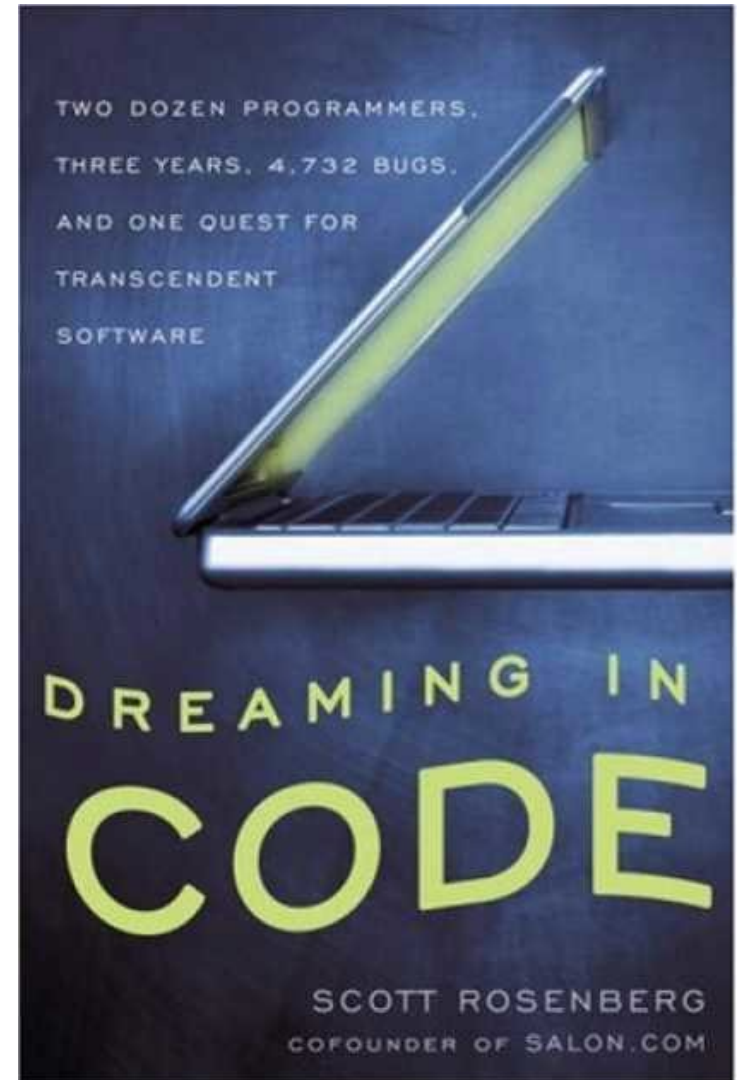




# Recommended Reading

- Because it's a great read
- Because it's got lots of useful insights into computer programming
- Because it will make you realize that you are not alone
- Because I will refer to it several times in this presentation



# So You've Decided to Write a Web App

- To provide users with a nice, friendly, well-understood interface to organizational data
- Server-centric so no distribution issues
- User interface is “standard” so no lock-in to proprietary UI
- It's all good...
  - Or maybe not (see *Dreaming in Code*)



# Initial Steps

- Decide what your application will do in broad terms
- Choose your technology



# Technology Choices

- Server technology
  - Model 204, of course!
  - Or get the heck out of here
- Client technology
  - HTML
    - Almost all data presented in web apps is HTML
    - Though sometimes in its internal representation (DOM)
  - Javascript
    - For local user interactions
- But how to move data between client and server?



# Why Model 204 As Your Web Server?

- Because that's where your data is (duh)
- But, it's also the best option available (in my humble unbiased opinion)
  - Still the only integrated database server, application server, and web server
    - Database more flexible than tabular SQL model
  - With Janus SOAP ULI, has one of the best O-O languages on the market
    - There's a strong industry consensus that O-O is the way to go (see *Dreaming in Code*)
  - Just in time compilation (with SirFact)
  - Very CPU and I/O efficient



# Moving Data From Model 204 to Browser: The Early Days

- HTML directly PRINTed or inside HTML/TEXT blocks
- Very simple but...
- Unstructured so messes were inevitable
- Nothing to prevent badly structured HTML, especially unmatched start/end tags
  - A common problem
  - Result could be very strange/confusing page on browser
- Lots of data transferred to browser for each page
  - So slow and bandwidth hungry
- Database/application code interspersed with HTML generation



# Moving Data From Model 204 to Browser: The Next Generation (cont...)

- Using the Janus SOAP XML API to produce HTML
- Start and end tags **guaranteed** to match up
  - Because the XML API generated XML
  - In fact, guaranteed to produce XHTML
- Code easier to structure
  - Because document/page does not need to be built in order
    - XML API allows out of order insertion of text
    - So application requirements can drive processing more than display requirements
    - But, HTML generation code still interspersed with database application code



# Moving Data From Model 204 to Browser: The Next Generation (...cont)

- Still lots of data transferred from server to browser
- Relatively CPU-intensive
  - Lots of XmlDocument API calls to add HTML details



