Kurogo Documentation

Release 1.0RC1

Modo Labs

March 26, 2011

CONTENTS

The Kurogo Framework is a PHP based web application that can help institutions efficiently deliver campus services and information to a wide array of mobile devices. Based on the MIT Framework, this open source project provides a modular way to present mobile versions of various data sources in an extendable, customizable fashion.

At a high level, the Kurogo Framework includes:

- A mechanism for detecting device characteristics and displaying content best suited for that device
- A object oriented system for retrieving, parsing and displaying data from a variety of external sources
- A robust templating system that utilizes themeable reusable parts to easily construct consistent user interfaces
- A series of prebuilt, customizable modules for gathering directory, news and event information
- A system of authentication and authorization for controlling access to content and administrative functions

This guide serves as a tour of the project source code and its features.

Note: This is documentation for a Beta product. Please be aware that certain details may change before it reaches final 1.0 status

CHAPTER

ONE

OVERVIEW

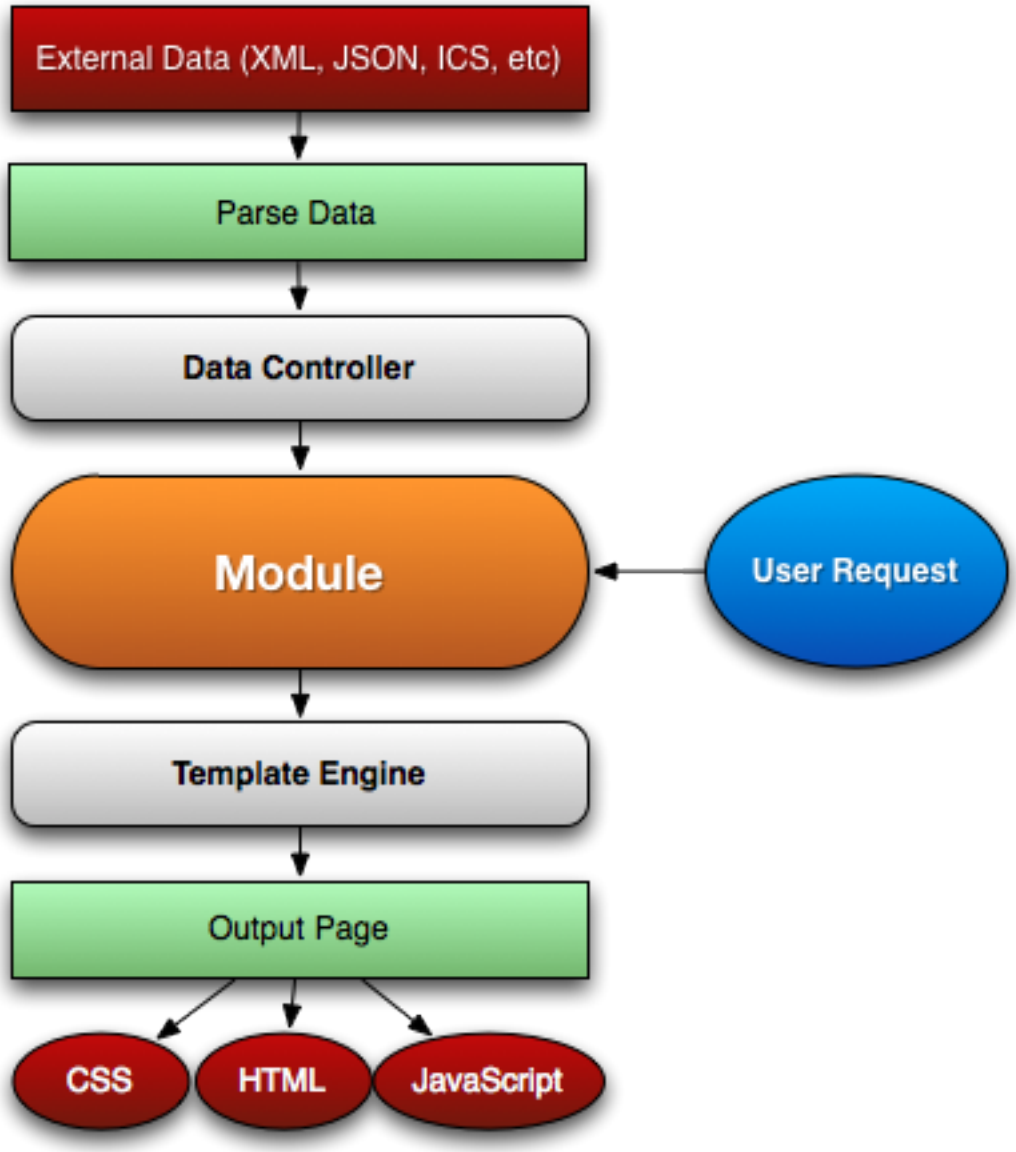
Kurogo is a PHP framework for delivering high quality, data driven customizable content to a wide range of mobile devices. Its strengths lie in the customizable system that allows you to adapt content from a variety of sources and easily present that to mobile devices from feature phones, to early generation smart phones, to modern devices and tablets.The mobile web component exists as a based web application served using PHP. This application is hosted on a web server and users access it using the web browser on their mobile device.

