Learning XoWiki

A Tutorial to the XoWiki Toolkit

Gustaf Neumann, Stefan Sobernig

Vienna University of Economics and Business Administration

Table of Contents

. Using XoWiki

. Some XoWiki Showcases

. XoWiki as Development Toolkit

0 Learning XoWiki
1 Table of Contents
2 What is XoWiki?
3.1 Using XoWiki
3.2 Links
3.3 Page Types
3.4 Prototype Pages
3.5 Composite Pages
3.6 XoWiki Forms
3.7 Mini Applications based on Forms
3.98 What did we not talk about...
3.99 End of Part 1/3
5 XoWiki based Applications
6 XoWiki as Development Toolkit
61 End of Part 3/3
62 References
What is XoWiki?

- Wiki for OpenACS
- Full text search, categories, general-comments etc
- Stores everything in the content repository
- Revisions, reusable content, multi-language, user-tracking
- Easy to use, few markup commands, Xinha rich-text editor
- Customizable
- Easy integrated into existing applications
- Performance

Using XoWiki

Basic Concepts

- Block-markers
- Links
- Page Types
- Prototype Pages
- Composite Pages
  - Includelets
  - Forms, Form-Pages, Form-Fields
Markup for Block-markers

- Divide page into section with DIV-tags
- CSS formats the DIV tags
- Predefined block markers
  - >>left-col<<
  - >>left-col30<<
  - >>right-col<<
  - >>right-col70<<
  - >>box<<
- Block markers are be closed with >>><<

Using Block-Markers
Links

- Intra-Wiki links: `[[pageReference|?link label?]]`
- File link: `[[file:fileName|?label?]]`
- Glossary link: `[[glossary:pageReference|?label?]]`
- Sub-class XoWiki::Link to create custom link types!

Glossary Link

- Used for definitions
- Browser reads pages using AJAX
- One XoWiki instance has to be named "glossary"
- Search sister-nodes then parent nodes until "glossary" is found
- Pages from other XoWiki instances refer to pages with the glossary link: `[[glossary:pageName|?label?]]`
Page Types

- XoWiki defines several different page types
- Most prominent page type ::xowiki::Page
- Other page types are subtypes, extending the basic functionality in various ways
- Differences:
  - Different Methods
  - Additional Attributes

Predefined Page Types
Page Types (2)

- xowiki::Page
  - most used page type, using rich-text editor (Xinha)
- xowiki::PlainPage
  - specialization of xowiki::Page, using plain textarea
- xowiki::File
  - stored files in XoWiki (images, documents etc.)
  - Linked to as file (download) or image (display in page)
- xowiki::PodcastItem
  - Specialization of xowiki::File
  - Extra meta data, different syndication
  - Published as podcast (http://server/xowiki/podcast)

Page Types (3)

- xowiki::Page Template predecessor of ::xowiki::Form
- xowiki::PageInstance predecessor of ::xowiki::FormPage
- older versions for backward compatibility
- Provide means to develop mini-apps (end-user programming)
- ::xowiki::Form and ::xowiki::FormPage introduced in next section
Page Types (3)

- \texttt{xowiki::Object} is a XOTcl object saved in the CR
- Contains code for dynamic pages
- Method "content" is called when rendered
- Require admin permissions to change
- Used for e.g the Weblog portlet

Example for ::xowiki::Object

```tcl
# -*- tcl-*-
# Small example for demonstrating an ::xowiki::Object
# When the prototype page is loaded, the object named
# CGI will be added to the XoWiki package
::xowiki::Object new -title "CGI" -text {
  proc content {} {
    return "Hello \[[[Wiki]]\]-World. It is now [clock format [clock seconds]]." 
  }
}
```

Rendered Content looks like:

Prototype Pages

Background:

. Prototyping is a Form of reuse in Programming Language Design
. Alternative to Inheritance

Idea:

. Take a full-functional Artifact
. Modify it to your needs

XoWiki Prototype Pages

Stored in the file system (packages/xowiki/www/prototypes/)

Can be loaded into the CR of an xowiki instance (version control after the initial import)

Typical prototype pages:

. index page
. news
. book
. podcast
. sitemap.xml
Composite Pages

XoWiki Pages can contain reusable units, such as

- other XoWiki pages
- configurable, dynamic content creators

Every includelet is configurable

- decoration portlet|plain|none
- title ...

