
<html>
        <head>
                <title>Hello World! with a little life</title>
        </head>

        <body>
                <p>Hello representative of the World!
                The current time is: Sat Mar 17 18:24:20 2007</p>
                <p>Tell me a little about yourself.</p>
                <p>
                        <form action="/demos/hw3/19631876-207386fd-495eaa76.a.1.5-7f00000115d0" method="post">
                                <input name="1.5.3" size="20" type="text" value="" maxlength="50" />
                                <input name="1.5.5" type="submit" value="What is your name?" />
                        </form>
                </p>
        </body>
</html>
```

Code and Architecture

And here's the URL of the response page.



The sessions and data are automatically tracked via data in the URL and form fields.

swiftcore iowa

Code and Architecture

The drawback to implicit sessions is that session requests all need to return to the same process. In a large scale deployment, this is a challenge, but one that can be handled (Swiftiply!).

The advantage to this, however, is that it eliminates a whole category of work for the developer. It makes writing applications easier because one really does not need to think about session/data transfer issues. It just works.

Code and Architecture

You aren't locked into implicit sessions, though. All of the ordinary sessioning techniques that are commonly used with other frameworks can also be used with IOWA. It's not exclusionary.

Code and Architecture

Since IOWA templates don't allow code to be embedded in them, how does one deal with things like conditionals and looping?

Code and Architecture

```
<if oid="show_list">
  <repeat oid="list" item="number">
    @number<br />
  </repeat>
</if>
```

Code and Architecture

IOWA provides a small set of special purpose tags which take care of all of the looping and conditional requirements within a template.

The set of these is small, so they are very easy to learn, but they provide all the capability needed to easy do anything one needs in a template, easily, without needing a special editor and without a lot of code clutter in the view.

Code and Architecture

Main.html

root@ubuntu60664m: ~

```
<html>
  <head>
    <title>
      Looping and Conditionals
    </title>
  </head>

  <body>
    <h1>Squares</h1>
    <table border="1">
      <repeat oid="tens" item="ten_count">
      <tr>
        <repeat oid="ones" item="one_count">
        <td><if oid="is_square?">*</if>@count</td>
        </repeat>
      </tr>
      </repeat>
    </table>
  </body>
</html>
~
~
                                                    1,1           All
```

Code and Architecture

Main.iwa

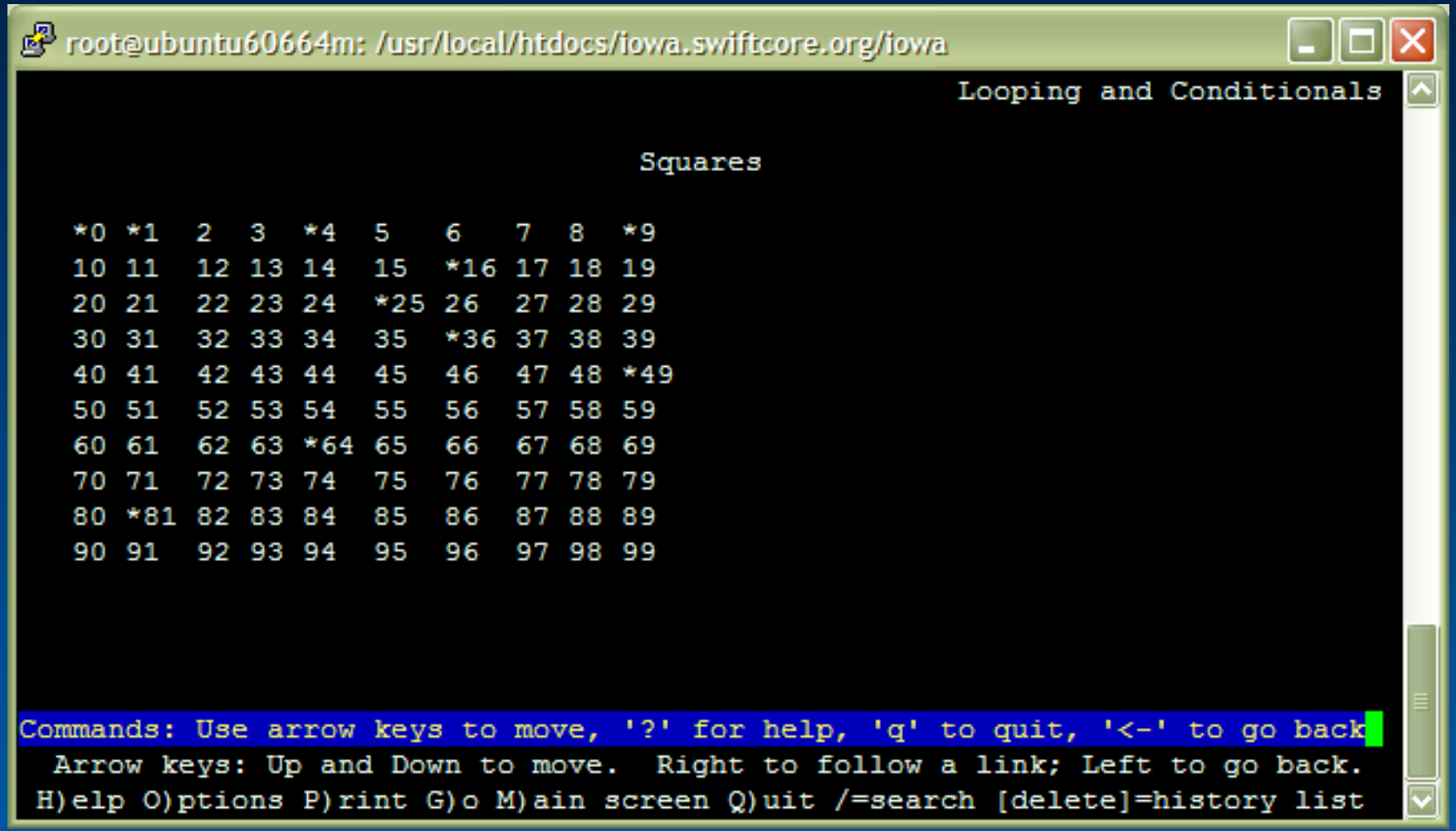
root@ubuntu60664m: /usr/local/htdocs/iowa.swiftcore.org/iowa/Demos/HW4

