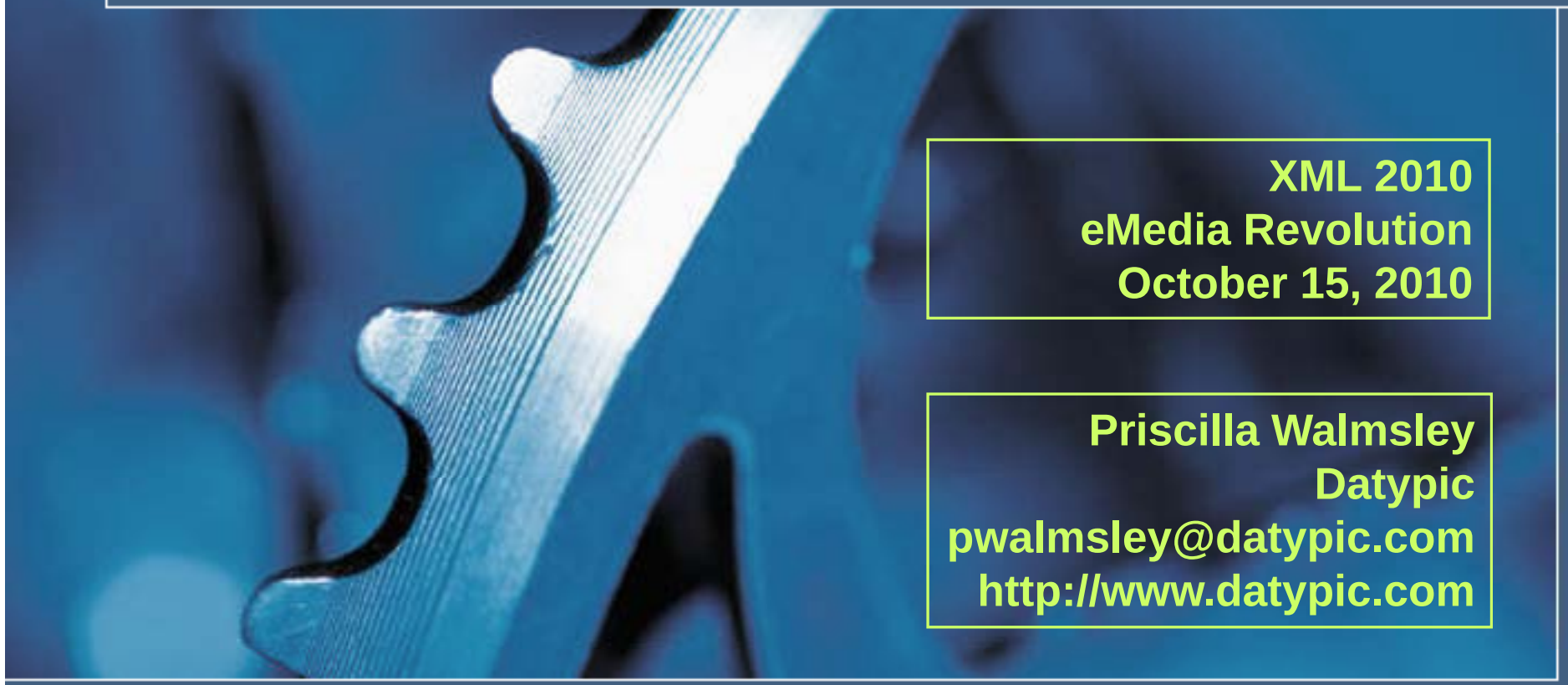
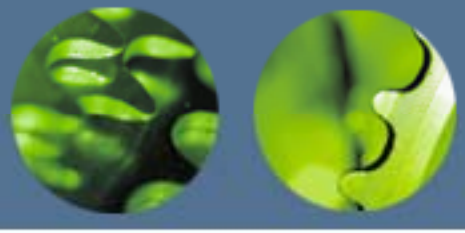
Adding Structure and Semantics with XSLT 2.0



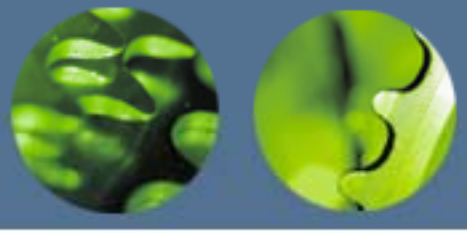
XML 2010
eMedia Revolution
October 15, 2010

