LATEX GALLEY PLUGIN
FOR OPEN JOURNAL SYSTEMS

by

Shiyi Chen

a Report submitted in partial fulfillment
of the requirements for the SFU-ZU dual degree of
Bachelor of Science
in the School of Computing Science
Simon Fraser University
and
the College of Computer Science and Technology
Zhejiang University

SIMON FRASER UNIVERSITY AND ZHEJIANG UNIVERSITY
Spring 2010

APPROVAL

Name: Shiyi Chen

Degree: Bachelor of Science

Title of Report: LaTeX Galley Plugin for Open Journal Systems

Examining Committee: Name the committee chair!
Chair

Dr. Qianping Gu, Professor, Computing Science
Simon Fraser University
Supervisor

Dr. Ramesh Krishnamurti,
Professor, Computing Science
Simon Fraser University
examiner

Date Approved: _____

Abstract

This project is based on a renowned open source project named Open Journal Systems (OJS), which is used by many online journals world wide. Open Journal Systems is an enterprise level web application that is highly structured for extension. This project aims to extend its functionality by adding a fresh new plugin without modifying much original code. This newly added plugin expands the choices of galleys[3] by allowing LaTeX files to be uploaded as a galley. By reading through this report, you will get to know important structures of OJS, crucial development concepts adopted by OJS and most importantly how to develop a plugin for OJS.

Contents

List of Tables

List of Figures

List of Programs

Chapter 1

Introduction

In this chapter, I first introduce the background of this project, then describe this project, and finally conclude with an example of its usage.

Firstly, this project is only part of efforts of a greater project, Public Knowledge Project, which develops open source software aiming to make knowledge more accessible.

1.1 About Public Knowledge Project

The Public Knowledge Project (PKP) is a research based project at Simon Fraser University, Stanford University and the University of British Columbia. The project was established by Dr. John Willinsky in 1998 and funded by the federal governments of Canada and US. Its main purpose is to improve the quality of academic research by developing innovative online publishing and knowledge-sharing systems. The project currently has three modules in production: Open Archives Harvester, Open Journal Systems, and Open Conference Systems, and two in development: Lemon8-XML and Open Monograph Press[14].

1.2 About Open Journal System

So what is Open Journal Systems (OJS)? As stated in its website[13], "Open Journal Systems is a journal management and publishing system, which assists with every stage of the refereed publishing process, from submissions through to on-line publication and indexing. "

OJS provides an on-line management system that has finely grained indexing of articles

and supports functions such as searching, editing and reviewing articles. This system is useful for refereed research and improves the quality of research by allowing open access to journal publications[13].

Open Journal Systems has undergone several stages of development. Its latest development release is version 2.3.1, though this project is based on its latest stable release, which is version 2.2.4. OJS as well as all other software of the PKP team is open source and freely distributed under the GNU General Public License. Notably, it has already been adopted worldwide because of its customizability and extensibility.

A screen shot of an OJS Demonstration Journal is shown in Figure 1.1. More information

Figure 1.1: An OJS Demonstration Journal

about OJS can be found at its website: http://pkp.sfu.ca/?q=ojs.

1.3 Editorial and publishing process

To understand how OJS works and what this project is about, it is necessary to take a look at the editorial and publishing process of a journal in OJS. OJS supports multiple roles, including those of journal manager, author, and editor. An authorized OJS user can play any of these roles and perform specific tasks. See Figure 1.2 below for more details.

As shown in the Figure 1.2, this process consists of five main steps: (1)submission queue, (2)submission review, (3)submission editing, (4)issue management and (5)publication. Details of these five steps are also given in the figure.

1.4 Goal of this project

This project is related to the third step of the editorial and publishing process of a journal. In this step, a layout editor needs to prepare galleys of submitted articles in HTML, PDF, PS, etc so that editors and section editors may use them for a final electronic publication. These galleys are usually transformed from copy-edited versions of a submission using some external software (e.g., use Dreamweaver to create a HTML galley, Adobe Acrobat for PDF)[7]. In Figure 1.1, every article in the journal has two galleys, namely a HTML galley and a PDF galley. By clicking on the appropriate buttons, journal subscribers can view the article in the desired format.

The goal of this project is to allow layout editors to upload LaTeX files as a galley. To make LaTeX files more viewable, the system may automatically convert a LaTeX galley into a PDF file depending on its system setting. Users can then view either the original LaTeX files or a PDF file that is generated.

1.5 Project implementation

Regarding the implementation of this project, I decided to add a plugin to fit into the code structure of OJS and follow its conventions. Basically, this plugin performs a quick check during the submission process to ensure that the submission is indeed valid LaTeX, and then automatically generates a PDF output from the LaTeX source.

Important considerations for the plugin implementation are listed below:

