WikiMap

# Developer Release Notes Alana Killeen, Dylan Horkin, Gordon Liu, Kimberly Koenig, Liem Dinh, Michael Amorozo, Robert Chu, Steven Kwan

## Draft 1.4

## 3/9/2011

## CSRocks, Inc. CSE 403 2011

Table of Contents

[1. Revision History 3](#_Toc287524593)

[2. Introduction 5](#_Toc287524594)

[3. Supported Browsers 5](#_Toc287524595)

[4. URLs 5](#_Toc287524596)

[5. Required Tools 5](#_Toc287524597)

[6. Feature List 6](#_Toc287524598)

[7. Known Issues 6](#_Toc287524599)

[8. Install/Run Instructions 7](#_Toc287524600)

[8.1. Package Overview 7](#_Toc287524601)

[8.2. Basic Installation 7](#_Toc287524602)

[8.3. Building/Compiling Components 8](#_Toc287524603)

[8.4. Connecting to Your Own Database 8](#_Toc287524604)

[8.5. Generating Your Own Data 8](#_Toc287524605)

[8.6. Setting up the database 9](#_Toc287524606)

[9. Database Schema 9](#_Toc287524607)

[9.1. Create Table Statements 9](#_Toc287524608)

[9.2. Database Location & Credentials 11](#_Toc287524609)

[9.3. Accessing the Database 12](#_Toc287524610)

[9.3.1. Using MySQL Workbench 12](#_Toc287524611)

[9.3.2. Running MySQL Queries in PHP 14](#_Toc287524612)

[9.4. Sample Queries 14](#_Toc287524613)

[10. Unit, Stress, and System Tests 15](#_Toc287524614)

[10.1. Testing Overview 15](#_Toc287524615)

[10.2. Backend JUnit Tests 15](#_Toc287524616)

[10.2.1. Overview 15](#_Toc287524617)

[10.2.2. Current JUnit Tests 16](#_Toc287524618)

[10.2.3. Adding Additional Tests 17](#_Toc287524619)

[10.2.4. Running JUnit Tests 17](#_Toc287524620)

[10.3. Frontend PHP Unit Tests 18](#_Toc287524621)

[10.3.1. Overview 18](#_Toc287524622)

[10.3.2. Current PHPUnit Tests 18](#_Toc287524623)

[10.3.3. Adding Additional Tests 18](#_Toc287524624)

[10.3.4. Running PHPUnit Tests 18](#_Toc287524625)

[11. Manual Testing 20](#_Toc287524626)

[11.1. Testing Overview 20](#_Toc287524627)

[11.2. Current Manual Tests 20](#_Toc287524628)

[12. Bug/Issue Tracking System 21](#_Toc287524629)

[12.1. Accessing the bug-tracking system 21](#_Toc287524630)

[12.2. Understanding bugs 21](#_Toc287524631)

[12.2.1. Controlling which issues are displayed 21](#_Toc287524632)

[12.2.2. Viewing more information on an issue 22](#_Toc287524633)

[12.2.3. Changing the status of a bug 22](#_Toc287524634)

[12.2.4. Bug Statuses and Issue Labels 23](#_Toc287524635)

[12.3. Filing a new bug 24](#_Toc287524636)

[13. Mercurial Source Control - Detailed Setup and Usage Instructions 26](#_Toc287524637)

[13.1. Overview 26](#_Toc287524638)

[13.2. Initial Setup 26](#_Toc287524639)

[13.2.1. Find your Google Code Repository Password 26](#_Toc287524640)

[13.2.2. Cubist Repository 26](#_Toc287524641)

[13.2.3. Local (“working”) repository 28](#_Toc287524642)

[13.2.4. Using Mercurial 29](#_Toc287524643)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Primary Author(s) | Description | Date |
| 1.0 | Kimberly Koenig | Set up initial document | 2/19/2011 |
| 1.1 | Kimberly Koenig | Added install/run instructions, changelog access instructions | 2/20/2011 |
| 1.2-1.3 | Kimberly Koenig | Updating for final release | 3/8/2011 |
| 1.4 | Kimberly Koenig | Updating per team feedback. | 3/9/2011 |

# Introduction

This source distribution documentation is for anyone who wants to continue developing WikiMap.   
  
We assume that you have a database [set up with the appropriate schema](#_Overview_&_Schema). Explaining the database setup beyond providing a schema and basic instructions is outside the scope of this document.

# Supported Browsers

WikiMap is supported in Chrome 9+, Firefox 3.6+ , and Safari 5+. We recommend using Chrome, Safari, or Firefox 4 for optimal treatment of HTML5 and Canvas.

# URLs

|  |  |
| --- | --- |
| Live Product URL | <http://wikimap.kimberlykoenig.com> |
| Team Site | <http://code.google.com/p/cse403-wi11-wikimap/> |
| Bug Database | <http://code.google.com/p/cse403-wi11-wikimap/issues/list> |
| Code Repository | <http://code.google.com/p/cse403-wi11-wikimap/source/browse/> |
| Binary Download | <http://www.kimberlykoenig.com/files/wikimap/wikimap_binary.zip> |
| Source Download | <http://www.kimberlykoenig.com/files/wikimap/wikimap_source.zip> |
| Install Script Download | <http://www.kimberlykoenig.com/files/wikimap/wikimap_install.sh> |

# Required Tools

For the purposes of running WikiMap we assume you have:

1. **Basic Install –** You use our database and do not want to make any changes to the back-end
   1. Firefox 3.6+, Chrome 9+, or Safari 5+ browser (Chrome recommended)
   2. Linux web server
2. **Custom Install –** You want use your own database and/or want to generate your own relationship data. Requires all of the above, plus:
   1. [Eclipse IDE for Java Developers](http://www.eclipse.org/downloads/moreinfo/java.php)
   2. [MySQL Workbench](http://wb.mysql.com/)
   3. [Ant 1.8 or higher](http://ant.apache.org/)
   4. MySQL database with write access
3. **Test Install –** You wish to run our test suites or write your own additional tests.
   1. PHP Unit Tests:
      1. [Zend Studio](http://www.zend.com/en/products/studio/)
      2. [PHPUnit 3.5](https://github.com/sebastianbergmann/phpunit/tree/3.5/PHPUnit)
      3. [PHP 5](http://www.php.net/) (may be included with Zend Studio)
   2. JUnit Tests:
      1. [JUnit 4](http://www.junit.org/) (may be included with Eclipse)