Priscilla Walmsley
Datypic
pwalmsley@datypic.com
http://www.datypic.com



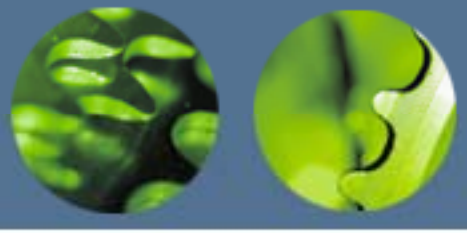
The Scenario

- You have content that is either:
  - almost completely unstructured
  - structured according to its presentation, not its meanin
  - structured according to the desired DTD/Schema, but not granularly enough
- You want it to add structure



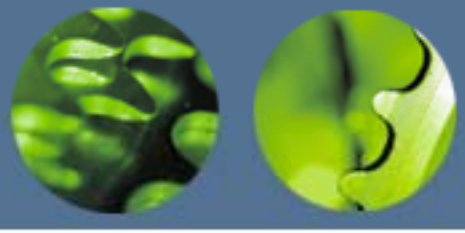
Why Add Structure?

- Enable links
  - links to URLs, email addresses, intra- or inter-document references
- Enable automatic generation of TOCs, indexes, summary views, etc.
- Enable more focused searching
- Ensure more consistent formatting



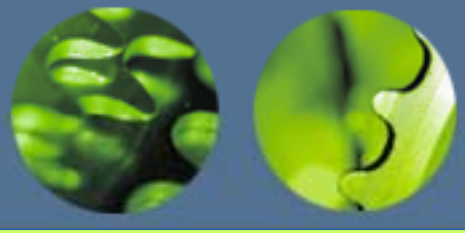
Why XSLT 2.0 for Adding Structure?

- XML-aware
  - Understands XML syntax, encoding, etc.
  - Understands namespaces
- Template processing provides necessary flexibility
  - "Push style" templates allow conversion to be driven by the content
  - Priorities allow templates to be overridden for special cases
- Advanced features (text matching, grouping)



Recognizing Textual Patterns

- XSLT 2.0 has excellent regular expression support
- **`xsl:analyze-string`** element splits string into matching and non-matching parts, based on a regex
  - **`xsl:matching-substring`** child specifies what to do with matching parts
  - **`xsl:non-matching-substring`** child specifies what to do with non-matching parts

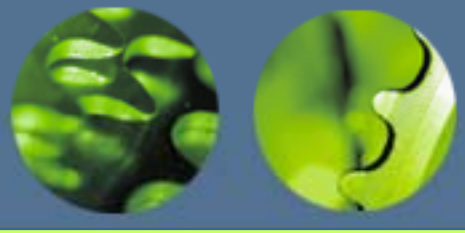


xsl:analyze-string Example

```
<xsl:function name="my:markUpPhone">
 <xsl:param name="theText"/>
 <xsl:analyze-string select="$theText"
                     regex="[0-9]{{3}}/[0-9]{{3}}-[0-9]{{4}}">
  <xsl:matching-substring>
   <xsl:element name="phone">
    <xsl:value-of select="."/>
   </xsl:element>
  </xsl:matching-substring>
  <xsl:non-matching-substring>
   <xsl:copy/>
  </xsl:non-matching-substring>
 </xsl:analyze-string>
</xsl:function>
```

```
can be reached at 231/555-1212 or...
```

```
can be reached at <phone>231/555-1212</phone> or...
```

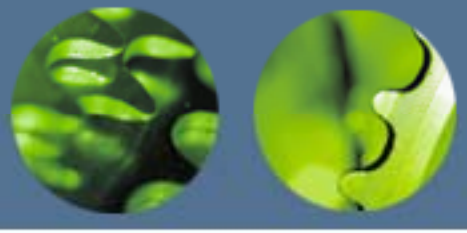


The regex-group Function

```
<xsl:function name="my:markUpPhone">
 <xsl:param name="theText"/>
 <xsl:analyze-string select="$theText"
                     regex="([0-9]{{3}})/([0-9]{{3}}-[0-9]{{4}})">
  <xsl:matching-substring>
   <xsl:element name="phone">
    <areaCode><xsl:value-of select="regex-group(1)"/></areaCode>
    <number><xsl:value-of select="regex-group(2)"/></number>
   </xsl:element>
  </xsl:matching-substring>  ....
```