Content Creators

Embeddable, configurable dynamic content

- Calling interface with default values, values from url, provided values

- List all: \{\{available-includelets\}\}

- Examples:

  - \{\{rss-button -span 10d -name_filter -title\}\}
  - \{\{digg -description "" -url\}\}
  - \{\{recent -max_entries 10\}\}
  - \{\{get -variable title -source cover\}\}
Example Includelet: Activity Graph

. {{activity-graph -max_activities 200 }}

Example Includelet: Timeline

. {{user-timeline -interval1 WEEK -interval2 YEAR}}
XoWiki Forms

::xowiki::Forms defines **input and output** behavior for ::xowiki::FormPages

**Input:**
- What variables should be gathered, when the page is edited

**Output:**
- How are is a page to be presented (with the embedded variables)

Relation between ::xowiki::Form and ::xowiki::FormPage

Two collaborating page types
- ::xowiki::Form:
  - specifies additional variables,
  - validators,
  - renderers,
  - mandatory parts, ...
- ::xowiki::FormPage:
  - associated data,
  - for one Form exist typically n FromPages
Properties of ::xowiki::Forms

. Bi-Directional Linkage between Form and FormPage
. Forms and Form-entries are stored in the CR
. One step definition of form entries (optional text content, optional attributes)
. attributes are defined via form fields (can be subclassed)
. Many more features: named/unnamed entries, summary tables, categories per form, entry specific localization of form fields, ...

Form Properties (2)

. Values are filled into HTML forms via tdom
. Forms can be built by a form-composer
  . example: Compound Excercise Compound Excercise Answer
. Combination of HTML and auto-generated form fields possible (e.g. submit button added automatically)
. Form-fields can be rendered full (with label) or inline
. HTML-Form-part can contain auto-generated fields as well
  . Example of inline rendering: Scales Scale-Entry
. Categories can be displayed on a per-form basis
Form Properties (3)

- Forms and FormPages are kept in the content repository
- Content repository keeps revisions of forms and answers (also answers can be corrected)
- Content-item lifecycle useful for questionnaires and exams
  - during fill-out: production
  - final submission: ready
- Approach independent, how the HTML form-content was generated
  - e.g. some xinha plugin
  - rendered form some xml (learn@wu-xml-format)
  - by hand...

Form Constraints

Form-constraints is a per-Form information specifying properties of form fields

Form-constraints define
- Semantic properties and output formats for form-fields
- Types from content-repository attributes are inherited, wherever applicable (e.g. boolean type for field "anonymous instances")
- Form fields can define validators, which are checked on auto-generated fields and on HTML fields
Syntax of Form Constraints

Syntactical Ideas:

- Similar interface for specifying form-fields like in non-positional arguments
  
  Use e.g. `some-field:boolean,required as usual`

- Simple syntax, address non-programmers as well (not so many curley braces like `ad_form`)

- Multiple refinements of form-fields possible

- Technically: Tcl list of form-field-specs

Form Fields

Building blocks of `::xowiki::Forms`

- Based on XOTcl classes (use inheritance, can be specialized, work with mixins...)

