Published in Code4Lib Journal, Issue 14, 2011, ISSN 1940-5758
http://journal.code4lib.org/articles/5732

WEB-BASED SOFTWARE INTEGRATION FOR DISSEMINATION OF ARCHIVAL IMAGES: THE FRONTIERS OF SCIENCE WEBSITE

Gary Browne,Development Programmer,

Library IT Services and Sydney eScholarship,

University of Sydney Library,University of Sydney,

Camperdown NSW 2006AUSTRALIA

Abstract

The Frontiers of Science illustrated comic strip of 'science fact' ran from 1961 to 1982, syndicated worldwide through over 600 newspapers. The Rare Books and Special Collections Library at the University of Sydney in association with Sydney eScholarship, digitized all 939 strips. We aimed to create a website that could disseminate these comic strips to scholars, enthusiasts and the general public. We wanted to enable users to search and browse through the images simply and effectively, with an intuitive and novel viewing platform.

Time and resource constraints dictated the use of (mostly open source) code modules wherever possible and the integration and customisation of a range of web-based applications, code snippets and technologies (DSpace, eXtensible Text Framework (XTF), OmniFormat, JQuery Tools, Thickbox and Zoomify), stylistically pulled together using CSS. This approach allowed for a rapid development cycle (6 weeks) to deliver the site on time as well as provide us with a framework for similar projects.

INTRODUCTION

The Rare Books and Special Collections Library at the University of Sydney was bequeathed the rights to the original strips, or "pulls", of the Frontiers of Science comic strips which were published in over 600 newspapers worldwide from 1961 to 1982. The series was co-written and produced by Professor Stuart Butler from the School of Physics at the University of Sydney and journalist and film-maker Bob Raymond. The early art work in the series was by Andrea Bresciani, continued later by David Emerson, with the aim of disseminating information about current topics in science in a novel and entertaining way.

Having been given permission to digitize the original strips, Sydney eScholarship at the University of Sydney sought to make the Frontiers of Science strips available once again, this time online, for wider access by scholars and enthusiasts. Our goal was to offer simple, robust, flexible discovery tools including search, facetted browse and a tag cloud, combined with a novel display mechanism. There was an upcoming exhibition of Frontiers paraphernalia at the Australian Broadcasting Commission (ABC) in Ultimo, Sydney which was to coincide with Science Week. We wanted the Frontiers of Science website to be launched to coincide with this, meaning the deadline at the time was 6 weeks away.

Many tasks needed prioritization and consideration within the short time frame. These included but were not limited to: importation of the digitized pulls to the institutional repository, metadata tagging of the items, processing of images (for thumbnails and for use with the image viewer), processing of metadata, indexing of image metadata for browsing and searching functionality, creating the browse and search front-end, tag cloud generation, deciding upon and implementing an image viewer, presentation and layout design (within University style constraints), and developing the site, including associated historical and descriptive content and complying with (as far as practicable) W3C standards. To expedite the process, it was decided to rely heavily on technologies that we were already familiar with, or that could be quickly integrated into existing (open source) architecture.

PREPARING THE IMAGES AND METADATA

SCANNING OF COMIC STRIPS

The original Frontiers of Science material was bequeathed to the Rare Books and Special Collections Library of the University of Sydney by Angela Raymond and Miriam Butler. Here, they were scanned in black and white on a Bookeye scanner (BE3-SCL-R1) at 600dpi by Nicholas Heath in January of 2009. Each resulting archival TIFF file was about 400kB in size.

CONTENT ARCHIVING

The Sydney eScholarship Repository, the University's institutional research repository, was used to archive the original scanned TIFF images. The repository employs a customized instance of the DSpace open source digital library software, which provides by default a facility for tagging items with qualified Dublin Core metadata. Most collections in the repository are tagged using the Dublin Core schema, so we decided to use this metadata schema to describe the Frontiers images.Tagging of the comic strips within the Sydney eScholarship Repository was completed by Professor Peter Harrowell from the School of Chemistry, University of Sydney. Due to time constraints, no controlled vocabulary was applied. So the metadata was less than ideal for data interchange purposes, but sufficient for our requirements of browsing and searching. Initially, about 200 pulls (each with one week's worth of comic strips) were tagged with keywords, with the aim of adding more tagged strips over time, following the launch of the site.

METADATA PROCESSING

In order to retrieve metadata from DSpace in a format useful for the search and browse system, the eXtensible Text Framework (XTF), a Java class was written (XTFItemExport.java) for DSpace, which accomplished the following:

- Created the Dublin Core metadata XML files with the necessary metadata fields
- Formatted the XML files for use with the default XTF metadata "prefilter"
- Named the XML files based on dc.identifier.other

We decided that it would be simpler and more efficient to do all three tasks in one Java class, rather than creating and naming the files this way and then customizing the XTF prefilter to suit the XML file format.

