



From DocBook to Wikipedia

An XML Publishing Case Study

Evan Lenz
XML 2008

About the presenter

- Evan Lenz
 - Independent XML consultant
 - <http://xmlportfolio.com>
 - Member of the XML Guild
 - <http://www.xmlguild.org>
 - Former member of the W3C XSL Working Group

About this project

- This case study is about a project I did for O'Reilly Media
- My background with O'Reilly
 - Reader, then
 - Tech reviewer, then
 - Author
 - *Office 2003 XML*
 - (co-authored with Simon St.Laurent & Mary McRae)
 - *XSLT 1.0 Pocket Reference*
 - Finally, consultant



What to expect

- Project overview
- MediaWiki basics
- Project details
- XSLT 2.0 highlights
- Implications



Project phase I (Spring 2008)

- Publish select books to a public wiki (O'Reilly Commons) for the developer community to maintain
- Target platform: MediaWiki
- Goal: Publish an entire book onto the wiki at one click of a button

Project phase 2 (Fall 2008)

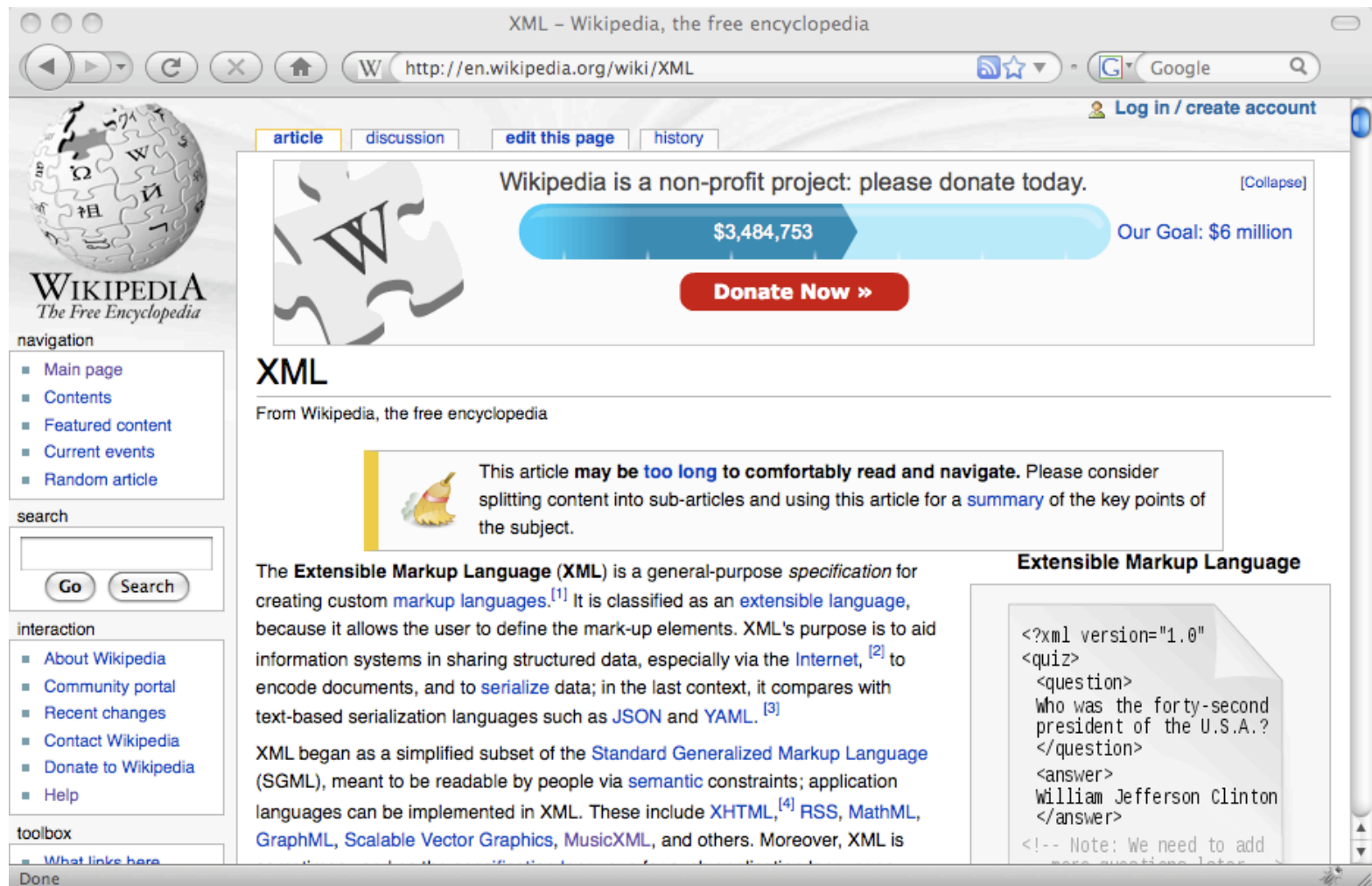
- A new request
- Publish a particular book, *Wikipedia: The Missing Manual* (by John Broughton), to the Help system on Wikipedia itself
 - slated for publication later this month or next month
- Ensure that the book content conforms to Wikipedia conventions

Intro to MediaWiki



- Open-source wiki platform
 - A wiki is a collaborative website whose pages support in-place editing
- Powered by PHP
- Originally written for, and still powers, Wikipedia

Clicking “edit this page” on this Wikipedia page...