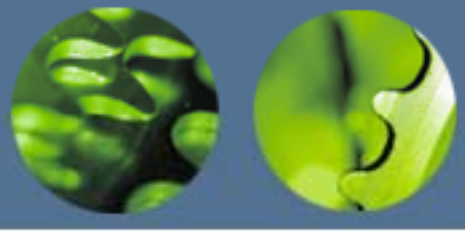
```
can be reached at 231/555-1212 or...
```

```
can be reached at
<phone><areaCode>231</areaCode><number>555-
1212</number></phone> or...
```



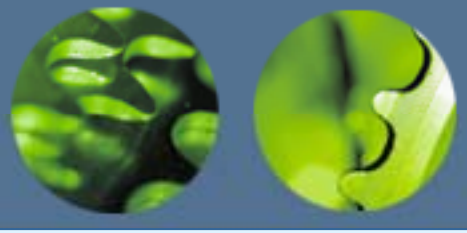
Grouping Sections and Lists

- Grouping features are very useful for adding container structures to content
- **`xsl:for-each-group`** element allows you to iterate through groups
- Two functions can be used within **`for-each-group`**:
  - **`current-group()`** returns members of current group
  - **`current-grouping-key()`** returns the current grouping key



Grouping Options

- **`group-by`**
  - groups based on a shared value
- **`group-adjacent`**
  - groups adjacent items with the same key together
- **`group-starting-with`**
  - creates a group of items starting with the specified element
- **`group-ending-with`**
  - creates a group of items ending with the specified element



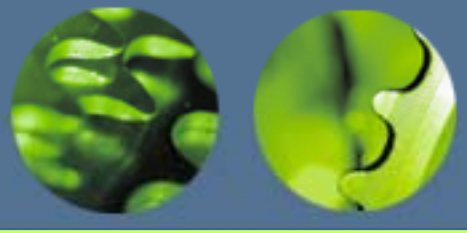
Grouping by Starting Element

```
<body>
 <h1>Chapter 1</h1>
 <h2>Section 1.1</h2>
 <p>In this section...</p>
 <p>More text</p>
 <h2>Section 1.2</h2>
 <p>In this section...</p>
</body>
```

Input document

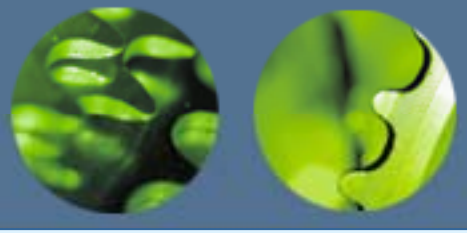
```
<section level="1">
  <heading>Chapter 1</heading>
  <section level="2">
   <heading>Section 1.1</heading>
   <p>In this section...</p>
   <p>More text</p>
  </section>
  <section level="2">
   <heading>Section 1.2</heading>
   <p>In this section...</p>
  </section>
</section>
```

Desired output document



Grouping by Starting Element

```
<xsl:template match="/">
  <xsl:for-each-group select="body/*" group-starting-with="h1">
   <section level="1">
    <xsl:for-each-group select="current-group()"
                        group-starting-with="h2">
    <xsl:choose>
      <xsl:when test="current-group()[self::h2]">
        <section level="2">
          <xsl:apply-templates select="current-group()"/>
        </section>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates select="current-group()"/>
      </xsl:otherwise>
    </xsl:choose>
    </xsl:for-each-group>
   </section>
  </xsl:for-each-group>
</xsl:template>
```



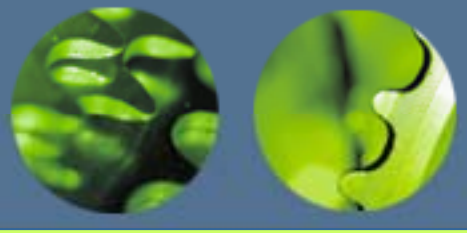
Grouping Adjacent Items

```
<body>
 <p>The following...:</p>
 <p>1. Open the file.</p>
 <p>2. Change it to...</p>
 <p>3. Save the file.</p>
 <p>As you can see...</p>
</body>
```