- All form-fields can be rendered inline or full (with label)

- Form-fields distinguish between internal and external representation (for e.g. sorting by month)

- Localization possible not only on connection locale, but also on entry-locale

- Uses DotLRN ZEN CSS
Predefined Form Field Types

Mini Applications based on Forms

FormPages can be used for developing small applications (mini-apps)

Two examples:

- Voting
- News
Voting

- Basic, down-stripped example
- User can give a vote
- No check, if the user voted already (could be realized via e.g. a prototype page)
- Modification of entries by other users could be realized via creator privilege in policy (everyone is allowed to create an entry, only creator and admin are allowed to change it)
- Entries are un-namned, no need to input a name

Vote: FormPage

[Image of Vote Form]

- Filling out the Form
- Answer Table

[Table showing votes]

- Gustaf Neumann: 17 votes
- No: 10 votes
- Yes: 7 votes
News

. More elaborate example:
  
  . prototype page in xowiki cvs
  . modifying publish-state via form
  . set publish-date (used for sorting)
  . use unnamed entries
  . provides image url
  . summarizer: prototype page news, similar to weblog

What did we not talk about...

. Tagging of pages
. Comments
. Notification
. Search
. Syndication (RSS, ...)
. etc ..
End of Part 1/3

XoWiki based Applications

- Real-word and prototype showcases
  - Managing mailing lists
  - Keeping "learning diaries"
  - Providing web application "mash-ups"

- Developer tools
  - How to create your own page types
  - How to create your own includelets
  - How to create your own link types
Create your own XoWiki Application

. Why would you like to do that? Some teasing examples

. Navigating through and managing mailing lists through XoWiki: *Hypermail2XoWiki*

. Keeping a "learning contract" by means of XoWiki: *iLogue*

. Creating "application mash-ups" based on XoWiki: *xomashup*

. XoWiki as development toolkit

. xotcl-core + XoWiki-specific idioms available to the developer

. Page types

. Typed links

. Includelets

. Form Fields

---

Showcase / Hypermail2xowiki

![Hypermail2xowiki screenshot](xotcl.wwo-wien.ac.at)
Showcase / Hypermail2xowiki

Hypermail2xowiki allows for ...

- tracking and archiving mailing lists (by replication)
- providing a convenient and enriched environment for interacting with archived postings:
  - Further structuring through category and tag facilities
  - Enhanced visibility: RSS feed generation, integration with full-text search facility of OpenACS
- Conversational enrichment: Support for comments and notifications

Design decisions:

- Conversion of lists' mbox files into HTML documents by means of hypermail (see http://www.hypermail-project.org/)
- Replication at regular intervals into representations of XoWiki pages (::xowiki::PlainPage); attachments to postings are preserved as ::xowiki::File instances.
Showcase / Hypermail2xowiki

Showcase / iLogue
Showcase / iLogue

iLogue is an attempt to ...

- support the pedagogical intervention instrument of "personal learning contracts", originally proposed by ...

- realize "personal learning contracts" that comprise ...
  - "Records of action": On-going reflection on learning progress
  - "Contract conversation": Elaboration and articulation of "goal sets"
  - "Review report": A reflection phase on the evolution of the contract and goal achievement

- Entirely built upon pre-existing XoWiki functionality which was simply refined by specializing toolkit features

- Added "in-place editing" on-top

- Get it from iLogue@sourceforge.net
Showcase/ Mupple

Mupple: infrastructure for personal learning environments based on the mash-up metaphor

- Presentation layer based on JS/Ajax toolkit adaptations
  - Grid-based screen with sortable columns
  - Windows presenting URLs (web application, includelets or feeds)
  - Windows can be reloaded, minimized, maximized or closed
  - New URLs can be added and launched
- XoWiki is used at the application/ data layer:
  - Prototypical use of XoWiki templates to share mash-ups of "web applets"
  - "Web applets" are realized using XoWiki's includelets
- Contact: felix.moedritscher@wu-wien.ac.at
XoWiki as Development Toolkit

. What are the most important ingredients for your XoWiki-empowered application?