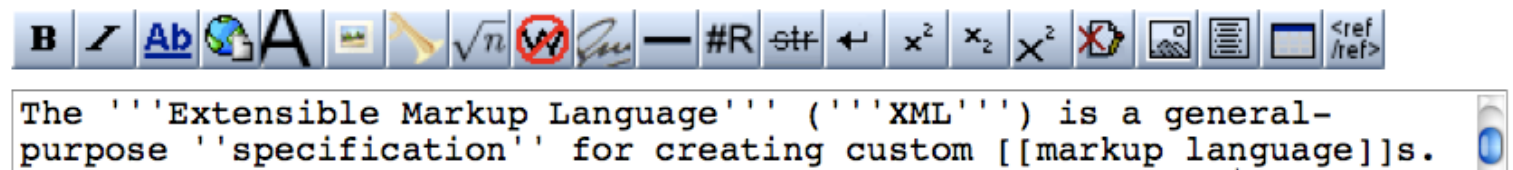


The screenshot shows the Wikipedia page for XML. The browser address bar displays 'http://en.wikipedia.org/wiki/XML'. The page title is 'XML - Wikipedia, the free encyclopedia'. The navigation tabs at the top are 'article', 'discussion', 'edit this page' (highlighted with a blue border), and 'history'. Below the tabs is a donation banner for Wikipedia, showing a progress bar at \$3,484,753 towards a goal of \$6 million, with a 'Donate Now' button. The article title 'XML' is prominently displayed, followed by the subtitle 'From Wikipedia, the free encyclopedia'. A yellow warning box with a bomb icon states: 'This article may be too long to comfortably read and navigate. Please consider splitting content into sub-articles and using this article for a summary of the key points of the subject.' The main text begins with 'The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages.[1]'. On the right side, there is a section titled 'Extensible Markup Language' containing an XML code snippet:

```
<?xml version="1.0"
<quiz>
  <question>
    Who was the forty-second
    president of the U.S.A.?
  </question>
  <answer>
    William Jefferson Clinton
  </answer>
<!-- Note: We need to add
more questions later -->
```

...gives you an edit window like this:

- Here's a portion of the edit window:



- And here's the corresponding result:

The **Extensible Markup Language (XML)** is a general-purpose *specification* for creating custom [markup languages](#).

Another example: previewing your edits

Preview

Remember that this is only a preview; your changes have not yet been saved!

This is **bold**.

Here's another paragraph. This is *italic*.

Here are links to other articles on Wikipedia: [XSLT](#), [Piano](#), and [Klavarskribo](#).



This is '''bold'''.

Here's another paragraph. This is ''italic''.

Here are links to other articles on Wikipedia:
[[[XSLT](#)]], [[[Piano](#)]], and [[[Klavarskribo](#)]].

Phase I: O'Reilly Commons

- Powered by MediaWiki



O'REILLY
commons

Navigation

- O'Reilly Commons
- Community portal
- Current events
- Recent changes
- Random page
- Help

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Evanlenz my talk my preferences my watchlist my contributions log out

article discussion edit history protect delete move watch

O'Reilly Commons

Welcome to the O'Reilly Commons. The purpose of this site is to provide content to communities that would like to create, reference, use, modify, update and revise material from O'Reilly or other sources.

Contents [hide]

- 1 Getting Started
 - 1.1 New Wiki Books
 - 1.2 Commons Pool
 - 1.3 Community Documentation

Getting Started

 [edit]

- Terms of Service
- Editing Policy
- Uploading/contribution Policy
- Discussion (Traditional Legal Frameworks, Open Project Frameworks, Differences)
- Characteristics of new media in the Internet age

New Wiki Books

 [edit]

This area is for original content that is written and edited directly in the MediaWiki framework. It is our hope that many interested individuals will contribute and make the resulting product more appealing and complete. Content that appears in this area may be used in other forms and venues. So you should make yourself aware of our terms of service and policies associated with participating in a unfettered group.

- The Art of Community

O'Reilly Commons

- <http://commons.oreilly.com>
- Purpose
 - “...to provide content to communities that would like to create, reference, use, modify, update and revise material from O'Reilly or other sources.”
 - See also:
 - http://commons.oreilly.com/wiki/index.php/Editing_Policy
- About O'Reilly books published there:
 - “...content contributed by O'Reilly Media to anyone who would like to edit, contribute and get involved in revising or modifying content that, today, may not make commercial sense as a printed book.”



16 books so far...

