



Current Topics

Version: 1.2

Release Date: 20/03/2008

Copyright © [Syger](#) 2007-2008. All rights reserved.

Table of Contents

Current Topics	4
Powerful Web Frameworks	4
High Octane Tools	4
REALbasic, the Alternative to Visual Basic 6?	4
Suitable Java Web Application Frameworks	5
Development Tools: Version Control and Unit Testing	5
Contacts	5
Ruby on Rails 2.0 – A First Look	6
Fast Prototyping	6
Documentation	6
Generators and Templates	7
Overall Impressions	8
Contacts	8
Grails – A First Look	9
Performance	9
The Wow! Factors	9
Fast Prototyping	10
Overall Impressions	11
Contacts	11
Groovy – a First Look	12
Documentation and Examples	12
The “Wow!” Factors	12
The Java to Groovy Transposition	12
Groovy Beans	13
Heredocs or Triple Quotes	15
Other Favourites	15
Contacts	16
Firebug and Aptana Studio	17
Firebug	17
Other Firefox Add-ons for Developers	18
Aptana Studio	19
Contacts	22
REALbasic, Heir to Visual Basic 6?	23
The Database Example	24
The Syger XML-RPC Example	26
Conversion Tools	29
Resources	30
Contacts	30
Java Web Application Frameworks	31
Past Practice	32
Current Practice	33
Probable Future Practice	33
Adobe Flex	33

The Google Web Toolkit and GWT-Ext	34
Grails and Groovy	35
Stripes	35
Wicket	36
Contacts	36
Version Control and Unit Testing	37
Version Control	37
Unit Testing	41
Contacts	42

© Syger 2007-2008. Content licensed according to the [Artistic License 2.0](#).

Current Topics

by [John Leach](#)

This article describes the current topics that Syger is actively investigating. They represent the type of consultancy and development work that Syger undertakes.

Powerful Web Frameworks

Thanks to the [programming course](#) I held at the beginning of 2008, I spent quite a bit of time with Ruby on Rails 2.0. Though the structure has changed somewhat, I really like the architecture. It is still one of the fastest development frameworks on the market. I've written a few notes on my impressions in [Ruby on Rails 2.0 – a First Look](#), on page 6.

The WebAlbum application covers quite a lot of the classic groundwork of a web application. I'd spent a few evenings reading the “[Definitive Guide to Grails](#)”, written by the Grails development lead, Graeme Rocher, so I decided to dedicate a couple or four days development work exploring this Groovy based framework. I've written up the notes of my experiences in [Grails – a First Look](#), on page 9.

High Octane Tools

I'm currently preparing two different studies. The first is a project using [Groovy](#), or “[Java for the 21st century](#)”, as it has been called, and the second is a [programming course](#) I'm holding. I've made some interesting discoveries, both [about the Groovy](#) language, and about [Firebug](#) and [Aptana Studio](#).

Groovy is an agile dynamic language for the Java platform, and I've written notes on my first impressions in [Groovy – a First Look](#), on page 12.

Firebug is an excellent (X)HTML / CSS / JavaScript debugging tool which runs inside the Firefox browser. Aptana Studio is an Eclipse based development tool for (X)HTML / CSS / JavaScript development, which seamlessly integrates with Firebug to produce a single development and debugging environment. Again, I've written notes on my first impressions in [Firebug and Aptana Studio](#), on page 17.

REALbasic, the Alternative to Visual Basic 6?

[REALbasic](#) is an object oriented version of the [BASIC](#) programming language. I first came across this language in 2005, when REALbasic was then at version 5.5. It has since changed its versioning policy to reflect the year of issue. Currently, the latest version of REALbasic is version 2007 release 5.

From a consultancy point of view, the most interesting question is if REALbasic is a valid alternative to [Visual Basic 6](#). Many companies have invested enormous amounts of effort and money in producing desktop applications with VB6, but Microsoft seems unmoveable in its policy of ending extended support for Visual Basic 6 in March, 2008. Mainstream support for Visual Basic 6 ended in March, 2005.

Syger discusses [switching to REALbasic](#), on page 23, as an alternative to using [Visual Basic.Net](#), or other Microsoft [.Net](#) technologies.

Suitable Java Web Application Frameworks

The genteel art of web application development in the Java language has taken great strides forward since the first implementations of [Java Servlets](#) and [JavaServer Pages](#). The open source community has embraced this development environment, producing almost countless frameworks, all with the promise of being “lightweight”, and “rapid”. Commercial ventures have also moved into the field, probably with less success in popularity than some of the open source offerings, though they have certainly been welcomed by large corporations.

So, in the age of Model, View, Controller ([MVC](#)), [AJAX](#) and Rich Internet Applications ([RIA](#)), and Ruby on Rails ([RoR](#)), who are the current and future winners in this endless battle? Syger discusses [this thorny topic](#), on page 31, looking at the problem from a low cost development budget point of view.

Development Tools: Version Control and Unit Testing

Modern professional application development requires much more than a text editor and compiler – unless you're a [Real Programmer](#), of course. Now that we've got highly sophisticated [Integrated Development Environments](#) (IDEs), the natural next steps are [version control systems](#) and [unit testing](#).

Syger has produced an [article](#), on page 37, discussing the relative merits of using a version control system, and the benefits of unit testing.

Contacts

Syger can be contacted for consultancy work on any of the topics mentioned in this article, by sending an email to info@syger.it.

Ruby on Rails 2.0 – A First Look

by [John Leach](#)

Ruby on Rails 2.0 is quite a change from the previous version 1.2. You may not like the “breaking” changes that were made, but at least it has the major advantage of not dragging all that backwards compatibility baggage with it.

This is a major version release. The developers seem to take notice of the user community, especially when it comes to dropping the less useful features. Just as components drifted away into oblivion, dynamic scaffolding has followed the same path. Scaffolding is still around, it's just that it now consists of generated code. Most of the changes are noted in “[Rails 2.0 It's Done!](#)”

Much of the fundamental framework still remains as it was, with both major and minor improvements in the robustness of the code. Ruby on Rails is now a very mature, solid framework, with a massive following and good support – especially from the excellently documented source code.

Fast Prototyping

More than anything else, Rails provides for very fast prototyping. It really can take just a few hours to get a basic application up and running. While developing the WebAlbum application for my [programming course](#), it took just a couple of hours to get the basic user / album / picture / image associations sorted out, including cascaded deletes, starting from a clean sheet of paper.

I actually spent three times more effort getting the passwords encrypted, and the ImageMagick library to produce the thumbnails, than I did preparing the data model.

Even if you can't, or won't, use Rails in your production environment, it's still worth investigating as a prototyping tool. Once the basic model has been ironed out, and all the difficult problems addressed, then you can transpose that model onto your preferred development environment, knowing that the model works.

Being able to convert ideas into working code in such a short time frame also allows you to be a little less cautious in the choice of a solution. If it takes two hours to build a “proof of concept” application, instead of two days, then you're more likely to try out different approaches, rather than sticking to some “traditional” solution.

Documentation

I spent much of my time with Dave Thomas' “[Agile Web Development with Rails](#)” open on my lap, but the second edition is now in need of an update, so I spent more time with the [API documentation](#), and the [Wiki](#). The [HowTos](#) Wiki page has some good tips, and the [code snippets](#) can be a source of good ideas too.

As I said before, scaffolding has changed, so the books and most of the tutorials are behind the times. The scaffolding scripts more or less explain themselves, just type

```
rails --help
```

before you generate your application (that's how I found out you can set the DBMS, and freeze Rails all in one step).

Once you've got your application, move to the application folder, then type

```
ruby script/generate --help
```

to see what generators are available, each of which supplies further information, by typing, say

```
ruby script/generate scaffolding --help
```

which gives information straight from the horse's mouth, as it were.

The API documentation is very complete (another sign of maturity for an open source project), but it's a shame there's no search facility. You can try guessing the function name – which worked for me about half the time – but some things just aren't that simple. There is a mirrored copy of the API documentation with search at <http://www.railsbrain.com>, otherwise it's back to Googling “ruby on rails api *whatever*”.

[Filtering](#) responses in the controller was one problem that took me a while to track down, because I was looking for `filter_something`, but the methods are actually `before_filter`, `after_filter`, or `around_filter`. Sigh.

Generators and Templates

The current state of the generators, or rather the code that is generated is pretty spartan, but it does the job - very much in style with Rails itself. I'm not totally convinced that this always helps, though. Although I like the very few lines of code, or HTML produced, there isn't always a hint as to what else you could add. On the one hand you have the `routes.rb` file, which has commented examples which you can change very easily:

```
ActionController::Routing::Routes.draw do |map|
  # The priority is based upon order of creation: first created -> highest priority.

  # Sample of regular route:
  # map.connect 'products/:id', :controller => 'catalog', :action => 'view'
  # Keep in mind you can assign values other than :controller and :action

  # Sample of named route:
  # map.purchase 'products/:id/purchase', :controller => 'catalog', :action =>
'purchase'
  # This route can be invoked with purchase_url(:id => product.id)

  # Sample resource route (maps HTTP verbs to controller actions automatically):
  # map.resources :products

  # Sample resource route with options:
  # map.resources :products, :member => { :short => :get, :toggle => :post },
:collection => { :sold => :get }

  # Sample resource route with sub-resources:
  # map.resources :products, :has_many => [ :comments, :sales ], :has_one => :seller

  # Sample resource route within a namespace:
  # map.namespace :admin do |admin|
  #   # Directs /admin/products/* to Admin::ProductsController
(app/controllers/admin/products_controller.rb)
  #   admin.resources :products
```

```
# end

# You can have the root of your site routed with map.root -- just remember to delete
public/index.html.
# map.root :controller => "welcome"

# See how all your routes lay out with "rake routes"

# Install the default routes as the lowest priority.
map.connect ':controller/:action/:id'
map.connect ':controller/:action/:id.:format'
end
```

With that amount of commenting, you'll rarely need to look up the documentation. At the other extreme you get the basic model class:

```
class User < ActiveRecord::Base
end
```

Hardly giving away trade secrets there. Yet we'll almost certainly need to add [association](#) and [validation](#) code.

Fortunately, the generator templates are on hand, and it's not such a daunting task to modify them for yourself. I've written a [companion tutorial](#) discussing this technique.

Overall Impressions

What is there not to like? Ruby is a clean and powerful language. Rails is a clean implementation of a complex problem. There are concerns raised about performance and scalability, but frankly, if and when such problems arise, there are solutions available. Hardware is, and even with Rails helping you, will always remain cheaper than software development.

Even though the Rails philosophy is convention over configuration, it is also a very configurable framework. The `WebAlbum` exercise required just two lines of configuration code, and not even one line of SQL. Admittedly it is a simple application, but Rails will let you specify almost any URL you like, and let you make all the SQL queries you want, if that's what you need to do.

If you're new to both Ruby and Rails the learning curve is steep. Start with Ruby, use it to create build scripts, or write a command line program. I used the [colophon](#) I use to build my sites as an exercise.

Once you've got Ruby under your belt, Rails won't be such a surprise. I have found that the learning curve is gradual. The most important thing to keep in mind is that "it has probably already been done". Take time to check out the documentation – there's probably already a method that does just what you're looking for. Better to spend an hour researching, and ten minutes implementing, than the other way round.

Contacts

Syger can be contacted for consultancy work on any of the topics mentioned in this article, by sending an email to info@syger.it.

Grails – A First Look

by [John Leach](#)

Grails is a very interesting modern framework for the Java platform. As its name implies, it has built on the Ruby on Rails philosophy, but with an interesting twist. It is based on a very stable and popular stack; the [Spring framework](#), [Hibernate](#), [SiteMesh](#), and [Quartz](#). On top of this, it offers [Groovy](#) to produce a clean and simple interface to the stack, whilst remaining completely compatible with the JEE environment – the entire web application can be packaged as a [war](#) file.



Performance

If you use a web application framework that makes heavy use of a dynamic language such as Groovy, you're going to pay the price in performance, right? Well, yes, Grails will be slower than a 'pure' Java / Servlet / JSP solution. In fact a couple of Sun engineers have produced some [benchmarks](#) comparing Ruby on Rails, Grails and JEE.

Although a little old now, the results seem to indicate that Grails is between four and two times slower than JEE with between 100 and 500 concurrent users (page 33 of the PDF). Obviously, this may be worrying if you expect such high application usage, but if your application works in the 'long tail' region, say 10 to 100 concurrent users, then the figures seem acceptable – considering the performance increase in development efficiency.

Personally, I would have guessed performance ten times slower than JEE, so I'm actually quite happy. Grails is still at version 1.0, and there probably is some room for improvement in terms of efficiency.

Grails itself is about 70% Java code, and 30% Groovy. Although you can use Java in your own code, most of it will be in Groovy, due to the Groovy based interfaces in the model, view and controllers.

The Wow! Factors

Grails works very hard to produce a similar development environment (and development experience) as Ruby on Rails. It also makes clever use of Groovy to produce a very small footprint in terms of hand written code. The most positive wow! factors in my experience writing the [WebAlbum](#) application were in developing the tag library, and the security filters.

