Contents in this Presentation

OUTLINE
Outline

• Ruby – What and Why...
• Rails – What, Why and Features...
• More on RoR – Active Records, Action Pack, CRUD, Migrations...
• Advantages
• Disadvantages
• Conclusion – Summary, References, Books and Tutorials...
What and why...

RUBY
What is RUBY?

• An Object oriented, Open source programming language
• Developed by Yukihiro Matsumoto in the 1990s.
• A blend of “Matz” favourite languages - Perl, Smalltalk, Eiffel, Ada, and Lisp
• Available for Windows, Mac OS, Unix/Linux, Java, .NET and Android.
RUBY – Why?

• According to ‘Matz’, Ruby is productive and fun because it was designed on following principles...
  
  – **Principle of Conciseness**
    “... short, concise code...”
  
  – **Principle of Consistency**
    “... a small set of rules covers the whole Ruby language...”
  
  – **Principle of Flexibility**
    “... should not restrict the flow of human thought ...”
RUBY – Syntax features...

• Whitespace is not significant (unlike Python)
• Statements separated by semicolons or carriage returns
• Statement can span a newline
• Parentheses can often be omitted
What, why and features...

RAILS
What is RAILS?

• An Open-source full stack web application framework for Ruby.

• **David Heinemeier Hansson** extracted Ruby on Rails from his work on Basecamp, a project management tool by 37signals.

• Open source and free. Growing community since 2004.
RAILS – Why?

• a lot less code
• a lot less configuration data
• bringing up basic functionality quickly
• building out new functionality incrementally
  integrated testing
RAILS – Features... (1)

• Can be used in multiple environments
  – Development, testing, production

• Rails embraces test-driven development
  – Unit testing, Functional testing, Integration testing

• Multiple database support
  – Oracle, DB2, SQL Server, MySQL, PostgreSQL, SQLite
RAILS – Features... (2)

• DRY principal
• Generate boilerplate code
• Full stack MVC Framework
  – The Framework provides all three MVC layers
• Convention over Configuration
  – No XML Configuration files
• Scaffolding
  – Automatically creates a full set of CRUD operations and views on any database table.
RAILS – MVC

1) Request

2) Forwards

3) Creates

4a) CRUDs

4b) Delivers

5a) Renders

5b) Redirects

6) Displays

1. Configure framework
2. Wrap CGI in request

 dispch.rb? controller=blog & action=display & id=5

/blog/display/5

Browser

Apache or WEBrick

Dispatcher

Action View

Action Controller

Active Record

Active Mailer
ActiveRecords, Action Pack, CRUD, Migrations...

MORE ON RoR...
Active Records... (1)

• “Database Wrapping” instead of “Database Mapping”
• Each active record object represents a row in a table
• Each record object has CRUD methods for database access
Active Records... (2)

• Adds attributes automatically, based on the columns in the database
• Adds relational management through a custom internal language
• Naming convention allow database to discover specific fields
• Schema migration “baked in” Rails
Action Pack... (1)

- Bundles both views and controllers
- The view and controller parts of MVC are pretty intimate
- The controller supplies data to the view
- The controller receives events from the pages generated by the views
- Rails provides a clear separation for control and presentation logic
Action Pack – Views...

• Creating either all or part of a page to be displayed in a browser

• Dynamic content is generated by templates
  – rhtml
    • Embeds snippets of Ruby code within the view’s HTML
  – rxml
    • Lets you construct XML documents using Ruby code
    • The structure of the generated XML will automatically follow that of the code
  – rjs
    • Allows you to create JavaScript fragments on the server which are to be executed on the browser
    • Great for creating dynamic Ajax interfaces
Action Pack – Controller...

• Coordinates the interaction between the user, the views, and the model
  – Rails handles most of this interaction behind the scenes
    • You only need to add the application-level functionality

• Other responsibilities
  – Routing external requests to internal actions
  – Managing caching
    • Give applications orders-of-magnitude performance boosts
  – Managing helper modules
    • Extend the capabilities of the view templates without bulking up their code
  – Managing sessions
    • Giving users the impression of ongoing interaction with the applications
CRUD...

Create  Read  Update  Delete
CRUD – Create... (1)

• Create row by creating object

```ruby
an_order = Order.new
an_order.name = “Dave Thomas”
an_order.address = “122 Main”
an_order.phone = 2125551212
an_order.save
```

```ruby
an_order = Order.new(
  :name => “Dave Thomas”,
  :address => “122 Main”,
  :phone => 2125551212 )
an_order.save
```

```ruby
Order.new do |o|
  o.name = “Dave Thomas”
  o.address = “122 Main”
  o.phone = 2125551212
  o.save
end
```

Note: We didn’t need to set a primary key. Rails assumes “id” is primary key and set autoincrement
CRUD – Create... (2)

- Can also use create method
- Creates a new object and saves it
- Takes a hash or an array of hashes

```ruby
an_order = Order.create(
  :name => "Dave Thomas",
  :address => "122 Main",
  :phone => 2125551212
)

an_order = Order.create(
  [ {  :name => "Dave Thomas",
       :address => "122 Main",
       :phone => 2125551212
    },
    {  :name => "Another Name",
       :address => "blah",
       :phone => 1234567890
    } ]
)
```
CRUD – Read... (1)