- *Programming Jabber*
- *QuickTime for Java: A Developer's Notebook*
- *Ubuntu Hacks*
- *SVG Essentials*
- *PHP Cookbook*
- *Essential CVS*
- *Learning Cocoa with Objective-C*
- *SpamAssassin*
- *XPath and Xpointer*
- *Greasemonkey Hacks*
- *Network Security Tools*
- *Test Driving Linux*
- *Linux in a Windows World*
- *Beyond Java*
- *Visual Basic 2005: A Developer's Notebook*
- *Open Sources 2*

Content organization for each book

- Chapters
 - 1 page per chapter
 - Using the same convention followed by wikibooks.org
- Optionally grouped into parts
 - 1 page per part
- Include the book's TOC on every page
- Page names follow this convention:
 - *Book_Name/Chapter_Name*, or
 - *Book_Name/Part_Name/Chapter_Name*

Example book page: PHP Cookbook


[article](#) [discussion](#) [edit](#) [history](#) [protect](#) [delete](#) [move](#) [watch](#)

PHP Cookbook

The *PHP Cookbook* is a collection of problems, solutions, and practical examples for PHP programmers. The book contains a unique and extensive collection of best practices for everyday PHP programming dilemmas. It contains over 250 recipes, ranging from simple tasks to entire programs that demonstrate complex tasks, such as printing HTML tables and generating bar charts -- a treasure trove of useful code for PHP programmers, from novices to advanced practitioners.

Contents [\[edit\]](#)

- [Preface](#)
- [Chapter 1: Strings](#)
- [Chapter 2: Numbers](#)
- [Chapter 3: Dates and Times](#)
- [Chapter 4: Arrays](#)
- [Chapter 5: Variables](#)
- [Chapter 6: Functions](#)
- [Chapter 7: Classes and Objects](#)
- [Chapter 8: Web Basics](#)
- [Chapter 9: Forms](#)



Example chapter page: PHP Cookbook/Preface

PHP Cookbook/Preface – WikiContent

http://commons.oreilly.com/wiki/index.php/PHP_Cookbook/Pr ☆ Google

Evanlenz my talk my preferences my watchlist my contributions log out

article discussion edit history protect delete move watch

PHP Cookbook/Preface

< PHP Cookbook

PHP is the engine behind millions of dynamic web applications. Its broad feature set, approachable syntax, and support for different operating systems and web servers have made it an ideal language for both rapid web development and the methodical construction of complex systems.

One of the major reasons for PHP's success as a web scripting language is its origins as a tool to process HTML forms and create web pages. This makes PHP very web-friendly. Additionally, it is a polyglot. PHP can speak to a multitude of databases, and it knows numerous Internet protocols. PHP also makes it simple to parse browser data and make HTTP requests. This web-specific focus carries over to the recipes and examples in the *PHP Cookbook*.

This book is a collection of solutions to common tasks in PHP. We've tried to include material that will appeal to everyone from newbies to wizards. If we've succeeded, you'll learn something (or perhaps many things) from the *PHP Cookbook*. There are tips in here for everyday PHP programmers as well as for people coming to PHP with experience in another language.


PHP, in source-code and binary forms, is available for download for free from <http://www.php.net/>. The PHP web site also contains installation instructions, comprehensive documentation, and pointers to online resources, user groups, mailing lists, and other PHP resources.

PHP Cookbook

- Preface
- Chapter 1: Strings
- Chapter 2: Numbers
- Chapter 3: Dates and Times
- Chapter 4: Arrays
- Chapter 5: Variables
- Chapter 6: Functions
- Chapter 7: Classes and Objects
- Chapter 8: Web Basics
- Chapter 9: Forms
- Chapter 10: Database Access
- Chapter 11: Web Automation
- Chapter 12: XML
- Chapter 13: Regular Expressions
- Chapter 14: Encryption and Security
- Chapter 15: Graphics
- Chapter 16: Internationalization and Localization
- Chapter 17: Internet Services
- Chapter 18: Files
- Chapter 19: Directories
- Chapter 20: Client-Side PHP
- Chapter 21: PEAR
- Colophon

Contents [hide]

- 1 Who This Book Is For
- 2 What Is in This Book
- 3 Other Resources
 - 3.1 Web Sites
 - 3.2 Books
- 4 Conventions Used in This Book
 - 4.1 Programming Conventions
 - 4.2 Typesetting Conventions
- 5 Comments and Questions
- 6 Acknowledgments
 - 6.1 David Sklar
 - 6.2 Adam Trachtenberg



O'REILLY
commons

Navigation

- O'Reilly Commons
- Community portal
- Current events
- Recent changes
- Random page
- Help

