Table of Contents

Introduction

This document is the developer documentation for Writing Annotation and Marking Platform (WAMP). In this document, you will find out about the code structure and the underlying technologies that are used to build WAMP.

Drupal

Drupal is the platform on which WAMP is based on. Drupal is a PHP-based Content Management System which is easily extensible. See drupal.org.

Directory Structure

```
drupal/
    wamp_includes/
        ui.html
    sites/
        all/
            modules/
                wamp/
                    essay/
                        css/
                        js/
                    modules/
                    research/
```

The 'css' and 'js' folders under 'wamp/essay' folder contain the cascading stylesheets and javascript code from annotator. They have to be copied from <repository_root>/annotator/release/<release_number>/css and <repository_root>/annotator/release/<release_number>/js respectively. The user interface for annotation, ui.html, is stored in 'wamp_includes' folder and has to be copied from <repository_root>/annotator/release/<release_number> too.

Drupal Basics

Please read the Drupal Module Developer's Guide and in addition the following guides to Database API, Form API, and Schema API. The Drupal API Reference will also be useful.

Database API

Database API Static Query Guide

Form API

Form API Guide for Drupal 6.x
Form API Reference Guide

Schema API

Schema API Quickstart Guide
Schema API Reference Guide
Schema API Data Types

Drupal Module Development

A drupal module consists on at least 2 files, a .module file and a .info file. A .install file is optional but it helps during module installation/update to create and alter database tables, as well as perform other necessary updates, if any.

A .info file gives Drupal information about your module. See wamp.module for more information.

A .module file is the heart of a Drupal module.

Drupal Modules

Modules that are shipped with a Drupal release can be found in 'modules' directory. As a best practice, additional modules should be put into 'sites/all/modules' in order to differentiate from official Drupal modules.

Community-contributed Drupal Modules

WAMP makes use of a number of community-contributed Drupal modules. They are as follows:

- Conditional Styles (Version 6.x-1.1)
    - This module allows us to attach stylesheets to a Drupal theme without modifying the template code
    - The code segment below is to be inserted into the .info file of the theme.

```
conditional-stylesheets[<conditional css clause>][all][] = css_file_name.css
```

- Hovertip (Version 6.x-1.x-dev)
    - This module makes it easy to attach mouseover hovertips.
    - The code segment below shows how to use hovertips

```
<span>Text</span><span class="hovertip">Hovertip Text</span>
```

- Jquery Update (Version 6.x-2.x-dev)
    - This module updates the version of jquery shipped with Drupal to the latest 1.3.2

- SMTP (Version 6.x-1.0-beta3)
    - This modules allow Drupal to send email via SMTP servers. It supports SSL-/TLS-enabled SMTP servers.

Drupal Modules Developed as part of WAMP

The Drupal component of WAMP consists of a Drupal module named "wamp" and 3 sub-modules "essay", "module" and "research".

WAMP Module

This module contains the navigational user interface for the annotators and spectators. It also contains the function for setting the operational mode, i.e., production (for firebug) and testing (for non-firebug).

Assumptions:

1. The Drupal role 'teacher' has to be created in the system.
2. A user has to be assigned the Drupal role 'teacher' in order to be allocated to a module as a tutor or a spectator.

Essay submodule

This submodule comprises of essay management functions for the administrator. Functions include essay creation, updating and deletion. The administrator can also view all annotations. The functions for annotating and viewing annotations are also contained here. The code from annotator javascript component is integrated in this submodule.