I have written a rather long article describing this development work in the [Grails WebAlbum](#) tutorial. Just to give you a small example of the tag library, this code extract from `grails-app/taglib/WebAlbumTagLib.groovy`:

```
class WebAlbumTagLib {
```

```

static namespace = "wa"
...
def pictureAnchor = { attrs, body ->
  def picture = attrs.remove('picture')
  def size = attrs.remove('size')
  out << "<a href=\"#{createPictureLink(picture.id, size).encodeAsHTML()}\">
  attrs.each { key, value ->
    out << " $key=\"#{value}\""
  }
  out << '>'
  out << body()
  out << '</a>'
}
...

```

together with this markup in `grails-app/views/picture/show.gsp`:

```

...
<tr class="prop">
  <td valign="top" class="name">Caption:</td>
  <td valign="top" class="value">
    <wa:pictureAnchor picture="{picture}" size="{Image.Original}" title="Show
Original" target="_blank">${picture.caption ? : '...'}</wa:pictureAnchor>
  </td>
</tr>
...

```

produces the following XHTML snippet:

```

<tr class="prop">
  <td valign="top" class="name">Caption:</td>
  <td valign="top" class="value">
    <a href="/image/7/1/7-Original.jpg" target="_blank" title="Show Original">One
Man, but Three Hands!</a>
  </td>
</tr>

```

It's interesting to note that I didn't have to write a Tag Library Descriptor file, and that the `picture` object in the GSP is 'passed through' to the `pictureAnchor` closure as is, so it can reference the properties (`picture.id`) immediately.

I didn't even have to stop the server to add `pictureAnchor` to the tag library, or to see the tag being used in the GSP.

Fast Prototyping

Similar to Ruby on Rails, Grails provides for very fast prototyping. It really can take just a few hours to get a basic application up and running. The major difference is that Grails, in the same way as Groovy, allows you to reuse all that hard earned JEE / Spring / Hibernate knowledge. For experienced Java programmers, Groovy and Grails is a far less traumatic transition than learning Ruby and Ruby on Rails.

Even if you can't, or won't, use Grails in your production environment, it's still worth investigating as a prototyping tool. Once the basic model has been ironed out, and all the difficult problems addressed, then you can transpose that model onto your preferred development environment, knowing that the model works. In addition, transposing a Groovy bean to a Java bean is relatively straightforward. Plus you've already got the Hibernate annotations worked out beforehand.

Overall Impressions

Groovy is a clean and powerful language, which provides a smooth and shallow learning curve. Grails is a clean implementation of a complex problem, which uses tried and trusted technologies. There are concerns raised about performance, but frankly, I'm prepared to pay that price given the dramatic development efficiency increase. Hardware will always remain cheaper than software development.

If you're new to both Groovy and Grails, you'll need a couple of weeks to get familiar with them. This is, however, a much shorter path than learning Ruby, and then Ruby on Rails, in my experience.

As with Ruby on Rails, I have found that the Grails learning curve is gradual. The most important thing to keep in mind is that “it has probably already been done”. Take time to check out the [reference documentation](#) and [user guide](#) – both of which are excellent.

Contacts

Syger can be contacted for consultancy work on any of the topics mentioned in this article, by sending an email to info@syger.it.

Groovy – a First Look

by [John Leach](#)

[Groovy](#) is an agile dynamic language for the Java platform. I've been using it for a week or so now on a project. So far I'm pleasantly impressed with the results. In fact, I'm sufficiently impressed that I'm taking a little time to outline my findings so far. This article will be expanded as the project progresses.



Documentation and Examples

The Groovy web site provides wiki style documentation, which can also be [downloaded](#) and is available as a PDF. Most of the information you will need is there, but it's not particularly well organised. Some of it refers to earlier versions (and syntax) of Groovy, which can be problematic. The “Getting Started Guide” section will do just that, it looks at installing Groovy and running the infamous 'Hello World' script. The various IDE plugins are covered in “IDE support”, I installed the Groovy plugin for Eclipse. The “User Guide” section runs through the most important aspects of the language, and the “Cookbook Examples” section provides both examples and a first look at some serious Groovy code.

After a couple of days with the wiki PDF, I needed something a little more organised. Fortunately, the [Pragmatic Programmers](#) have just issued a couple of Groovy books, though they're still in beta and currently only available in PDF, and will be until March 2008. The first is “[Programming Groovy](#)”, by Venkat Subramaniam. Unfortunately, as of writing (January 2008) the book is only about half finished. The second book is “[Groovy Recipes](#)”, by Scott Davis. Scott's book seems pretty complete, so that's what I've been using for the time being.

The “Wow!” Factors

If you're a long time Java programmer like myself, and have strayed off the “one true path”, by using dynamic languages such as [Ruby](#), then you'll understand the “Wow!” factors. Ruby has so many elegant features that I've found going back to Java a real struggle. But it's also a huge problem trying to sell Ruby (and [JRuby](#)) in the JavaWorld. The Groovy team seem to have understood that problem right from the start. What they've produced is something which can start out very close to pure Java syntax, and then move on towards idiomatic Groovy. That way, the learning curve is not so steep.

So, from a hectic few days of Groovy programming, here is a short list of “Wow!” factors that I really like.

The Java to Groovy Transposition

Let's see just how easy it is to move to idiomatic Groovy from idiomatic Java. Here's

a simple 'Hello World' application which simply lists its arguments, this is HelloWorld.groovy:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
        for(String arg : args) {
            System.out.println(arg);
        }
    }
}
```

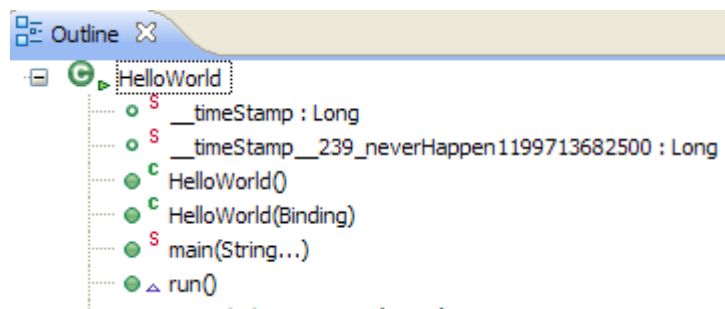
Looks pretty much like Java, doesn't it? But Groovy can take us a little further, firstly because it adds a static println() method to Object, and secondly because semicolons are optional, and thirdly because the compiler knows that the class name matches the filename:

```
public static void main(String[] args) {
    println("Hello World!")
    for(String arg : args) {
        println(arg)
    }
}
```

That's a little less typing. Now Groovy also knows that any code outside of a method needs to be wrapped in a static main method. Groovy won't be too troubled by missing parentheses in the method calls either. It's also a dynamic language, so we don't really need to tell it that arg is a String. Oh, and it also provides an each() method for collections, so:

```
println "Hello World!"
args.each { arg ->
    println arg
}
```

That's about the least amount of typing you'll need. But what about the compiled class file? What does that look like to Java? Well, it looks like this (when decompiling the bytecode):



Which is pretty close to what you'd have got writing a pure Java class.

Groovy Beans

We can continue that type of compactness when writing plain old Groovy objects (POGOs), which are very much like POJOs, but without the boilerplate baggage:

```
package net.sf.bifocal.table

class Cell {

    Table parent = null
}
```

```

Row parentRow = null

String styleName = null

int span = 1

String valueType = "string"

def value = null

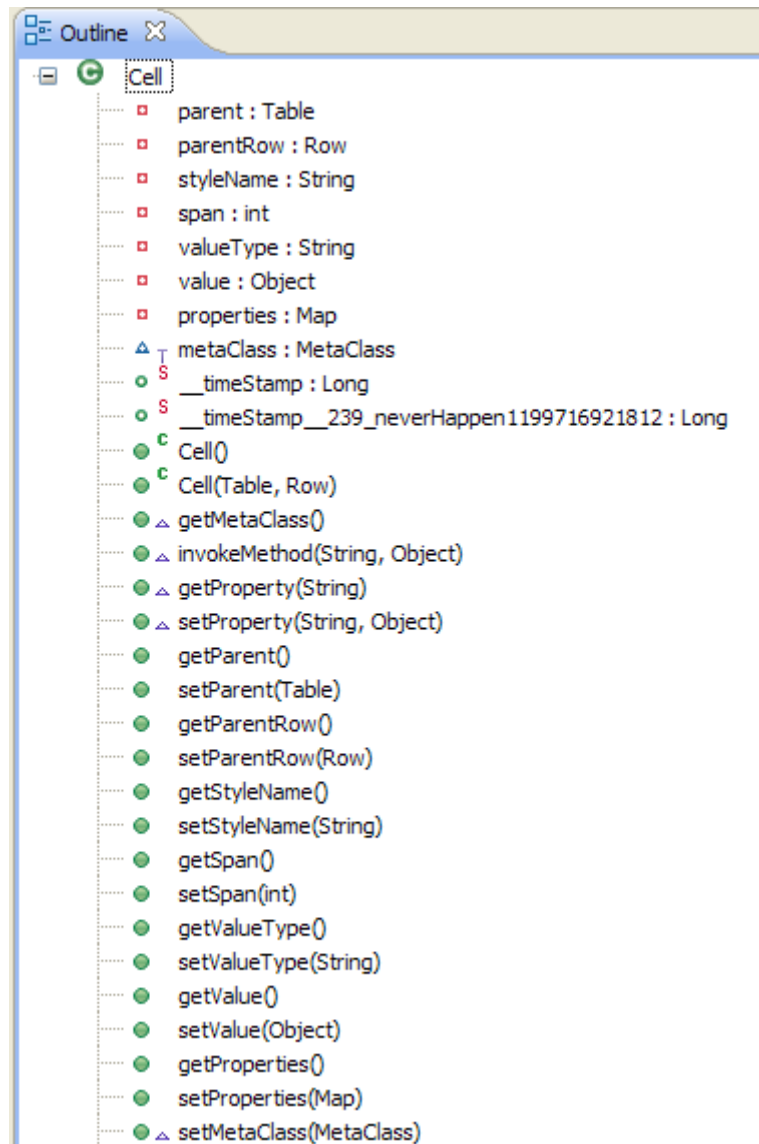
Map properties = [:]

Cell() {
}

Cell(Table table, Row row) {
    parent = table
    parentRow = row
}
}

```

Well, we're used to the missing semicolons by now. The `def` keyword defines a property or method – in this case we'll have an `Object` type for `value`. Note also how easy it is to create an empty `Map` object – just `[:]`. But is this `GroovyBean` really a `JavaBean`? Again let's take a look at the bytecode:



Looks like one to me. Wow! Look at all those getters and setters. That's a very low signal to noise ratio.

Here-docs or Triple Quotes

What if you needed a string like this:

```
<office:automatic-styles>
  <style:page-layout style:name="pml">
    <style:page-layout-properties fo:page-width='21.001cm' fo:page-height='29.7cm'
      fo:margin-bottom='1.499cm' fo:margin-left='2.499cm' fo:margin-right="2.499cm">
      <style:background-image/>
      <style:columns fo:column-count="1" fo:column-gap="0cm"/>
    </style:page-layout-properties>
    <style:header-style/>
  </style:page-layout>
</office:automatic-styles>
```

Care to write that in Java? Try this in Groovy:

```
String openDocContent = '''
<office:automatic-styles>
  <style:page-layout style:name="pml">
    <style:page-layout-properties fo:page-width='21.001cm' fo:page-height='29.7cm'
      fo:margin-bottom='1.499cm' fo:margin-left='2.499cm' fo:margin-right="2.499cm">
      <style:background-image/>
      <style:columns fo:column-count="1" fo:column-gap="0cm"/>
    </style:page-layout-properties>
    <style:header-style/>
  </style:page-layout>
</office:automatic-styles>'''
```

Wasn't too difficult was it? This makes unit testing XML so much easier.

Other Favourites

Although you've got property getters and setters, Groovy lets you use the (private) property names instead, so:

```
value = myTable.getRow().getCell().getValue();
```

becomes:

```
value = myTable.row.cell.value
```

And what if the table, row, cell, or value are null? This:

```
if (myTable != null) {
  myRow = myTable.getRow();
  if (myRow != null) {
    myCell = myRow.getCell();
    if (myCell != null) {
      myValue = myCell.getValue();
    }
  }
}
```

becomes:

```
myValue = myTable?.row?.cell?.value
```

That's Groovy's safe dereferencing.

Finally, even the ternary expression gets a reworking:

```
myValue = myValue != null ? myValue : myDefaultValue;
```

becomes:

```
myValue = myValue ?: myDefaultValue
```

Here, by the way, `myValue` evaluates as `true` when it's not `null` as far as Groovy is concerned.

Contacts

Syger can be contacted for consultancy work on any of the topics mentioned in this article, by sending an email to info@syger.it.

Firebug and Aptana Studio

by [John Leach](#)

The first developer's rule should be “Never stop looking for better tools”. I've just finished preparing some material for a [programming course](#) I'll be holding, and I needed to define the development environment for the familiar [\(X\)HTML](#) / [CSS](#) / [JavaScript](#) trio. I had originally planned on using a plain text editor, together with Firefox, and my previous favourite development add-ons; Chris Pederick's [Web Developer](#), and [Venkman](#).

Fortunately, I decided to spend a little time checking out if anything new had appeared recently on the horizon. Well, it has, in the form of Joe Hewitt's [Firebug](#). I'll probably still keep Web Developer, but Firebug has plenty to offer, over and above the rest.

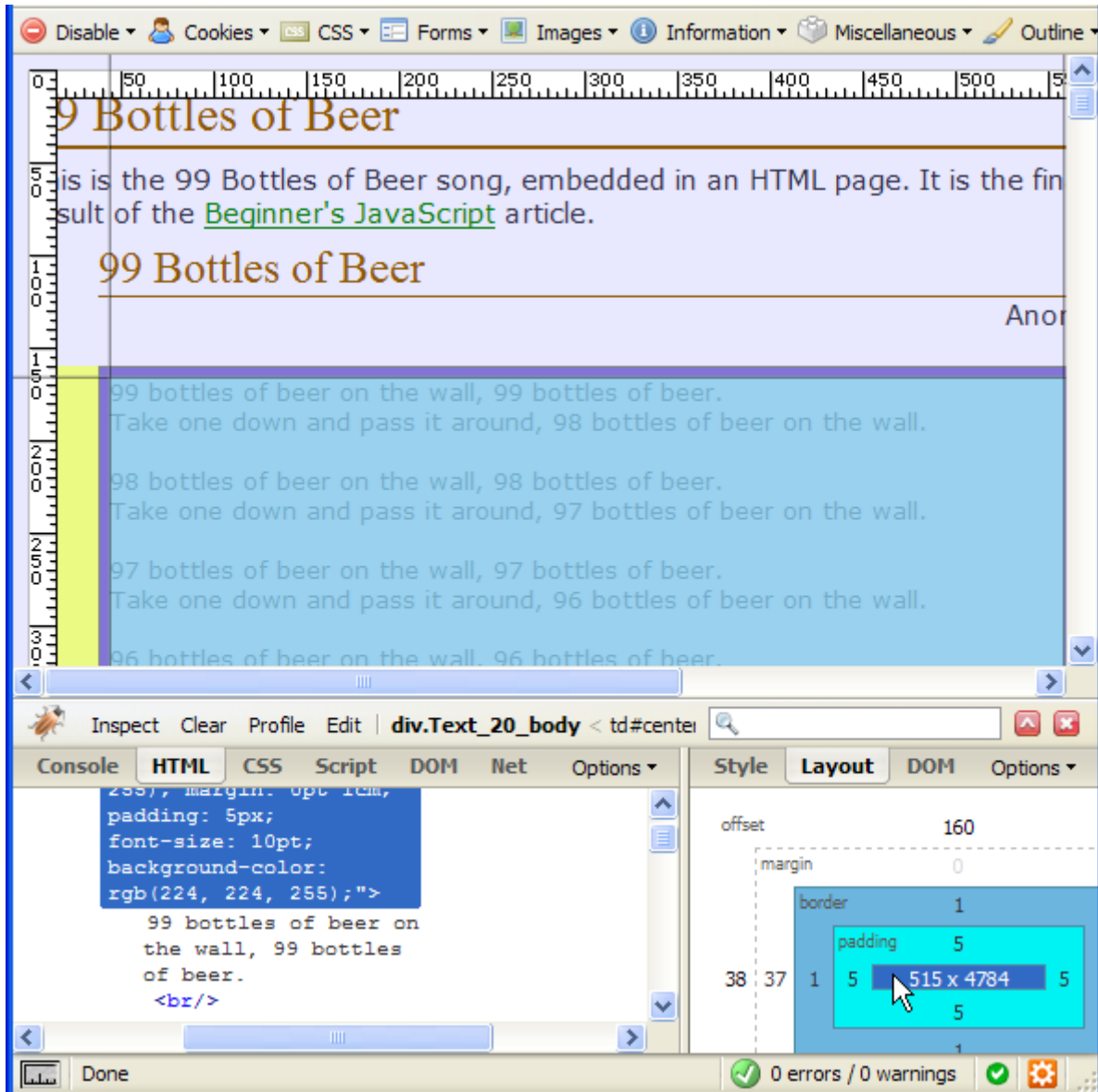
Firebug



Firebug gives you XHTML, CSS, and JavaScript editing and debugging all within one add-on. Jesse Newland has made a nice [screencast](#) of Firebug, showing the most salient points.

Joe's “[get Firebug](#)” web site also shows screenshots and features of this excellent add-on.

I've used a [simple JavaScript tutorial](#) I wrote for my screenshot example:



Beneath the location toolbar is the Web Developer toolbar. At the bottom of the screen is Firebug, displaying the song lyrics (which were generated by JavaScript, so Firebug is showing the in-memory DOM), and the right hand pane is showing the div element layout. The border, background-color, and padding styles were added by hand using Firebug's CSS editor.

Note that while it is possible to edit the HTML, CSS, and JavaScript, these modifications are not permanent, and will be cleared if the page is refreshed.

The most important key strokes are F12 to open Firebug in the browser pane, or Ctrl F12 to open Firebug in it's own window.

Other Firefox Add-ons for Developers

I have found the [UrlParams](#) add-on invaluable on occasions when debugging server side code. It can also be used to experiment with "code breaking" parameter combinations.

The [HTML Validator](#) makes sure that I'm not making life any harder for myself by

creating invalid HTML. Based on Dave Raggett's [HTML Tidy](#).

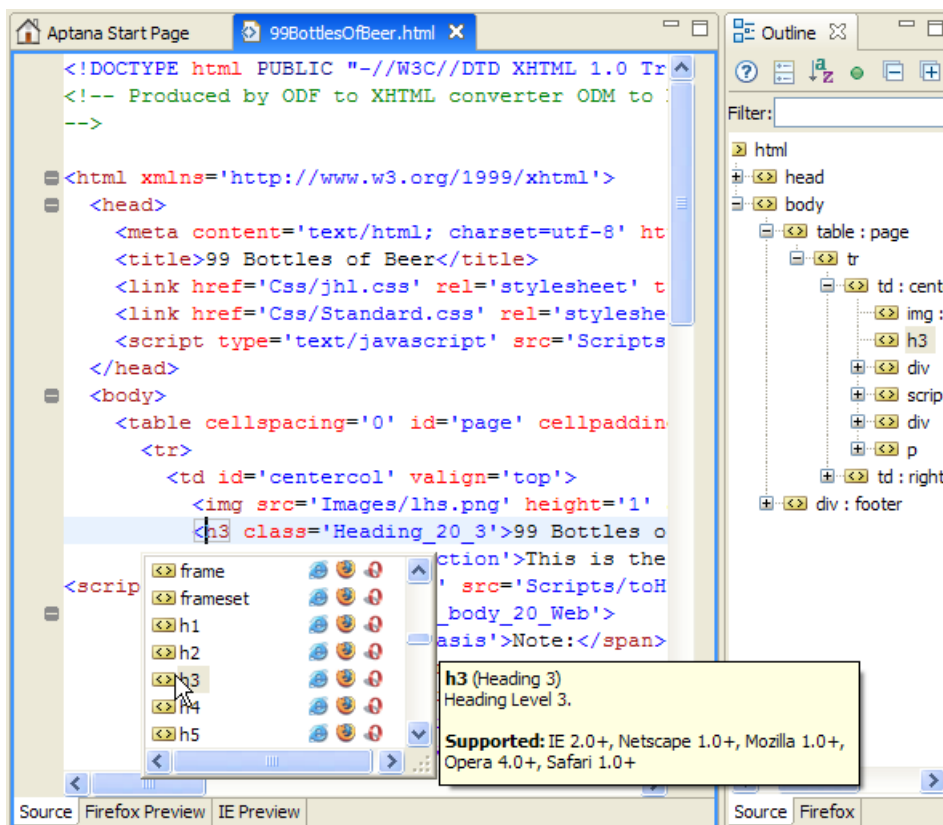
Aptana Studio

Just when I thought things couldn't get better, I decided to check out the Aptana site, to see if the original Aptana IDE had been updated. Low and behold, it had.



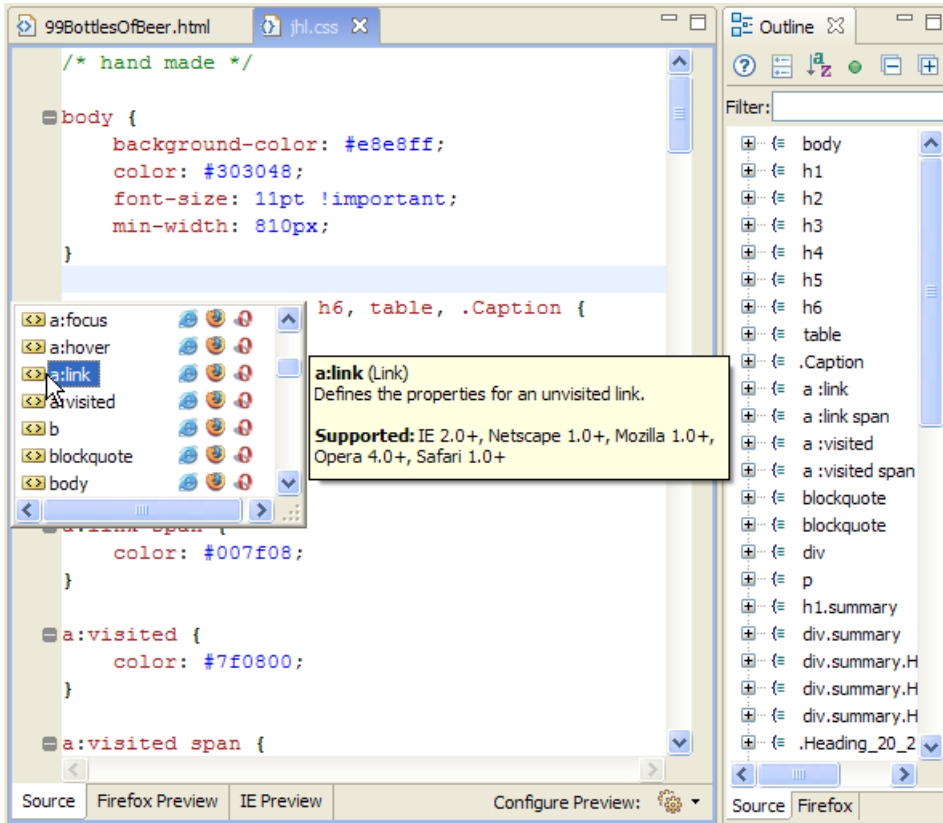
The product is now called Aptana Studio, and has a special version of Firebug which allows debugging within Firefox to be controlled from the IDE. The integration is not exactly seamless. You can set breakpoints in Aptana Studio which Firebug will see, and you can make changes to the source code, but you'll have to reset the debugger to see those changes executed.

The HTML editor offers a DOM outline, and code completion with browser specific notes:

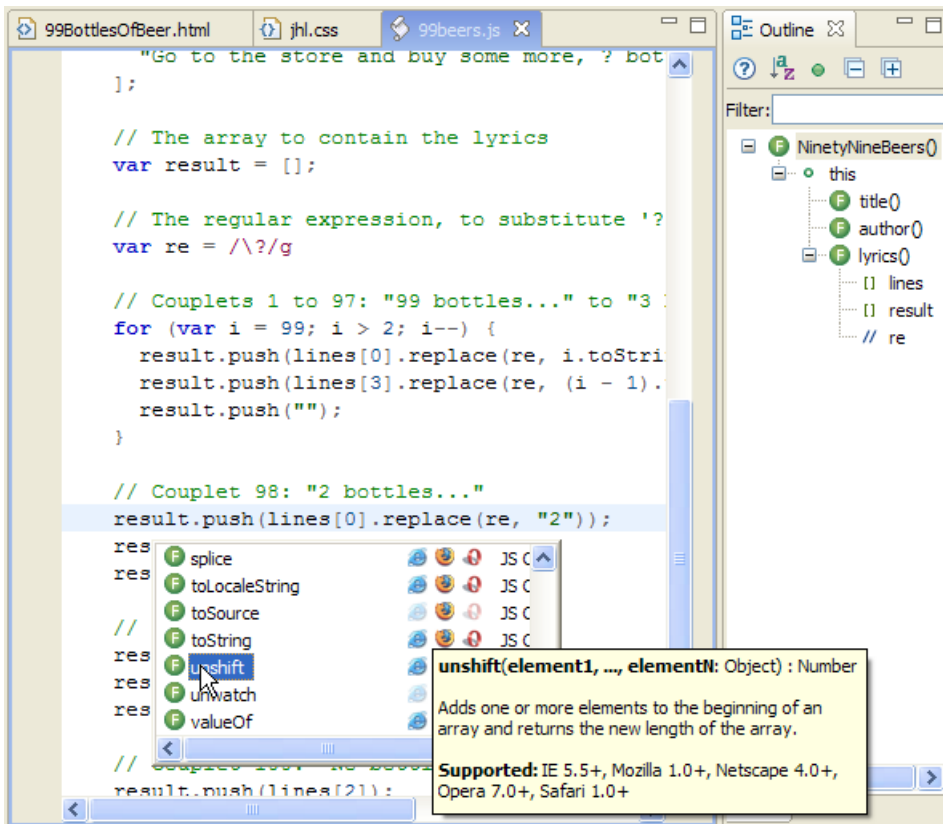


Unfortunately, all tags are offered at any one point – there are no checks made against the DTD to limit your choice to only valid tags. Pity.

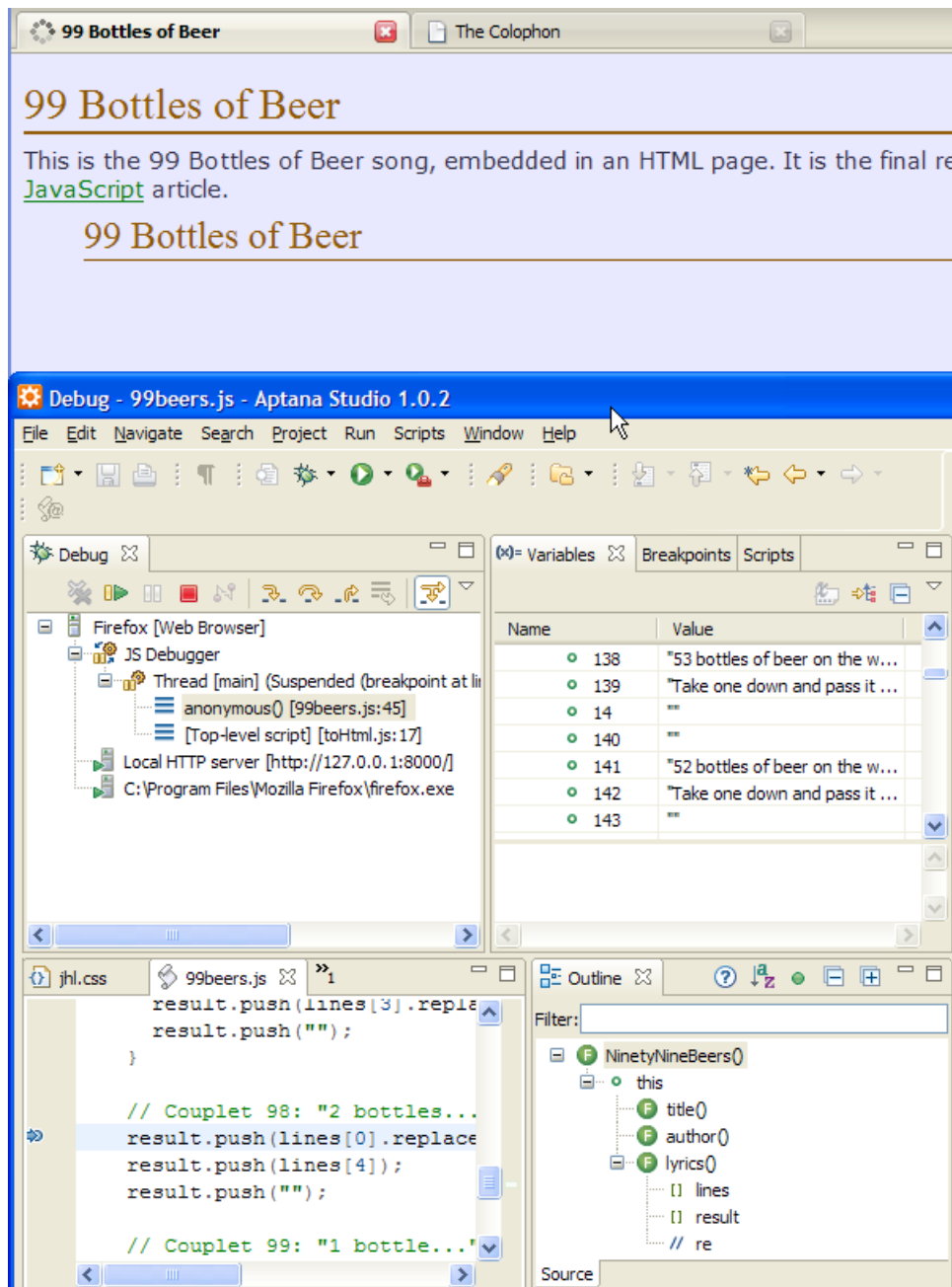
The CSS editor also offers a styles outline, and the same type of information for code completion:



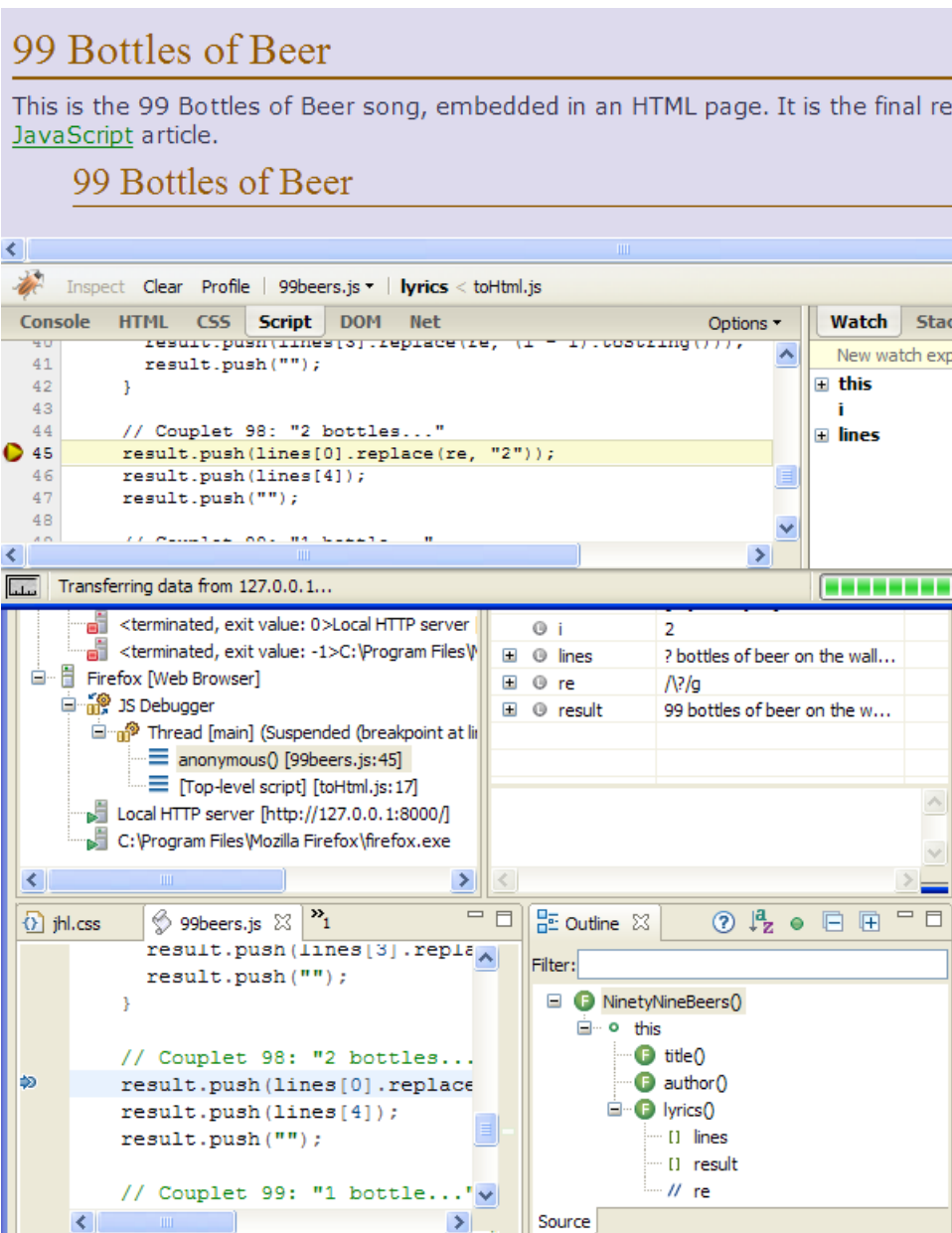
The JavaScript editor offers an environment outline, and provides browser specific code completion:



The debugger launches Firefox, but allows breakpoints to be stores between sessions (Firefox window shown in the top half, Aptana Studio in the bottom half):



Opening the Firebug window shows further information at the breakpoint set in Aptana Studio (Firefox and Firebug windows in the top half, Aptana Studio window shown in the bottom half):



As I said before, Aptana keeps your breakpoints, but there's no integration of the editors, so you'll have to start a new debugging session after making (and saving) your changes.

Contacts

Syger can be contacted for consultancy work on any of the topics mentioned in this article, by sending an email to info@syger.it.

REALbasic, Heir to Visual Basic 6?

by [John Leach](#)

[REALbasic](#) is an object oriented version of the [BASIC](#) programming language. It shares some similarity with Visual Basic 6, but also has differences. There is a slightly outdated [whitepaper](#) by Hank Marquis, which points out the most significant differences.



Most importantly, REALbasic as a language is probably closer in architecture to [Visual Basic.Net](#) and even [C#](#), than it is to Visual Basic 6. The samples in this Code Project [article](#) comparing Visual Basic.Net and C#, shows the syntactical similarity of Visual Basic.Net and REALbasic.

Which means that REALbasic can, and should, be considered as a possible alternative to moving to, or using, [.Net](#) technology for a specific project.

The two leading arguments for using REALbasic are its capability of producing a single file executable, and cross compiling for the Windows, Mac OS X, and Linux platforms. The second argument is gradually becoming more blurred, with the advent of a [Visual Basic.Net compiler](#) from the [Mono project](#). Similarly, for software houses that develop only for the Windows platform, cross compilation has little commercial significance. Nonetheless, the single file executable is still a notable advantage over having to ensure the correct configuration of (possibly) some specific version of the .Net or Mono runtime environment, on each client's computer.

Making the decision to change from Visual Basic 6 to some other platform has to be looked at from two different and potentially clashing prospects. The software house, in the form of a product manager, will be looking for the most cost efficient choice. The software developer will probably be more interested in convincing the product manager to choose the “latest and greatest” in terms of programming languages and environments.

Making the choice depends on a series of factors. Most Visual Basic 6 projects were designed to produce either stand alone Windows desktop applications, or client applications communicating with back end servers, usually databases.

The software house can choose to continue developing and maintaining their original Visual Basic 6 projects. For the short term, this is unlikely to create problems, but eventually the applications may fail to work on new versions of Windows, or may not have the same “look and feel” as newer applications, and Visual Basic 6 programmers may become more difficult to find.