- We need to specify which rows we want
  - Rails will return objects containing the data from those rows in the database
- Use the find method with one or more primary keys
  - `an_order = Order.find(27)`
  - `product_list = Order.find(params["product_list"])`
- `find()` will throw a RecordNotFound exception if any of the requested primary keys cannot be found
CRUD – Read... (2)

- `find()` also has other options
  - can pass :all or :first along with other parameters
    - :conditions => “name = ‘Dave’”
      - corresponds to WHERE clause
    - :order => “name”
      - corresponds to ORDER BY clause
    - :limit => pagesize
      - corresponds to LIMIT
    - :offset => pagenum * pagesize
      - use in connection with :limit to step through query results

- `an_order = Order.find(:first, :conditions => "name = ‘Dave Thomas’")`
- `orders = Order.find(:all, :conditions => "name = ‘Dave’", :order => "pay_type, shipped_at DESC", :limit => 10)"
Allowing for externally generated parameters

- `pname = params[:name]`
  `orders = Order.find(:all, :conditions => ['name = ?', pname])`
- `orders = Order.find(:all, :conditions => ['name = :name', {:name => pname}])`

Can also write your own SQL

- `orders = Orders.find_by_sql('select * from orders')`
- single parameter - SQL string
- May also be an array where first element is SQL with place holders. The rest is a list of values or hash
- Nice for hard queries or performance
CRUD – Update... (1)

• Simple
  – find the row or rows using find
  – update necessary fields
  – save

• Also works with an array for multiple update
  – orders = Order.find(:all, :conditions => “name like ‘Dave%’”)”
  orders[0].name = “Fred”
  etc.

• May also use update() or update_all()
  – order = Order.update(123, :name => “F”, :address => “blah”)
    • finds, updates, saves, and returns object
  – result = Order.update_all(“set clause”, “where clause”)
    • returns number of rows updated

order = Order.find(123)
order.name = “Fred”
order.save
CRUD – Delete... (1)

- **delete & delete_all**
  - `Order.delete(123)`
  - `Order.delete([1,2,3,4])`
  - `Order.delete_all([“price > ?”, maxprice])`

- **destroy & destroy_all**
  - `order.find(123)`
  - `order.destroy`
  - `Order.destroy_all([“price > ?”, maxprice])`

- **destroy and destroy_all** ensure that ActiveRecord callback and validation functions are invoked
  - preferred methods
Migrations... (1)

- Rails is set up to encourage agile development
  - always making changes
  - even to the database
- To support this, Rails provides a mechanism to set up and modify the database
- Goal 1: Apply only those changes necessary to move a database from version x to version y
- Goal 2: Shield the developer from the specific implementation details of the underlying database
Migrations... (2)

• Migration skeleton files are created every time you generate a model
  – contained in db/migrate
• Run the migration using rake
  – rake db:migrate
• Migration files have a sequence number
  – acts as a version number
  – apply all migrations with sequence number greater than the database version
• Can pick a specific version
  – rake db:migrate VERSION=12
Migration Methods...

• **create_table**
  – accepts a table name and a ruby block

• **add_column and remove_column**
  – accepts table name and column name
  – and column type if adding a column

• **rename_column**
  – accepts table name, column name, new column name

• **change_column**
  – accepts table name, column name, new type

• **drop_table**
  – accepts table name
Pros and Cons...

ADVANTAGES AND DISADVANTAGES...
Disadvantages

• No big corporate backer
• Very few expert Ruby programmers, and universities and TAFEs have not picked it up
• Runs slowly (Java ~ 5 times faster but Ruby may be improved by new VM - YARV)
• Poor editor support and very slow debugger
• No clustering, failover
• No two-phase commit
• Does not support compound primary keys
• Internationalization support is weak
• No off-the-shelf reporting tool
Advantages

• Standard directory structure for source
• Can build prototype very quickly
• Can add to and change prototype easily
• Can generate scaffolding, if app is more complex, and build on this
• Very powerful, high-level commands
• Ruby has great short-hand code for common patterns, eg the Value Object
• Built in testing, migration, and some version control
• Does not constrain the programmer like other frameworks
CONCLUSION
Points to noted...

• Can only be used for web-based, specifically HTML-based, applications
• Designed for small to medium CRUD-based applications
• Cross-platform
• Large tools and software base
• Ruby and Rails are each very powerful in their own right.
References...

• The Ruby Programming Language - David Flanagan and Yukihiro Matsumoto
• Engineering Long Lasting Software - Armando Fox and David Patterson
• http://en.wikipedia.org/wiki/Ruby_on_Rails
• http://www.ruby-lang.org/en/
• http://rubyonrails.org/
Books...

- **The Ruby Programming Language** - David Flanagan and Yukihiro Matsumoto
- **Ruby on Rails Bible** - Timothy Fishe
- **Agile Web Development with Rails** - Sam Ruby, Dave Thomas and David Heinemeier Hansson
- **Ruby on Rails : Up and Running**

Online Resources...

- [http://ruby.railstutorial.org/](http://ruby.railstutorial.org/)
- Rails mailing list
David Heinemeier Hansson (Ruby on Rails creator) explained, "Once you've tried developing a substantial application in Java or PHP or C# or whatever," he says, "the difference in Rails will be readily apparent. You gotta feel the hurt before you can appreciate the cure."

FROM THE CREATOR...