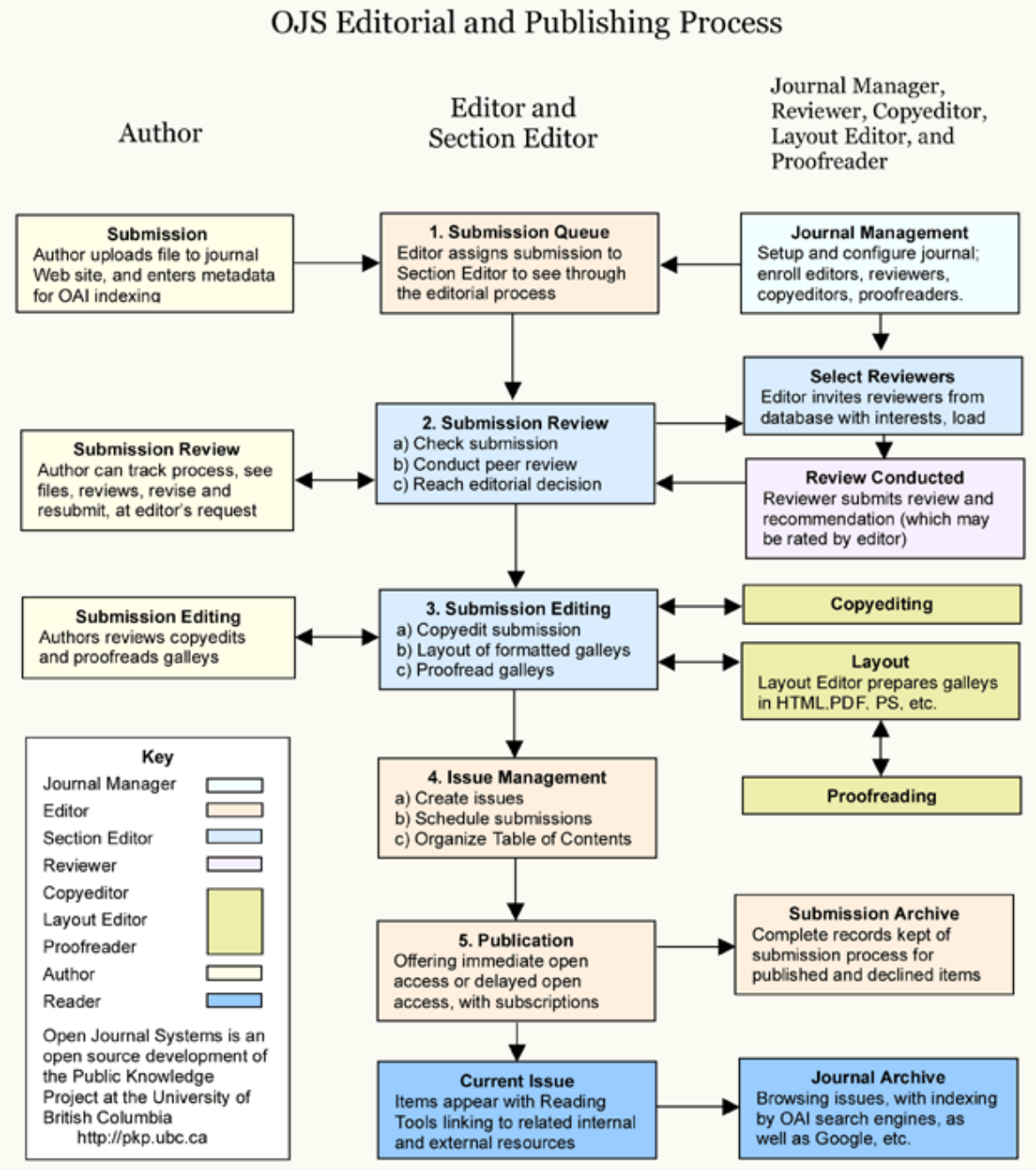

Figure 1.2: Editorial and publishing process[6]

- Layout editors should be required to upload a zipped/compressed document, as a document written in LaTeX may include multiple files and subdirectories. The exchange file format during the editorial process is ZIP or any other compression format for LaTeX galleys.
- The LaTeX galley plugin should unzip/decompress the uploaded document before verifying its LaTeX syntax or generating a PDF output.
- The plugin should be able to verify LaTeX galleys upon request, (e.g., by clicking a "view proof" button).
- It should be possible to enable and disable the plugin. If enabled, it should display the LaTeX galley in PDF format after the galley has been published.

1.6 A taste of the final product

Figure 1.3: A taste of the final product

It can be seen from Figure 1.3, in addition to the PDF and XML galleys, there is a LaTeX galley for the article named Test1. If PDF conversion is enabled and the LaTeX source is syntacticly correct, users can view the galley in PDF format by clicking the "LATEX" link.

The rest of the report is organized as follows:

Chapter 2 gives some preliminaries, including some technical features of Open Journal Systems that are important to the development of the plugin.
Chapter 3 describes some of the most important concepts of plugin development in OJS.
Chapter 4 describes the implementation of this plugin.
Chapter 5 concludes the report.

Chapter 2

Preliminaries

This chapter gives some preliminaries, including some technical features of Open Journal Systems that are important to the development of the plugin.

2.1 Technologies

This section describes some core technologies used in developing OJS and this project.

2.1.1 PHP language

PHP is the abbreviation for Hypertext Preprocessor, which is a scripting language widely used for developing dynamic web applications[4]. Most of the code for this project is written in PHP. Its syntax is more or less similar to that of C++ and Java. One distinguishing feature of PHP is that it is weakly typed, in which programmers do not need to define variables and their types (e.g., integer type or float type) before using them. It can help save some time to code a program, but on the other hand it usually takes more time to debug it.

Starting from a set of Common Gateway Interface binaries written in the C programming language[4], PHP has become a general-purpose programming language that can support the object oriented programming paradigm. OJS uses this feature of PHP a lot and is highly object oriented.

Figure 2.1 gives a sample of PHP code snippet.

Every article in OJS is represented as an ArticleFile Object, which extends and inherits

```php
class ArticleFile extends DataObject {

    /**
     * Constructor.
     */
    function ArticleFile() {
        parent::DataObject();
    }

    /**
     * Return absolute path to the file on the host filesystem.
     * @return string
     */
    function getFilePath() {
        $articleDao = &DAORegistry::getDAO('ArticleDAO');
        $article = &$articleDao->getArticle($this->getArticleId());
        $journalId = $article->getJournalId();

        return Config::getVar('files', 'files_dir') . '/journals/' . $journalId .
        '/articles/' . $this->getArticleId() . '/' . $this->getType() . '/' . $this->getFileName();
    }
```

Figure 2.1: A PHP code snippet

features of DataObject. As shown in the figure, ArticleFile class has a member method "getFilePath" which returns the absolute path to the article on the host file system.

2.1.2 Smarty template system

For the purpose of separating the PHP code from the HTML tags, Smarty Template System is introduced as a presentation layer in the OJS, which renders the user interface of OJS[15].

Every web page of OJS is formed by using Smarty templates. A template is a combination of HTML tags, Smarty variables and Smarty functions. HTML tags are responsible for formatting the structure of a web page. Smarty variables are often used to provide the page content, for example, a web page displaying an article should have `$headline`, `$abstract`, `$author` and `$body` variables which will be assigned values according to a specific article. Smarty functions are either predefined in the system or defined by developers, and can be used by simply calling them and passing parameters in a template. Smarty functions help hide the application logic in the presentation layer.