Switching development to languages such as [Java](#) or [C#](#) will require a great deal of time to retrain Visual Basic 6 programmers, and to learn the enormous number of libraries available. On the other hand, finding Java and C# programmers is an easier task, as both are popular languages. However, the risk of failure is very high, and both languages require a runtime environment for the programs to work. Java has not had a particularly successful track record for client application development.

Switching development to Visual Basic.Net, which seems to be the commonest

choice, is still prone to risk. The language is too similar to Visual Basic, while being a radically different architecture, which can be a cause of confusion. The same runtime environment problems are applicable as for C#. The [Visual Basic 6 Resource Center](#) provides information on the [upgrading process](#).

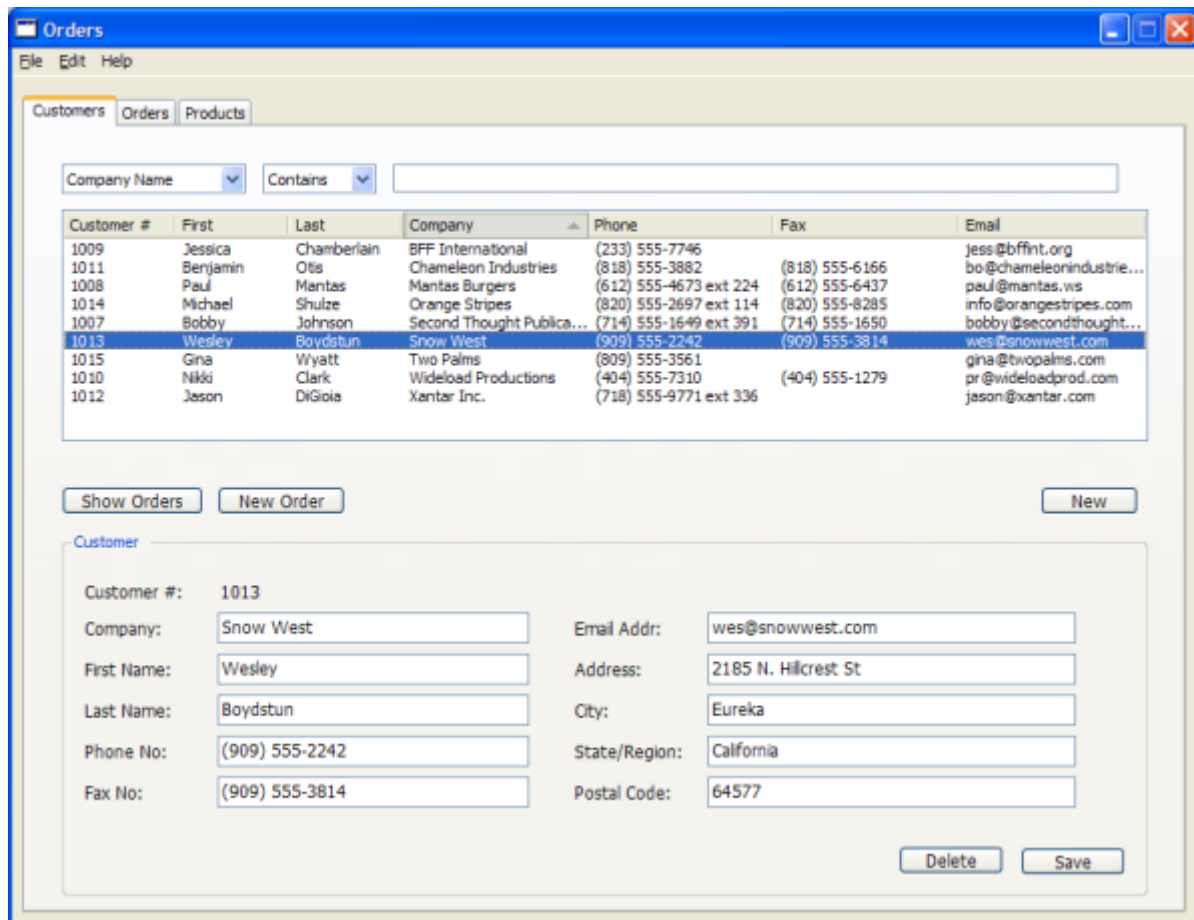
Switching to REALbasic is also prone to some risks. As for Visual Basic.Net, project conversion programs exist, but cannot be considered to be a completely 100% automatic process. However, the learning curve for Visual Basic 6 programmers is lower, the syntax is similar without being radically different in architecture, and the support libraries are smaller. Being object oriented, the language syntax is not too difficult for Visual Basic.Net, C#, or Java programmers to learn quickly. There are little or no runtime environment problems.

The rest of this article explores a couple of applications, discusses project conversion, and provides further resources for REALbasic programming.

The Database Example

REALbasic comes with a surprisingly detailed database example, including a 66 page PDF development manual. The application demonstrates normal CRUD (Create, Read, Update and Delete) operations on three associated database tables, customers, products, and orders.

The first diagram, below, shows the Customers panel, running under Windows XP.



The second diagram, below, shows the orders panel, using the filter “no” on the

company name.

The screenshot shows a software window titled "Orders" with a menu bar (File, Edit, Help) and tabs for Customers, Orders, and Products. The "Orders" tab is active. At the top, there is a search filter: "Company Name" (dropdown), "Contains" (dropdown), and a text input field containing "no". Below this is a table of orders:

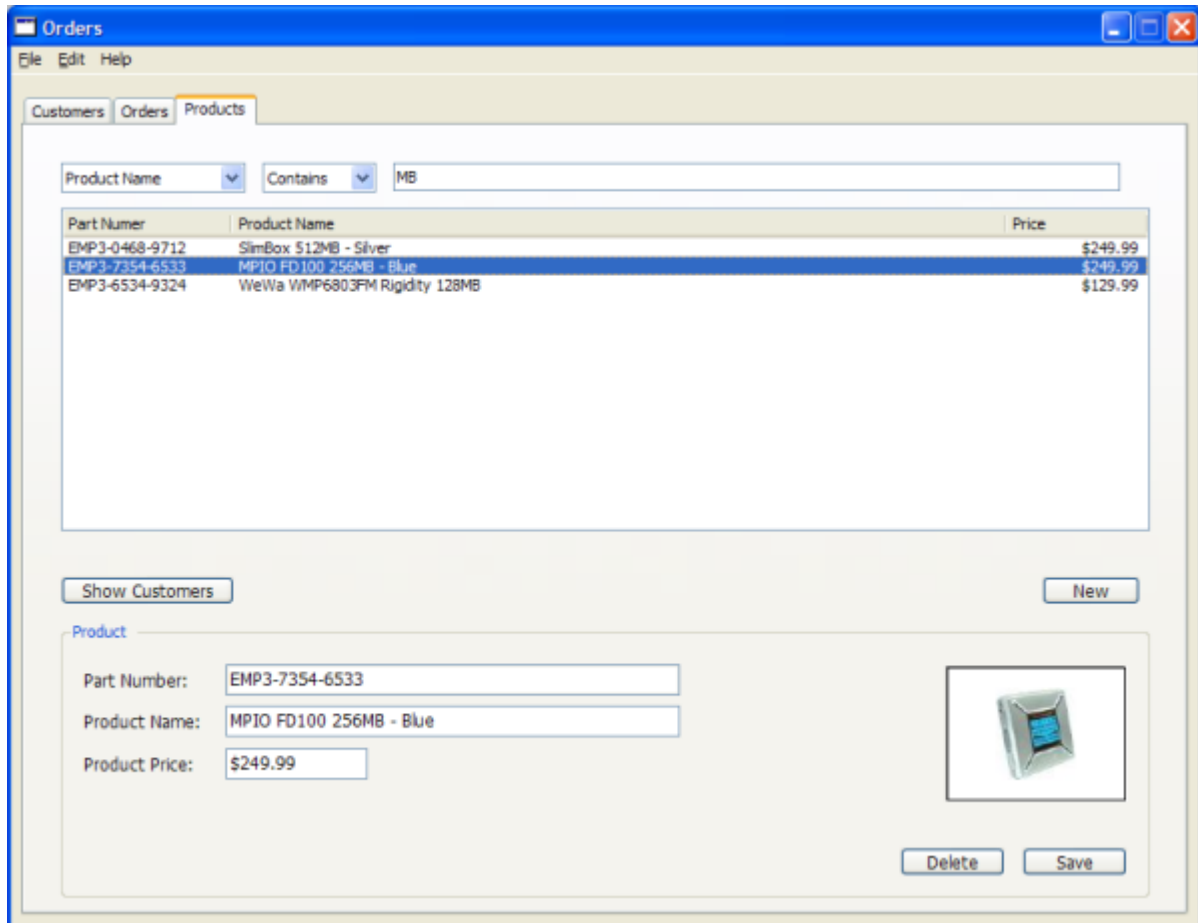
Order #	Cust. #	Company	Last Name	Ordered	Total
10022	1013	Snow West	Boydston	2004-01-21	\$386.64
10034	1013	Snow West	Boydston	2004-03-15	\$1,615.15

A "New" button is located to the right of the table. Below the table, the "Order" section is set to "10034". It contains several input fields: "P.O. #", "Tax Rate: 0.08", "Ordered On: 2004-03-15", "Shipped On: 2004-03-16", "Customer #: 1013", and "Ship Method: Download" (dropdown). Below these is a table of products in the order:

Product ID	Product Name	Qty	Price	Line Total
EMP3-0468-9712	SlimBox 512MB - Silver	1	\$249.99	\$249.99
EPDA-4656-3794	Palm Zire 71	1	\$249.99	\$249.99
EVID-6746-9721	Canon Optura 300 Mini DV	1	\$999.00	\$999.00

At the bottom left, there are input fields for "Part #", "Quantity", and "Price Each: \$0.00", along with "Update", "Delete", and "Add" buttons. At the bottom right, a summary box shows: "Sub Total: \$1,498.98", "Total Tax: \$116.17", and "Order Total: \$1,615.15". "Delete" and "Save" buttons are at the very bottom.

The third diagram, below, shows the customers panel, again with a filter set.



Standard display widgets are used in this program, to maintain multiplatform compatibility, which produce an acceptably professional interface. REALbasic has a richer collection of GUI widgets when compared to Visual Basic 6.

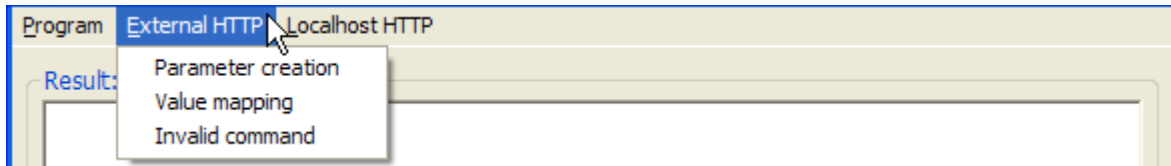
In particular, the data table, which lists the records for each database table, allows the user to sort (ascending or descending) by column name. Clicking on a table line fills in the form below the table.

The Syger XML-RPC Example

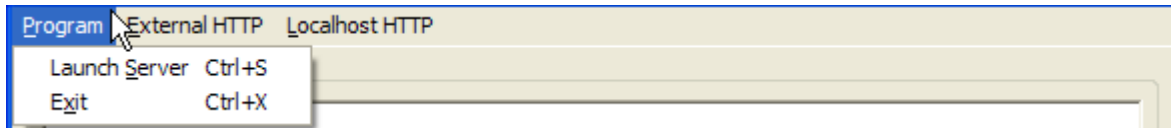
This is one of our own examples, which is a reworked version of Jan G.P. Sijm's open source [XMLRPCCOM](#) project, which was written in Visual Basic 6. This small REALbasic project was designed to show some fairly straightforward object oriented design principles to Visual Basic 6 programmers, and as a proof of concept for an XML-RPC based client application.

The [source code](#) is released under the [GNU Lesser General Public License](#). Both project files need to be built in order for the complete set of tests to be run.

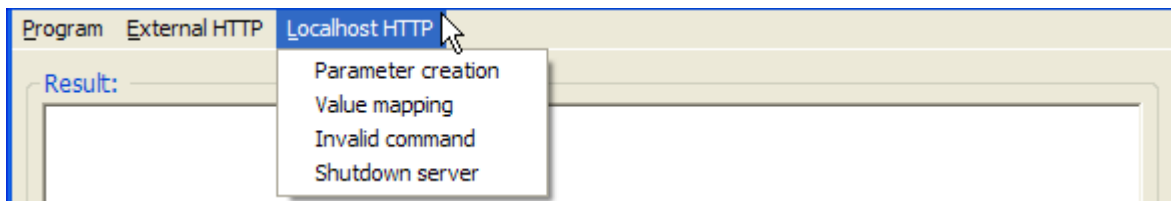
The client program provides a series of tests via its menu bar. The External HTTP menu item provides three commands which are sent to the University of Calgary XML-RPC server.



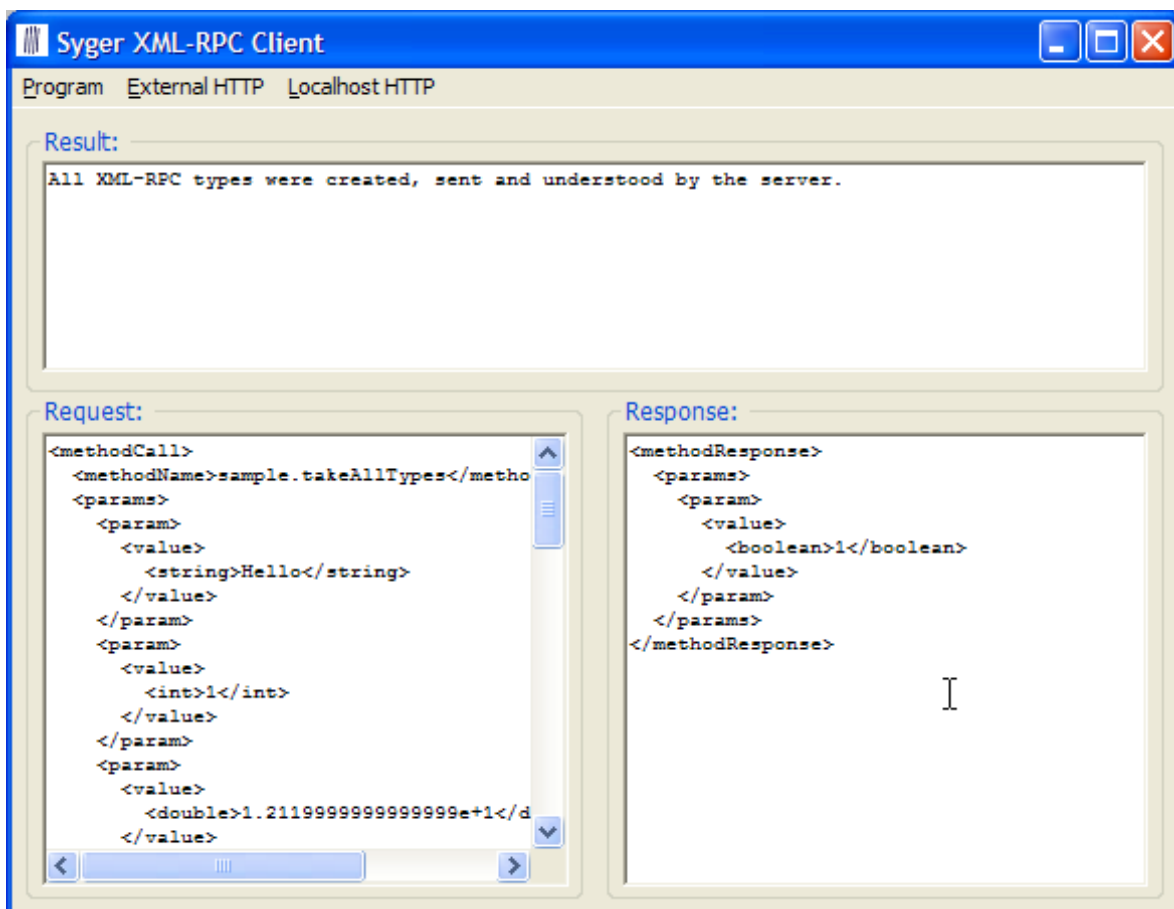
Before sending commands via the Localhost HTTP menu item, the server must be started. This can be achieved by launching the application normally, or via the Program menu item.



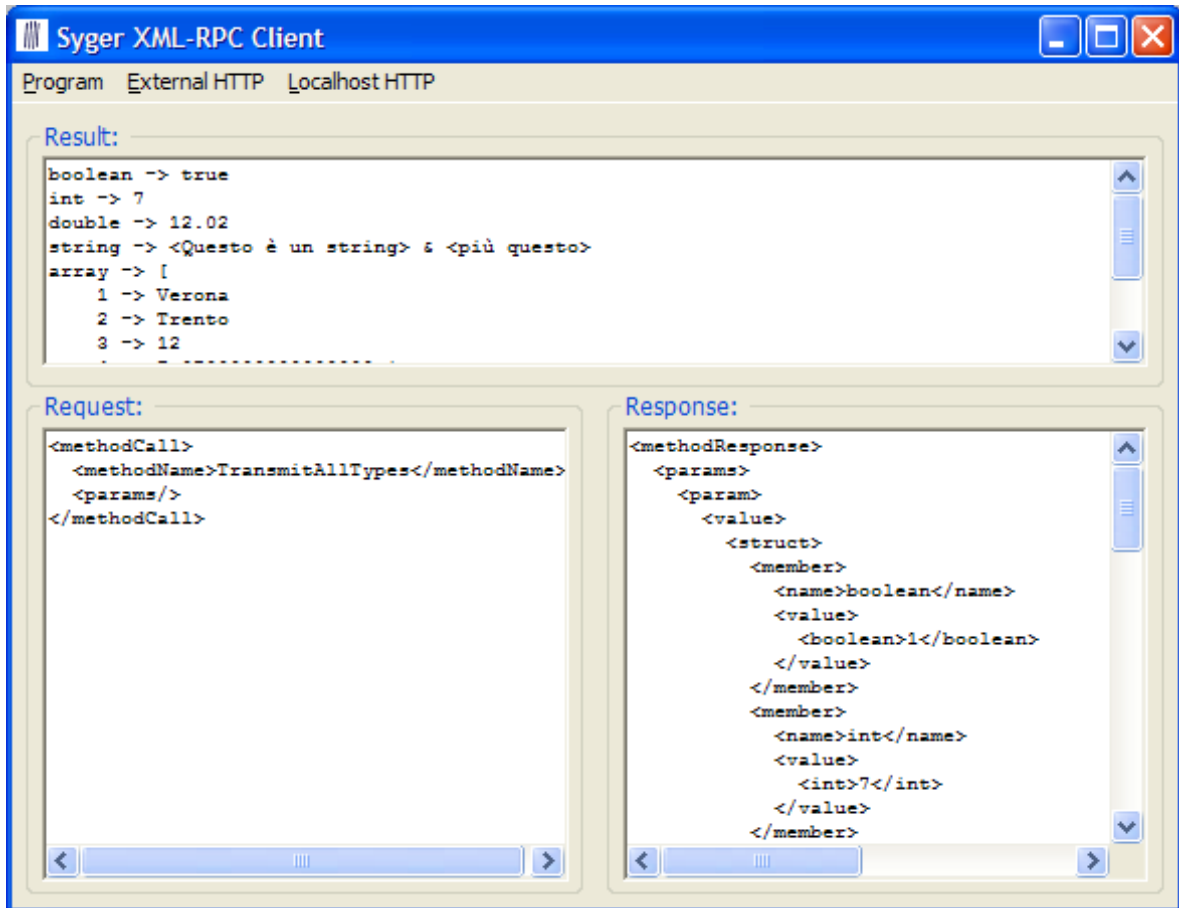
The Localhost HTTP menu item provides four commands, the first three of which are identical to the external HTTP server. The last command shuts the server down.



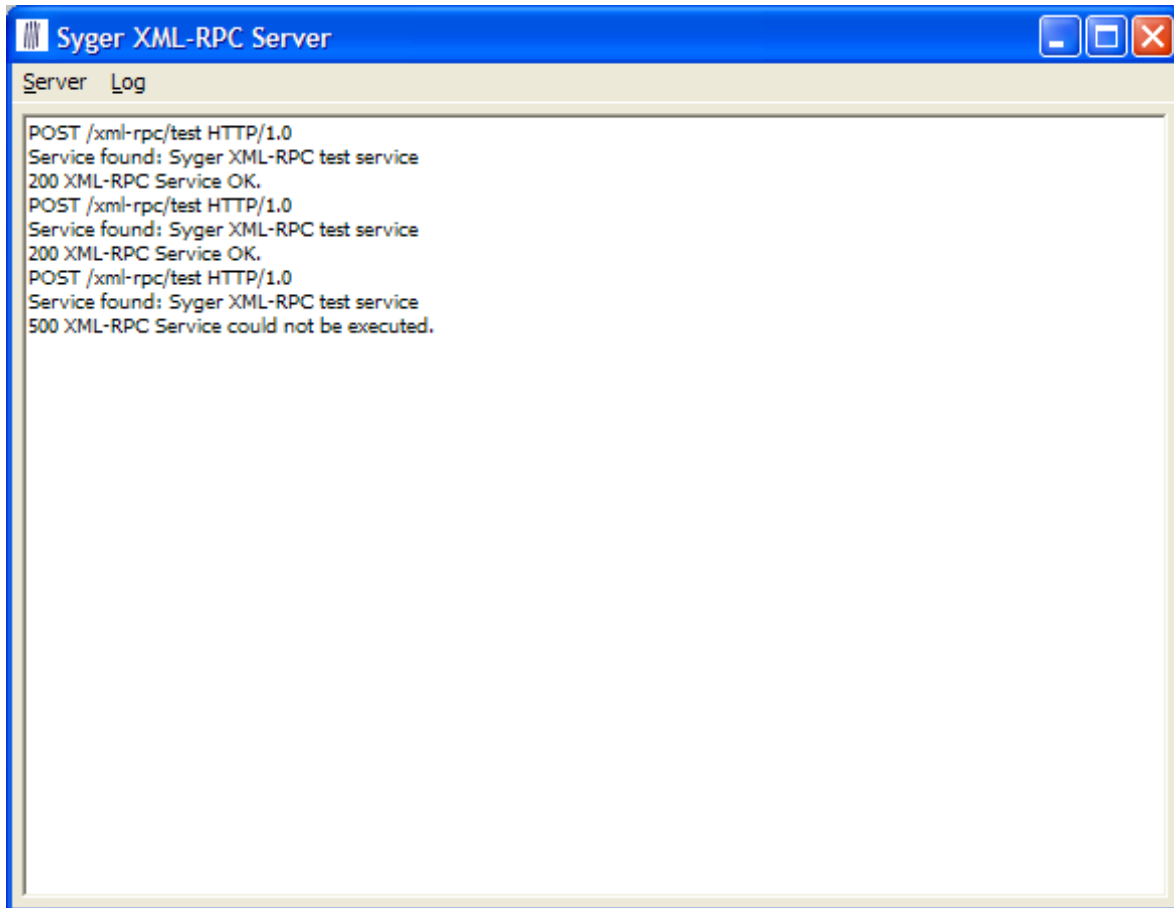
The following screenshot shows the client application communicating with the external HTTP server, displaying the result of calling the “Parameter creation” function.



The second screenshot shows the client application communicating with the localhost server, displaying the result of calling the “Value mapping” function.



The final screenshot shows the logging information in the localhost server application.



Conversion Tools

Since REALbasic is a more object oriented programming language than Visual Basic 6, any converted Visual Basic 6 project will probably be, at best, the equivalent of a poorly written REALbasic project. Nonetheless, the idea of being able to convert from one to the other, using some automatic conversion tool, is a tantalising prospect.

Although REAL Software provides a [conversion tool](#) called VB Project Converter, it seems that, according to [this article](#) by Bob Keeney, there are better tools available, in particular a product called [VB Convert!](#). Bob's software company also provides [conversion services](#).