# Feature List

# UI

|  |  |
| --- | --- |
| **All Views** | 1. Searching for an article 2. Swapping between “Full Map” view and “Full Article” view 3. Accessing site documentation |
| **“Full Map View”** | 1. Hovering over nodes displays a preview image and summary text 2. Clicking and dragging the map allows exploration of peripheral relationships 3. Single-clicking on an article node builds a map for it 4. Clicking on a link in the sidebar builds a map for it |
| **“Full Article View”** | 1. Clicking on an article link displays the article and mini-map for it 2. Hovering over article links displays a preview image and summary text 3. Clicking on a mini-map node displays the article for it 4. Clicking on sidebar links (in “Full Map” view) builds a map for it |

# APIs/Communication

* + 1. Relevancy Tree Builder
    2. Relevancy Tree Caching
    3. Image and Summary Tree Caching

# Logic

* + 1. Relevancy algorithm
    2. Dump downloader
    3. Wikipedia XML Parser

# Known Issues

**Bug database:** <http://code.google.com/p/cse403-wi11-wikimap/issues/list>

**Bug list coming soon…**

|  |  |  |
| --- | --- | --- |
| Title | ID | Link |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Install/Run Instructions

## Package Overview

1. **Front-End – PHP, HTML, CSS, JavaScript (web server components)**
   1. **UI** – Everything that should be placed on your web server. These are all the front-end components. They do not require compilation due to being JavaScript, PHP, and CSS files.
2. **Back-End – PHP, Java**
   1. **Logic –** Components to parse Wikipedia data, build relationships for the database, and detect updates to the Wikipedia XML dump. Requires compilation.
   2. **Communication –** Components for connecting to and updating the database. Requires compilation.
   3. **DB –** Database setup scripts.

|  |
| --- |
| **Note:** We have two installation scripts – one for the binary distribution, one for the source distribution. They should be installed to a web server, as they automatically place WikiMap’s front-end components in a **/www/ui/** folder, and the back-end components into a **/wikimap/** folder. |

## Basic Installation

This is a basic, out-of-box installation. It connects to our database.

1. Download the binary or source distribution installation script to your web server’s **root** directory using wget:  
     
   **wget** [**http://www.kimberlykoenig.com/files/wikimap/wikimap\_binary\_install.sh**](http://www.kimberlykoenig.com/files/wikimap/wikimap_binary_install.sh)

**wget** [**http://www.kimberlykoenig.com/files/wikimap/wikimap\_source\_install.sh**](http://www.kimberlykoenig.com/files/wikimap/wikimap_source_install.sh)

1. Change permissions on the shell script to allow execution:  
     
   **chmod +x wikimap\_<type>\_install.sh**Where <type> is “source” or “binary”, depending on which you installed.
2. Install WikiMap by running the shell script:  
     
   **./wikimap\_<type>\_install.sh**

|  |
| --- |
| **Note:** this script assumes that your web server uses a **www** directory to deliver web content. If a **www** folder already exists, WikiMap files and folders will be copied into it. Otherwise, a new **www** directory will be created. If your web server uses a different schema, such as **public\_html**, just rename the **www** directory. |

This installs front-end components to **/www/wikimap\_ui/** and back-end components to a **/wikimap/** directory.

1. WikiMap will now be available at: http://<your\_web\_server>/ui/

|  |
| --- |
| **Security Note:** On your web server, PHP files should be prevented from being directly accessed, otherwise you will expose your database credentials.  We do not provide instructions for doing so because the method may vary, but it typically involves modifying your .htaccess to prevent access to certain filetypes (some tips available here: <http://www.ducea.com/2006/07/21/apache-tips-tricks-deny-access-to-certain-file-types/>) |

## Building/Compiling Components

If any Java components in the back-end are modified, the WikiMap package will need to be recompiled. These instructions will guide you through that process.

1. Navigate to the back-end components in the **wikimap** directory. Ensure that there is a file called **“build.xml”** in the folder.
2. Clean and build the WikiMap package by running our ANT build script:

**ant clean all**

Where targets include: clean, init, all, communication, parser, relbuilder, dumpupdater, test-init, and test-run. <TODO: more elaboration on what each build target does needed>

## Connecting to Your Own Database

If you want to use your own database, some components will have to be reconfigured. Note that you will have to recompile any Java files that are altered.

1. Coming soon…

## Generating Your Own Data

These instructions will guide you through the process of generating your own relationship data to be inserted into your database. This data is takes the form of a .sql file containing a batch of INSERT statements generated using our parsing algorithm and the Wikipedia XML dump.

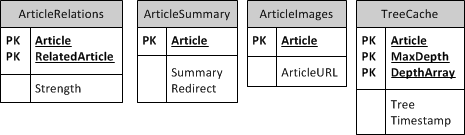
|  |
| --- |
| **Note**: This is a very time, processor, and memory-intensive process. We recommend designating a server specifically to perform data generation, or using AWS or similar to run this algorithm and to host your database. |

1. Navigate to the “logic” directory in the WikiMap back-end files (/wikimap/logic/).
2. Grant execute permissions to the “parser.sh” script:  
     
   chmod +x parser.sh
3. Run parser.sh to download, parse, and calculate relevancy data for the latest Wikipedia dumps from their directory:   
     
   ./parser.sh
4. This will generate… TODO: complete this

## Setting up the database

Database setup is currently outside the scope of this document. We assume that your database has been set up with the appropriate schema, permissions, etc. The details of our database setup are outlined below.