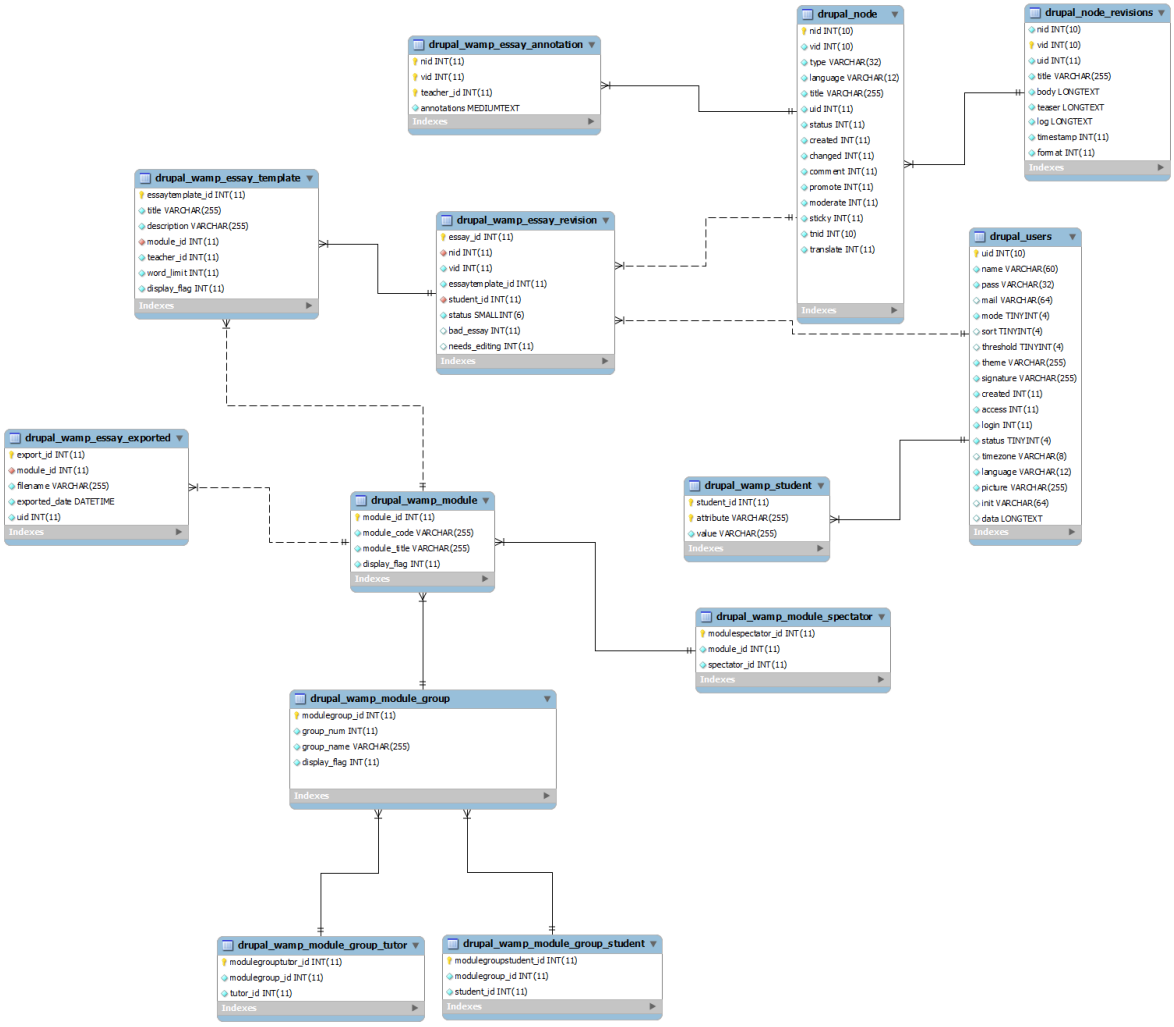
Module submodule

This submodule comprises of module management functions for the administrator. A module comprises of 1 or more groups. Each group can be taught by 1 or more tutors and can be attended by 1 or more students. As a group can have 1 or more tutors, an essay submitted by a student in each group can be annotated by the tutors in the group. Spectators, on the other hand, are on a per-module basis.

Research submodule

This submodule comprises of research-related data management for the administrator. These consist of import and export of XML data files functions. Imported XML files can also be processed to automatically create module, groups, students and assign them to module groups.

Data Model



Limitations

1. A user cannot be tutoring one module and spectating another at the same time, i.e., tutoring and spectating are activities that have to be separated into different users.

Annotator

This section covers all the Javascript code used for the **Annotator** (for making annotations), **Corrector** (for viewing a corrected version of essays), and **Converter** (for converting WAMP-exported XML files to a desired format) modules. All Annotator modules use jQuery, and ierange-m2*custom.

Overview of SVN head directory structure (/annotator/Annotator/)

Only important files within the SVN **/annotator/Annotator** directory are listed here.

/ui.html

Contains all WAMP Annotator User Interface Elements. **Required by the Annotator module.**
A selectable (drop-down) list in #wamp_form also contains definitions of all possible annotation Mistake Types.