  . **Custom Content Types:** XoWiki allows to define refined types of pages, so-called "page types"

  . **Customized Link Types:** You may also provide dedicated navigation units, so-called "typed links"

  . **Custom Includelets:** In case you want to enrich the authoring environment of users by "programmatic" elements that go beyond mere mark-up re-use

  . **Custom Form-Fields:** Provide Input/Output behaviour
**Toolkit Components**

Beyond XoWiki extension infrastructure, the entire functionality of XOTcl is at your hand:

- Package and scope management
- Object-relational DB Access Layer
- See tutorial on XOTcl Core + related materials

**Operations on Pages**

Before focussing on advanced extension techniques (custom types etc.), some hints on basic interaction (CRUD) with XoWiki pages is appropriate:

- (C)reate
- (R)ead
- (U)pdate
- (D)elete

This functionality is provided by xotcl-core through the Content Repository management layer, (see xotcl-core tutorial)
Prerequisites

. Prerequisites for managing XoWiki pages
  . Define, which package instance to use: e.g.: http://yoursite.org/xowiki
  . Locate its content folder (per-package-instance)
  . Every XoWiki instance has its own folder
  . The ID of the folder object is also the folder id in the CR
  . We need this for read/write/lookup operations on pages

CRUD Example

# Prerequisites
# - Initialize package instance
::xowiki::Package initialize -url /xowiki

# - Get the id of the Content Repository Folder (folder_id)
set folder_id [$package_id folder_id]

# READ operation: Check, whether a page_name is already used
set page_name "my_page_[clock scan now]"
if {{::xo::db::CrClass lookup -name $page_name -folder_id $folder_id} == 0} {
  # We can CREATE the page
  # First create an XOTcl Object in memory
  ::xowiki::PlainPage create $page_name \
  -set text {This is the content of the new Page} \ 
  -set title {This is the Title of the new Page} \ 
  -name $page_name \ 
  -parent_id $folder_id \ 
  -package_id $package_id

  # persist the page in the content repository
  $page_name save_new

  # optionally, destroy the XOTcl object
  $page_name destroy
}
Read Operations

::xo::db::CrClass lookup -name ... -folder_id ... :

  . Check, if an item with the given name exists in the content repository. If yes, the item_id is returned

::xo::db::CrClass get_instance_from_db -item_id ... :

  . Load a page from the content repository

Get information from the page

  . $page title
  
  . $page get_content

Creating/Updating Pages

Typical pattern

  . Create an XOTcl page in memory
  
  . Populate page object with actual values
  
  . CREATE: persist page with method "save_new"
  
  . UPDATE: create new revision with method "save"
Page Types

Page types specify the following behavior for XoWiki-specific content items:

- Rendering
- Editing (by association of appropriate form types)

Subclassing XoWiki's Page Types

By refining XoWiki's page type, one can reuse its functionality.

Steps involved:

- Step 1: Define a new Page Type as a subtype of one of the XoWiki Page types
- Step 2: Define a Form for editing your values
- Step 3: Provide custom rendering behavior for the page type, i.e. by introducing a type-specific method `render`. 
Example: Geo Page Type

Step 1: Define a custom page type

. A page that will be geographically tagged, by WGS84 geographic data (longitude, latitude)

Step 2: Provide a type-specific form declaration

. In our geo-annotated page type, we want to be allowed to provide longitude/latitude pairs

Step 3: A refined rendering method

. In our example, we want to render the geo information into a "geo" microformat

Includelets

Includelets are

. configurable content creators, available to page authors.
. Included in pages via {{{rss-button}}}

Defining a custom includelet:

. Step 1: Define a custom includelet class
. Step 2: Provide an appropriate rendering method for this class

Example: Embedding microformat geo-data in ordinary XoWiki pages:

{{{geo-longitude 90.516667 -latitude 14.616667}}}
### Custom Includelet

# Define a custom includelet which receives as input location coordinates
# (longitude and latitude) according to the WGS84 specification
# and renders it in micro-format mark-up
::xowiki::IncludeletClass create geo 
   -superclass ::xowiki::Includelet 
   -parameter {
      {__decoration plain} 
      {parameter_declaration {
         {-longitude} 
         {-latitude} 
      }} 
   } 

# Provide a custom renderer for the microformat output
# (following http://microformats.org/wiki/geo)
geo instproc render {} {
   my get_parameters 
   return [subst {
      <div class="geo">Coordinates: 
      <span class="latitude">$latitude</span>
      <span class="longitude">$longitude</span>
   </div>
   ]
}

---

### Typed Links

Typed links allow content authors to establish different relationship types between XoWiki pages.