The main `writeMetadata` method of this class (Code Listing 1) sets up the XML file for XTF by defining the root element as `<dc>`, the XTF default. Within the `<dc>` tags, we tested only for the following metadata elements: title, date, identifier and subject.

```
// output the item's dublin core into the item directory

    private static void writeMetadata(Context c, String schema, Item i, File destDir)

            throws Exception

    {

        // id to rename file with

        String id="";        String filename;

         if (schema.equals(MetadataSchema.DC_SCHEMA)) {

             filename = "dc.xml";

         } else {

             filename = "metadata_" + schema + ".xml";

         }


         File outFile = new File(destDir, filename);
```

```
        System.out.println("Attempting to create file " + outFile);


        if (outFile.createNewFile())
        {
            BufferedOutputStream out = new BufferedOutputStream(
                    new FileOutputStream(outFile));


            DCValue[] dcorevalues = i.getMetadata(schema, Item.ANY, Item.ANY, Item.ANY);


            // XML preamble
              byte[] utf8 = "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n"
                    .getBytes("UTF-8");
            out.write(utf8, 0, utf8.length);


              // "dc" instead of "dublin_core"
            String dcTag = "<dc>\n";
            utf8 = dcTag.getBytes("UTF-8");
            out.write(utf8, 0, utf8.length);


            for (int j = 0; j < dcorevalues.length; j++)
            {
                DCValue dcv = dcorevalues[j];
                String qualifier = dcv.qualifier;
               // get element
               String element = dcv.element;


               // test only for elements we want
               if(element.equals("title")){
                       utf8 = ("  <title>" + Utils.addEntities(dcv.value) +
"</title>\n").getBytes("UTF-8");
                       out.write(utf8, 0, utf8.length);
               }
               else if(element.equals("date") && qualifier.equals("issued")){
                       utf8 = ("  <date>" + Utils.addEntities(dcv.value) +
"</date>\n").getBytes("UTF-8");
                       out.write(utf8, 0, utf8.length);
```

```
                }
                else if(element.equals("identifier") && qualifier.equals("other")){
                        utf8 = ("  <identifier>" + Utils.addEntities(dcv.value) +
"</identifier>\n").getBytes("UTF-8");
                        out.write(utf8, 0, utf8.length);
                        id = dcv.value;
                        System.out.println("WROTE ID: " + id);

                }
                else if(element.equals("subject")){
                        utf8 = ("  <subject>" + Utils.addEntities(dcv.value) +
"</subject>\n").getBytes("UTF-8");
                        out.write(utf8, 0, utf8.length);

                }
                else{
                        System.out.println("NO MATCHING ELEMENT FOUND");
                }

            }

            // close "dc" tag
            utf8 = "</dc>\n".getBytes("UTF-8");
            out.write(utf8, 0, utf8.length);

            out.close();

            // rename file - prepend identifier
            File idOutFile = new File(destDir, id + ".dc.xml");
            outFile.renameTo(idOutFile);
        }
        else
        {
            throw new Exception("Cannot create dublin_core.xml in " + destDir);
        }
    }
```

Code Listing 1. The writeMetadata method to setup XTF-ready Dublin Core files

The output of this program was one directory for each image. Each directory contained the XTF-formatted Dublin Core metadata file (named $dc.identifier.other.dc.xml where $dc.identifier.other is the issue week) and the "pull" image plus other accessory files. This directory structure is the standard DSpace submission information package (SIP) configuration. The ".dc.xml" extension is recognized by XTF as a Dublin Core metadata file.

When loaded into DSpace, the TIFF file names were disparate. Some were simply numeric ("064.TIF") and others had varying prefixes ("FoS" or "FoS_"). To rectify this, another Java program was written to standardize the naming of the TIFF files in line with the metadata files i.e. $dc.identifier.other.tif, where $dc.identifier.other is the issue week. The main method (Code Listing 2) which retrieves the identifier from the Dublin Core XML file uses the Java API for XML Processing (JAXP) [1].

```
    /**
      * Reads the Frontiers of Science identifier from an image's Dublin
      * Core metadata file (exported from DSpace using ItemExporter)
      *
      * @param file - the file from which to read
      * @return FOSID - the Frontiers of Science image ID
      */
     public String readDCFileFOSIdentifierValue(File file){

        // the FOS identifier value
        String FOSID="";

        try {

               DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
                 domFactory.setNamespaceAware(true); // never forget this!
                 DocumentBuilder builder = domFactory.newDocumentBuilder();
                 Document doc = builder.parse(file);

                 XPathFactory factory = XPathFactory.newInstance();
                 XPath xpath = factory.newXPath();
                 XPathExpression expr
                  = xpath.compile("//dcvalue[@element='identifier' and
@qualifier='other']/text()");
```

```
            Object result = expr.evaluate(doc, XPathConstants.STRING);
            FOSID = result.toString();


      }
      catch (SAXParseException err) {
       System.out.println ("** Parsing error" + ", line "
            + err.getLineNumber () + ", uri " + err.getSystemId ());
       System.out.println(" " + err.getMessage ());


       }catch (SAXException e) {
       Exception x = e.getException ();
       ((x == null) ? e : x).printStackTrace ();


       }catch (Throwable t) {
       t.printStackTrace ();
       }
            return FOSID;
  }
```