VB Convert! goes to [great lengths](#) to explain why a 100% conversion is an unrealistic proposition. Syger has had a similar experience of converting a Cobol program to Java. After some cleaning up of the original Cobol program, about 80% of the program was successfully converted, the remaining 20% (mostly database connections, and queries) were translated by hand. The resulting program was closer to being a Cobol emulator in Java, rather than being a well written Java program. Nonetheless, it achieved its goal – the program worked correctly, with exactly the same functionality of the original, and within the project development time frame. A complete rewrite in Java would have taken at least three to five times the amount of work.

Using VB Convert! is probably a good starting point, as the program is relatively

inexpensive (about US\$50), and can be expected to convert a major portion of the Visual Basic 6 code. As the author of VB Convert! points out, this may involve cleaning up the original Visual Basic code, and results may also depend on the amount of external COM and ActiveX objects used.

Resources

The [REALbasic](#) web site provides a starting point to information on [REALbasic](#), the [language](#), the [community](#), and [support](#) available.

[RBDevZone](#) hosts a small set of open source projects, including [RBUnit](#), a unit testing framework for REALbasic.

[Einhugur](#) and [MonkeyBread Software](#) provide a collection of commercial plugins and controls for REALbasic. [RB Garage](#) has a large collection of links to resources for REALbasic developers.

Software In Motion provides a code analyser called [Reality Check](#).

[Great White Software](#) has a list of REALbasic modules.

Electric Butterfly provides a cross platform help system for REALbasic, called [UniHelp](#).

Various blogs are available, including [declareSub.com](#), [Envied Design](#), [Ramblings](#), [Software Made Simple](#), [This Much I Know](#), and [nilobject](#).

Contacts

Syger can be contacted for consultancy work on any of the topics mentioned in this article, by sending an email to info@syger.it.

Java Web Application Frameworks

by [John Leach](#)



Since the advent of [Java Servlets](#) and [JavaServer Pages](#), we have seen an explosion of web application frameworks for the Java platform. The last time I looked there were 54 open source web application frameworks listed on [Java-Source.net](#). Where does one begin?

To avoid wasting other people's time, web application development and consequently choosing a web application framework, depends on a multitude of factors including, but not limited to, the size of the development team and the deployment environment. In this article I am targeting the small software house, say from a single developer to a five man team. Larger software houses will probably have enough resources to make their own decisions. I will also further limit the discussion by defining the deployment environment as targeting the lower end of the scale, 10 to 50 concurrent users, requiring lower memory size servers with lower bandwidth. I am also assuming that you can actually choose the web application framework, rather than being obliged (even contractually) to use a specific framework.

[Matt Raible](#) has been giving presentations on Java web application frameworks since 2004 and is also the founder and lead developer of the [AppFuse](#), and [AppFuse Light](#) projects. In his [personal blog](#), he has made two slide shows available that he prepared for presentation at the ApacheCon 2007. In those presentations, his preferences produce the following list:

- [Adobe Flex](#)
- [Grails](#), together with [Groovy](#)
- [Google Web Toolkit](#) (GWT), together with [GWT-Ext](#)
- [JavaServer Faces](#) (JSF), together with [Facelets](#)
- [Seam](#)
- [Spring MVC](#), which is a part of the [Spring Framework](#)
- [Stripes](#)
- [Struts 2](#)
- [Tapestry](#)
- [Wicket](#)

That's still 10 frameworks. In another of his [articles](#), Matt talked to founders and lead developers of some of these frameworks, asking some pertinent, but perhaps also leading questions. Matt's [response](#) to an [article by Tim O'Brien](#) also sheds some light on his personal views of some of today's web application frameworks in the previous list.

For reasons which he highlights, which have been repeated by others, and given the

scope of this article that I have outlined above, I'll remove:

- [JavaServer Faces](#) – a complex technology, with a steep learning curve, more suited to larger development teams where separation of concern is fundamental. Though expensive, a good working knowledge of JSF does, however, help when looking for work.
- [Seam](#) – designed for the Enterprise ([JBoss](#) based), more suited to larger development teams, and larger deployment environments. As an Enterprise solution, however, Seam does provide several interesting possibilities, and removes some wrinkles from JavaServer Faces.
- [Spring MVC](#), which is a part of the [Spring Framework](#) – designed to resolve fundamental criticisms of the Struts 1 framework, is perhaps too dependent on [declarative programming](#), and has a steep learning curve due to its flexibility. Probably a good choice, if you are an experienced Struts 1.x programmer.
- [Struts 2](#) – while an improvement on the popular Struts 1 framework, the community has become fragmented, and specific version 2 documentation is difficult to find. There seems to be no specific solution to the double POST problem. It is more likely that potential clients or employers will ask for Struts 1.x knowledge.
- [Tapestry](#) - little or no backward compatibility between versions, steep learning curve.

Which leaves us with five frameworks; Adobe Flex, Grails, Google Web Toolkit, Stripes and Wicket. Not all frameworks are born equal, or rather, they have particular strengths within the Model, View, Controller pattern. Adobe Flex is a client side framework. The Google Web Toolkit, Stripes and Wicket emphasise the Controller and View components, leaving the Model (or at least model persistence) to other libraries or frameworks, such as [Hibernate](#). Grails is a complete framework in that it provides Model, View and Controller components, using [Spring](#) and Hibernate internally.

Before taking a closer look at these five, it is important to take a look at past, current and probable future practices in web application development, as this will help put each framework into prospective.

Past Practice

Classic web applications, which are still being produced today, consist in combinations of HTML, CSS, and JavaScript, all sent from the web server, using the HTTP protocol. The granularity of the web application is the HTML web page. Any modifications made require a request-response cycle for the complete HTML web page.