It is available under a liberal open source MIT license. This mean you are free to download, install, copy, modify and use the software as you see fit.

1.1 Modules

The building block for Kurogo is modules. Each page request is handled by a module that parses the url and displays one of its *pages* with content. Modules are contained pieces of code that (typically) connect to external services, process the data and display it on the device using standard HTML and CSS. Kurogo is designed to accept data from a variety of external sources, parse and process that data and prepare it in a format suitable for presentation.

A typical user request would include querying a data source and parsing the resulting data using a *Data Controller*. Then the module would pass the data to the *Template Engine* which outputs the page using appropriate HTML, CSS and Javascript for the device.



1.2 Device Detection

An important feature of Kurogo is the ability to detect the device being used. Because mobile devices have different capabilities and performance characteristics, classifying devices is critical to giving the user the best experience possible while still supporting a wide variety of devices.

Kurogo classifies devices by *type* and *platform*. The type is a generic bucket of devices that outlines the device's basic capabilities: it's level of CSS support, javascript, image handling,etc. The platform is the specific operating system/browser used. Each of these values can be used to provide a customized experience for devices. Kurogo already has a series of templates and css files that provides this experience for the included modules and common user interface elements.

1.3 Customization

From the beginning, Kurogo is built to be customized. You have full control of how data gets into a module, how it is parsed, and how it gets presented to the user. The modular nature of the software allows you to customize its behavior at any step of the process.

1.3.1 Data Customization

Each module gives you the opportunity to choose the data source and processing methods for getting data into your application. By abstracting the details of retrieving and parsing data, your application can respond to a variety of data sources. Most modules will use a subclass of *DataController*. Each data controller has an object that is a subclass of *DataParser* that takes the resulting data and creates an appropriate PHP structure. Through configuration you can customize which DataController and DataParser are used in a request which can influence the structures that get used.

1.3.2 Code Customization

Each module is a PHP object that inherits from the *WebModule* class. Developers can create their own modules or just subclass existing ones. When extending, you can choose only to override certain methods. This allows you to concentrate on the features and behaviors that make your module unique.

1.3.3 Output Customization

Once the data has been prepared for output, you have several means to customize the presentation. Each page is associated with a HTML document. These templates can be customized and overridden as needed and there is a library of existing fragments that contain common user interface elements and pieces, each customized for the appropriate device. Along with HTML, you can also customize the style sheets associated with the page using the cascading nature of CSS.

CHAPTER

TWO

GITHUB REPOSITORY

Kurogo is an open source project. You are free to download, update and use the code for your own use without charge. The project uses the Git distributed version control system. The Git repository is hosted by GitHub. It can be found at https://github.com/modolabs/Kurogo-Mobile-Web.

For those not familiar with Git or GitHub, please view the GitHub Help Site.