# Database Schema

Our database is a MySQL server running on AWS. The database has the following schema/tables:  


## Create Table Statements

delimiter $$

CREATE TABLE `ArticleImages` (

`Article` varchar(300) NOT NULL DEFAULT '',

`ArticleURL` varchar(500) DEFAULT NULL,

PRIMARY KEY (`Article`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1$$

CREATE TABLE `ArticleRelations` (

`Article` varchar(300) NOT NULL DEFAULT '',

`RelatedArticle` varchar(300) NOT NULL DEFAULT '',

`Strength` int(11) DEFAULT NULL,

PRIMARY KEY (`Article`,`RelatedArticle`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1$$

CREATE TABLE `ArticleSummary` (

`Article` varchar(300) NOT NULL,

`Summary` varchar(10000) DEFAULT NULL,

`Redirect` int(1) DEFAULT NULL,

PRIMARY KEY (`Article`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1$$

CREATE TABLE `TreeCache` (

`Article` varchar(250) NOT NULL,

`MaxDepth` int(2) NOT NULL,

`DepthArray` varchar(50) NOT NULL,

`Tree` varchar(4500) DEFAULT NULL,

`Timestamp` int(11) DEFAULT NULL,

PRIMARY KEY (`Article`,`MaxDepth`,`DepthArray`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1$$

## Database Location & Credentials

The WikiMap MySQL database is located on AWS and is read-only accessible.

1. Customers/staff may access the database read-only using the **MySQL Database Username** and **Password** we provide:

**Username:** wikiread

**Password:** WikipediaMaps123

**Server:** cse403.cdvko2p8yz0c.us-east-1.rds.amazonaws.com

1. The WikiMaps database table is called **wikimapsDB**.

More detailed access instructions (including using MySQL Workbench and PHP) are provided below.

## Accessing the Database

### **Using MySQL Workbench**

To query the database, view its tables, or otherwise access it from a front-end without having to write PHP code, use MySQL workbench as follows:

1. Download Workbench from this link: <http://www.mysql.com/downloads/workbench/>
2. After installing, click on **Open Connection to Start Querying**

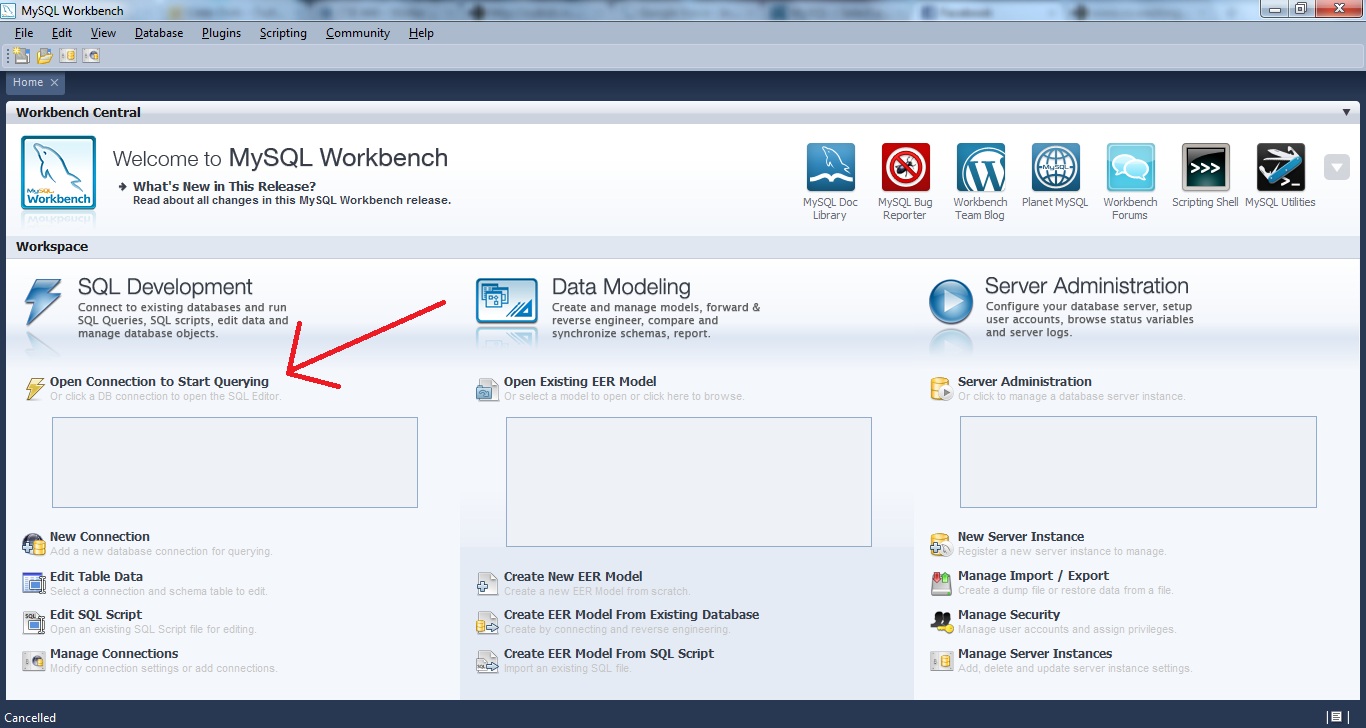
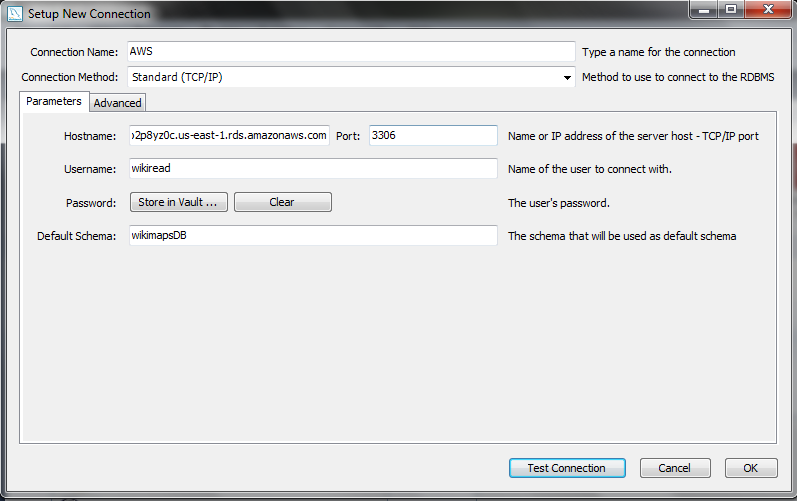


Figure : Open Connection to Start Querying

1. Enter the following settings (see next page for a sample settings screen):
   1. For **Connection Type** select **Standard TCP/IP**
   2. Change the **MySQL Hostname** to *cse403.cdvko2p8yz0c.us-east-1.rds.amazonaws.com*
   3. Change the **MySQL Username** to *wikiread*.
   4. Enter your **MySQL password** by clicking **Store in Vault**, entering the password *WikipediaMaps123*, and saving it.