Current Practice

Current web applications are still based largely on the classic model of HTTP request-response cycles of a complete HTML web page. However, the major difference is that sub-page elements can be modified using AJAX request-response cycles and DOM manipulation. The major benefits are the reduced traffic of data to and from the web server, and hence more responsive web applications for the user.

This has had the effect of making web applications more complex, but also more flexible. The increase in complexity is due to the fact that they have to respond to the AJAX (HTTP/XML) requests, as well as the usual HTTP/HTML requests. The flexibility lies in the fact that these AJAX requests-responses can be made public in the form of APIs, (a form of [Service Oriented Architecture](#)), which other programmers can use in their own web applications, known as [mashups](#).

Probable Future Practice

The obvious evolution would be to remove the classic HTTP/HTML request-response cycle from the web application entirely. HTML pages can continue to be served by the web server, as for any other static content, leaving the web application to honour AJAX HTTP/XML request-response cycles.

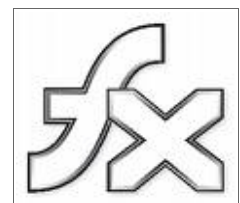
This separation of the front-end from the back-end makes it possible to create new front-ends, such as traditional desktop applications, or substitute back-end servers with newer technologies. This can only be achieved, obviously, if the AJAX services requested and provided remain compatible.

Once this decoupling process is achieved, the prospects of off-line web applications, as proposed by [Google Gears](#), becomes much easier to implement.

See [Life above the Service Tier](#), and [Does the Rise of Service Oriented UI \(SOUI\) Mean the Death of Server-Assisted MVC?](#), for other similar prospectives. There are also [proposals](#) for improving HTML. James Ward provides a good [definition](#) of the Rich Internet Application.

Adobe Flex

[Adobe Flex](#) is not a Java based product. It is a combination of [ActionScript](#), XML and CSS, which is used to produce a rich user interface, based on Flash technology.



The quality of the user interface is as good as it gets, going much further than anything possible with HTML / CSS / JavaScript combinations. With its Flash parentage, good eye-catching animation effects are also possible. It also benefits from being open source, and backed by Adobe. The Wikipedia [article](#) gives an overview of Flex's history and parentage.

The [Flex development site](#) provides specific information for Java programmers for [interfacing Flex to Java web applications](#). The [OpenLaszo](#) project, which provides

similar though incompatible functionality, may also be a good source for components. Flex applications can be built using the free Flex Software Development Kit (SDK), but Adobe also sells a development tool, Adobe Flex Builder, which starts at about US\$250.

The [Flex Showcase](#) shows a list of web applications built using Flex. The excellent article [Crash Course in Next-Gen RIA: AIR, Silverlight, and JavaFX](#) by Alexey Gavrilov, illustrates this technology, and compares it with Microsoft's [Silverlight](#), and Sun's [JavaFX](#). Adobe are also moving this technology to the desktop, via the [Adobe Integrated Runtime](#) (AIR).

There are some problems with Flex, other than those mentioned by Matt Raible (search engine unfriendly, HTML rendering and printing problems). The size of the client application can become very large. The Flex Showcase comes in at 705 KB, the [Picnik](#) site, which has a truly appealing user interface, weighs in at 1,350 KB.

The Google Web Toolkit and GWT-Ext



The [Google Web Toolkit](#) (GWT) is an interesting concept. It produces a rich user interface that is composed of HTML, CSS and JavaScript, however the actual user interface is developed in Java, similar to designing a Swing user interface. This Java user interface is then translated into HTML for static content, CSS for styling, and JavaScript for dynamic content. The HTML/CSS/JavaScript user interface communicates with the back-end server using AJAX.

The toolkit is open source, and comes with powerful tools, which can be integrated into Eclipse, for development and debugging. The Wikipedia [article](#) on the GWT gives a concise overview. The GWT web site also provides extensive documentation, articles and tutorials. The [GWT-Ext](#) project, among several others, provides additional components for the toolkit. The Timepedia [blog](#) provides some interesting insights into the GWT.

A great deal of effort has also been taken to make the interface fast and responsive. The intent is clearly to make the user interface “feel like a traditional web application, just better”.

As for Flex, GWT can provide a complete separation of the front-end from the back-end. Even with [externally supplied widgets](#), which use the [Script.aculo.us](#) libraries, GWT cannot fully compete with Flex for animations and effects. Taking the [Kitchen Sink](#) example, the client application code size seems to be smaller than Flex. GWT does not have the disadvantages of Flex, but the [security problems](#) inherent in any JavaScript application remain.

I am less inclined to subscribe to Matt Raible's list of disadvantages. From the documentation, I don't see why it should be difficult to incorporate a partial GWT user interface inside a classic Java web application. I can't see how GWT could have implemented Java 5 syntax in the compiler, it is a limiting but natural constraint – though this may change in the future. GWT seems to be getting more and more

support from widget designers.

One interesting point will be to see if the global programming community, as opposed to just Java programmers, will use GWT with say a Python, Ruby, or PHP back-end. This [blog](#) from Palantar, loosely explains how he integrated GWT with Django (Python).

Grails and Groovy

[Groovy](#) is a scripting language for the JVM. The Wikipedia [article](#) gives a quick overview of the language. It shared some similarity with Ruby, but is able to use the great wealth of Java libraries natively. It can be used with an interpreter, and can also be compiled to JVM bytecode.



[Grails](#) was heavily inspired by Ruby on Rails, and by the Groovy language, which was itself, at least partially, inspired by Ruby. The Wikipedia [article](#) gives a concise overview of the framework. This [article](#), by Harshad Oak, gives a short introduction to Groovy and Grails.