All the template files in OJS have a tpl suffix. A sample template of OJS is shown in Figure 2.2. A typical smarty variable is `$pageToDisplay`, which is compared to strings "active" and "completed" and decide the class attribute of the list tag <li> accordingly. The code segment {`if ($pageToDisplay == ''active'')`} `class=''current''` {`/if`} will render the string "class="current"" in the template if the `$pageToDisplay` equals "active".

```
{assign var="pageTitle" value="common.queue.long.$pageToDisplay"}
{include file="common/header.tpl"}

<ul class="menu">
    <li{if ($pageToDisplay == "active")} class="current"{/if}>
        <a href="{url op="index" path="active"}">
            {translate key="common.queue.short.active"}
        </a>
    </li>
    <li{if ($pageToDisplay == "completed")} class="current"{/if}>
        <a href="{url op="index" path="completed"}">
            {translate key="common.queue.short.completed"}
        </a>
    </li>
</ul>

<br />

{include file="author/$pageToDisplay.tpl"}

<h4>{translate key="author.submit.startHereTitle"}</h4>
{url|assign:"submitUrl" op="submit"}
{translate submitUrl=$submitUrl key="author.submit.startHereLink"}<br />

{include file="common/footer.tpl"}
```

Figure 2.2: A sample Smarty template

So the list tag is formatted as <`li class=‘‘current’’`> in the page output.

A typical Smarty function is “translate”. The code segment
{`translate key=‘‘common.queue.short.active’’`}
translates the string indexed by the “key” parameter, which is “common.queue.short.active”. The reason that OJS does not put the string directly in this template file is to make it easy to manage such strings and change them upon request. Readers may refer to Section 2.4 Internationalization for more details.

2.1.3 ADODB database abstraction library

ADODB is an open source database abstraction library for PHP[1]. It is used to provide APIs that make it easy to access a database. The API is simple and straightforward. After getting a database instance by calling `$dbconn = &ADONewConnection($driver)`, developers can connect to and disconnect from that database and execute queries by calling its member functions. For example, `$dbconn ->Execute(‘‘select a from b’’)` will execute that query string and return a result set. As in this project, a database instance is already established, which is used globally to interact with the database in every data access object (DAO).

2.2 Model View Controller structure

As stated before, OJS is highly object oriented. In order to make it more extensible, flexible and robust, starting from OJS 2.X, OJS is also heavily structured in the Model View Controller (MVC) pattern. Models are usually objects that represent database entities, views are user interfaces that interact with users, and controllers contain the application logic and control how models are rendered into views.

The source code of OJS can be classified into six categories, which are Smarty templates, page classes, action classes, model classes, data access objects and support classes, which can also be divided according to the MVC classification[8].

- Smarty templates, based on which HTML pages are rendered, are the views.
- Page classes and action classes are the controllers. The main responsibility of page classes is to delegate the required processing to other classes after receiving user requests from web browsers and call up the appropriate Smarty template to generate a response. Action classes are used by page classes to perform non-trivial processing of user requests. Page classes always have the name pattern [PageName]Handler.inc.php, while action classes have the name pattern [ActionName]Action.inc.php.
- Model classes, which are subclasses of the DataObject class and represent database entries in memory, are the models. They are used by controllers to update the views, while controllers use data access objects to update them. Data access objects (DAOs) are those associated classes that generally provide update, create, and delete functions for their model classes. For example, an article model contains all the data of an article, which is fetched from the database by an article DAO, and page classes (the controller) use getTitle or getContent member methods of the article model to render a Smarty template.

Support classes just provide common functionality that can be used in any class.

2.3 URL request handling

Unlike normal web applications, where a Uniform Resource Locator (URL) is directly used to locate its source file, a URL in OJS does not correspond to the location of its source script

in the file system. In order to understand how OJS handles HTTP request, it is necessary to understand one of the Common Gateway Interface environment variables, PATH INFO, which stores extra path Information following a script's path in the URL.

All URLs of OJS have the following pattern:
`site_name/index.php/journal_id/page_name/operation_name/arguments....`
The string "index.php" always appears halfway through every URL. ALL requests in OJS will be forwarded to the root script index.php first, and then be delegated to a specific page class (the controller) according to the value stored in PATH INFO, which is part of the URL string:
`journal_id/page_name/operation_name/arguments....`
After parsing PATH INFO in index.php, OJS will delegate the control to a method named "operation_name" in the page class `/pages/page_name/page_nameHandler.inc.php`. The partial string "arguments" is passed to that method as parameters[9].

2.4 Internationalization

Like many software programs that have global users, OJS can be easily internationalized and adapted to multiple languages. It separates the contents of its web pages from their Smarty templates and puts together these contents into resource files. These contents are indexed by keys and can be used by calling Smarty function "translate", as shown in Section 2.2. Usually there is one copy of resources files for each language.

There are multiple locations that are specifically reserved for storing such resources files in OJS. For example, the locale/en_US folder is used to store the main resource files written in US English and the help/ja_JP folder contains resource files for help pages written in Japanese.

2.5 Caching

OJS uses the caching mechanism to speed up database access. Whenever OJS wants to fetch data from a database, it will first check if there is cached data available. If available, OJS will use the cached data if it is up to date. On the other hand, whenever OJS wants to update database tables, it will create or update its cached data as well. In order to manage the cache, OJS implements a global class called "CacheManager". All the cached data is

put together in the cache folder. It is useful to check the cache folder to make sure the caching routines work properly.

Chapter 3

Plugin Development

This chapter describes some of the most important concepts of how a plugin is generally developed in OJS.

3.1 Plugin and hook

Plugins are pieces of code that extend or modify OJS. If they are put in an appropriate directory, OJS will execute the code and the plugins will take effect by registering themselves.

There are many plugins that have been implemented in OJS. Plugins are all placed in the plugins folder. Each plugin extends a base class that defines a category and is responsible for implementing methods of the class. Existing categories include Authorization Plugins, Block Plugins, Citing Format Plugins, Gateway Plugins, Generic Plugins, Implicit Authentication Plugins, Import/Export Plugins, Payment Plugins, Report Plugins and Theme Plugins[10]. The base classes of all plugins are in the classes/plugins folder. If you consider Authorization Plugins, its base class is `classes/plugins/AuthPlugin.inc.php`, which defines all the user authentication tasks that are needed to be implemented, such as CreateUser and DeleteUser.

Another important concept is the hook. A hook is some code that intercepts function calls and alters or augments the default behaviours of these functions. Thus the way that a plugin integrates with OJS is by registering hooks, which overrides some built-in behaviours without changing the code elsewhere.