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Done



**Q. How do you get a lot of content
onto a wiki?**



Q. How do you get a lot of content onto a wiki?

- Answer: make it look like it was exported from another wiki

MediaWiki's export function

- Export any number of pages via the Special:Export page
- MediaWiki will give you a big XML file

special

Export pages

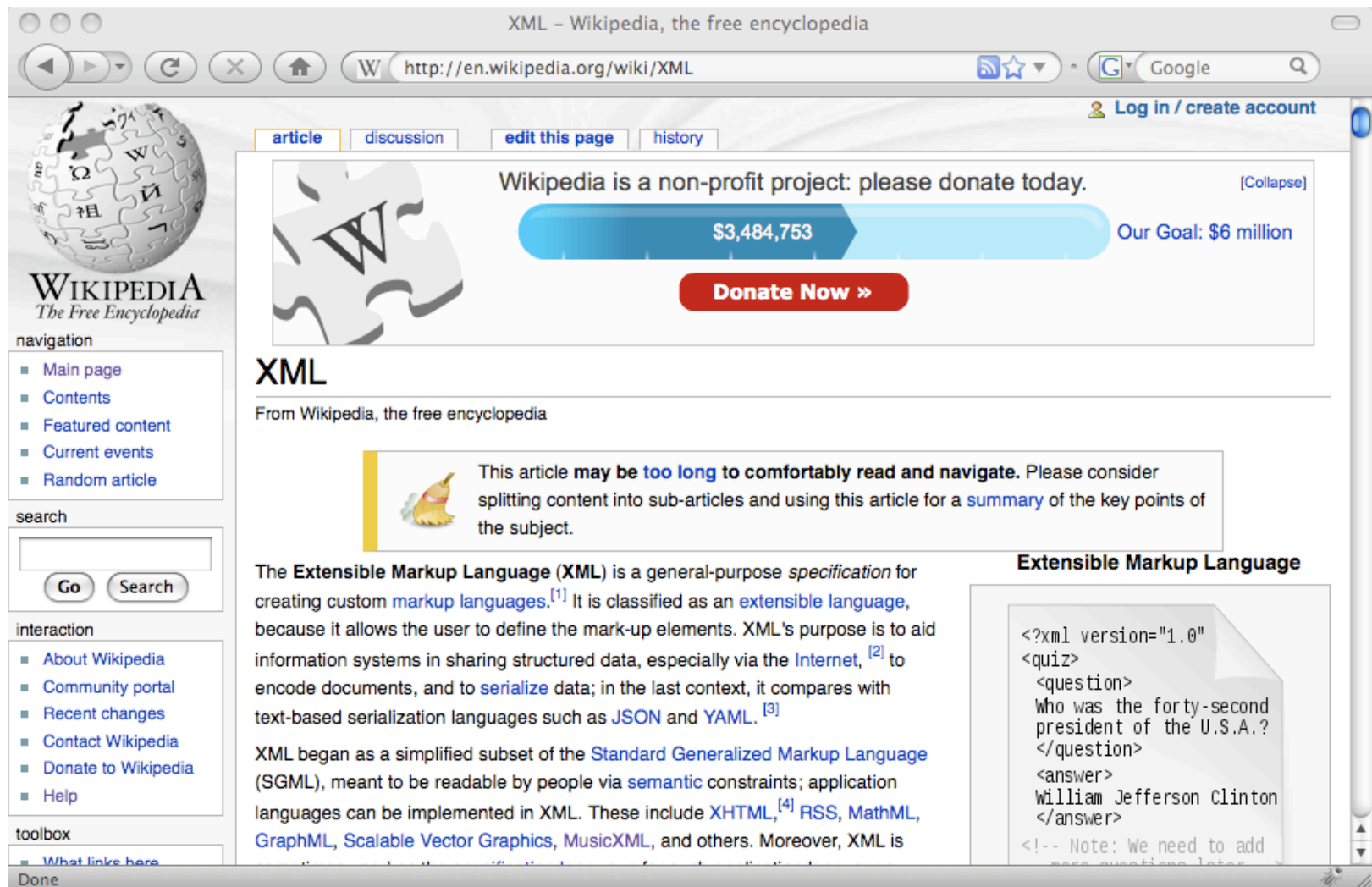
You can export the text and editing history of a particular page or set of pages wrapped in some XML. This can be imported into another wiki using MediaWiki via the [import page](#).

To export pages, enter the titles in the text box below, one title per line, and select whether you want the current version as well as all old versions, with the page history lines, or just the current version with the info about the last edit.

In the latter case you can also use a link, e.g. [Special:Export/O'Reilly Commons](#) for the page "O'Reilly Commons".

Add pages from category:

For example, exporting this page...



The screenshot shows the Wikipedia page for XML. At the top, the browser address bar displays 'http://en.wikipedia.org/wiki/XML'. Below the address bar, there are navigation tabs for 'article', 'discussion', 'edit this page', and 'history'. A prominent blue banner at the top right encourages donations, showing a progress bar at \$3,484,753 towards a goal of \$6 million, with a 'Donate Now' button. The main heading 'XML' is followed by the text 'From Wikipedia, the free encyclopedia'. A yellow warning box with a cartoon character states: 'This article may be too long to comfortably read and navigate. Please consider splitting content into sub-articles and using this article for a summary of the key points of the subject.' The main text begins with 'The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages.' To the right, a code block titled 'Extensible Markup Language' shows an XML snippet:

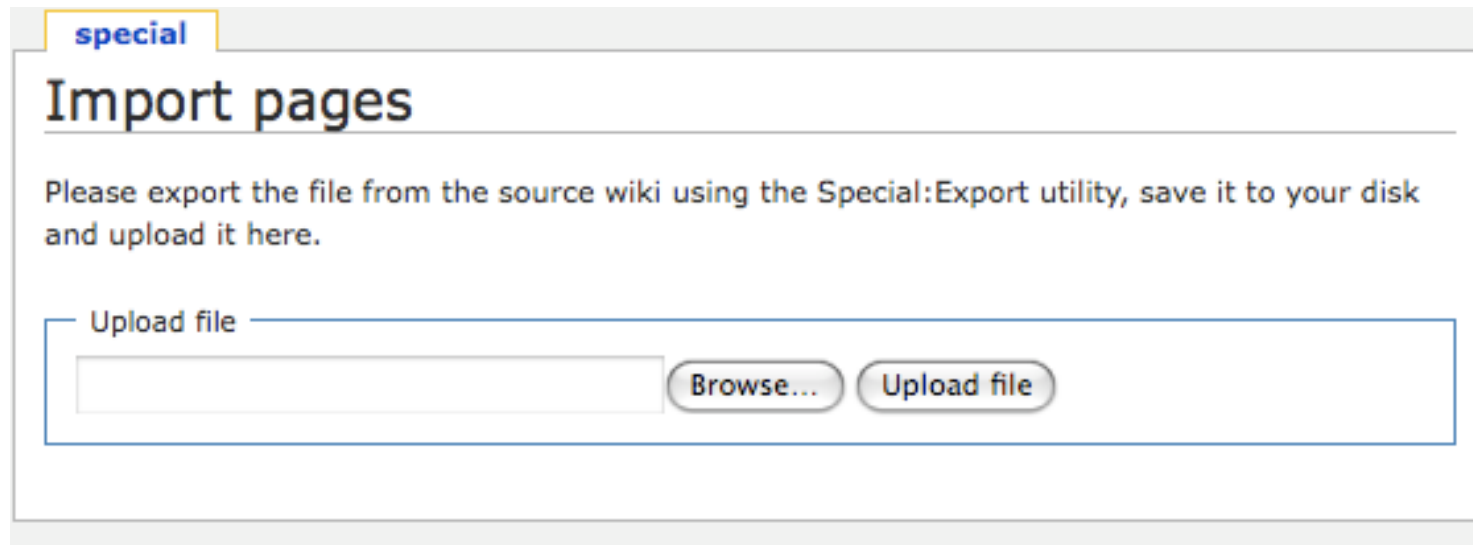
```
<?xml version="1.0"
<quiz>
  <question>
    Who was the forty-second
    president of the U.S.A.?
  </question>
  <answer>
    William Jefferson Clinton
  </answer>
<!-- Note: We need to add
more questions later -->
```


...results in a file that looks like this:

```
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.3/"
            version="0.3" xml:lang="en">
  <page>
    <title>XML</title>
    <revision>
      <timestamp>2008-12-06T18:17:39Z</timestamp>
      <contributor>
        <username>JLaTondre</username>
      </contributor>
      <comment>fix incorrect link to island using
[[Project:AutoWikiBrowser|AWB]]</comment>
      <text>The '''Extensible Markup Language''' ('''XML''') is a
general-purpose '''specification''' for creating custom [[markup
language]]s.
      ... Etc. ...
    </text>
  </revision>
</page>
</mediawiki>
```

Importing content to another wiki

- Once you have the MediaWiki XML dump, you can upload it to another wiki via the Special:Import page



The screenshot shows the 'special' tab selected in a MediaWiki interface. Below the tab is the heading 'Import pages'. A text instruction reads: 'Please export the file from the source wiki using the Special:Export utility, save it to your disk and upload it here.' Below this is a form with the label 'Upload file' and a text input field. To the right of the input field are two buttons: 'Browse...' and 'Upload file'.

special

Import pages

Please export the file from the source wiki using the Special:Export utility, save it to your disk and upload it here.

Upload file

Browse... Upload file



Bulk-loading new content

- MediaWiki's export format can also be used to add brand new content to a wiki, provided that it's packaged as MediaWiki expects it
- You can create any kind of page:
 - Regular content pages, image pages, template pages, etc.

What content did we have?