Input document

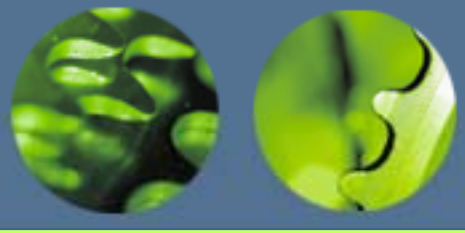
```
<body>
  <p>The following...:</p>
  <ol>
    <li>Open the file.</li>
    <li>Change it to...</li>
    <li>Save the file.</li>
  </ol>
  <p>As you can see...</p>
</body>
```

Desired output document



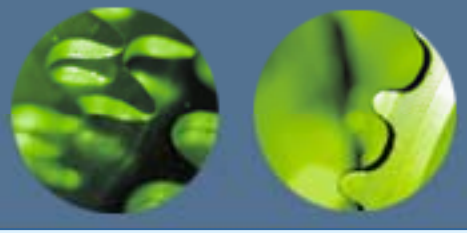
Grouping Adjacent Items

```
<xsl:template match="body">
  <body>
    <xsl:for-each-group select="*"
                        group-adjacent="my:is-a-list-item(.)">
      <xsl:choose>
        <xsl:when test="current-grouping-key() = true()">
          <ul>
            <xsl:for-each select="current-group()">
              <li><xsl:apply-templates/></li>
            </xsl:for-each>
          </ul>
        </xsl:when>
        <xsl:otherwise>
          <xsl:copy-of select="."/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each-group>
  </body>
</xsl:template>
```



Grouping Adjacent Items (cont.)

```
<xsl:template match="text()">
  <xsl:choose>
    <xsl:when test="my:is-a-list-item(parent::p)
                    and my:is-a-list-item(.)
                    and (. is parent::p/node()[1])">
      <xsl:value-of select="replace(.,'^\s*\d+\.\s*','')"/>
    </xsl:when>
    <xsl:otherwise>
        <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:function name="my:is-a-list-item">
  <xsl:param name="node"/>
  <xsl:sequence select="matches($node,'^\s*\d+\.')"/>
</xsl:function>
```



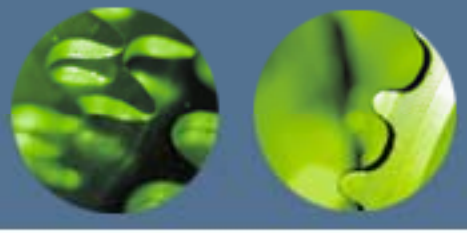
Inferring Structure from Section Numbers

```
<body>
 <p>Chapter 1</p>
 <p>1.1 Introduction</p>
 <p>In this section...</p>
 <p>More text</p>
 <p>1.2 Next Steps</p>
 <p>In this section...</p>
</body>
```

Input document

```
<chapter>
  <title>Chapter 1</title>
  <section level="1">
   <h1>1.1 Introduction</h1>
   <p>In this section...</p>
   <p>More text</p>
  </section>
  <section level="1">
   <h1>1.2 Next Steps</h1>
   <p>In this section...</p>
  </section>
</section>
```

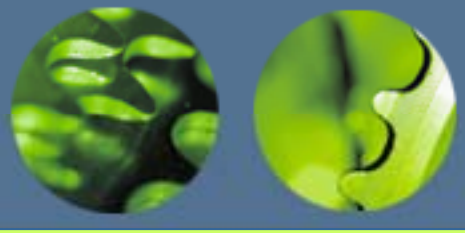
Desired output document



Inferring Structure from Section Numbers

- Use two steps:
  - Convert the p's to their desired names
  - Group based on the element names, as in the first example

```
<xsl:template match="body">
  <xsl:variable name="renamed" as="element()*">
    <xsl:apply-templates mode="rename"/>
  </xsl:variable>
  <body>
    <xsl:for-each-group select="$renamed"
                        group-starting-with="title">
      <!-- same logic as first grouping example -->
    </xsl:for-each-group>
  </body>
</xsl:template>
```