 Figure : Sample settings

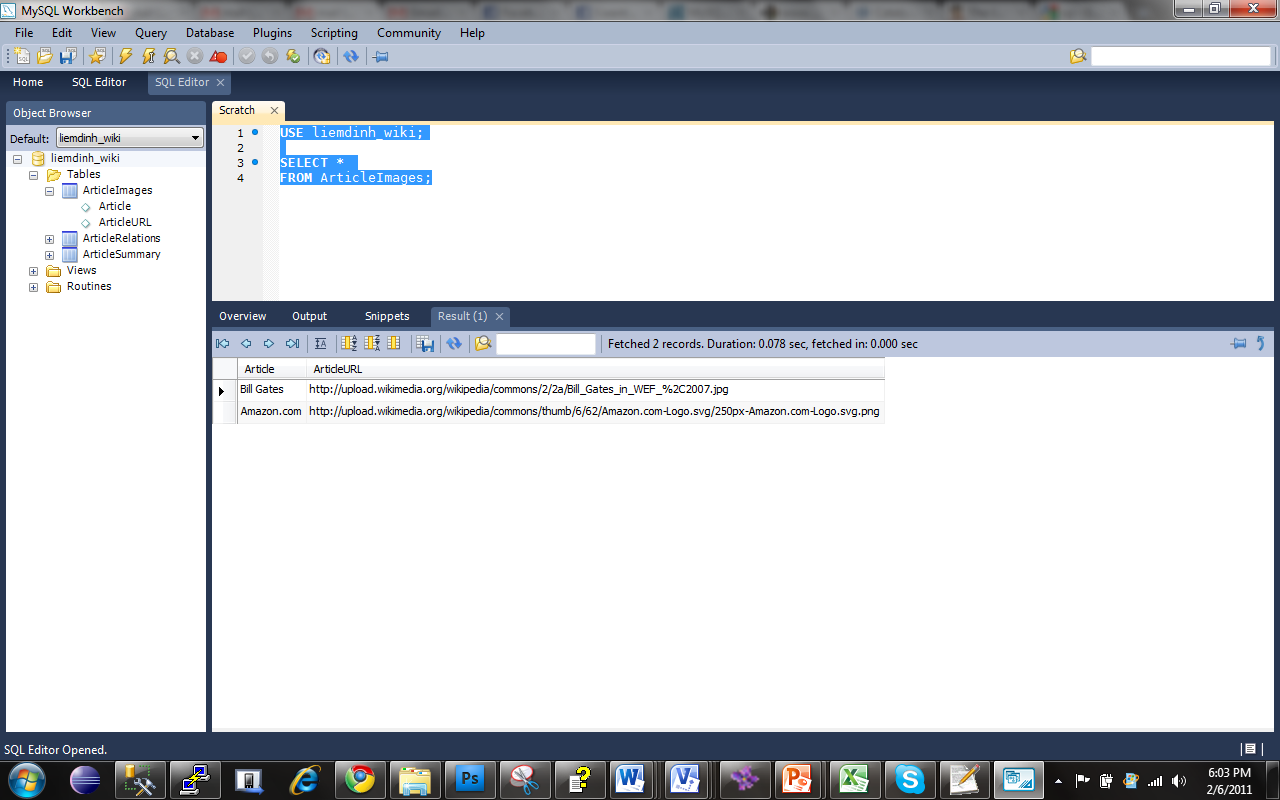
1. Now you can run a new query from the **Scratch** area. See [Sample Queries](#_Sample_Queries) for some example queries that can be run on our database. 

Figure : “Scratch” Area

### Running MySQL Queries in PHP

Here is a sample MySQL PHP query that queries the “wikimapsDB” database, retrieves all the data from the “ArticleImages” table, and prints out each row.

**<?php**

**// Make a MySQL Connection  
mysql\_connect("**cse403.cdvko2p8yz0c.us-east-1.rds.amazonaws.com**", "wikiread", "WikipediaMaps123*"*) or die(mysql\_error());  
mysql\_select\_db("wikimapsDB") or die(mysql\_error());**

**// Retrieve all the data from the "ArticleImages" table  
$result = mysql\_query("SELECT \* FROM ArticleImages") or die(mysql\_error());  
while($row = mysql\_fetch\_array( $result )) {  
// Print out each row**

**echo $row['Article'];  
echo " ";   
echo $row['ArticleURL']."</br>";}**

**?>**

## Sample Queries

1. Select all rows from the “ArticleImages” table:

**USE wikimapsDB;  
SELECT \*   
FROM ArticleImages;**

1. Select all rows from the “ArticleSummary” table:

**USE wikimapsDB;  
SELECT \*   
FROM ArticleSummary;**

1. Select all rows from the “ArticleRelations” table:

**USE wikimapsDB;**

**SELECT \***

**FROM ArticleRelations;**

1. Select the Article Summary for “Bill Gates”:

**USE wikimapsDB;**

**SELECT s.Summary**

**FROM ArticleSummary s**

**WHERE s.Article = 'Bill Gates';**

1. Return the summaries for all articles related to “Amazon.com”:

**USE wikimapsDB;**

**SELECT s.Article, s.Summary**

**FROM ArticleRelations r, ArticleSummary s**

**WHERE r.Article = 'Amazon.com' AND r.RelatedArticle = s.Article;**

# Unit, Stress, and System Tests

## Testing Overview

The unit, stress, and system tests were automated to validate the functionality of features in the backend and in the frontend.   
  
For backend unit testing, the team chose to go the route of testing only the methods that had **protected** and **public** accessors. We assumed that private method functionality would reveal inconsistencies on the output and would be handled by the developer. There exist many **private** methods that were implemented to make the source code more readable to future developers. All backend unit tests were written with JUnit.