/converter.html & /converter.jse

Files essential for running the **Converter** module from Windows.
To run:

1. Make sure Windows Script Host (WSH) 5.6 or above is installed (preinstalled in Windows XP and above).
2. Make sure Internet Explorer 6 and above (IE8 recommended) is installed, and "Allow active content to run in files on My Computer" is turned on (setting found in Tools/Internet Options, in the Advanced tab, Security settings).
3. To run from Windows Explorer, double click on converter.jse. Script will run in the background (useful if GUI is preferred).
4. To run from a console window, enter "cscript.exe converter.jse" (in the root Annotator directory). Script will block console until completed (useful for running batch conversions).

converter.jse accepts 2 optional command-line arguments: e.g. "**cscript.exe converter.jse input.xml output.xml**"

- input.xml: The path to a WAMP-exported XML file to be converted.
- output.xml: The path to save the converted XML file to.

Load and/or save dialog boxes will appear if both/output argument(s) are/is missing (only for XP?).

/corrected.html

Sample file to show usage of Corrector module. Uses **/another.xml**.

/index.html

Sample file to show usage of Annotator module in complicated content (with paragraphs, bullet/numbered lists, pre-formatted text, and other inline formatting). Uses **/annotations.xml**.

/another.html

Another sample file to show usage of Annotator on a real-world essay. Uses **/another.xml**.

/js/jquery-1.3.2.js

jQuery 1.3.2 is the essential Javascript framework used by most, if not all classes.

/js/ierange-m2.custom.js

Based on the ierange-m2 library, ierange-m2*custom is the original ierange-m2, modified and debugged to sufficiently accurately emulate the required functionality of the W3C DOM2 Range object, and the Mozilla Selection object.
All changes done by me (Pua Jun Hong) are annotated with "PJH:" comments

/js/converter.js

This file defines the Converter static class used by the Converter WSH script to convert WAMP-exported XML files into the desired format.

Converter

Convert XML exported from WAMP into the desired format.

/js/corrector.js

This file defines the Corrector class used to display a corrected version of an annotated content body.

Corrector

Replace each annotated mistake in a content body with the suggested correction of that annotation.

/js/ui.js

This file defines the WAMP Annotator User Interface classes, which emulate the classical single-inheritance model.
Current class hierarchy: (fundamental ancestor) BaseUI -> PrintUI -> ViewUI -> EditUI (highest-level descendant).
(i.e. EditUI, in addition to editing functionality, also has printing, viewing and highlighting/labelling functionality inherited from all it's ancestor classes).

BaseUI

The base user interface class, which loads and prepares all UI Elements for use by subclasses, loads the annotations, and marks (highlights and labels) the mistakes in the specified content body.

PrintUI

Add printing table of annotations and indexes to labels, for easier referencing of annotations in printed output.

ViewUI

Add the functionality to select and view details of annotations.

EditUI

Add the functionality to create new annotations, and delete existing annotations.

/js/annotator.js

This file defines the core classes used by WAMP Annotator UI, Corrector and Converter.

Annotations

Load, manage, and save annotations in XML format.

Highlighter

Highlight the text of DOM Ranges.

Labeller

Add labels to highlighted Ranges.

Locator

Locate a position within a specified DOM subtree, converting the location to/from an address string.

Popup

Create absolutely-positioned popup "windows", with various options.

Whitespace

Tools to deal with (user-definable) whitespace in Strings.

XMLTools

Group of useful functions for parsing XML.

/css/annotator.css

The CSS file **required by all modules** (Annotator, Converter, Corrector), for the formatting of screen and print output.

/test/

Unit and integration test cases for annotator.js core classes available here. Uses the QUnit (jQuery unit testrunner) library to run the test cases.

Annotation XML Format

This is the intermediate format used by WAMP to store annotations specific to a single instance (single content body), by a single annotator.
It is also the format used by the **Annotations** class to internally store and manage annotations (in fact, it is the Annotations class that defines/creates this format, since the WAMP database only stores/retrives the XML as plain text, and is internally ignorant of the XML format of the annotations data).

Annotation XML DTD

```
<!DOCTYPE annotations [
<!ELEMENT annotations (mistake*)>

<!ELEMENT mistake (type, correction, comments)>
<!ATTLIST mistake id ID #REQUIRED>
<!ATTLIST mistake start CDATA #REQUIRED>
<!ATTLIST mistake end CDATA #REQUIRED>

<!ELEMENT type (#PCDATA)>
<!ELEMENT correction (#PCDATA)>
<!ELEMENT comments (#PCDATA)>
]>
```

Mistake Attributes: id

Unique identifier for the annotation (unique throughout entire annotations list).