- Lots of DocBook XML content stored in an XML database powered by MarkLogic
- Retrievable via an internal Web-based application at O'Reilly called “deli”
 - Give it an ISBN number and get back a zip file containing:
 - Full XML content of the book
 - All supporting files, such as images
- The goal: push a button and make a book appear on the wiki

The final implementation

- A bash shell script that:
 1. Retrieves and unzips the content, given an ISBN
 2. Transforms content from DocBook XML to MediaWiki's export XML format
 3. Imports the resulting dump and images using server-side PHP scripts:
 - importDump.php
 - importImages.php

Here's the “button” (script) to push:

```
Terminal
#!/bin/bash

IMAGE_RENAMING_SCRIPT=copy-image-files.sh
RENAMED_IMAGES_DIR=renamed-images
WIKI_MAINTENANCE_DIR=/var/www/www.wikicontent.com/htdocs/wiki/maintenance

# Require ISBN to be supplied
if [ -z "$1" ]
then
    echo "USAGE: `basename $0` <isbnNumber>"
    exit 1
fi

echo -n "Downloading $1.zip..."
wget --quiet --output-document=$1.zip "https://deli.oreilly.com/content/Book/isbn13?isbn=$1&format=docbookzip"
echo "Done."

echo -n "Unzipping $1.zip..."
unzip -q $1.zip
echo "Done."

# Change to the unzipped directory
cd $1

echo -n "Transforming Docbook XML to MediaWiki import XML (and generating image-renaming script)..."
java -jar /home/evan/saxon9/saxon9.jar $1.xml /home/evan/doc2wiki/doc2wiki.xml \
    copy-script-name=$IMAGE_RENAMING_SCRIPT \
    renamed-images-dir=$RENAMED_IMAGES_DIR \
    DEBUG=$2 \
    >WIKI_PAGES.xml

echo "Done."

echo -n "Renaming image files..."
sh $IMAGE_RENAMING_SCRIPT
echo "Done."

echo "Importing book pages to MediaWiki..."
sudo php $WIKI_MAINTENANCE_DIR/importDump.php WIKI_PAGES.xml
echo "Done."

echo "Importing book images to MediaWiki..."
sudo php $WIKI_MAINTENANCE_DIR/importImages.php $RENAMED_IMAGES_DIR
echo "Done."

~
~
```



The heavy lifting

- Step 2 (transforming the XML) is where the real work happens
- The final stylesheet, doc2wiki.xsl:
 - Constructs a <mediawiki> export doc
 - Creates a <page> element for each <chapter> in the DocBook source
 - Inserts the content of each chapter into the MediaWiki <text> element

...like this:

```
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.3/"
            version="0.3" xml:lang="en">
  <page>
    <title>PHP Cookbook</title>
    <revision>
      <timestamp>2008-03-07T19:17:37Z</timestamp>
      <contributor>
        <username>Docbook2Wiki</username>
      </contributor>
      <comment>Initial conversion from DocBook</comment>
      <text>[[Image:PHP Cookbook cover.gif|right]]
```

The ''PHP Cookbook'' is a collection of problems, solutions, and practical examples for PHP programmers.

```
        <!-- etc.... -->
      </text>
    </revision>
  </page>
  <!-- rest of pages -->
</mediawiki>
```


Design goals of phase I

- Philosophy used:
 1. Make it user-friendly for wiki readers
 - Preserve structural integrity using HTML tags when necessary
 - E.g., preserve list nesting structure and alignment
 2. Make it user-friendly for wiki editors
 - Minimize use of HTML tags in the wiki markup
 - Use wiki shorthands whenever possible

How XSLT 2.0 features helped



This is a continuum, of course. XSLT 1.0 is Turing-complete, so theoretically everything is possible in 1.0. But that doesn't mean everything is practical.

Making things possible

1. Finding special wiki characters in code examples and escaping them
2. Finding HTML markup in code examples and escaping them
 - But don't just escape every "<"
 - Selectively escape only those tag names that are interpreted as HTML by MediaWiki
 - This supports the design goal of keeping things friendly for wiki editors

I. Escaping wiki markup in content

- Code example from *PHP Cookbook*:

```
<programlisting>  
$array[3] = $array['foo'] = '';  
</programlisting>
```

- Naively converts to, simply:

```
$array[3] = $array['foo'] = '';
```

But the result doesn't look right

- The apostrophes are missing and the colon is italicized:

```
$array[3] = $array['foo'] = ;
```

- Because ' ' means italicize in wiki markup

I could have punted

- Wrap `<nowiki>` around every single result text node
 - i.e. everything but the wiki markup
- For example, convert this DocBook:
 - `<para>Here is some <emphasis role="bold">bold text</emphasis>.</para>`
- to this wiki markup:
 - `<nowiki>Here is some </nowiki>' '<nowiki>bold text</nowiki>' '<nowiki>.</nowiki>`



But that would be horrible

- I remembered design goal #2:
 - user-friendliness for wiki editors
- So I needed to selectively apply the `<nowiki>` tag to text content that contains wiki markup

XSLT 2.0's regular expression support makes that easy

- Here's the template rule I used to wrap a `<nowiki>` tag around text nodes that contain wiki markup:

```
<!-- Escape Wiki markup that can be escaped using <nowiki> -->
<xsl:template mode="escape-wiki-markup"
               match="text()[matches(.,$wiki-regex,'x')]"
               priority="1">
    <nowiki>
        <xsl:copy/>
    </nowiki>
</xsl:template>
```