# Example of a Web Page Built with XmlDocument API

```
b
%doc    is object XmlDocument
%html   is object XmlNode
%body   is object XmlNode
%table  is object XmlNode
%tr     is object XmlNode
%i      is float

%doc = new
%html = %doc:addElement('html')
%html:addElement('head'):addElement('title', 'Squares of 1-20')
%body = %html:addElement('body')
%table = %body:addElement('center'):addElement('table')
%table:addAttribute('border', '2')
for %i from 1 to 20
    %tr = %table:addElement('tr')
    %tr:addElement('td', %i):addAttribute('align', 'right')
    %tr:addElement('td', %i:toPower(2)):addAttribute('align', 'right')
end for

$web_type('text/html')
%doc:webSend('INDENT 2')
end
```



# Moving Data From Model 204 to Browser: The Latest and Greatest

- Sending the data as XML that represents the data's structure but no HTML
  - So no HTML generation code interspersed with application code
  - Minimal data sent to browser so bandwidth friendly
  - Only one XmlDocument call per data element so very CPU-friendly
- Transforming the data to HTML using XSLT
- Centrelink's been doing this for years
  - And I thought they were daft
    - I was wrong



# XSLT

- XSLT = eXtensible S Stylesheet L Languages T Transformations
  - XSLT really has nothing to do with “style”
    - CSS is really what you would use to style documents
- A language that is expressed in XML
  - With HTML embedded inside XSLT programs also as XML
  - Which means HTML produced by XSLT is guaranteed to be XHTML
    - Start tags and end tags guaranteed to match up
    - Though programmer has to enter them
- A functional language
  - Which means a somewhat different mindset required
  - But I've found that functional language claims of greater robustness are largely true



# XSLT and XPath

- XSLT uses XPath to access/navigate data in the source XmlDocument
- The same XPath used by the XmlDocument API
- But, because XSLT is such a minimalist language, you'll probably sometimes use more complex Xpath than you would with the XmlDocument API



# The Downsides of Using XML+XSLT to Send Data from Model 204 to Browser

- Application now in at least two places:
  - UL code in UL procs
  - XSLT in some other proc
  - But maybe that's a good thing
- Programmer needs to know both UL and XSLT (or need two programmers)
  - But XSLT is **easy**
    - Think of XSLT as HTML statement on steroids
  - And no more difficult than developing and learning some XmlDocument wrapper API for building XHTML
  - And **lot's** of web resources to help you get started
    - The spec is pretty readable: <http://www.w3.org/TR/xslt>



# XSLT Browser Support

- What if a user's browser doesn't support XSLT?
  - LOL, as-if. This hasn't been an issue in like 8 years.
  - People (including Roadway) used to to use middleware XSLT transformation engines to avoid this problem
    - But this reduces bandwidth benefits of XSLT
    - And introduces another moving (and overworked?) part
    - Totally unnecessary now, anyway
- What about browser inconsistencies in XSLT support?
  - AFAIK, there are none!
  - If there are any, they're minor
  - Because XSLT is so simple



# Example of a Web Page Built with XSLT: The UL that Produces the XML

```
b
%doc      is object xmlDoc
%squares  is object xmlNode
%number   is object xmlNode
%i        is float

%doc = new
%doc:version = '1.0'
%doc:addPi('xml-stylesheet', 'type="text/xsl" href="foo.xsl"')
%squares = %doc:addElement('squares')
for %i from 1 to 20
    %number = %squares:addElement('number')
    %number:addAttribute('value', %i)
    %number:addAttribute('square', %i:toPower(2))
end for
$web_type('text/xml')
%doc:webSend('INDENT 2')
end
```



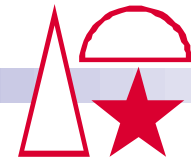
# Producing XML For XSLT To Transform

- While you could do it using Print/Text/Html statements, don't
- I believe this is what Roadway did many years ago
  - Because XmlDocument API was not available
- But now, use the XmlDocument API
  - Easier to read the code
  - Easier to get the code right
    - Start and end tags generated for you
  - Character encoding issues handled automatically for you
  - Slightly more CPU overhead but, because it's only one tag per data element, the extra overhead is minimal



# Example of a Web Page Built with XSLT: The XSLT to Produce the HTML

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/squares">
    <html>
      <head>
        <title>Squares of 1-20</title>
      </head>
      <body>
        <center>
          <table border="2">
            <xsl:for-each select="number">
              <tr>
                <td><xsl:value-of select="@value"/></td>
                <td><xsl:value-of select="@square"/></td>
              </tr>
            </xsl:for-each>
          </table>
        </center>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```



# Structuring Your Stylesheets

- Have a common stylesheet to contain your site's standard headers, footers, and nav bars
- If nav bars vary in the site, the common stylesheet should use `<xsl:if>` tags to conditionally create the right nav bars (my recommendation)
- Only have one unnamed `<xsl:template>` tag for the outermost element (my recommendation)
- Have callbacks (named templates) in the common template for key parts of your HTML (my recommendation)



# Questions? Comments?



Sirius Software, Inc.