Front end unit testing follows use cases as well as checking for load behavior and system integrity. They were written with PHPUnit.

## Backend JUnit Tests

### Overview

Our current JUnit test suites are listed in the below tables.

All JUnit tests use a separate database to keep their changes isolated from other test components. This database is already configured and is called **wikimapsDB\_unit\_test**. Its schema is identical to the live database, but it is not populated. Data population should be handled by the setup of each test suite.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Location | Test Target | Description | Type | Status |
| /test/JUnitTests/DatabaseUpdaterTest.java | /communication/DatabaseUpdater.java | Tests each public method of DatabaseUpdater.java using a test database. | Automated Unit Test | Done |
| /test/JUnitTests/WikiMapTestCase.java | n/a | Abstract class – provides database connectivity for all JUnit test cases | Superclass for Automated Unit Tests | Done |

### Current JUnit Tests

**DatabaseUpdaterTest.java**

|  |  |
| --- | --- |
| ***testNullQueries()*** | Tests the all DatabaseUpdater methods with null data. Null data should not be able to be added to the database.Methods should be able to handle the null case. |
| ***testAddRelevantNodes()*** | Tests the DatabaseUpdater.UpdateRelevantNodes method.Test will add new relevancy data to RelatedArticles table.  Data should be searchable. |
| ***testAddPreviewText()*** | Tests the DatabaseUpdater.UpdatePreviewText method.Tests will add new preview texts to the ArticleSummary table.Data should be searchable. |
| ***testAddImageURL()*** | Tests the DatabaseUpdater.UpdateImageURL method. Tests will add new image urls to the ArticleImageURL table. |
| ***testUpdateRelevantNodes()*** | Tests the DatabaseUpdater.updateRelevant method.Given that there already exists an article with relevant nodes,***u***pdate the relevant nodes and verify. |
| ***testUpdatePreviewText()*** | Tests the DatabaseUpdater.updatePreviewText method.Given that there already exists an article with preview text. Update the text and verify. |
| ***testUpdateImageURL()*** | Tests the DatabaseUpdater.updateImageURL method.Given that there already exists an article with the image URL , update the image url and verify. |
| ***testRemoveArticle()*** | Tests the DatabaseUpdater.RemoveArticle method.Adds and then deletes a series of articles from the database, verifying that they were removed correctly.  This should update all tables that have the article title in it. |
| ***testLongTitleTruncate()*** | Tests the DatabaseUpdater.updatePreviewText method.Test behavior if we insert titles that are much longer than db can allow.Article titles should truncate or error. Final behavior has not been defined. |
| ***testLongArticlePreviewTruncate()*** | Tests the DatabaseUpdater.updatePreviewText method.Test behavior if we insert article previews that are longer than the allowed amount. Article previews should truncate or error. Final behavior has not been defined. |
| ***testLongURLTruncate()*** | Tests the DatabaseUpdater.updateImageUrlTest behavior if we insert URL's that are longer than they should be article image URLs should truncate or error. Final behavior has not been defined. |

### Adding Additional Tests

We have abstracted basic database connectivity into a WikiMapTestCase.java file that extends JUnit’s TestCase class.

Additional test suites should be stored in the **test.JUnitTest** package and inherit from **WikiMapTestCase.java** to have access to these database properties.

It is the responsibility of the developer to create the test data to perform tests on because the unit testing database content is purged every time a test runs super.setup().

Once test cases have been written, developers must add the new tests to the build.xml file that ant runs:

* + - 1. Depending on what your new test suite tests, you first must make sure that the test-init target contains the target of the class you are testing.
      2. For a new test class, the following line must be updated in the test-run target. ***The highlighted section must be changed to the the .java filename of your new test.***  
           
         <javac srcdir="${test.JUnitTests.dir}" destdir="${out.test.JUnitTests.dir}" includes="DatabaseUpdaterTest.java">
      3. Also in the test-init target in the junit section, add a line for your test class using the full test class package name.  
           
         <test name="test.JUnitTests.DatabaseUpdaterTest"/>

### Running JUnit Tests

Running tests is simple. We are utilizing ant to run all tests at once. Run ant test-run. This will compile any missing dependencies.

## Frontend PHP Unit Tests

### Overview

Our PHP tests are our “use case” tests. Currently we test the first use case in our SRS (“User searches for an article”. Current tests are located in test/FrontEndUI/retrieverTest.php since we are only testing the interaction between the frontend on database.

**Note:** We did not test for SQL injection from the frontend because we chose a PHP function that would make it impossible to use a semicolon to delimit a query and allow for a SQL query to be injected in the search.

### Current PHPUnit Tests

**retrieverTest.php**

|  |  |
| --- | --- |
| ***testArticleInDB()*** | Test correct behavior for article found in database. |
| ***testArticleNotInDB()*** | Test correct behavior for article found in database. |
| ***testRedirects()*** | Redirects should return the same tree. |
| ***testTextVariations()*** | Text variations should all return the same tree. |
| ***testStress1()*** | Search for Michael Jackson 100 times. |
| ***testStress2()*** | Asynchronously retrieve 10 random articles from Wikipedia. |
| ***testHTMLInjection()*** | Test tree returned correctly despite inclusion of HTML tags. |
| ***testUnusualCharacters()*** | Tests that the frontend can interpret non-western characters and return the appropriate article and article image. |

### Adding Additional Tests

To be added

### Running PHPUnit Tests

To run PHP unit tests:

* + - 1. Create a new project in Zend Studio. Right click on the Project in the Project Explorer, and import all PHP files for the WikiMap front-end UI from the UI folder and its subfolders.
      2. These tests are compatible with PHPUnit 5. Make sure Zend Studio is using that version of PHP by right-clicking on the project and then selecting **Properties > PHP Interpreter**.
      3. Also add the PHPUnit tests to the file from test/FrontEndUI/\*.php.
      4. Run the tests by right-clicking on the test and selecting **Run As...>PHPUnit test**

# Manual Testing

## Testing Overview

Originally many of the UI specific features were to be tested with Selenium, but we ran into compatibility issues that prevented us from specifying coordinate specific actions. We instead performed manual tests on Windows 7, Mac OS X, and Ubuntu 10 with Firefox 3.6+, Chrome 9+, and Safari 5+.