Regular expression

- And here's the regular expression I used to find wiki markup in content:

```
<xsl:variable name="wiki-regex">
  \{\{      <!-- template references -->
| \}\}
| \[\[      <!-- page links -->
| \]\]
| \{\|      <!-- table start delimiter -->
| '''      <!-- bold -->
| ''       <!-- italic -->
</xsl:variable>
```

Result looks correct now

- Now converts to:

```
<nowiki> $array[3] = $array['foo'] = '';</nowiki>
```

- And looks good in the result:

```
$array[3] = $array['foo'] = '';
```

2. Heavier-duty example

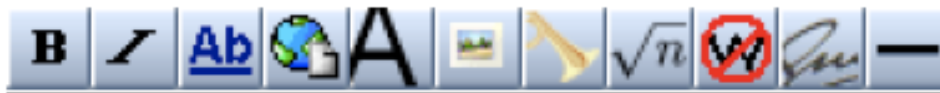
- XSLT 1.0 could have handled the last example by doing a bunch of contains() tests
- But this one would be tougher:

```
<lt;( (!--) |  
    (/(?( nowiki |  
        ref | references |  
        b | del | i | ins | u | font | big | small | sub | sup |  
        h1 | h2 | h3 | h4 | h5 | h6 | cite | code | em | s | strike |  
        strong | tt | var | div | center | blockquote | ol | ul | dl |  
        table | caption | pre | ruby | rt | rb | rp | p | span | u |  
        br | hr | li | dt | dd | tr | td | th  
        )[^a-zA-Z]  
    )  
)
```

- (matches HTML start & end tags and HTML comments)

Resulting benefit: cleaner code examples featuring XML or HTML

- Code examples that look like this:



```
<section>
  <title>Example</title>
  &lt;p>This is <emphasis>mixed</emphasis>.&lt;/p>
</section>
```

- Instead of this:



```
&lt;section>
  &lt;title>Example&lt;/title>
  &lt;p>This is &lt;emphasis>mixed&lt;/emphasis>.&lt;/p>
&lt;/section>
```



Making code beautiful

- a.k.a. engineering good software
- a.k.a. having fun programming 😊
- In the Ruby world, the DRY principle
 - “Don’t Repeat Yourself”



First example:

Multiple-mode template rules

- A common pattern I used was the identity transform
- I used a 6-stage pipeline, in which 5 out of 6 transformations made only incremental changes to the input

First, here's the pipeline

<!-- STAGE 1 -->

```
<xsl:variable name="pre-processed">
  <xsl:apply-templates mode="pre-process" select="."/>
</xsl:variable>
```

<!-- STAGE 2 -->

```
<xsl:variable name="wiki-markup-escaped">
  <xsl:apply-templates mode="escape-wiki-markup" select="$pre-processed/*"/>
</xsl:variable>
```

<!-- STAGE 3 (the main transformation from DocBook to MediaWiki XML) -->

```
<xsl:variable name="intermediate-result">
  <xsl:apply-templates mode="mediawiki-xml" select="$wiki-markup-escaped/*"/>
</xsl:variable>
```

<!-- STAGE 4 -->

```
<xsl:variable name="html-serialized">
  <xsl:apply-templates mode="serialize-html" select="$intermediate-result/*"/>
</xsl:variable>
```

<!-- STAGE 5 -->

```
<xsl:variable name="code-examples-formatted">
  <xsl:apply-templates mode="format-code-examples" select="$html-serialized/*"/>
</xsl:variable>
```

<!-- STAGE 6 -->

```
<xsl:variable name="final-result">
  <xsl:apply-templates mode="change-to-default-ns" select="$code-examples-formatted/*"/>
</xsl:variable>
```

Next, here's the space-saving code

- Assigning more than one mode to a given template rule (a new feature in XSLT 2.0) saves me from having to duplicate a lot of code.
- Here's the identity transform, listed just once as a baseline for all 5 pipeline stages that use incremental transformations

```
<!-- Re-use this same basic identity rule for multiple stages -->
<xsl:template mode="pre-process
                escape-wiki-markup
                serialize-html
                format-code-examples
                change-to-default-ns" match="@* | node()">
  <xsl:copy>
    <xsl:apply-templates mode="#current" select="@* | node()"/>
  </xsl:copy>
</xsl:template>
```

- In XSLT 1.0, I'd have to repeat myself a bunch (using 5 template rules instead of 1)



Second example:

User-defined functions in patterns

- User-defined functions are new in XSLT 2.0
- They can be particularly useful in avoiding duplication in template rule “match” patterns

First, we have patterns with long lists of elements

```
<!-- delimiter for italicized elements -->
```

```
<xsl:template mode="bold-or-italic-mark"
  match="replaceable
    | parameter
    | filename
    | emphasis
    | systemitem
    | citetitle
    | foreignphrase
    | lineannotation
    | command
    | firstterm">'</xsl:template>
```

```
<!-- delimiter for bold elements -->
```

```
<xsl:template mode="bold-or-italic-mark"
  match="userinput
    | emphasis[@role='bold']">'</xsl:template>
```