Mistake Attributes: start, end

Addresses to the starting node-offset and the end node-offset that defines the DOM Range within the XML node that is annotated.
Please see http://www.w3.org/TR/DOM-Level-2-Traversal-Range/ranges.html for a proper definition of DOM Ranges.

Address Format

The address is a series of slashes and integers, followed by an optional ending dot and integers.
For example, for the Range "**/0/0.3**" to "**/1/0.2**" in the XML (node)
<body> Hello World </body>
marks the portion "**lo Wo**".
It is based on the Marginalia's pointer format (http://www.geof.net/code/annotation/technical), which is in turn based on the XPointer child sequence format (http://www.w3.org/TR/WD-xptr#child-seqs).

Each integer *n* after a slash "/" locates the position just before the *normalised n*th child Node (*not* Element) in the list of childNodes of the last located parent Node (usually an Element, since only Elements can have childNodes).
The interger *i* after the dot ".", if it exists, locates position just before the *normalised i*th character in the string of text (i.e. text offset) from the start of the last located parent

Node.

Note:
The following parts define how to calculate n and i respectively. Blue parts indicate the rationale for calculating in the given method, while red parts are only relavant when having to handle content with highlights and labels, inserted elements that are not part of the original content (as in the case for the Annotator module). When just parsing the original document to get the original range, without anything extra added in the mix, taking care of ignorable or normalisable elements is not necessary (since they do not exist in the original document).
One can assume that Locator.getAddressFromNodeOffset(), with no normaliseClasses[] or ignoreClasses[] defined, wo

Normalising n (integer after a slash '/')

The counting of node offset seems unnecessarily complicated, but there is meaning in the chaos. IE removes whitespace-only text nodes in between block-level elements (e.g. IE removes what cannot be seen by the user). Some XML parsers remove comment nodes, while others keep them in the DOM. Also there may a need to insert content (e.g. annotation markers) into the document, which is not part of the original annotated document (and thus must be ignored/accounted for) And, the act of highlighting a certain range of text (surrounding text nodes with a new tag) also changes the DOM tree of the original document. All these reasons create the need for n to be normalised as such.

There are 3 types of nodes to be considered: nodes that can be *ignored*, nodes that can be *normalised*, and *significant* nodes (i.e. all other nodes).

Significant nodes are non-ignorable and non-normalisable nodes, and each significant node increments the offset n once, when walking through the list of childNodes.
(e.g. **`<div>Abc<i>D</i>E</div>`**, the address to the location just before E is "**/2**")

Nodes that can be ignored are:
- Text nodes consisting of only whitespace.
- Element nodes with a defined as ignorable.
- All other non-text and non-element nodes (e.g. comments).

Nodes that can be ignored to not increment the offset n when walking through the list of childNodes.
(e.g. **`<div>A<!--comment--><p>C</p><b class="ignore">B</b>D</div>`**, assuming the class "ignore" is defined as ignorable, the address to the location just before D is "**/2**").

Nodes that can be normalised are:
- Text nodes
- Element nodes with a class defined as normalisable.

Adjacent nodes that can be normalised are considered as a single node, and thus *only increment the offset n once*, regardless of the number of normalisable nodes, when walking through the list of childNodes.
Note that nodes that can be ignored are not considered as part of the list of childNodes, and so a pair of normalisable nodes with a ignorable node in between are still considered adjacent to each other, and are thus only counted as once.
(e.g. **`<div>A<b class="ignore">B</b><p class="normalise">D</p><i>E</i>F</div>`**, assuming the class "ignore" is defined as ignorable, and class "normalise" is defined as normalisable, the address to the location just before E is "**/1**".)

Normalising i (integer after the dot '.')

The use of word-character offset (like Marginalia) is not used, because word boundaries cannot be reliably defined (Is "HelloWorld" 2 words or 1? What about Hello World "? Is "Alan's" and "one-word" 2 words or 1? What about other languages that do not use whitespace to demarcate word boundaries?). Thus, I can only use a normalised and condensed character offset, to be consistent across languages and various HTML/XML representations of the document. Left-trim is done, since Internet Explorer trims whitespace in text nodes at the start/end of block-level elements)

The text offset i is typically counted from the start of a group of adjacent nodes that can be normalised, and is the length of the left-trimmed and condensed text substring found from the start to offset i.