Code Listing 2. Main method to retrieve identifier from Dublin Core XML files

The format of a typical dc.xml file is shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<dc>
  <date>1963-03-18</date>
  <identifier>79</identifier>
  <subject>Physics</subject>
  <subject>Space</subject>
  <subject>Lunar surface</subject>
  <subject>Tom Gold</subject>
  <subject>powder oceans</subject>
  <subject>Tycho crater</subject>
  <subject>moon landing</subject>
  <title>The surface hazard</title>
</dc>
```

This naming convention and metadata file format standardization would assist in further processing and dynamic rendering of the images, as well as indexing for the search and browse interface.

IMAGE PROCESSING

CREATING SCROLLABLE PREVIEW IMAGES

In keeping with the format of the comic strips, we aimed to make the style of the website very visual and high contrast. The facetted browse interface and search results would allow users to scroll through preview images in an image slider. These previews were created by processing the images output from the DSpace export using the OmniFormat (free) document conversion utility. JPEGs at 200px x 400px were created for scrolling in the image slider (Figure 1) and we were very happy with the image quality.

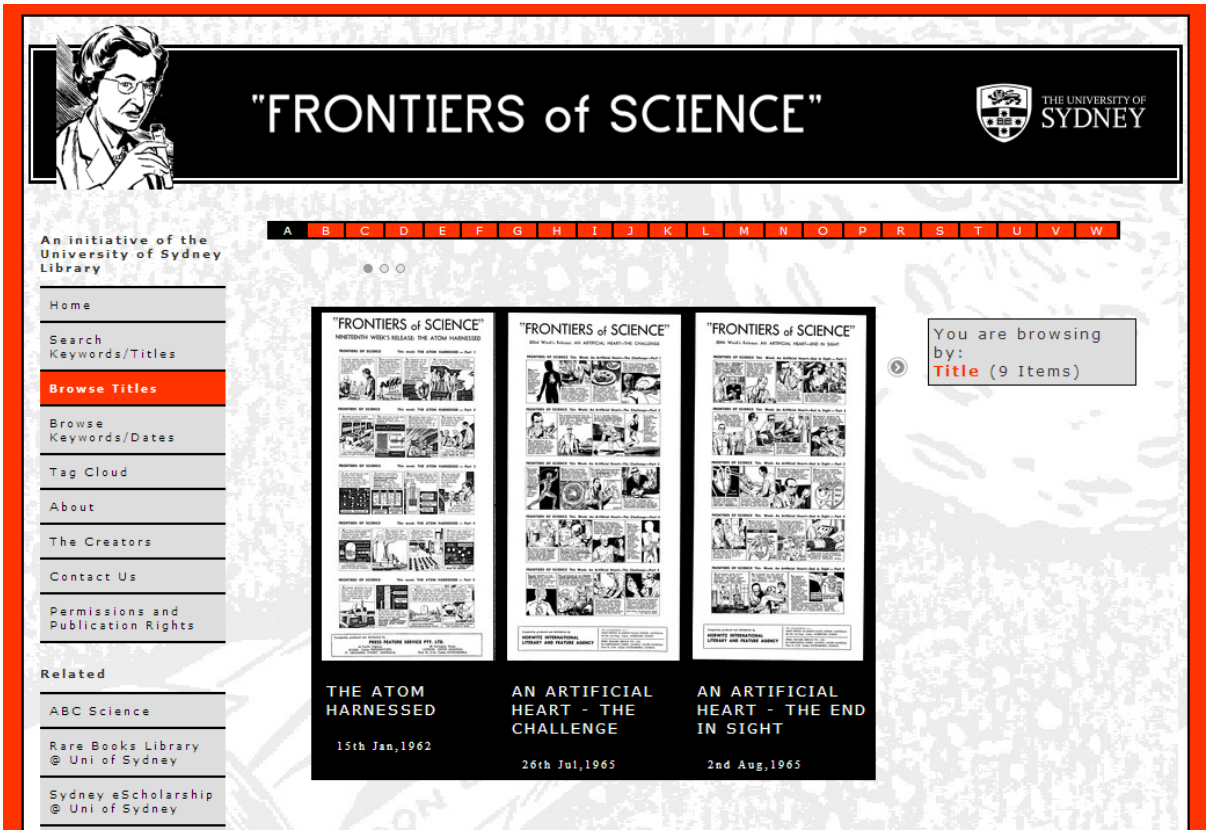

Figure 1. Scrollable preview images for browsing

CREATING TILES FOR ZOOMIFY DESIGN

Zoomify is commercial software enabling zooming and panning of high-resolution images. The decision to purchase a commercial licence for ZoomifyDesign4-Win rather than code an image

zoomer and panner in, for example, jQuery was made due to time and resource constraints. Technically, this worked well in terms of development time. However, due to Zoomify being Flash-based, it meant that the site would not work on iPads or iPhones. This was an oversight at the time of development and the change over to jQuery for zooming could be a possible future site enhancement.

Zoomify requires the creation of image “tiles”, which were created by dragging and dropping images onto the Zoomify Converter.exe program icon in Windows. This processing produced one directory corresponding to each TIFF image (below), consisting of subdirectories of “tile groups”, collections of small JPEGs which are small portions of the original image.