## Current Manual Tests

|  |  |  |
| --- | --- | --- |
| Test Objective | Test Description | Result |
| Windows7 : Determine that the modules are rendered well in Safari 5, Firefox 3.6, and Chrome 9 | 1. Load wikimap.kimberlykoenig.com 2. Search for a topic 3. Resize the window vertically and horizontally at random and make sure it is still usable according to specs |  |
| Mac OS X: Determine that the modules are rendered well in Safari 5, Firefox 3.6, and Chrome 9 | 1. Load wikimap.kimberlykoenig.com 2. Search for a topic 3. Resize the window vertically and horizontally at random and make sure it is still usable according to specs |  |
| Ubuntu: Determine that the modules are rendered well in Safari 5, Firefox 3.6, and Chrome 9 | 1. Load wikimap.kimberlykoenig.com 2. Search for a topic 3. Resize the window vertically and horizontally at random and make sure it is still usable according to specs |  |
| Firefox 3.6: Switching between article view and map view updates the url and ui | 1. Enter from searching for an article. 2. Switch between article view and map view. 3. Verify that the UI elements match |  |
| Safari 5: Switching between article view and map view updates the url and ui | 1. Enter from searching for an article. 2. Switch between article view and map view. 3. Verify that the UI elements match |  |
| Chrome 9: Switching between article view and map view updates the url and ui | 1. Enter from searching for an article. 2. Switch between article view and map view. 3. Verify that the UI elements match |  |
| Firefox 3.6: Switching between article view and map view updates the url and ui | 1. Enter from .../wikiSearch.php?s=food&view=article 2. Switch between article view and map view. 3. Verify that the UI elements match |  |
| Safari 5: Switching between article view and map view updates the url and ui | 1. Enter from .../wikiSearch.php?s=food&view=article 2. Switch between article view and map view. 3. Verify that the UI elements match |  |
| Chrome 9: Switching between article view and map view updates the url and ui | 1. Enter from .../wikiSearch.php?s=food&view=article 2. Switch between article view and map view. 3. Verify that the UI elements match |  |
| Search Term Integrity using special characters | 1. Search for topics with '&', '/', '%' and see if it respects the characters as is in the UI |  |

­

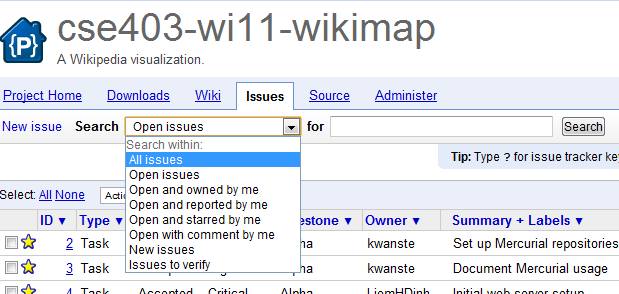
# Bug/Issue Tracking System

## Accessing the bug-tracking system

WikiMap bugs and tasks are tracked in our [Google Code Issue Tracking tool](http://code.google.com/p/cse403-wi11-wikimap/issues/list). This tool is publicly accessible as part of our Google Code repository.

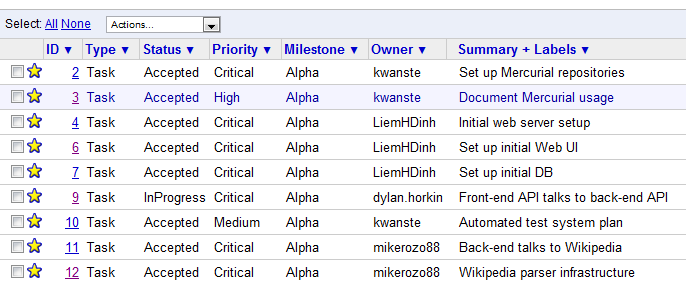
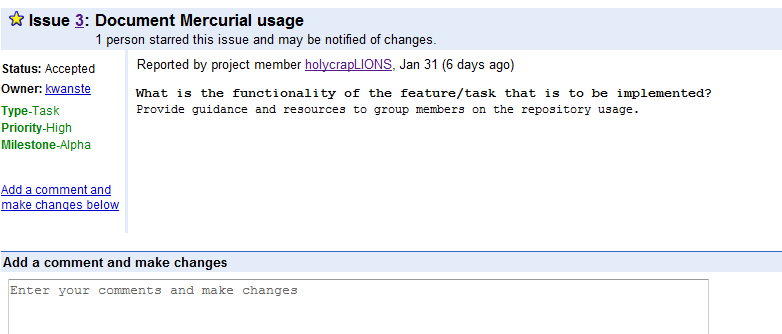
## Understanding bugs

### Controlling which issues are displayed

By default, only **Open Issues** are displayed. To control which are displayed at a given time, click the drop-down menu at the top of the page next to the **Search** text:  


Then click the **“Search”** button on the right, which will filter for the desired issues.

### Viewing more information on an issue

1. In the Issues List, click on the issue for which you want to view more information:  
   
2. The following screen will display. Note that Issue Labels, Priorities, Milestone, etc. displays on the left in green:  
   

The description of the Issue/bug is in the center of the page.

### Changing the status of a bug

1. Click on the issue from the main list, as in [4.2.2](#_Viewing_more_information).
2. Click on the textbox in the **Add a comment and make changes** region (see above picture).
3. The comment box will expand and allow you to make changes to the issue.