3.2 Hook registration and callback

Plugins enhance OJS by registration hooks. Functions that implement the enhanced behaviour should be passed to hooks as well when hooks are being registered. Such functions are called callbacks, which are called in a hook after it intercepts function calls. Callbacks are formally known as references to executables. Plugins pass references of its callback functions to a hook when registering the hook.

An example of hook registration is shown below:

```
HookRegistry::register(
    'Templates : : Manager : : Index : : ManagementPages',
    Array(&$this, 'managePage')
);
```

The first parameter of the register function is "Templates::Manager::Index::ManagementPages", which is the hook name. The second parameter is a reference to the callback function with the function name "managePage". The "
t &$this" argument is a reference to the object where the callback function "managePage" is a member. In the body of HookRegistry::register, it keeps an array of references to callback functions that are indexed by hook names.

An example of calling that hook is shown below:

```
HookRegistry::call(
    'Templates : : Manager : : Index : : ManagementPages',
    array(&$article, &$galley, &$fileId)
)
```

The first parameter is the name of the hook that HookRegistry::call is calling. The second parameter is an array of arguments that will be passed to callback functions. The body of HookRegistry::call will call those callback functions that have been registered against this hook name.

The signature of the callback function managePage is shown below.

```
function managePage ($hookName, $args) {
//code here
}
```

The first parameter gets the name of the hook that transfers control to this function. The second parameter "`$args`" gets the second parameter of HookRegistry::calls, which is "`array(&$article, &$galley, &$fileId)`" in this case. In the body of "managePage", the application logic for improving OJS is defined.

Moreover, there can be many callback functions registered against the same hook. HookRegistry::call calls such callback functions sequentially. If the current callback returns false, other callbacks after this one will still have a chance to handle the hook call. Otherwise, the HookRegistry will consider the hook call to be successfully handled, stop calling the remaining callbacks and exit the hook call. So the return value of a callback function is very important.

3.3 HTTP form and form object

In order to customize plugins, setting pages are usually required by plugins. See Figure 3.1 for a sample setting page.

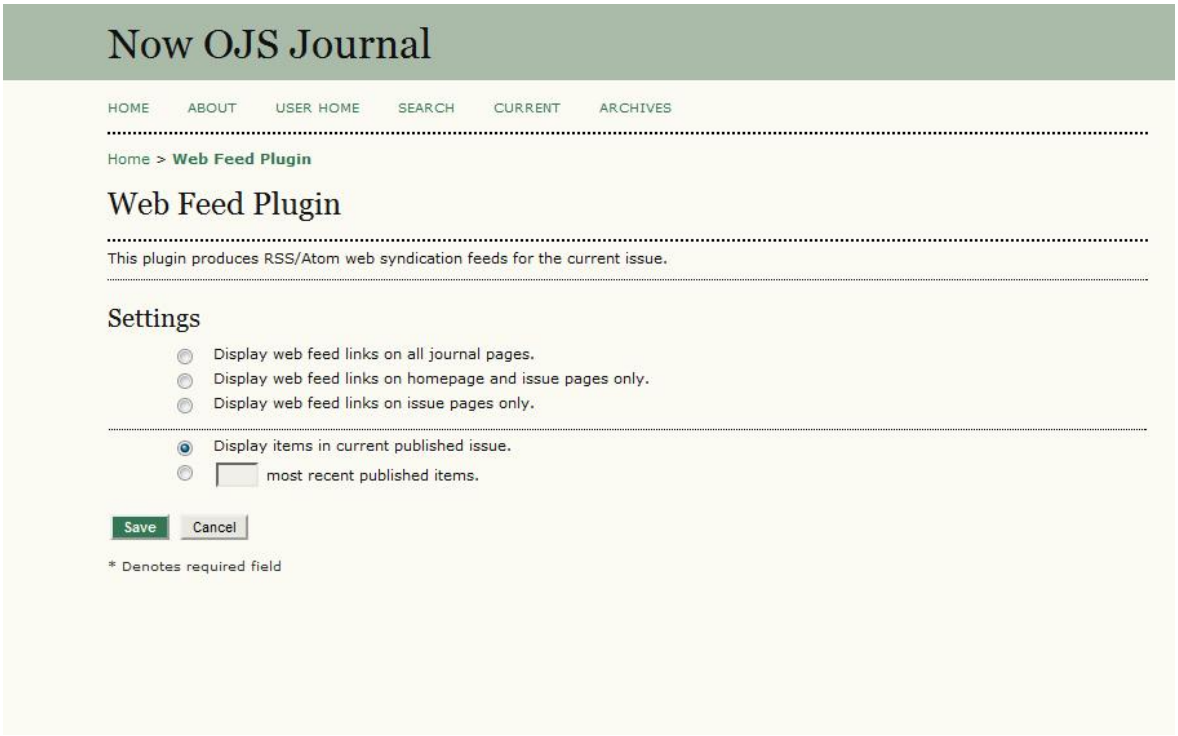

Figure 3.1: A sample setting page

Normally, a page like the one in Figure 3.1 usually contains HTML forms with text fields to be filled in or options to be selected, which will be submitted to the server. In OJS, data in these HTML forms are also be stored in a database after they are submitted. OJS handles form data with form objects. A typical form object extends the Form class

`classes/form/Form.inc.php`, which defines all the functions used in processing forms such as validation and error handling[11].

Generally, a subclass of Form overwrites initData, readInputData and executes methods to customize specific forms.

- initData: When users of OJS go to a setting page, this method is called to initialize the form with the current set of data.
- readInputData: After a form is submitted, this method is called to read the input data and ensure the required data are not blank.
- Execute: This method is called at places where the input data can be committed, thereby updating both the cache and the database.

3.4 Plugin management

Figure 3.2 shows OJS's plugin management page .