```
1

|-- ImageProperties.xml

|-- TileGroup0

|   |-- 0-0-0.jpg

|   |-- 1-0-0.jpg

|   |-- 1-0-1.jpg

.

.

.

|-- TileGroup1

|   |-- 5-0-10.jpg

|   |-- 5-0-11.jpg

|   |-- 5-0-12.jpg

.

.

.

2

|-- ImageProperties.xml

|-- TileGroup0

|   |-- 0-0-0.jpg

|   |-- 1-0-0.jpg

|   |-- 1-0-1.jpg

|   |-- 2-0-0.jpg

|   |-- 2-0-1.jpg
```

These tile group directories were then ready to be presented using the Zoomify viewer, a Shockwave Flash file which is bundled with Zoomify Design.

BUILDING THE SITE

THE SEARCH AND BROWSE PLATFORM

At the University of Sydney Library we have used the eXtensible Text Framework (XTF) for several projects over the last few years and have found it to be flexible enough to meet our needs for developing customized search and facetted browse functionality for a variety of content types and metadata schemas. Having some expertise in creating XTF sites, we decided to use this open source platform for the search and browse functions of the Frontiers of Science site.

Given the time and resource constraints, we wanted to keep things simple, so we pared down the full-featured XTF advanced search engine to allow searching only on keyword or title. We used the term "keyword" since the subject metadata values were not controlled. Figure 2 shows the simple keyword/title search forms page. Clear examples of available Boolean operations were provided.

Figure 2. Keyword and title search page showing sample searches

XTF comes with a highly configurable text indexing facility which we used to generate a Lucene index from the Dublin Core files. The Lucene index is read for searching by the "crossQuery" servlet. This is also highly configurable through XSL files and templates.

The search engine produces a results page which is powered by XTF but which integrates the use of the JQuery Tools' Scrollable module. Figure 3 shows the salient features of this search results page and the scrollable module. Firstly, the search itself is displayed on the right hand sidebar and the number of matches is shown. The results can be sorted by title or date. Through XTF's facetted browsing feature, users are able to narrow their search results further. Once again, we decided to keep it simple, only defining "date" and "keywords" as facetted browse fields.

Figure 3. Main features of the search results page.

The display of the actual results is visual, showing the JPEG files in a scrollable component which may be navigated either by clicking the arrows beside the component, or the circles above (thus, users can jump to any part of the list of results, which is particularly useful when there are a large number of matches). Each JPEG has a title and publication date associated with it. This display uses the JQuery Tools library, included in the document <head>:

```
<script src="script/jquery/jquery.tools.min.js" type="text/javascript"/>
```

Also included are two CSS files for the styling of the module:

```
<link rel="stylesheet" type="text/css" href="css/default/scroll.css"/>
<link rel="stylesheet" type="text/css" href="css/default/scrollable-navig.css"/>
```

XTF uses XSL stylesheets to control and configure the Java classes which drive it. The main scrollable module is defined within an XSL template in $XTF_HOME/style/crossQuery/resultFormatter/default/resultFormatter.xsl, which is called on the condition that a search returns at least one match (“docHit” in Code Listing 3).

```
<xsl:if test="docHit">
      <!-- scrolling images with zoomable image overlaid onclick -->
      <xsl:call-template name="scrollable">
         <xsl:with-param name="browseOrSearch" select="docHit"/>
      </xsl:call-template>
.
.
.
</xsl:if>
```

Code Listing 3. The test for search term matches.

The “scrollable” XTF XSL template defines container classes and ids such as “scrollable”, “thumbs”, “thumb” and “meta”, the styles of which are set in the two CSS files, “scroll.css” and “scrollable-navig.css”. The collection of pulls included some that had not been previously published and, therefore, had no publication date. A workaround was put in place for this - if the pull did not have a date greater than zero, instead of displaying a formatted date, the text "Unpublished" was displayed (see “unpublished material hack” in Code Listing 4).

```
<xsl:template name="scrollable" exclude-result-prefixes="#all">
  <xsl:param name="browseOrSearch"/>
  <!-- navigator -->
  <div class="navi"></div>
  <!-- prev link -->
  <a class="prevPage"></a>
  <!-- root element -->
  <div class="scrollable">
    <!-- container for items -->
    <div id="thumbs">
      <!-- loop through browse/search results, add each to scroller -->
      <xsl:for-each select="$browseOrSearch">
        <xsl:variable name="id" select="meta/identifier[1]"/>
        <xsl:variable name="title" select="meta/title[1]"/>
        <!-- unpublished material hack -->
```

```
          <xsl:variable name="published" select="if(number(substring(meta/date[1],1,4)) > 0)
then format-date(meta/date[1], '[D1o] [MNn,*-3],[Y]') else 'Unpublished'"/>

          <!-- setup the scrollable "thumbnails" -->

          <div>

            <div id="thumb">

                <a class="thickbox" href="#TB_inline?height=700&amp;
       width=1000&amp;inlineId={$id}">

                  <img src="img/jpgs/{$id}.jpg" alt="{$title}" title="{$title}" />

               </a>

            </div> <!-- end thumb div -->

            <div id="meta">

              <a class="thickbox" href="#TB_inline?height=700px&amp;
width=1000&amp;inlineId={$id}">