Next, another rule that applies to the same (bold or italic) elements

- In XSLT 1.0, I'd have to copy and paste that long list of elements
 - Actually, I'd probably define them as an entity in the internal DTD subset, but I digress

```
<xsl:template mode="inline"
               match="THAT LONG LIST OF ELEMENTS GOES HERE
```

Then, a user-defined function

- This function tests whether a given element is in my long list of elements

```
<xsl:function name="f:is-bold-or-italic"
              as="xs:boolean">
  <xsl:param name="element" as="element()"/>
  <xsl:variable name="mark">
    <xsl:apply-templates mode="bold-or-italic-mark"
                        select="$element"/>
  </xsl:variable>
  <!-- Return true if this element is in the list -->
  <xsl:sequence select="boolean(string($mark))"/>
</xsl:function>
```

Finally, we call the function from the match pattern

- Now we don't have to repeat that long list of element names:

```
<xsl:template mode="inline"
              match="*[f:is-bold-or-italic(.)]">
  <xsl:apply-templates mode="bold-or-italic-mark" select="."/>
  <xsl:apply-templates mode="inline">
    <!-- [some plumbing omitted here] -->
  </xsl:apply-templates>
  <xsl:apply-templates mode="bold-or-italic-mark" select="."/>
</xsl:template>
```



Design goals of phase I (summarized again)

- In order of importance:
 1. Make it user-friendly for wiki readers
 2. Make it user-friendly for wiki editors

Phase 2 design goals are the reverse

- In order of importance:
 1. Make it user-friendly for Wikipedia editors
 - Editors should only see wiki markup and *never* any HTML tags
 2. Make it look reasonable
 - even if structure is lost
 - even if nested lists aren't aligned perfectly



Implementation of phase 2

- Only applicable to one book, so essentially a one-off conversion
- Consists of a new stylesheet that imports doc2wiki.xsl, making selective customizations

Phase I example result

Adding Text

You edit Wikipedia articles in a big, white text box in the middle of the window. To get to that box, you must go into edit mode.

1. In the search box on the left side of the screen, type **WP:SAND**, and press Return to go to the sandbox.

You'll do all your work in this chapter in the sandbox, so you won't actually change any Wikipedia articles.

2. From the sandbox page (**Figure 1-1**), click the "edit this page" tab.

You're now in edit mode, complete with the edit box shown in **Figure 1-2**.

Note:

If the bottom of **Figure 1-2** looks intimidating, don't worry: There are only about two dozen items that editors actually use, except in exceedingly rare circumstances. If you're curious, [A Tour of the Wikipedia Page](#) provides a complete cross-reference to everything on the bottom of **Figure 1-2**, as well as all the icons on the edit toolbar.

3. Delete everything but the first three lines, which are instructions.

- Wiki markup does *not* support this layout without using HTML elements (``, ``, and `<div>`).

Phase 2 example result

Adding Text

You edit Wikipedia articles in a big, white text box in the middle of the window. To get to that box, you must go into edit mode.

1. In the search box on the left side of the screen, type **WP:SAND**, and press Return to go to the sandbox.

You'll do all your work in this chapter in the sandbox, so you won't actually change any Wikipedia articles.

2. From the sandbox page (**Figure 1-1**), click the "edit this page" tab.

You're now in edit mode, complete with the edit box shown in **Figure 1-2**.

Note:

If the bottom of **Figure 1-2** looks intimidating, don't worry: There are only about two dozen items that editors actually use, except in exceedingly rare circumstances. If you're curious, [A Tour of the Wikipedia Page](#) provides a complete cross-reference to everything on the bottom of **Figure 1-2**, as well as all the icons on the edit toolbar.

3. Delete everything but the first three lines, which are instructions.

- Uses 100% pure wiki markup (and plain text numbering)
- Phase 2 design goal achieved: make it look...reasonable



A difference in philosophy

- Personally, the phase 2 example offends my sensibilities, but perhaps I'm just being an XML bigot
- Bottom line: Wikipedia editors won't be scared away by HTML tags
- Round-tripping seems remotely possible with phase 1, but not with phase 2
 - Of course, round-tripping was never the goal

Categorizing transformations according to where the data lives

	Source (XSLT input)	Result (XSLT output)
Presenting/querying data	✓	
Creating data ("up-conversions")		✓
Migrating data (to a new format)	✗ →	✓
Forking data (multiple copies)	✓	✓

Categorizing transformations according to where the data lives

	Source (XSLT input)	Result (XSLT output)
Presenting/querying data	✓	
Creating data ("up-conversions")		✓
Migrating data (to a new format)	✗	✓
→ Forking data (multiple copies)	✓	✓



An XML “forking” case study

- The book content starts a new (separate) life on Wikipedia
 - Over time it will essentially be a different work
- Apparent wiki philosophy:
 - Throw structure to the wind
 - With an army of active, passionate authors and editors working in their free time, perhaps structure ain't so important
- Highlights the advantages of the XML approach
 - particularly when you don't have such a generous army of workers at your disposal



Questions?