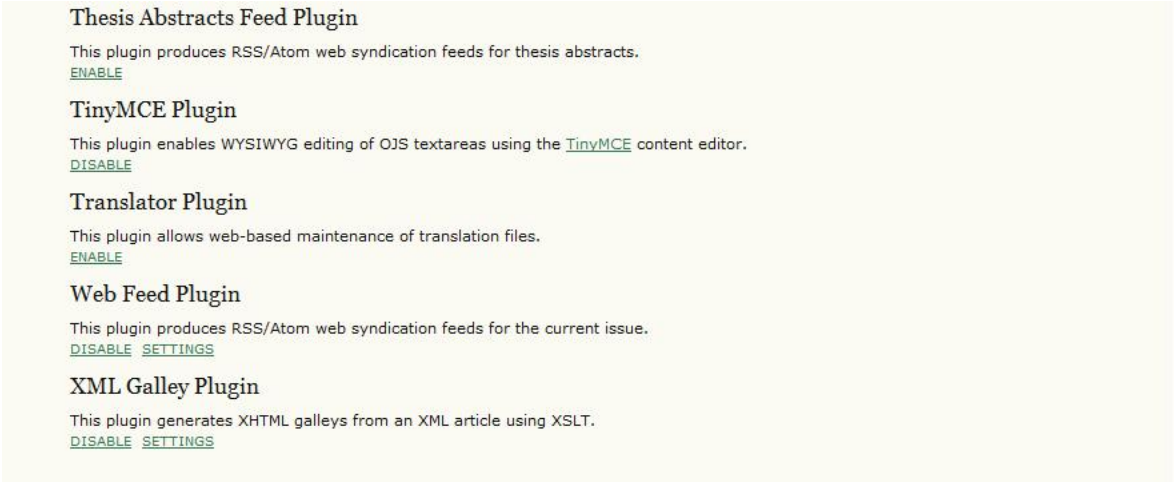

Figure 3.2: OJS's plugin management page

Those underlined links such as "disable", "settings", "enable" all have a URL pattern of `/index.php/journal_id/manage/plugin /plugin_category/plugin_name/action_verb`. As said in Section 2.3 URL request handling, these links will all be processed by a method named "plugin" in the page class located at `pages/manager/manageHandler.inc.php`. The "plugin" method takes care of all plugin-specific management by calling a common member

function for all plugins, namely `manage($verb, $args)`. The "action_verb" string in the URL is passed to the argument `$verb` of the manage method. Common values of `$verb` are "enable", "disable" and "setting". The manage methods manage plugins according to these action verbs.

Code below shows a sample structure of the manage method, which processes two action verbs named func1 and func2[12].

```
function manage($verb, $args) {
    if (!parent::manage($verb, $args))
        switch ($verb) {
            case 'func1':
            // Handle func1 here.
            break;
            case 'func2':
            //Handle func2 here.
            break;
            default:
            return false;
        }
    return true;
}
```

Chapter 4

Implementation

With most of the background knowledge covered, I now describe the implementation of this plugin in this chapter.

4.1 Environment setup

The first step in implementing a plugin is to set up the development environment. Following software programs are required to host a PHP website: PHP executables for running PHP scripts, MySQL[5] for a database server, Apache[2] for an HTTP Server. Generally, it is very time consuming to install and configure these software programs correctly before they can run a website. However, I used a program called WAMP[16] which combines all of these software programs and contains a user interface which is intuitive to use.

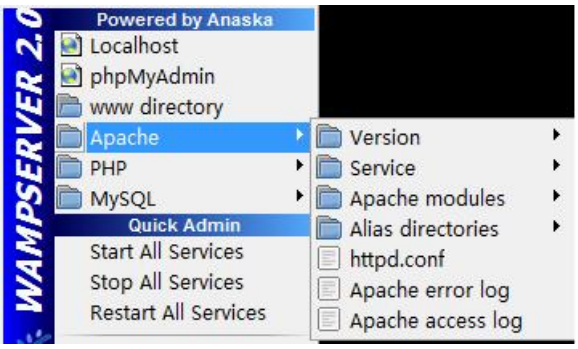

Figure 4.1: WAMP console

In Figure 4.1, WAMP has a console which can be used to easily configure Apache, PHP and MySQL.

After WAMP has been installed, a website can be set up by simply putting its source code under WAMP's web root directory named "WWW". In order to develop and debug PHP code, an integrated development environment is also required. I combined Eclipse with the PHP Development Tools plugin to develop the project. The debugger used for PHP is the Zend Debugger[17]. As always, all these software programs should be configured properly before they can be used.

4.2 Structure and dependence

As stated in Section 3.1, the source code of a plugin should be put in the plugins folder and every plugin belongs to one of the existing categories. OJS categorizes different plugins by putting them in folders whose names are their specific categories, which are placed one level below the plugins folder. As the LaTeX galley plugin is in the category "generic", it is put in the plugins/generic folder. The folder directly containing its source code is named "latexGalley". It has a code structure and dependence as follows:

4.2.1 Locale folder

This folder has many XML files (resources files), which store all the indexed strings needed by this plugin to be translated into various languages

4.2.2 Index.php

The loader stub is responsible for instantiating and returning the plugin object LaTeXGalleyPlugin.inc.php

4.2.3 LaTeXGalleyPlugin.inc.php

This is the main class representing the LaTeX galley plugin, which is also a subclass of GenericPlugin. This script rewrites some behavior dependent methods inherited from its parent class. Some of the most important methods are

- function `register($category, $path)`
  This is the function that will be called when OJS loads plugins. Plugin are initialized

here as well as hooks are registered here. Hook registration defines behaviors that this plugin wants to improve and modify. Hooks which are registered in this plugin are listed below:

`ArticleGalleyDAO::_returnGalleyFromRow`
The default behaviour of this hook is to return a specific galley model from a galley entry in the database. There are many galley classes defining different kinds of galleys. For example, ArticleXMLGalley, ArticleHTMLGalley and ArticleLaTeXGalley are all galley models. This plugin should return an ArticleLaTeXGalley model.

`ArticleHandler::viewFile`
`SubmissionEditHandler::viewFile`
`ArticleHandler::downloadFile`
As their names indicate, these hooks are used for viewing and downloading files. By registering against these hooks, the plugin can deal with specific cases related to viewing and downloading LaTeX galleys. When the plugin is enabled, its correct behavior should be that a PDF file is returned when the user wants to view or download a LaTeX galley.

- `manage($verb, $args)`
  This is the function used for plugin management. As described in Section 3.4, page classes (the controllers) manage plugins by calling this method.

4.2.4 settingsForm.tpl

This Smarty template describes the setting page of this plugin, which contains some HTML tags, Smarty variables and functions. This is the view layer of the MVC structure.

The page rendered from this template contains an HTTP form as described in Section 3.3. In order to successfully convert a compressed LaTeX galley to a PDF document, external unzip and pdfLaTeX executables are required. Unzip is used to decompress the LaTeX galley and pdfLaTeX to transform the LaTeX sources to a PDF document. The HTTP form for setting paths to these two commands is defined using a mix of HTML tags and Smarty variables in the template.