                <h4><xsl:value-of select="$title"/></h4>

              </a>

              <p><xsl:value-of select="$published"/></p>

            </div>

            <xsl:call-template name="overlay">

              <xsl:with-param name="id" select="$id"/>

            </xsl:call-template>

          </div>

        </xsl:for-each>

      </div><!--  end thumbs div -->

    </div><!--  end scrollable div -->

    <!-- next link -->

    <a class="nextPage"></a>

               <script type="text/javascript">

               // only execute scripts when DOM is ready

               $(function() {


                       // initialize scrollable

                       $("div.scrollable").scrollable({

                               size: 3,

                               items: '#thumbs',

                               hoverClass: 'hover'

                       });
```

```
                });

                </script>

        </xsl:template>
```

Code Listing 4. The "scrollable" XSL template

Within each "thumb" and "meta" , a hyperlink is setup which has a class of "thickbox" and passes some parameters in the URL, most notably the $id of the pull to show, as follows:

```
<a class="thickbox" href="#TB_inline?height=700&amp; width=1000&amp;inlineId={$id}">

<img src="img/jpgs/{$id}.jpg" alt="{$title}" title="{$title}" /></a>
```

The $id is, in all cases, standardized and represents:

- The issue number, or publication week
- The prefix of the *.dc.xml metadata file
- The prefix of the *.jpg thumbnail file
- The prefix of the Zoomify image tile directory

Once a user wishes to view a particular pull in detail, they may click on the JPEG (or its title). An overlay with zoom and pan controls then allows the user to view each week's comic strip in detail.

THE VIEWING PLATFORM

Rather than click through to a standalone page for viewing the pulls in Zoomify, we decided to integrate Zoomify with XTF through the use of an overlay. We chose ThickBox for creating the overlays, primarily for its ease of use (both from a user and developer perspective). The following Javascript and CSS files were required for the thickbox component:

```
                <!-- thickbox -->

                <script type="text/javascript" src="script/thickbox/jquery.js"></script>

                <script type="text/javascript" src="script/thickbox/thickbox.js"></script>

                <link rel="stylesheet" href="css/default/thickbox.css" type="text/css"
media="screen" />

                <link rel="stylesheet" href="css/default/overlay.css" type="text/css"
media="screen" />
```

When the pull thumbnail or title is clicked in the scrollable component (from browse or search results), the XTF overlay XSL template (Code Listing 5) is called to display the zoomable and pannable pull.

```
    <!-- ============================================================ -->

   <!-- Overlay Template                                             -->
```

```
   <!-- ============================================================= -->

        <xsl:template name="overlay">

                <xsl:param name="id"/>

                <div class="overlay" id="{$id}">

                        <p>

                        <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=6,0,40,0
" width="1000" height="750" id="theMovie">

                        <!-- this works for IE -->

                        <param name="FlashVars"
value="zoomifyToolbarSkinXMLPath={$zoomifyToolbarSkinXMLPath}&amp;zoomifyImagePath={$zoomPath}
{$id}&amp;zoomifyNavigatorVisible=1&amp;zoomifyInitialZoom=10&amp;zoomifySplashScreen=0&amp;zo
omifyToolbarLogo=0&amp;zoomifyToolbarTooltips=1&amp;zoomifyNavigatorHeight=230&amp;zoomifyInit
ialX=0&amp;zoomifyInitialY=0&amp;zoomifyNavigatorX=870&amp;zoomifyNavigatorY=460" />

                        <param name="menu" value="false" />

                        <param name="src" value="{$zoomifyViewerPath}" />

                        <!-- this works for Firefox -->

                        <embed
FlashVars="zoomifyToolbarSkinXMLPath={$zoomifyToolbarSkinXMLPath}&amp;zoomifyImagePath={$zoomP
ath}{$id}&amp;zoomifyZoom=180&amp;zoomifyNavigatorVisible=1&amp;zoomifyInitialZoom=10&amp;zoom
ifySplashScreen=0&amp;zoomifyToolbarLogo=0&amp;zoomifyNavigatorHeight=230&amp;zoomifyInitialX=
0&amp;zoomifyInitialY=0&amp;zoomifyNavigatorX=870&amp;zoomifyNavigatorY=500"
src="{$zoomifyViewerPath}" menu="false"
pluginspage="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveF
lash"  width="1000" height="750" name="theMovie"></embed>