Condensing a string means to replace all groups of adjacent whitespace with a single space, and to left trim means to remove all adjacent whitespace at the start of the string.
(e.g. **`<div>  AB  CD  E</div>`**, assuming class "normalise" is defined as normalisable, the address to the location just before E is "**/0.6**")

Text within adjacent normalisable nodes are considered as consecutive (i.e. part of the same text string), and the offset can be counted continually across adjacent normalisable nodes' boundaries.
(e.g. **`<div>A<i class="normalise">B</i>C</div>`**, assuming class "normalise" is defined as normalisable, the address to the location just before C is "**/0.2**")

Text within ignorable elements (i.e. element nodes with a class defined as ignorable) with within and in between adjacent normalisable nodes are not counted as part of the text substring, and thus do not increment the text offset i.
Note that whitespace-only text nodes within adjacent normalisable nodes (not at the start, or at the end, but in between 2 other non-whitespace-only adjacent normalisable nodes) are not ignored, but instead considered significant (i.e. they are counted as part of the text substring to i).
(e.g. **`<div>A<i class="normalise">  <b class="ignore">C</b>D</i>  <i class="normalise">E</i>F<b class="ignore">G</b>H</div>`**, assuming the class "ignore" is defined as ignorable, and class "normalise" is defined as normalisable, the address to the location just before H is "**/0.6**").

Import XML DTD

```
<!DOCTYPE wamp [

<!ELEMENT wamp (module+)>

<!ELEMENT module (group+)>
<!ATTLIST module code CDATA #IMPLIED>

<!ELEMENT group (assignment+)>

<!ELEMENT assignment (description?,instance+)>
<!ATTLIST assignment title CDATA #IMPLIED>
```

```
<!ELEMENT description (#PCDATA)>
<!ELEMENT instance (context,annotation+)>
<!ATTLIST instance matric CDATA #IMPLIED>

<!ELEMENT context (#PCDATA)>
]>
```

Initial Export XML DTD

```
<!DOCTYPE wamp [

<!ELEMENT wamp (module+)>

<!ELEMENT module (group+)>
<!ATTLIST module code CDATA #IMPLIED>

<!ELEMENT group (assignment+)>

<!ELEMENT assignment (description?,instance+)>
<!ATTLIST assignment title CDATA #IMPLIED>

<!ELEMENT description (#PCDATA)>
<!ELEMENT instance (context,annotation+)>
<!ATTLIST instance matric CDATA #IMPLIED>

<!ELEMENT context (paragraph+)>

<!ELEMENT paragraph (#PCDATA)>

<!ELEMENT annotation (mistake*)>
<!ATTLIST annotation annotator CDATA #IMPLIED>


<!ELEMENT mistake (type,correction,comments)>
<!ATTLIST mistake start CDATA #REQUIRED>
<!ATTLIST mistake end CDATA #REQUIRED>

<!ELEMENT type (#PCDATA)>
<!ELEMENT correction (#PCDATA)>
<!ELEMENT comments (#PCDATA)>
]>
```

Notes

1. The start and end attributes in the mistake element follow the format described in Annotation_XML_Format.

WAMP Final (Converted) XML DTD

This is the format of the converted XML output of the **Converter** module (assuming the input WAMP-exported XML is correct in the first place).

```
<!DOCTYPE wamp [
<!ENTITY % Digits "CDATA">

<!ELEMENT wamp (module+)>

<!ELEMENT module (group+)>
<!ATTLIST module code CDATA #IMPLIED>

<!ELEMENT group (assignment+)>

<!ELEMENT assignment (description?,instance+)>
<!ATTLIST assignment title CDATA #IMPLIED>

<!ELEMENT description (#PCDATA)>
<!ELEMENT instance (context,annotation+)>
<!ATTLIST instance matric CDATA #IMPLIED>

<!ELEMENT context (paragraph+)>

<!ELEMENT paragraph (#PCDATA)>

<!ELEMENT annotation (mistake*)>
<!ATTLIST annotation annotator CDATA #IMPLIED>


<!ELEMENT mistake (type,correction,comments)>
<!ATTLIST mistake startParagraph %Digits; #REQUIRED>
<!ATTLIST mistake startOffset %Digits; #REQUIRED>
<!ATTLIST mistake endParagraph %Digits; #REQUIRED>
<!ATTLIST mistake endOffset %Digits; #REQUIRED>

<!ELEMENT type (#PCDATA)>
<!ELEMENT correction (#PCDATA)>
<!ELEMENT comments (#PCDATA)>
]>
```