- Define different kinds of relationships (see e.g. in W3C link types)
- How to provide new relationship types to XoWiki content authors?
  - Step 1: Define a new link class type
  - Step 2: Provide a type-specific renderer that needs to support
    - resolving the referenced page and
    - render the link text to be embedded into the rendering page
Custom Link Types

. Example

. Referring to other XoWiki pages as literature references: `[[cite:<page>|<link text>]]`

. Rendered links will expand into a quotation microformat fragment, following an example from microformat.org

Custom Link Definition

```ruby
# define link class
Class create ::xowiki::Link::cite -superclass ::xowiki::Link
::xowiki::Link::cite instproc render_found {href label} { # resolve the referenced page
  set item_id [my resolve]
  set reference [::xo::db::CrClass get_instance_from_db -item_id $item_id]

  # generate citation information
  set creation_date [::xo::db::tcl_date [reference set creation_date]]
  set creation_stamp [clock scan $creation_date]
  set user_id [reference set creation_user]
  set author(url) [export_vars -base /shared/community-member {user_id}]
  set author(pretty_name) [::xo::get_user_name $user_id]
  set published(pretty_date) $creation_date
  set published(raw) [::xo::ical clock_to_iso $creation_stamp]

  return [subst {
    <cite class="hcite">
      <a class="fn url" href="$href">$label</a>
      <span class="author vcard">
        <a href="$author(url)" class="url fn">$author(pretty_name)</a>
      </span>
      <abbr class="dtpublished" title="$published(raw)">
        $published(pretty_date)
      </abbr>
    </cite>
  }
```

Rendered Result

. What do we get from this new "cite" link type?

. [[cite:marimba | See the Article on the Marimba]]

<a class="fn url" href="/xowiki/en/marimba">See the Article on the Marimba</a>

<span class="author vcard">
  <a href="/shared/community-member?user_id=512" class="url fn">Stefano Sobernig</a>
</span>

<abbr class="dtpublished" title="20080210T164625">2008-02-10 16:46:25</abbr>

</cite>
Custom Form Fields

Custom form fields allow content authors to provide form entries and custom renderers in ::xowiki::Forms.

Required Steps:

1. Step 1: Define a class for a custom form field
2. Step 2: Provide a renderer
3. Step 3: Use custom Form Field in an ::xowiki::Form

Example: Pretty Printed Program Listings

Basic Idea:

1. Use HTML textarea for input
2. Use OpenACS pretty-printer (from api-browser) for output
3. Define a new form field type "code_listing"
4. Define a form using @listing@ in the content area
5. Form constraints: listing:code_listing,required
Form-Field for Program Listings

namespace eval ::xowiki {
    #
    # ::xowiki::FormField::code_listing
    #
    #
    Class FormField::code_listing -superclass ::xowiki::FormField::textarea -parameter {
        {rows 20}
        {cols 80}
    } FormField::code_listing instproc pretty_value {v} {
        [my object] do_substitutions 0
        return "<pre class='code'>[api_pretty_tcl [my value]]</pre>"
    }
}

End of Part 3/3
References

- XoWiki Documentation
  - http://media.wu-wien.ac.at/download/xowiki-doc/
  - http://alice.wu-wien.ac.at:8000/xowiki-doc
  - http://alice.wu-wien.ac.at:8000/xowiki-faq