                        </object>

                        </p>

                </div>

        </xsl:template>
```

Code Listing 5. The "overlay" XSL template

Figure 4 shows the zoomed pull. The zoom and pan controls are shown at the bottom centre (green border), and the navigation window is shown at the bottom right of the overlay (yellow border).

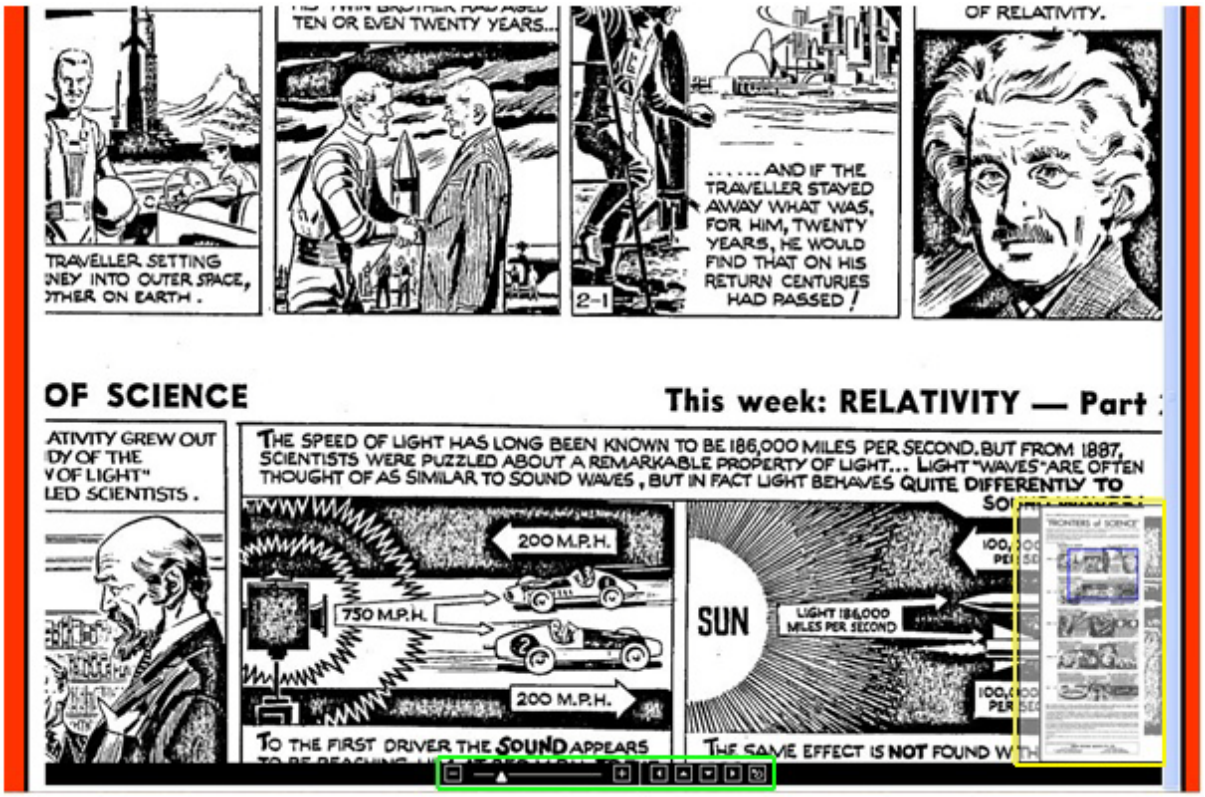

Figure 4. The Zoomify overlay showing zoom and pan controls, and the navigation window.

The Zoomify window provides a convenient method of navigating through and reading the pulls and gives the user the opportunity to look in detail at the high resolution (tiled) images. Clicking on the "Close" link, using the "Esc" key or clicking anywhere outside of the overlay window closes the Zoomify overlay window.

THE HOME PAGE

The home page features 5 selected strips presented in a vertical tabbed format (Figure 5).

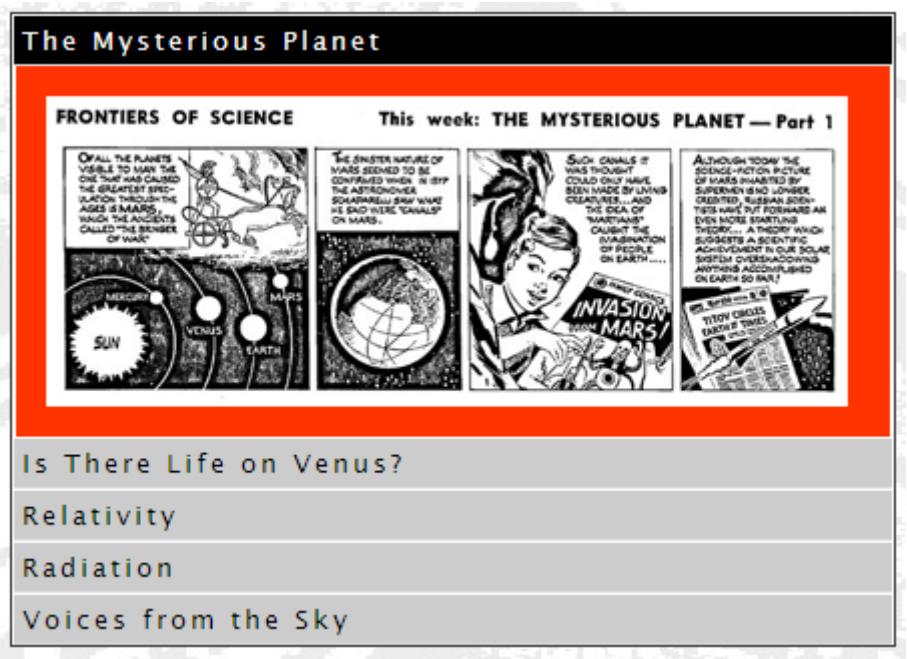

Figure 5. Vertical accordion tabs on the home page.

We used JQuery Tools accordion tabs (Code Listing 6), with a "fade" effect, rather than sliding.