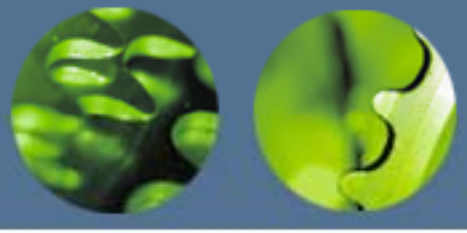
Inferring Structure from Section Numbers

```
<xsl:template match="p" mode="rename">
  <xsl:choose>
    <xsl:when test="my:is-a-chapter(.)">
     <title>
      <xsl:apply-templates/>
     </title>
    </xsl:when>
    <xsl:when test="my:is-a-level-1(.)">
     <h1>
      <xsl:apply-templates/>
     </h1>
    </xsl:when>
    <xsl:otherwise>
     <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

```
<body>
 <p>Chapter 1</p>
 <p>1.1 Introduction</p>
 <p>In this section...</p>
 <p>More text</p>
 <p>1.2 Next Steps</p>
 <p>In this section...</p>
</body>
```

```
<body>
 <title>Chapter 1</title>
 <h1>1.1 Introduction</h1>
 <p>In this section...</p>
 <p>More text</p>
 <h1>1.2 Next Steps</h1>
 <p>In this section...</p>
</body>
```

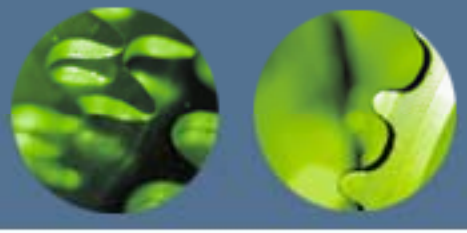
Intermediate Format



Using Style Information

- Named styles are usually your most important clue regarding what something means
- Ideally, you have named styles to work with
  - Word paragraph and character styles
  - HTML div and span classes
  - etc.

Turning Word Styles into Element Names



Style/Element Mapping Document

```
<styles>
  <style>
    <name>Heading2</name>
    <transformTo>h2</transformTo>
  </style>...
```

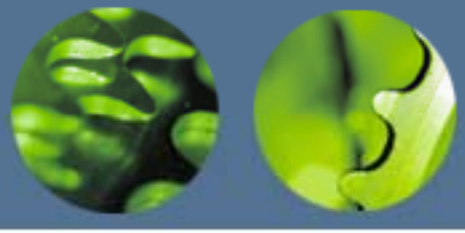
```
<w:p wsp:rsidR="005103A7"
wsp:rsidRPr="005103A7"
wsp:rsidRDefault="005103A7"
wsp:rsidP="000B6D1E"><w:pPr><w:pSt
yle w:val="Heading2"/><w:spacing
w:line="240" w:line-
rule="auto"/><w:ind
w:left="0"/><w:rPr><w:b/><w:b-
cs/><w:color w:val="808080"/><w:sz
w:val="24"/><w:sz-cs
w:val="24"/></w:rPr></w:pPr><w:r
wsp:rsidRPr="005103A7"><w:rPr><w:b
/><w:b-cs/><w:color
w:val="808080"/><w:sz
w:val="24"/><w:sz-cs
w:val="24"/></w:rPr><w:t>Introduct
ion</w:t></w:r></w:p>
```

Word document

XSLT