```
class Main < Iowa::Component
  attr_accessor :ten_count, :one_count

  def tens
    [0,10,20,30,40,50,60,70,80,90]
  end

  def ones
    (0..9).to_a
  end

  def is_square?
    ((count ** 0.5).to_i ** 2) == count
  end

  def count
    @ten_count + @one_count
  end
end
~
~
~
"Main.iwa" 19L, 253C                                  1,1           All
```

swiftcore iowa

Code and Architecture

To see it yourself:
http://iowa.swiftcore.org/demos/hw4/



swiftcore iowa

Code and Architecture

Classes and objects represent pages and ***building blocks of pages***.

IOWA is component based. This means that it is well suited to breaking a website layout or an application into smaller pieces, into reusable pieces that can be used separately or combined in different ways.

swiftcore iowa

Code and Architecture

Consider two component templates:

History.html

```
<header oid="_" />
<content oid="history" />
<footer oid="_" />
```

Performance.html

```
<header oid="_" />
<content oid="performance" />
<footer oid="_" />
```

swiftcore iowa

Code and Architecture

<header oid="_" />

If there is a Header component, this tag will insert it into this template. Because the oid, in this case, is being used only as a flag to the template parser, we don't really care what the value is. "_" just says that this is an anonymous object.

Code and Architecture

<content oid=”history” />

This inserts the Content component, and assigns an id of history to it. This component makes use of its oid to tell it what content it should contain.

Code and Architecture

Content is a real component that implements a generic content container. What makes it useful is that it can be setup to pull its content from any source, and that content is, in turn, passed through the Iowa::TemplateParser and transformed into a dynamically defined template. This means that it can, itself, embed other components into it, including other Content components.

Code and Architecture

```
<Content oid=”history_slide1” />
<Content oid=”history_slide2” />
<Content oid=”history_slide3” />
```

It is a very useful component that is included in the small but growing Swiftcore component library.

Code and Architecture

```
<header oid="_" />
<content oid="history" />
<footer oid="_" />
```

All together, those three tags in a template construct a unique page.

http://iowa.swiftcore.org/demos/hw5/history.html
http://iowa.swiftcore.org/demos/hw5/performance.html

swiftcore iowa

Code and Architecture

IOWA is a full featured web development framework that tries to keep your work simple, stay out of your way, and provide the flexibility to handle whatever you ask of it.

Give it a try!

http://iowa.swiftcore.org
wyhaines@gmail.com

swiftcore iowa