```
<link rel="stylesheet" type="text/css" href="css/default/tabs-accordion.css">
<link rel="stylesheet" href="css/default/thickbox.css" type="text/css" media="screen">
<script src="script/jquery/jquery.tools.min.js" type="text/javascript"> </script>
<script type="text/javascript" src="script/thickbox/thickbox.js"></script>
.
.
.
<script type="text/javascript">
$(function() {
  $("#accordion").tabs("#accordion div.pane", {
        tabs: 'h2',
        effect: 'fade'
  });
});
</script>
```

Code Listing 6. JQuery Tools Accordion Tabs

Clicking on a strip title fades in that particular strip's "thumbnail". Clicking on the thumbnail itself employs the same methods as the search/browse functionality to display a zoomable and pannable pull.

THE TAG CLOUD

As an alternative method for browsing the archive, it was suggested that a tag cloud be developed. I like Steven York's [3] approach to this, so I used a slightly modified version of his "accessible tag cloud in PHP and CSS (with MySQL)". Our "tags" were embedded in the $dc.identifier.other.dc.xml files, so a Java class (TagCloudFromDC.java) was written to extract the list of <subject> tag values from them, once again using the JAXP. The program outputs a flat text file with one tag per line, by traversing a directory of the dc.xml files. Steven York's PHP code was then modified to generate the HTML for the tag cloud from this text file (Figure 6).

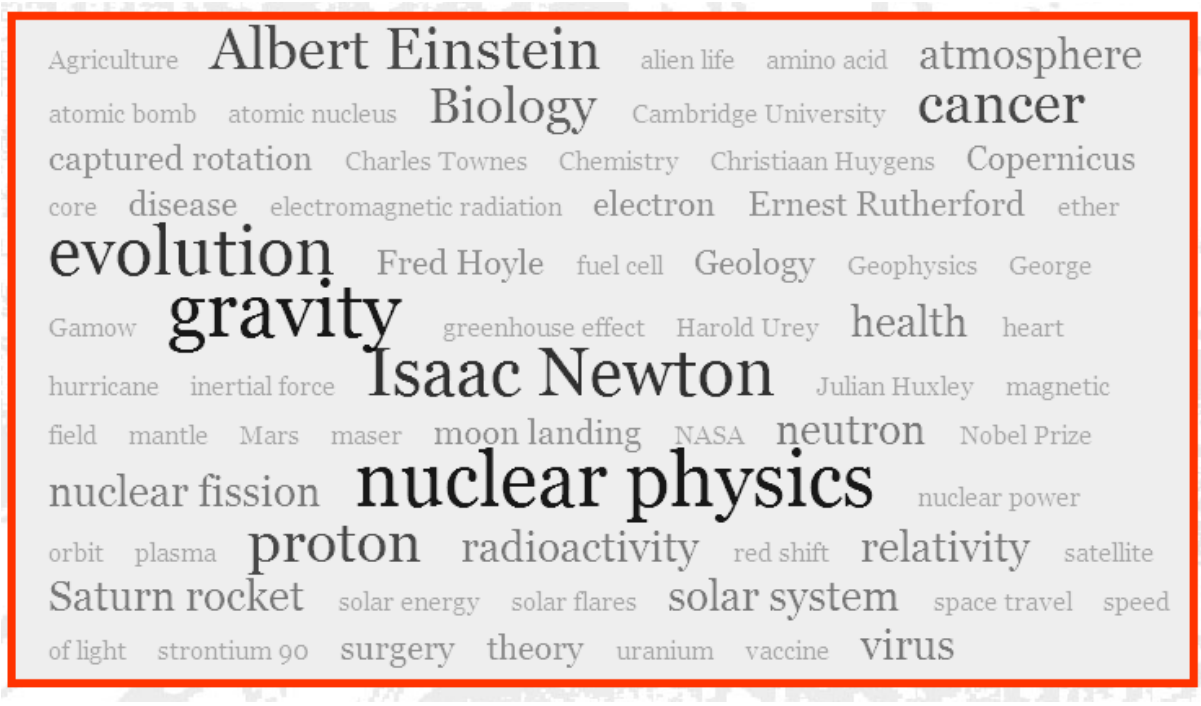

Figure 6. The Tag Cloud

THE NAVIGATION MENU

Highlighting the current page's menu item was accomplished using the technique described at http://www.456bereastreet.com [2] among others. A unique id or class is added to the body element of the page. Then CSS rules are written to match each body id or class with the appropriate menu item anchor tag:

```
/* highlight current page's corresponding navigation menu item */

body#home a#homenav, body#tagcloud a#tagcloudnav,

body#about a#aboutnav, body#help a#helpnav,

body#contact a#contactnav, body#search a#browseallnav,

body#browseall a#browseallnav, body#browsetitle a#browsetitlenav,

body#searchform a#searchnav, body#creators a#creatorsnav,

body#permissions a#permissionsnav{

        font-weight: bold;

        background: #ff3300;

        color: #fff;