```
<h2>Introduction</h2>
```

Simplified intermediate document

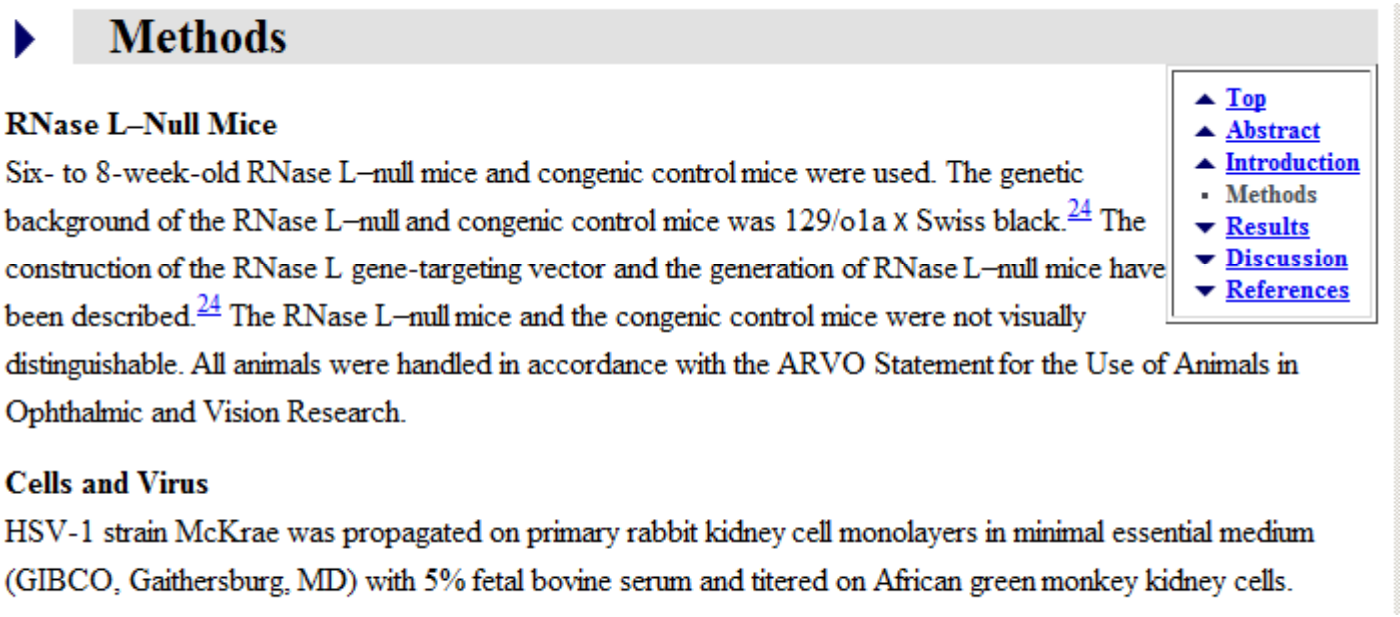


Turning Word Styles into Element Names

```
<xsl:template match="w:p">
  <xsl:variable name="newElName">
    <xsl:sequence select="doc('stylemap.xml')/styles
      /style[name=current()/w:pPr/w:pStyle/@w:val]/transformTo"/>
  </xsl:variable>
  <xsl:element name="{if ($newElName != '')
                      then $newElName
                      else 'p'}">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

Using Formatting Information

- In the absence of named styles, formatting information provides important clues about the structure of content



Methods

- Top
- Abstract
- Introduction
- Methods
- Results
- Discussion
- References

RNase L–Null Mice

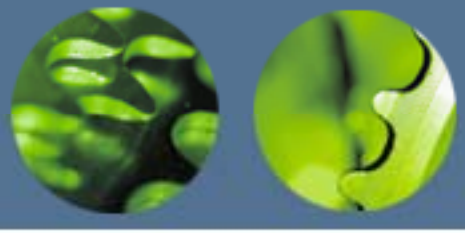
Six- to 8-week-old RNase L–null mice and congenic control mice were used. The genetic background of the RNase L–null and congenic control mice was 129/o1a x Swiss black.[24] The construction of the RNase L gene-targeting vector and the generation of RNase L–null mice have been described.[24] The RNase L–null mice and the congenic control mice were not visually distinguishable. All animals were handled in accordance with the ARVO Statement for the Use of Animals in Ophthalmic and Vision Research.

Cells and Virus

HSV-1 strain McKrae was propagated on primary rabbit kidney cell monolayers in minimal essential medium (GIBCO, Gaithersburg, MD) with 5% fetal bovine serum and titered on African green monkey kidney cells.