4.2.5 LaTeXGalleySettingForm.inc.php

This class is a subclass of class Form, which represents a form object described in Section 3.3. It processes all the data submitted by the setting page rendered using the template settingForm.tpl and can initialize the form, validate the form and display the template after initializing its form.

4.2.6 ArticleLaTeXGalley.inc.php

This class is a sub class of ArticleGalley and is the model class for LaTeX galleys. It has all the data fields that represent a LaTeX galley and all the methods that deal with LaTeX galleys. For example, its main methods are unzipping a zipped LaTeX submission and automatically converting LaTeX files to PDFs.

4.2.7 ArticleLateXGalleyDAO.inc.php

This is a data transfer object used by the model class ArticleLateXGalley to interact with the database (e.g. inserting, deleting LaTeX galleys). At present, there is no code implemented in this file, as the functionality inherited from its parent class ArticleGalleyDAO is sufficient. This script is reserved for further extensions of the plugin.

4.2.8 Transform folder

This folder has two files test.tex and test.zip which are used to test whether the external unzip and pdfLaTeX commands are set properly.

4.3 Core functionality

4.3.1 Plugin management

Enable and disable

As described in Section 3.4, it should be possible to enable or disable plugins, which can be achieved easily by setting a flag of the plugin to either true or false. Every time the code of that plugin is processed, the flag is checked first. If it is set to false, that part of the code is simply ignored.

Settings

If the "settings" verb is hit, OJS will go to the setting page of the plugin, as shown in Figure 4.2. In addition, if users click the save button, the request will go to the same page as well. In HTML, a form submits itself by transferring its data to its destination page. So it is easy to distinguish if the setting page is for saving the setting or not by checking if there is a field called "save" in the HTML request.

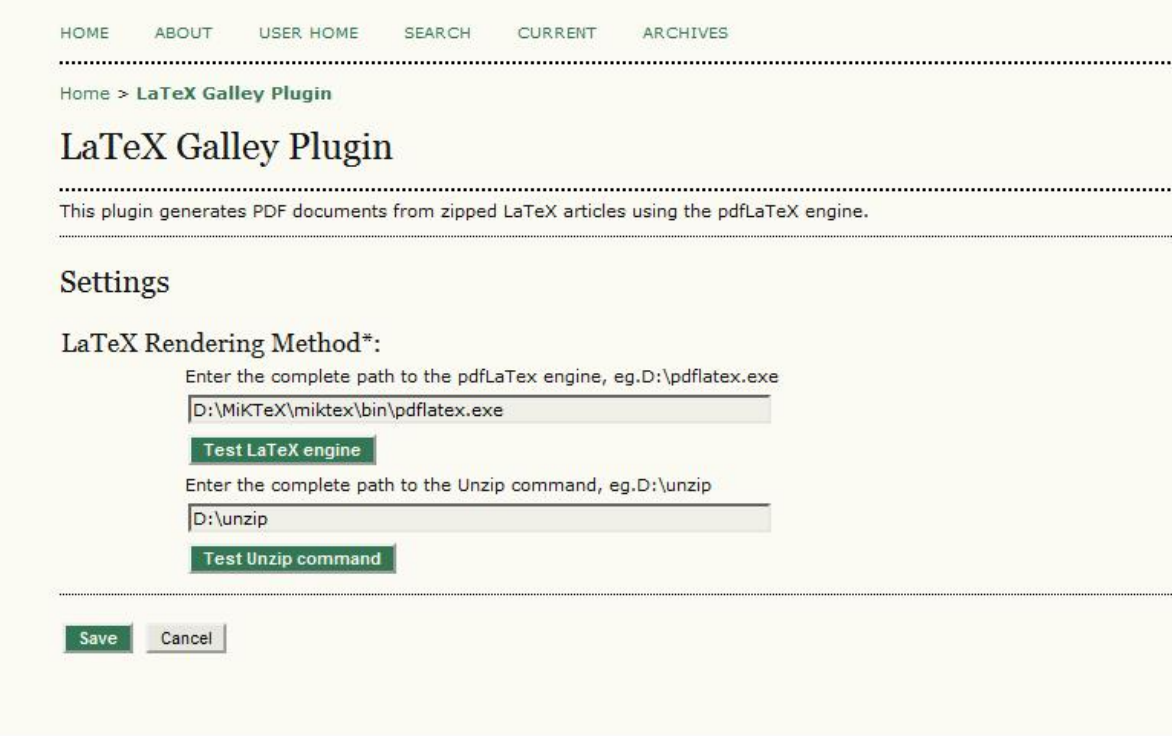

Figure 4.2: The setting page of the plugin

Figure 4.3 shows the code of the "setting" verb. The `$form` object is an instance of the LaTeXGalleySettingForm class. Lines 222 to 228 deal with the case of saving the settings. After line 225 validates the submitted data (blank strings or not), line 226 commits the data to the database. Lines 224 and 230 `$form->initData()` initialize the form with the current set of the data, while lines 228 and 231 `$form->display()` render the template settingForm.tpl.

```
220             case 'settings':
221                 // if we are updating settings
222                 if (Request::getUserVar('save')) {
223                     $form->readInputData();
224                     $form->initData();
225                     if ($form->validate()) {
226                         $form->execute();
227                     }
228                     $form->display();
229                 }else{
230                     $form->initData();
231                     $form->display();
232                 }
233                 return true;
```

Figure 4.3: The "setting" verb

Testing verbs

In addition to the "enable", "disable" and "settings" verbs, the plugin also has "testPdfLaTeX" and "testUnZip" verbs corresponding to the buttons "Test LateX engine" and "Text Unzip command" in the setting page shown in Figure 4.2.

These two buttons will test whether the pdfLaTeX engine and the unzip command are valid by applying them respectively to the test.tex and test.zip files in the transform folder. The PHP code `exec($command, $contents, $status)` can be used to execute shell commands. After examining their return values, a form object of LateXGalleySettingForm will be used to redraw the page, which will show the test results.

Figure 4.4 shows the case of a successful test on the LaTeX engine.