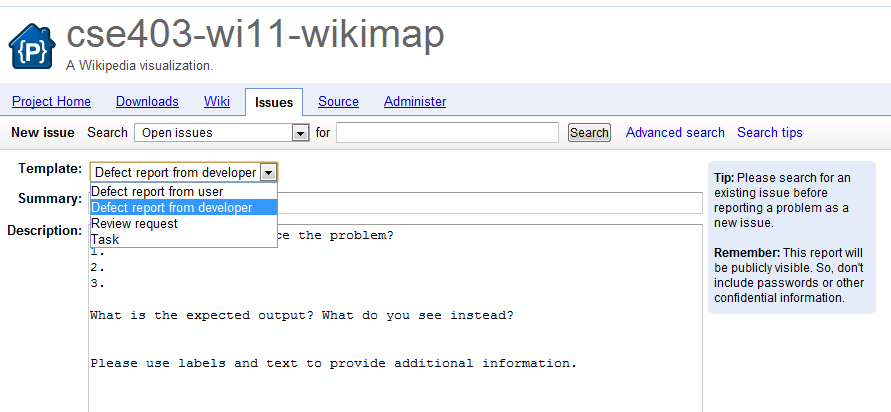
### Bug Statuses and Issue Labels

Various tags can be assigned to a bug. These tags and their meanings include:

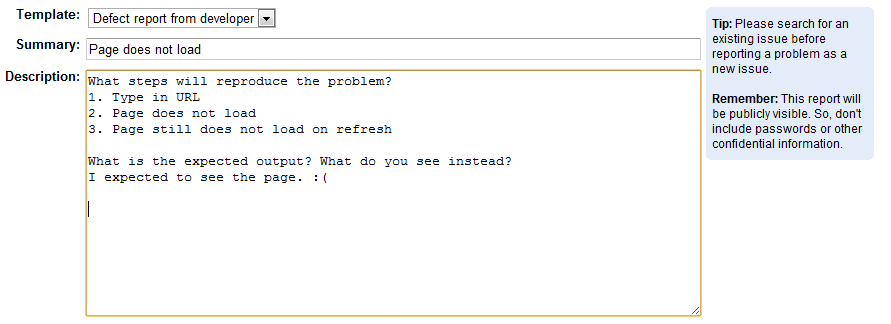
|  |  |
| --- | --- |
| **Status** | **Description** |
| New | Issue has not had initial review yet |
| Accepted | Problem reproduced and/or need to fix acknowledged |
| InProgress | Work on this issue is in progress |
| Fixed | Developer made source code changes, QA should verify |
| Verified | QA has verified that the fix worked |
| Invalid | This was not a valid issue report |
| Duplicate | This report duplicates an existing issue |
| WontFix | We decided to not take action on this issue |
| Done | The requested non-coding task was complete |
| Revisit | This issue is tabled but may be revisited at a later date. |
| Cut | This feature was cut. |

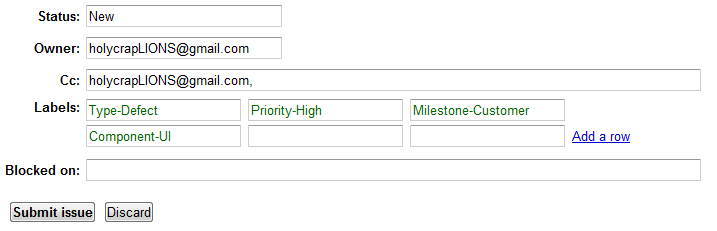
|  |  |
| --- | --- |
| **Issue Labels** | **Description** |
| Type-Defect | Report of a software defect |
| Type-Enhancement | Request for enhancement |
| Type-Task | Generic work item |
| Type-Review | Request for a source code review |
| Type-Other | Some other kind of issue |
| Priority-Critical | Must resolve in the specified milestone |
| Priority-High | Strongly want to resolve in the specified milestone |
| Priority-Medium | Normal priority |
| Priority-Low | Might slip to later milestone |
| OpSys-All | Affects all operating systems |
| OpSys-Windows | Affects Windows users |
| OpSys-Linux | Affects Linux users |
| OpSys-OSX | Affects Mac OS X users |
| Browser-Chrome | Affects Chrome users |
| Browser-Firefox | Affects Firefox users |
| Browser-Safari | Affects Safari users |
| Milestone-NotAssigned | A milestone has not been set for this issue |
| Milestone-Alpha | All essential functionality working (2/7) |
| Milestone-Beta | Functionality related to the beta release (2/21) |
| Milestone-Customer | Functionality that we want the customer to experience (2/28) |
| Milestone-Final | Functionality related to the final release (3/11) |
| Milestone-Stretch | Functionality that is not essential to the course deadline |
| Component-API | Issue relates to APIs |
| Component-UI | Issue relates to program UI |
| Component-Logic | Issue relates to application logic |
| Component-Persistence | Issue relates to data storage components |
| Component-Scripts | Utility and installation scripts |
| Component-Docs | Issue relates to end-user documentation |
| Security | Security risk to users |
| Performance | Performance issue |
| Usability | Affects program usability |
| Maintainability | Hinders future changes |
| User-reported | Customer reported issues |

## Filing a new bug

1. Click the “New Issue” link at the top of the Issues page. 
2. Select what kind of issue template you want to fill out:  
   

**Bugs** will most likely be “defect report from developer”. **Deliverables** will most likely be “task”. **Usability bugs reported by our customers** are “defect report from user”, and **requests for code review** are “review request”.

1. Fill out the Summary and Description fields, being sure to answer all applicable fields in the Description:  
   
2. Apply all relevant labels/classifications to your issue:



1. And click submit!

# Mercurial Source Control - Detailed Setup and Usage Instructions

## Overview

The source control repository consists of three layers:

1. **Team repository:** Google Code
2. **Personal repository:** Cubist
3. **“Working” repository:** Local machine

The **Team Repository** holds all of the latest, fully committed code changes. This is the main branch of the WikiMap team’s code.

A **Personal Repository** on Cubist holds an individual’s personal “pristine” copy of the **Team Repository**, as well as any changes that individual was comfortable enough committing to Cubist. Cubist is backed up, so this is an extra failsafe.  
  
A **Working Repository** is a local clone of the **Personal Repository**. It’s for a majority of individual changes before committing changes, preventing conflicts and allowing a “safe” copy of the repository to reside both on Cubist and the **Team Repository.**

Any changes in which a team member is confident should be committed up to the main Google Code branch, but only after resolving any conflicts.

## **Initial Setup**

### **Find your Google Code Repository Password**

In order to push to the main repository, you will need to know your Google Code password. This is different than your Google (Gmail) account password.  
  
This password can be obtained from <https://code.google.com/hosting/settings>.  
   
You have the option on this page to use your Google account password if you feel this password is secure enough.

### Cubist Repository

The main Google Code repository is already set up for you, so the next step is to set up your **Personal Repository** on Cubist.