Using Formatting Information

```
<table width="100%" bgcolor="#e1e1e1"><tr><th valign="middle"
width="95%" align="left"><font
size="+2">Methods</font></th></tr></table>

<p><strong>RNase L-Null Mice</strong><br/>

Six- to 8-week-old RNase L-null mice and congenic control mice
were used. The genetic background of the RNase L-null and congenic
control mice was 129/o1a <font face="ariel,helvetica">x</font> Swiss
black.<sup><a href="#B24">24</a></sup> The construction...</p>

<p><strong>Cells and Virus</strong><br/>

HSV-1 strain McKrae was propagated on primary rabbit kidney cell
monolayers in minimal essential medium (GIBCO, Gaithersburg, MD) with
5% fetal bovine serum and titered on African green monkey kidney
cells. </p>
```

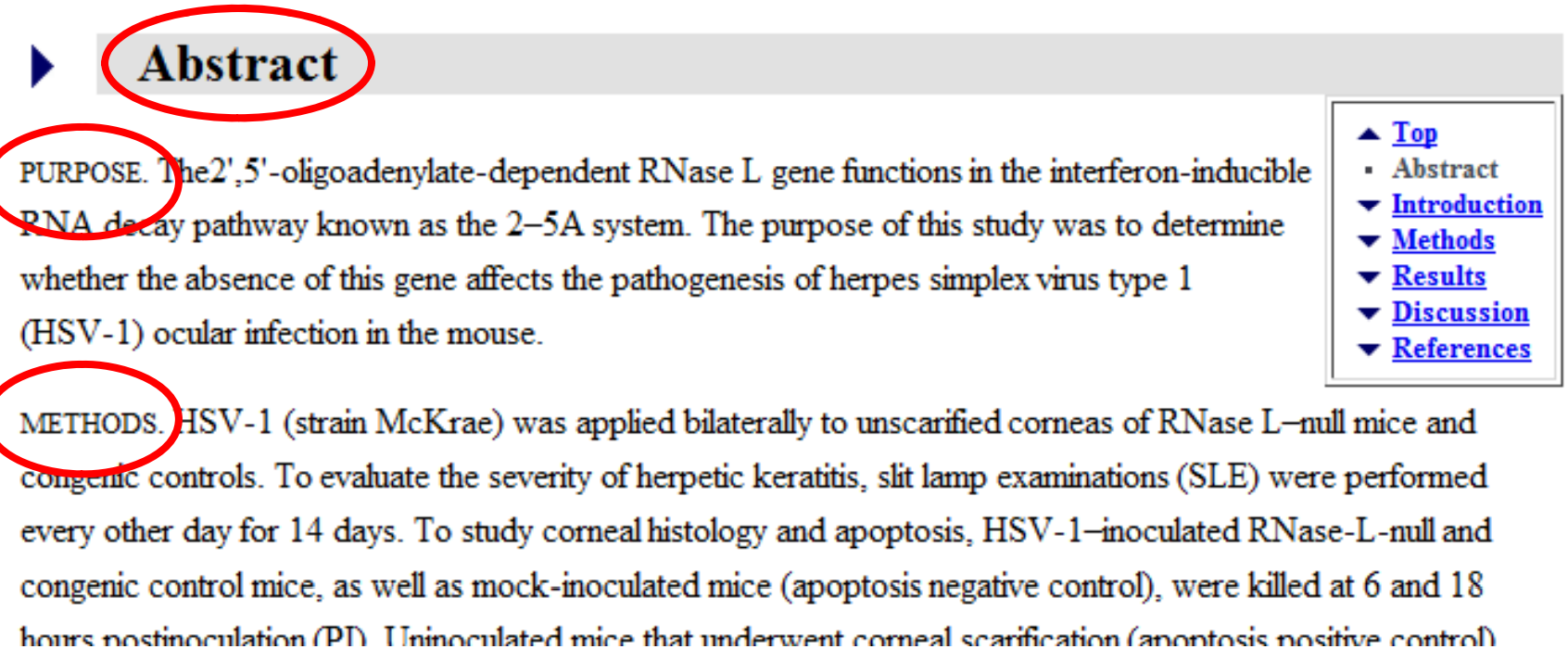


Using Formatting Information

```
<xsl:template
      match="table[@bgcolor='#e1e1e1'][tr/th/font[@size='+2']]">
  <h1>
     <xsl:apply-templates/>
  </h1>
</xsl:template>

<xsl:template match="p">
  <xsl:variable name="firstBr" select="br[1]"/>
  <h2>
    <xsl:apply-templates select="node()[. &lt;&lt; $firstBr]"/>
  </h2>
  <p>
    <xsl:apply-templates select="node()[. >> $firstBr]"/>
  </p>
</xsl:template>
```