After a first look, you might be wondering why any experienced Java programmer would consider such a radically different framework. It has a non-Java language, and uses the so called “convention over configuration” MVC framework.

If you have had any experience using [Ruby](#) and / or [Ruby on Rails](#) you will understand that together they create a very fast development cycle, while necessarily limiting some aspects of flexibility, due to the conventions used. The development process works very efficiently where those limits are acceptable. The development efficiency increase is so great that the limitations can become of secondary importance. This comes at the cost of creating a web application which is less performant than traditional Java web frameworks.

The Grails web site, much as the Ruby on Rails website provides [tutorials](#), [screencasts](#), and [plugins](#). Marcel Overdijk has written [two articles](#) on integrating Adobe Flex with Grails, and a third article is available from [Jacob von Eyben](#). A [plugin](#) has been created for GWT integration.

Stripes



[Stripes](#) uses Java [annotations](#) and [generics](#) to provide a “convention over configuration” MVC framework. It also makes use of more recent versions of [Servlets](#) (2.4) and [JavaServer Pages](#) (2.0). The web site provides a [quick start guide](#), and a more serious [sample application](#). Interestingly, the site also provides an [article](#) comparing Stripes with Struts 1.x. Whilst pointing out some of the leading problems with Struts 1.x, it also indicates the similarities in structure, which have been more closely analysed by Rick Smith in an [article](#) on porting Struts 1.x applications to Stripes.

The [How-To page](#) provides articles on file uploading, AJAX, localisation, state management, and multi-page wizard construction, among others. [Examples](#) are provided for using both the Spring and Hibernate frameworks.

Similar to the Grails philosophy, speed of development is considered a primary concern, limiting some aspects of flexibility. This is one of the critiques made by Matt Raible, that URLs are hard coded to ActionBeans. Stripes supports incremental building of web applications, web pages can be prepared quickly without having to create the model components, which can be developed at a later stage.

According to the framework's inventor, Tim Fennell, Stripes is a high quality, action oriented framework, with powerful type conversion, binding, and validation, which shines in applications with lots of complex data interactions.

Kai has an [article](#) integrating Stripes with the GWT.

Wicket

[Wicket](#) provides a mechanism for complete separation of the view components. Everything else is handled by Java code. As with other modern frameworks listed here, it takes full advantage of the latest enhancements to the Java language, such as [annotations](#) and [generics](#). The templating mechanism uses HTML with (a very few) special attributes and tags which are bound to the Java components.



The programming experience is much closer to Swing application development than traditional web applications. Wicket has the concept of application, as well as page and component. The `WebApplication` class is responsible for setting the configuration for the entire application.

The framework's inventor, Jonathan Locke, [describes Wicket](#) as being a compact, focused, and powerful framework, designed to solve one very specific problem well; enabling component oriented, programmatic manipulation of markup, and very little else.

Contacts

Syger can be contacted for consultancy work on any of the topics mentioned in this article, by sending an email to info@syger.it.

Version Control and Unit Testing

by [John Leach](#)

In many software houses version control and unit testing are the poor relatives of software development. Everyone knows that a programmer needs a text editor and a compiler (or interpreter) to build an application. Modern IDEs have demonstrated that they can have a phenomenal impact on the development cycle, despite their high learning curves. But what are the advantages of learning about, and using version control or unit testing?

This article explains the advantages of using these development tools, and will, hopefully, convince you that they are quite as essential in the development process as the compiler is.

Version Control

Almost everything I write on my computer gets stored in my version control repository. I just committed this web site project, since I'm starting this article, and there were 7 additions, and 8 changes since I last committed the project. There were also two images I'd forgotten to add to the project from the previous article. Later on today, I'll backup the repository, which will save all the projects that I'm currently working on.

I convert this [OpenOffice.org](#) document into a web page using the [odm to xhtml](#) program, which has changed considerably since I first started working on it. I also know exactly how many changes, and where they are. That's because I use [Subversion](#) to keep the source files under version control. Let me show you the `textparser.rb` file history:

From: 03/10/2007 To: 28/11/2007 Messages, authors and paths

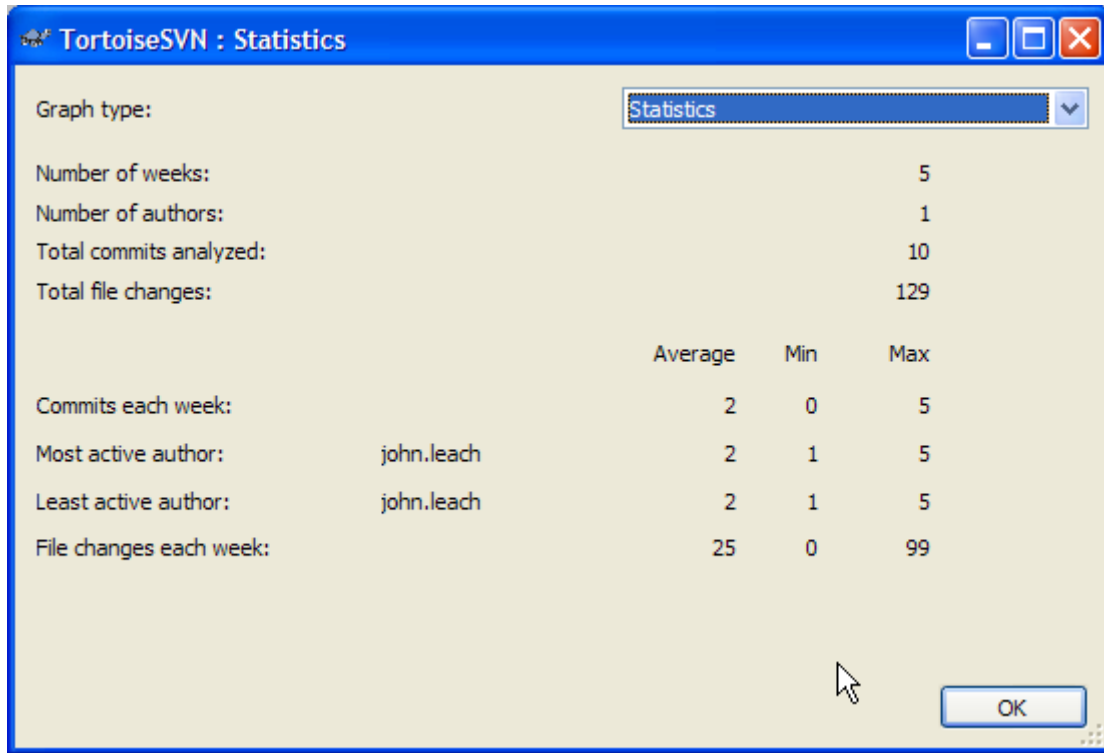
Revision	Actions	Author	Date	Message
88		john.leach	09:45:26, 28 November 2007	Minor bug fix for Literal
84		john.leach	15:10:23, 27 November 2007	Simplified XHTML pretty printing
76		john.leach	13:09:54, 13 November 2007	Updated documentation for public release
75		john.leach	11:09:14, 13 November 2007	Minor bug fixes Removed unused CSS classes from s
72		john.leach	16:59:38, 07 November 2007	Minor bug fixes
54		john.leach	10:35:13, 15 October 2007	Minor bug fix
49		john.leach	23:55:19, 07 October 2007	Added literal style to inject markup in the output doc
45		john.leach	18:23:36, 05 October 2007	Fixed text writing bug Fixed empty string matching b
36		john.leach	16:44:32, 03 October 2007	Minor modifications to the Ruby code, and docs. Rer
15		john.leach	11:03:17, 03 October 2007	First import of the Odt2Xhtml project to the trunk

First import of the Odt2Xhtml project to the trunk

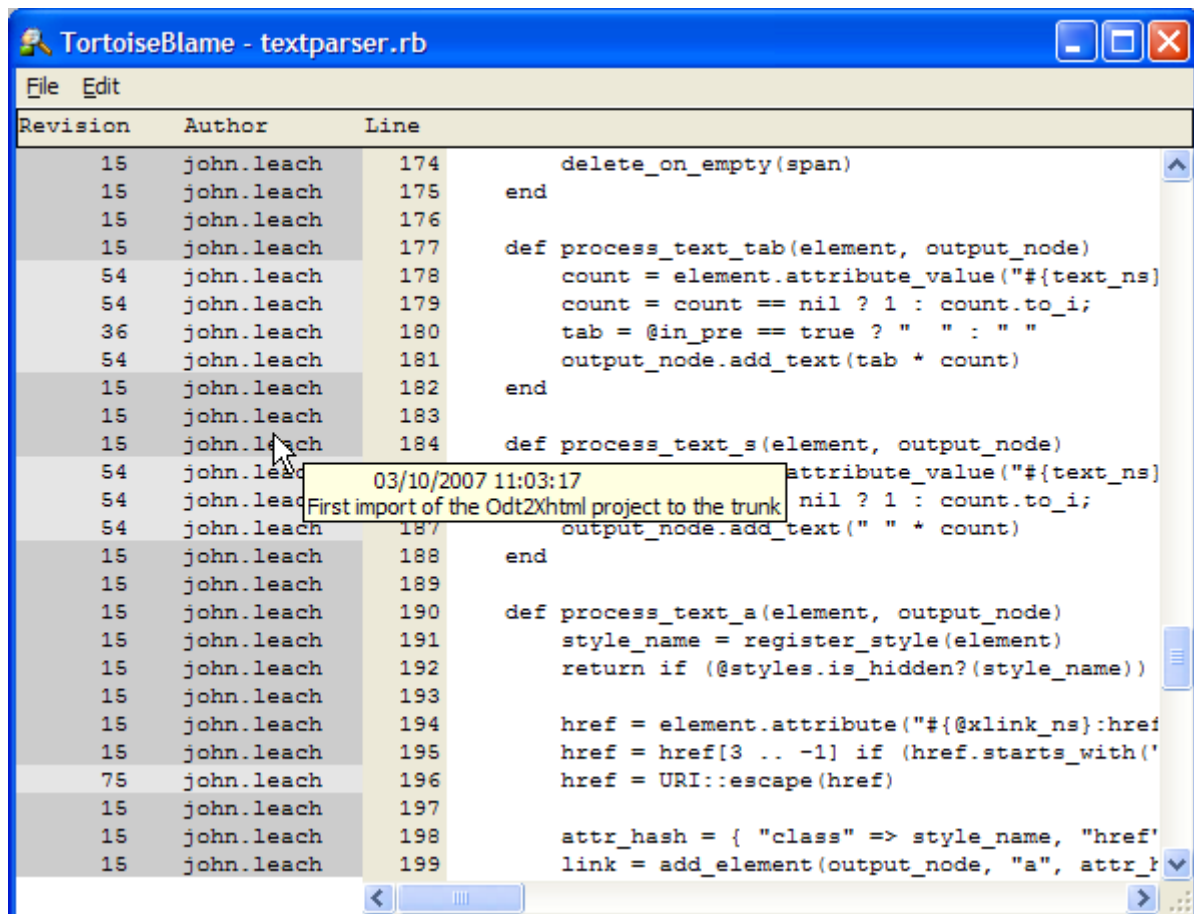
Action	Path	Copy from path	Revision
Added	/Odt2Xhtml/trunk/changelist.txt		
Added	/Odt2Xhtml/trunk/docs		
Added	/Odt2Xhtml/trunk/docs/Pictures		
Added	/Odt2Xhtml/trunk/docs/Pictures/Thumbs.db		
Added	/Odt2Xhtml/trunk/docs/Pictures/campanile-venezia.jpg		
Added	/Odt2Xhtml/trunk/docs/Pictures/measure.jpg		
Added	/Odt2Xhtml/trunk/docs/Pictures/police-car-accident-miami.jpg		
Added	/Odt2Xhtml/trunk/docs/Pictures/syger-logo.jpg		
Added	/Odt2Xhtml/trunk/docs/Pictures/tree-winter-xxx.jpg		