#### Create personal Incoming/Outgoing Clones

1. Log into your Cubist account by SSHing through PuTTY (or an equivalent).   
   ***Connect to:* cubist.cs.washington.edu  
   *Username:* <CSE Username>  
   *Password:* <Kerberos password>**
2. Run the following lines of code in your root directory (or in a directory of your choice), but replace *<name>* with your own name or username:

**hg clone https://cse403-wi11-wikimap.googlecode.com/hg/ cse403-wi11-wikimap-*<name>*-incoming  
  
hg clone https://cse403-wi11-wikimap.googlecode.com/hg/ cse403-wi11-wikimap-*<name>*-outgoing**

***(Important note: the “wi11” uses the numeral “1”, not the lowercase letter “l”)***

#### Configuration settings

1. Change directory into the “outgoing” repository directory:  
   **cd cse403-wi11-wikimap-*<name>*-outgoing**
2. If you enter an ls –la command you should see an .hg folder.
3. Change directory into the “.hg” folder:  
   **cd .hg**
4. Enter the **ls –la** command once again. You should see an “hgrc” file.
5. Open the “hgrc” file in your text editor of choice, such as by entering the command:

**emacs hgrc**

1. You will see the following line in the “hgrc” file:  
   **[paths]   
   default = https://cse403-wi11 wikimap.googlecode.com/hg/**
2. Update this line to look like the text below, but change <USERNAME> to your Google Account username is and <PASSWORD> to your Google Code password that you found in [step 2.2.1](#_Find_your_Google).   
   **[paths]  
   default = https://<USERNAME>%40gmail.com:<PASSWORD>@cse403-wi11-wikimap.googlecode.com/hg/**  
     
   ***(Important note: the “wi11” uses the numeral “1”, not the lowercase letter “l”)***
3. Save the file and exit the editor.

### Local (“working”) repository

#### Installing TortoiseHg on your computer

Windows users will need to install Hg onto their computers. This guide explains the installation and configuration of Hg using TortoiseHg, which may be installed from:  
<http://tortoisehg.bitbucket.org/>

For Linux-style command line control, Cygwin is recommended. Explaining how to install and configure Cygwin is not within the scope of this document, but those who wish to use Cygwin may download it here: <http://www.cygwin.com>

For other Hg installation instructions (including Mac OS X and Linux), see:   
<http://hgbook.red-bean.com/read/a-tour-of-mercurial-the-basics.html#sec:tour:install>

#### “Incoming” Clone & Outgoing Push

Once you have Hg (and Cygwin) installed, you need to clone from your incoming folder, and configure to push to your outgoing.   
  
***Set your incoming repository***

1. In Windows Explorer, browse to the folder into which you want to store your working repository (or create a new folder). Right click on this folder and select **TortoiseHg > Clone**.
2. Enter the following address into the source line where you will replace <USERNAME> with your own and <NAME> with your own.

ssh://**<USERNAME>**@cubist.cs.washington.edu/cse403-wi11-wikimap-<**NAME>**-incoming

1. Hit **“Clone”**. You will have to authenticate with your KERBEROS password.

***Configure your working directory & “push” path***

1. Open the incoming repository directory you configured in the previous step, and then browse to the**.hg** folder.
2. Edit the **hgrc** file to look like below while replacing the appropriate fields with your username.   
     
   **[paths]  
     
   default = ssh://<USERNAME>@cubist.cs.washington.edu/cse403-wi11-wikimap-<NAME>-incoming  
     
   default-push = ssh://<USERNAME>@cubist.cs.washington.edu/cse403-wi11-wikimap-<NAME>-outgoing  
     
   [ui]  
     
   username = <NAME>  
     
   [tortoisehg]  
   summarylen = 70  
   messagewrap = 80  
   closeci = True  
   pushafterci = False**
3. And that’s it! Everything is set up.

### Using Mercurial

#### Getting the latest repository changes

#### Pulling and Updating from Google Code

When you need to work on a new feature, SSH into Cubist and pull down the newest Google Hg Repository revision and update your incoming folder to reflect those new changes.

1. Change directory to your “incoming” directory  
   **cd cse403-wi11-wikimap-*<name>*-incoming**
2. Enter the following command:  
   **hg pull –u**
3. If you receive an error, you may have to enter the following two commands consecutively:

**hg pull  
hg update**

***Pulling and Updating to your Local (“Working”) Repository***

From your working repository, you can pull the changes down, or if you choose, re-clone the Cubist repository to create another working repository specifically for the feature on which you are currently working.

1. Right click on the folder you set to be your incoming repository. Select **TortoiseHg > Synchronize**. (If you are using a command prompt/Cygwin, you may do **hg pull –u** from within your working repository directory).
2. In the Synchronize UI which appears, click the “Pull” button.
3. Finally, right click on the folder you set to be your incoming repository and select **TortoiseHg > Update**. The “default” update should be sufficient.

**Note:** There is no need to ‘Checkout’. However, when you make code changes, you must commit for there to be a record. Commit often.

#### Committing changes (from your Local Repository to Cubist)

***Commit from Local Repository to Cubist***

1. To commit a change in your working repository, run the following line:  
   **hg commit**  
   Or, right click on your repository and select **Hg Commit.**
2. This will open a text editor where you are to put a detailed explanation of what change you just made. It will detect the recently edited files for you.

#### Pushing changes back to Google Code

To get your changes back to Google Code, some integration must occur.

1. First, push the record of commits from your working (local) directory to your outgoing repository on Cubist by running the following line:  
     
   **hg push**  
     
   You’ll have to authenticate to get onto Cubist.
2. Once that is done, swap over to Cubist through PuTTY (or an equivalent).  
     
   From your outgoing repository’s folder, run the following line to get the changes you just pushed up.  
     
   **hg update**  
   There should be no conflicts here.
3. Next, pull down the recent changes from the Google Code repository with:  
     
   **hg pull –u**
4. You may find there are conflicts, and have to run the following line to resolve the conflicts.  
     
   **hg merge**
5. Once all those are resolved, you can finally perform a push to the Google Hg repository and all your changes will show up in the source tracker!  
     
   **hg push**

TESTING VERSION CONTROL ON WORD DOCS.