Identifying Semantics

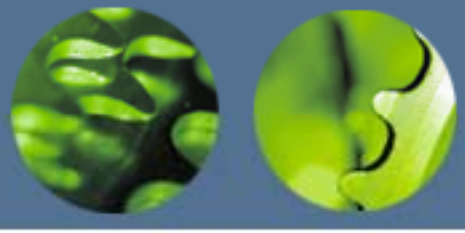
- Formatting combined with content can help identify semantics



Abstract

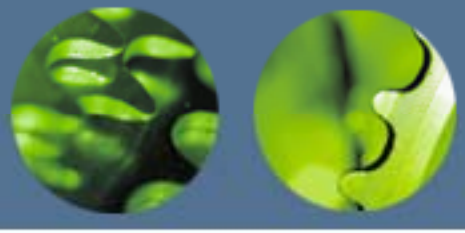
PURPOSE. The2',5'-oligoadenylate-dependent RNase L gene functions in the interferon-inducible RNA decay pathway known as the 2–5A system. The purpose of this study was to determine whether the absence of this gene affects the pathogenesis of herpes simplex virus type 1 (HSV-1) ocular infection in the mouse.

METHODS. HSV-1 (strain McKrae) was applied bilaterally to unscarified corneas of RNase L–null mice and congenic controls. To evaluate the severity of herpetic keratitis, slit lamp examinations (SLE) were performed every other day for 14 days. To study corneal histology and apoptosis, HSV-1–inoculated RNase-L-null and congenic control mice, as well as mock-inoculated mice (apoptosis negative control), were killed at 6 and 18 hours postinoculation (PI). Uninoculated mice that underwent corneal scarification (apoptosis positive control)



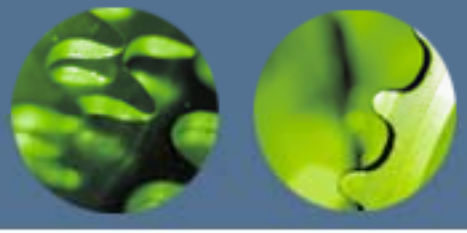
Identifying Semantics

```
<body><table width="100%" bgcolor="#e1e1e1" cellpadding="0"
cellspacing="0"><tr><th valign="middle" width="95%" align="left"><font
size="+2">Abstract
</font></th></tr></table>

<p><font size="-1">PURPOSE.</font> The2',5'-oligoadenylate-dependent
RNase L gene functions in the
interferon-inducible RNA decay pathway known as the ...</p><p>

<font size="-1">METHODS.</font> HSV-1 (strain McKrae) was applied
bilaterally to unscarified corneas of
RNase L-null mice and congenic controls. To evaluate the severity of
herpetic keratitis, slit lamp examinations (SLE) were performed every
other day for 14 days. To study corneal histology and apoptosis,
HSV-1-inoculated RNase-L-null and congenic control mice, as well as
mock-inoculated mice (apoptosis negative control),.... </p>
```



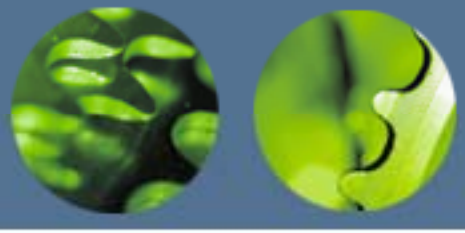
Identifying Semantics

```
<xsl:template match="p">
  <xsl:choose>
    <xsl:when test="matches(font[@size='-1'],'^\s*PURPOSE[.\s]*$')">
      <sec type="objectives">
        <xsl:apply-templates/>
      </sec>
    </xsl:when>
    <xsl:when test="matches(font[@size='-1'],'^\s*METHODS[.\s]*$')">
      <sec type="methods">
        <xsl:apply-templates/>
      </sec>
    </xsl:when>
    <xsl:otherwise>
      <p><xsl:apply-templates/></p>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```



Caveats

- Be careful not to inadvertently flatten mixed content
- Don't assume consistency
  - test your assumptions in your code frequently and write warning messages
- Beware of unexpected whitespace
- Human review is always recommended



Conclusions

- XSLT 2.0 has many useful features for adding structure to content
  - string matching
  - grouping
  - lookup/mapping documents
  - ability to make multiple passes via modes
  - positional testing using << and >>
- Questions?