Part of the management code on the "testUnzip" verb is shown below.

```
1   $journal =& Request::getJournal();
2
3   $templateMgr = &TemplateManager::getManager();
4   $templateMgr->register_function('plugin_url', array(&$this, 'smartyPluginUrl'));
5
6   $this->import('LateXGalleySettingsForm');
7   $form =& new LateXGalleySettingsForm($this, $journal->getJournalId());
8
9   switch ($verb) {
10      case 'testUnZip':
11          $unzipEngine=$this->getSetting($journal->getJournalId(), 'externalZipEngine');
12          $destination=dirname($_SERVER['SCRIPT_FILENAME']) .
```

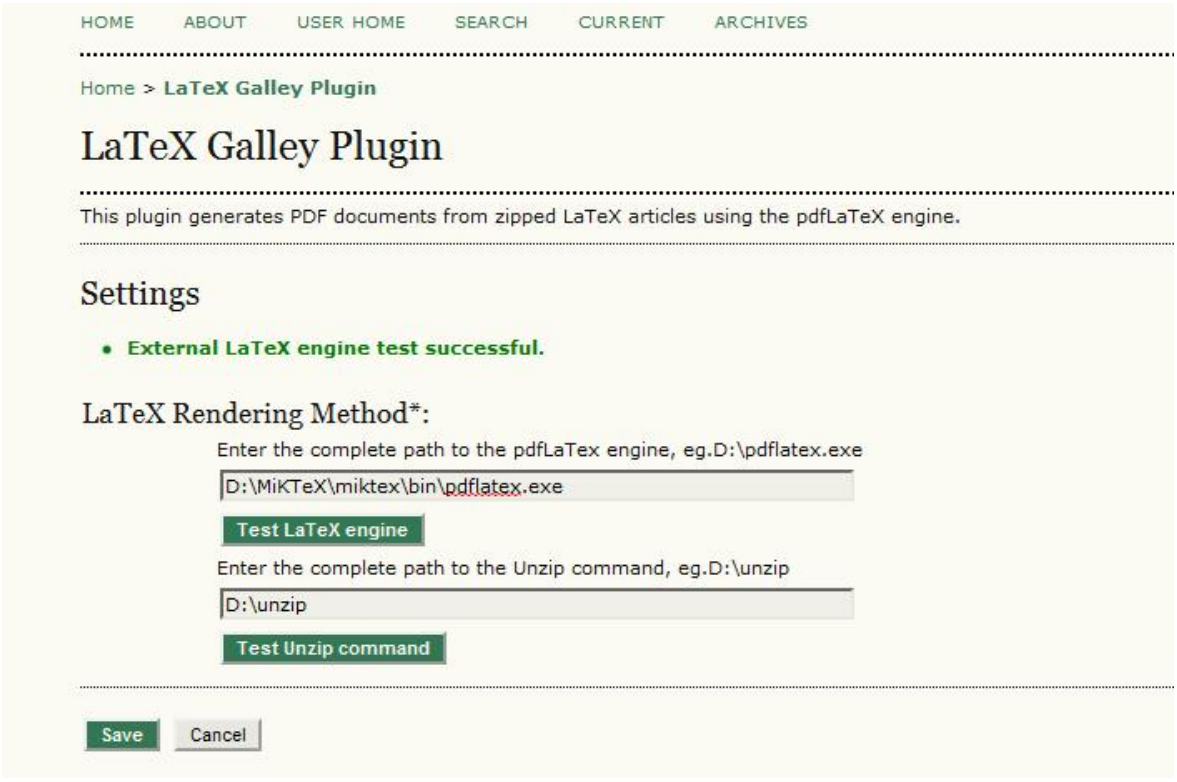

Figure 4.4: A successful test on LaTeX engine

```
                            DIRECTORY_SEPARATOR . $this->getPluginPath() . '/transform';
13          $zipFile= dirname($_SERVER['SCRIPT_FILENAME']) .
                            DIRECTORY_SEPARATOR .$this->getPluginPath() . '/transform/test.zip';
14          $unzipCommand = $unzipEngine.' -o '.$zipFile.' -d '.$destination;
15          $unzipCommand = str_replace('\\','/',$unzipCommand);
16          if( !ini_get('safe_mode') )    {
17              $unzipCommand = escapeshellcmd($unzipCommand);
18          }
19          exec($unzipCommand, $contents, $status);
20          if ($status != false || $contents =='') {
21              $form->addError('content',
                        Locale::translate('plugins.generic.latexGalley.settings.externalZipEngineFailure'));
22          } else{
23              $templateMgr->assign('unzipTestSuccess', true);
24          }
25          $form->initData();
26          $form->display();
27          return true;
```

Line 3: `$templateMgr` is the variable which manages the Smarty template system. It is used to register a Smarty function "plugin_url" for use in templates, as shown in line 4. The Smarty function "plugin_url" is implemented by a callback function named "smartyPluginUrl", which is referenced by `&$this->smartyPluginUrl`. In the "settingsForm.tpl" file, there is a case of applying the Smarty function "plugin_url", `<href=''{plugin_url path=''testUnZip''}''>`, which constructs the URL for testing the unzip command for the plugin. Similarly, line 23 assigns the Smarty variable "unzipTestSuccess" the Boolean value true, which will be picked up in settingsForm.tpl as well. Line 10 to 18 constructs the unzip command. Line 19 executes it. Lines 20 to 24 modify the template to reflect the execution result. Lines 25 and 26 initialize the form and display the template.

4.3.2 View or download a LaTeX galley

When a user wants to view or download a galley, OJS will try to call the hooks ArticleHandler::viewFile or ArticleHandler::downloadFile. The LateXGalleyPlugin class implements callback methods and registered them against these hooks, so that when users want to view or download a LaTeX galley, these callback methods will get called.

The registration code is listed as follows:

```
HookRegistry::register( 'ArticleHandler::viewFile', array(&$this, 'viewLateXGalleyFile') );
HookRegistry::register( 'ArticleHandler::downloadFile', array(&$this, 'viewLateXGalleyFile') );
```

The callback method that is registered against these hooks is "viewLateXGalleyFile". If the article is a LaTeX galley, this method will try to convert the LaTeX galley to a PDF document and return it for viewing or downloading. In the body of "viewLateXGalleyFile", it uses the model class ArticleLateXGalley.inc.php who actually takes care of the application logic of converting a LaTeX galley.

The detailed logic is as follows:

First step: Check if there exists a PDF cache of the LaTeX galley. If the result is true and the cache is up to date, it will simply return that PDF. Otherwise, a cache miss event is triggered and the flow goes to the second step.