}
```

Presentation and Copy Styles

Aside from the functionality, the website needed to be attractive in a way that reflected the history of the strips. Hence, the fonts chosen were those that were in keeping with the style of the original comic strips. Images for the site headers and page background were actual cropped portions of the original pulls. The colors used were exciting and bright to denote the entertaining aspect of the content. The copy itself was also written in a fun, lighthearted, but informative sense to match the mood of the strips.

The general styles and layout were loosely based on the University of Sydney stylesheets, so there were some constraints on the graphic design.

Serving the Site: System Architecture

Part of what made it possible to do this site within a limited time was our established infrastructure. It has been made simple to clone a virtual server with the required components to run an XTF website here at the University of Sydney Library. Table 1 outlines our standard setup.

Component	Version	Comments
Operating System	RedHat Enterprise Linux 5	
Java	Sun Java 5	
Web Server	Apache httpd v2.2	Handles incoming requests,

		passing them to Tomcat (using mod_proxy_ajp)
Servlet container	Tomcat v5.5	Serves the XTF servlets
Proxying	mod_proxy_ajp	pushes port 80 requests to port 8080 (Tomcat)

Table 1. University of Sydney Library IT Services standard server setup

The mod_proxy_ajp setup is contained within the Apache virtual host for Frontiers:

```
ProxyPass       /       ajp://localhost:8009/

ProxyPassReverse        /       ajp://localhost:8009/
```

In this manner, we can serve the home page as:

http://frontiers.library.usyd.edu.au

instead of:

http://frontiers.library.usyd.edu.au:8080

CONCLUSIONS

OUTCOMES

Referral data from Google analytics and our in-house web usage statistics indicate that there have been various sources of incoming traffic to the site, including:

- Direct traffic (primarily as a result of the launch)
- The Sydney University website
- The Sydney University Library website
- Wikipedia
- The Sydney Morning Herald site (article)
- The California Digital Library XTF website (featured application)
- Google search
- Facebook

We have had a lot of qualitative positive user feedback, with comments such as:

- "…the design is innovative and fresh…", Martin Haye, California Digital Library

- "…it is beautifully done and really showcases a great way of displaying and granting access to such an ephemeral art form as the newspaper comic strip", Bernard Caleo, comic book maker and publisher
- "…I'm delighted to discover that Frontiers of Science is being preserved for posterity!", Andrew Collier Cameron, Professor of Astronomy at the University of St Andrews

We have also seen traffic peaks during times of media interest, such as the Frontiers of Science exhibition (and comic drawing competition) at the University of Sydney SciTech Library in February 2011. This points to the value of marketing and media used creatively in conjunction with the website. Of course, there are budgetary constraints on these activities.

Issues

Following the launch of the site, there were several issues that came to our attention. Firstly, to get the site done quickly, we used Zoomify. This worked well, but constrains the use of the site to Flash-compatible devices. Therefore, zooming and panning the pulls does not work on iPhones and iPads. Ideally, a javascript-based mechanism would be employed for the zooming and panning functionality.

The site was W3C validated as far as practicable for HTML 4.01 Transitional or XHTML 1.0 Transitional. Validating proved difficult primarily due to the embed code for Zoomify. Changing from a Flash-based script to javascript would circumvent these issues.

In some cases, CSS code for the layout and styling proved difficult and was less than ideal. For example, aligning the search/limit results boxes with the JQuery Tools scrollable component could be better. Also the arrows for scrolling were not clear to some users, but changing them would have taken some time.

These issues may have been avoided had the project received more funding for technical staff (I was the sole programmer available to work on the site, along with my other existing projects).

Future Work

One of the major enhancements planned for the site is using OCR to scan the original pulls and obtain full text for indexing. Full text searching would then be a possibility. Digitization and tagging of additional pulls is also yet to be done, to expand the current database of material available.

Our standard server setup is now moving to Open JDK rather than the Sun Java JDK, so this will be instigated in the future.

Pending funding, there is also the possibility of video interviews and other multimedia additions to the site. Social networking and blogging would also be valuable tools for the future of the site, allowing greater user interaction with the site and with the general and academic community.

ACKNOWLEDGEMENTS

Judy Campbell and Professor Bob Hewitt – initial ideas

Julie Price and Nicholas Heath, Rare Books & Special Collections Library, University of Sydney – site planning and scanning of pulls

Susan Murray-Smith, Sydney eScholarship, University of Sydney – finalizing contracts

Alison Muir, Science Foundation for Physics – grant to enable digitization

Abbie Thomas, ABC Science – Frontiers exhibition

Digital and Print Media, University of Sydney – advice on University of Sydney style guidelines

Miriam Butler and Angela Raymond – rights and licensing

Professor Peter Harrowell – for tagging of the pulls

REFERENCES

1. The Java XPath API, http://www.ibm.com/developerworks/library/x-javaxpathapi.html

2. Highlighting the current page's menu item using CSS, http://www.456bereastreet.com/archive/200503/setting_the_current_menu_state_with_css/

3. Creating an accessible tag cloud in PHP and CSS (with MySQL), http://stevenyork.com/tutorial/creating_accessible_tag_cloud_in_php_css_mysql