Added	/Odt2Xhtml/trunk/docs/Templates		
Added	/Odt2Xhtml/trunk/docs/Templates/Manuals.xml		
Added	/Odt2Xhtml/trunk/docs/convert_odm.bat		
Added	/Odt2Xhtml/trunk/docs/convert_odt.bat		

Since I am a lone developer for this project, you'll only see me as the author. On multi-developer projects, you'd see the login names of each author making the [commits](#). The messages are stored for each commit, and give a brief reminder of why the commit was made. The bottom window shows all the other files that were committed at the same time as `textparser.rb`.

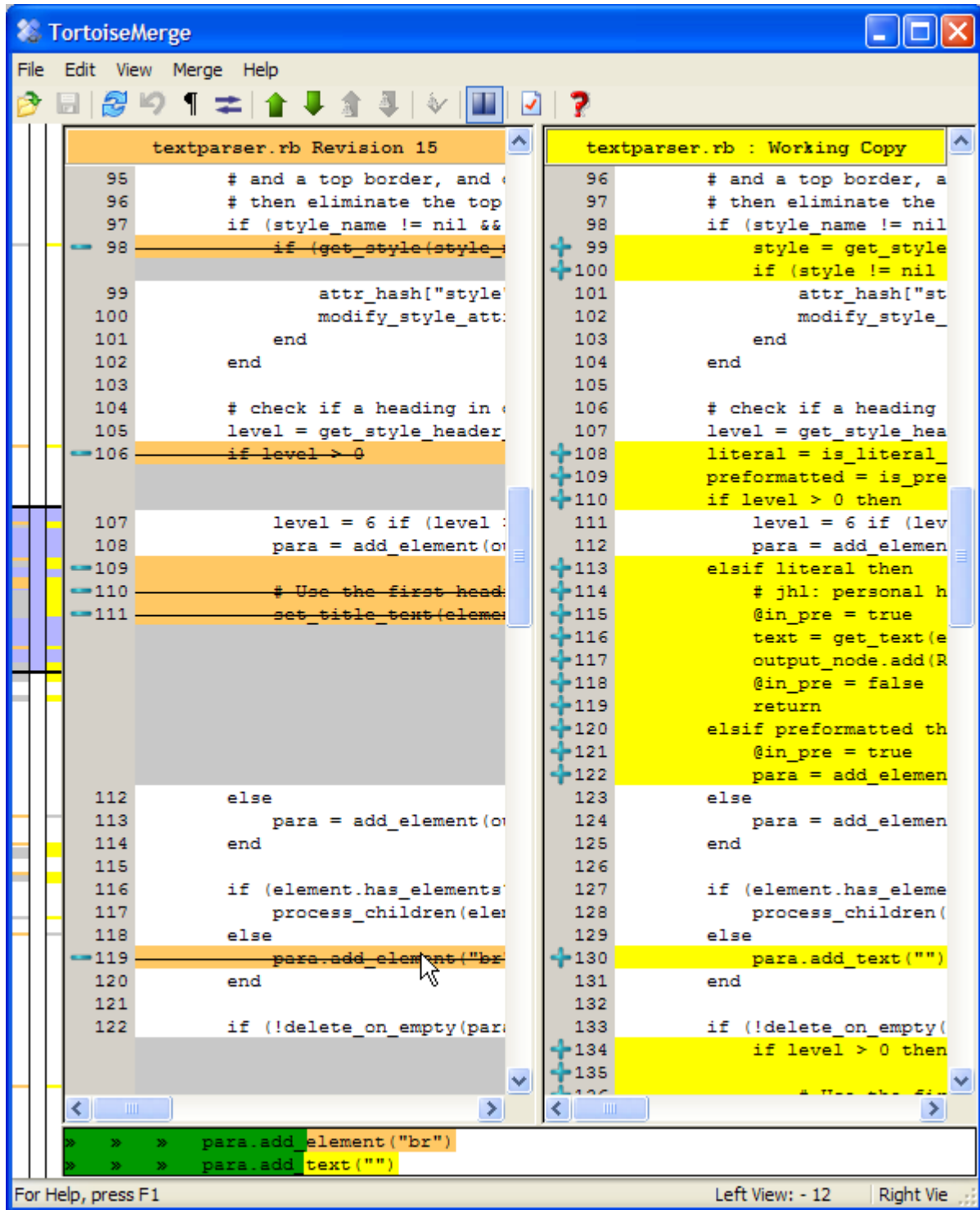
I can also pull out some statistics, which is why I know exactly how many changes there are:



I can see how the file looks today, showing each line by latest revision (it's called blame, because I can see who is responsible for each line of code in the current version):



I can also see what the changes are, from any specific revision to another:



The left hand side shows revision 15 (the first time the file was committed to the repository), and the right hand side shows my current working copy on my hard disk. Where the cursor is pointing, I can see the specific changes on a character by character basis in the bottom text box.

That's enough of pictures, but version control doesn't stop there by any means. When a version is released I can create a [branch](#) at that point. I'll probably continue working on the project in the future, because I tend to add features, as and when needed. But if I get a bug report for the released version, I can restore that version to my working copy. So despite the fact that I may now have a heavily modified

development version of the program, which may or may not be working properly, I can set up my working copy with exactly the version specified in the bug report.

When I've finished fixing the bug, I can update the version branch, and re-release it. I can then [merge](#) the bug fix back into later version releases, and my development version, by checking exactly what code I modified to fix the bug, and then adding those modifications into each version branch, and the [trunk](#) (the development branch).

[Subversion](#) has become a successful open source project, with plugins for IDEs such as [Subclipse](#), GUIs for [Windows](#), and [multi-platform](#), integration with [Apache](#), and web based issue [tracking](#) software. There is also a freely available book from [O'Reilly](#).

Unit Testing

Whenever you make a software product available to others, whether it be a library, command line, desktop or web application, it will be tested – by the user. The user expects that every test he makes will succeed – the program will perform correctly. When the user's test fails, you have a problem, and probably an upset user.

Naturally enough, the development team will also test their software. The usual practice is to write code corresponding to a particular problem to be solved, then execute the code feeding in valid and possibly invalid values. Once the tests have been checked, and the code modified to work correctly, the programmer will move on to the next problem.

Although this may seem a satisfactory process, it has several flaws.

- Tests are usually uncontrolled, undocumented, inconsistent, and rarely repeated.
- The larger the quantity of code to be tested, the less enjoyable the process. Most programmers do not enjoy testing and debugging code when that process takes a long time.
- Programmers do not enjoy spending time testing code written by others, even on the same team.
- They do not enjoy fixing the code of others, and may not be able to provide sufficient information to other programmers in order to fix the problem.
- There is a natural tendency for the programmer to supply valid values, testing input for a name, for example, they may test with "John", or the empty string "", but will rarely test with something like "...".
- They are unlikely to test for rare events, such as a disk read or write failure, database or socket connection failure, which can be difficult to simulate while executing some specific piece of code.
- Programmers are not users of the software. Users will perform broader and deeper testing of the code, especially if they have to use the program regularly

in their work.

[Unit testing](#) is a process of applying general programming practices to develop suites of tests which can be repeated, and eventually automated. [Ron Jeffries' site](#) provides a [list of unit testing frameworks](#) for a variety of languages, as does the [opensourcetesting.org](#) web site. Tim Burns has written a brief article entitled “[Effective Unit Testing](#)”, which takes a balanced view of the topic.

Many articles have been written on unit testing, practices, design patterns, and philosophy. Fortunately, the original ideas presented by Kent Beck in his article “[Simple Smalltalk Testing: With Patterns](#)”, still hold true, irrespective of the great deal of hype subsequently generated.

Unit testing is not a recipe for perfect “bug free” software. It does require extra effort by the programmer, which is an extra cost. It is perfectly possible that the unit testing code itself contains bugs. In some circumstances, it becomes necessary to construct [mock objects](#), particularly for database transactions. Modifications to the code may require changes to the unit testing code. Unit testing can produce over confidence, especially where the tests do not cover 100% of the code. Code coverage of unit tests can be difficult to analyse.

Apart from the obvious task of creating a controlled, documented (the unit testing code itself), consistent and repeatable set of tests, the process of writing unit testing code can also produce other benefits. The extra effort required increases disciplined programming, and induces the programmer to write the minimum necessary to complete the task. The effort of isolating the code, to increase testability, makes the final code more loosely coupled, which increases re-usability. The unit tests provide a reference which allows the programmer to refactor code more confidently and aggressively. Unit testing code becomes the first “user” of publicly defined interfaces and libraries, thus guaranteeing the consistency of those public definitions, both during compilation, and at run-time. Finally, unit testing code can also provide a historical database of previously fixed bugs, thus ensuring that those bugs do not reappear in future releases.

Contacts

Syger can be contacted for consultancy work on any of the topics mentioned in this article, by sending an email to info@syger.it.