Second step: Use the external unzip command to unzip the LaTeX galley and the external pdfLaTeX command to convert the LaTeX galley to a PDF document. In order to use pdfLaTeX, the main LaTeX file of the galley should be passed to pdfLaTeX as a parameter. As there is no easy way to know which file is the main LaTeX file, the convention here is that a zipped LaTeX galley to be uploaded should share the same name as its main LaTeX file (Note: the name of a galley will be changed when it is uploaded and stored in OJS, so a bookkeeping of the original name is required.). Also, the main LaTeX file should be at the top level of the directory hierarchy after being unzipped. For example, if a zipped LaTeX galley has the name main.zip, its main LaTeX file should have the name main.tex and should be located at main.zip/main.tex when zip files are interpreted as folders. If the PDF document is successfully generated, it will be returned to the user and put into the cache for future use; otherwise, the original zipped galley will be returned.

The key to realize the logic above is to use the PHP method "`exec($command, $contents, $status)`" cleverly, especially when constructing the command variable `$command`. For example, for security reasons, the plugin should escape any characters in the commands of either pdfLaTeX or unzip that might be used to trick the system into executing arbitrary commands. The solution is to apply a method called "escapeshellcmd" to the command strings before executing them. Another case is that the the name of an uploaded LaTeX galley can contain spaces. Such a name is generally valid in an operating system, but will cause problems when being used to execute shell commands.

4.3.3 Verify LaTeX syntax

The way to verify the LaTeX syntax of a LaTeX galley is to use pdfLaTeX to convert the LaTeX galley to a PDF document. If it succeeds, it means that the LaTeX galley has a valid

syntax; otherwise, even if the LaTeX galley plugin is enabled, the original zipped LaTeX sources will be returned when journal subscribers want to view a LaTeX galley.

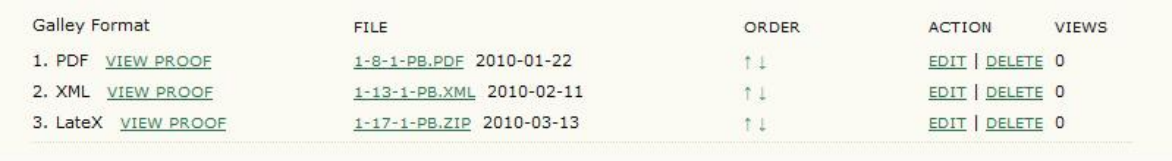

Figure 4.5: The "VIEW PROOF" link

During the editorial process of OJS, for layout editors who prepare galleys, there is a "VIEW PROOF" link besides every galley in the editing page, which can be used to check the validity of a LaTeX galley (See Figure 4.5). As the logic of verifying LaTeX syntax is the same as viewing or downloading LaTeX galleys, I solved the problem of verifying LaTeX syntax by making the "VIEW PROOF" link call a hook named "SubmissionEditHandler::viewFile" and registered the callback function "viewLateXGalleyFile" against the hook. Its registering code is

```
HookRegistry :: register( 'SubmissionEditHandler::viewFile', array(\&\$this, 'viewLateXGalleyFile') ).
```

Chapter 5

Conclusions

Starting from a rough idea provided by the PKP team at SFU, I have implemented the project in accordance with the protect proposal.

5.1 About the implementation

As the project is based on a pre-existing large code base, I had to understand the original code and try to fit my code into its structure and follow its conventions. A lot of time was spent on learning the system by trial and error, which was by tracing the flow of the system step by step or making a small modification to the system and seeing its result. One of the implementation requirements was to reuse the code as much as possible for clarity and consistency. For example, OJS already has features to interact with the database (e.g., galleys will be stored in the database after being uploaded). In order to deal with cases related to LaTeX galleys, I reused the original database routines by adding some code specific to LaTeX.

This project was supervised by the PKP team. We had weekly or biweekly Skype calls to check the progress of the project. They also offered some guidance on the technical implementation which was quite helpful.

5.2 About further improvement

There are many possible ways of improving the final product. Two things worth further consideration are listed below.

- Make commands MakeIndex and BibTeX available to LaTeX galleys. The MakeIndex command can make an index of the generated PDF document from LaTeX sources, and the BibTeX can generate the bibliography. In order to accomplish this, two more fields should be added to the setting page of this plugin. See Figure 4.2 for the setting page.

- After converting a LaTeX galley to a PDF document, delete those files that are a result of decompressing the LaTeX galley. Such files are no longer required, but will take up disk storage and possibly lead to instances of duplicate filenames as multiple galleys are stored in the same location.

Bibliography

[1] *ADOdb Database Abstraction Library for PHP*. http://adodb.sourceforge.net/.

[2] *The Apache HTTP Server*. http://httpd.apache.org/.

[3] *Galleys*. http://en.wikipedia.org/wiki/Galleys.

[4] *Hypertext Preprocessor Language*. http://en.wikipedia.org/wiki/PHP.

[5] *MySQL Database*. www.mysql.com/.

[6] *OJS in an Hour*. http://pkp.sfu.ca/files/OJSinanHour.pdf. page 12.

[7] *OJS in an Hour*. http://pkp.sfu.ca/files/OJSinanHour.pdf. page 156, Layout Editors.

[8] *OJS Technical Reference*. http://pkp.sfu.ca/ojs/OJSTechnicalReference.pdf. pages 26-35.

[9] *OJS Technical Reference*. http://pkp.sfu.ca/ojs/OJSTechnicalReference.pdf. pages 12-13.

[10] *OJS Technical Reference*. http://pkp.sfu.ca/ojs/OJSTechnicalReference.pdf. pages 41.

[11] *OJS Technical Reference*. http://pkp.sfu.ca/ojs/OJSTechnicalReference.pdf. pages 29.

[12] *OJS Technical Reference*. http://pkp.sfu.ca/ojs/OJSTechnicalReference.pdf. pages 44-45, Plugin Management.

[13] *Open Journal Systems Overview*. http://pkp.sfu.ca/?q=ojs.

[14] *Public Knowledge Project Overview*. http://pkp.sfu.ca/node/1410.

[15] *Smarty Template Engine*. http://www.smarty.net/.

[16] *The Wamp Server*. http://www.wampserver.com/en/.

[17] *The Zend Debugger*. www.zend.com/topics/Debugger-Install.pdf.