2.1 Forking and Managing your repository

If you simply want to download the code, you should clone the repository using `git clone git://github.com/modolabs/Kurogo-Mobile-Web.git`

If you are interested in maintaining your own project you should fork the project.

1. Log into GitHub
2. Browse to https://github.com/modolabs/Kurogo-Mobile-Web
3. Click the **fork** icon in the upper right portion of the page
4. (Optional) You may wish to rename your project
5. Clone your project to your local machine.
6. Set up an upstream remote:
    - `git remote add upstream git://github.com/modolabs/Kurogo-Mobile-Web.git`
    - `git fetch upstream`
    - `git checkout -b upstream upstream/master`
7. When new changes come down you can run:
    - `git checkout upstream` Change to upstream branch
    - `git pull` Pull down changes
    - `git checkout master` Change to master branch
    - `git merge upstream` Merge changes into master branch

There are certainly other ways to manage your repository, but this method provides flexibility and will allow you to maintain a branch that represents the current development in the project.

2.2 Creating your site

Because your own project will contain elements that are not part of the master project (i.e. your own site's image and css assets), we recommend you keep a separate **upstream** branch. This branch will remain clean and can be merged into your master branch. By creating an upstream branch it also allows you to more cleanly handle submitting changes back to the project.

From your master branch, make a copy of the *site/Universitas* folder. This is the template site. You should rename this to match a concise name for your site. Most, if not all, of your coding will take place in this folder. You can read more about *creating additional modules*, *extending existing modules* and *theming your site*. Unless you have the need to change core libraries, you should not need to edit any files in the *lib* or *templates* files of the root project directory.

2.3 Submitting your changes

If you have fixed a bug in the project or would have a new feature to share, you can submit a pull request. This informs the project maintainers that you have code you wish to be part of the mainline project.

It is **strongly** recommended that you initiate pull requests in the following manner:

1. Make sure your upstream branch is up to date
2. Make a new branch that implements the fixes/features.
3. Browse to your GitHub project in your web browser
4. Switch to the branch with your fix/feature
5. Click the **pull request** icon
6. Include a description regarding the nature of your work. If there is not sufficient detail, then your request may not be accepted.
7. If you do not initiate your pull request from a separate branch you will likely have to click the **change commits** button and select the various commits that include your fix.
8. Click the send pull request when the changes are appropriate.

By utilizing this method, you can insure that only the changes appropriate for the project are included in your request. It also allows for alterations to be included without affecting your main branch of work. Sometimes it can take discussion to resolve any issues regarding coding style, questions regarding your patch and then final integration.

CHAPTER

THREE

SETUP AND INSTALLATION

Kurogo is a PHP web application. As such it must be installed on a web server. As of this version, the only web server that has been qualified for use is Apache version 2.x on a unix based system.

3.1 System Requirements

- Apache Web Server (tested on 2.x)
  - Requires mod_rewrite module
  - Requires .htaccess support (AllowOverride)
- PHP 5.2 or higher with the following extensions:
  - xml
  - dom
  - json
  - PDO (SQLite required, also can use MySQL)
  - mbstring
- Some PHP modules are optional depending on whether you need their backend functionality
  - LDAP
  - curl

3.2 Installation

Please note that some of these instructions assume that you have basic system and web server administration knowledge. For more information please see the documentation for your system and the Apache Documentation. For development environments, we recommend MAMP on Mac OS X.

1. Extract the files to a location accessible by your web server
2. Set the DocumentRoot of your web server to the *www* folder.
3. Ensure that .htaccess files are enabled. AllowOverride must be set to at least *FileInfo*.
4. In the *site* directory, make a copy of the *Universitas* folder, including all its contents. The name of this site is up to you, but it would be prudent for it to refer to your site's name. We will refer to this folder as *SITE_FOLDER*

- **Critical:** Make sure the web server user (typically *apache* or *www*) has write access to all the contents *SITE_FOLDER*.

5. In the *config* directory, make a copy of the *kurogo-default.ini* file called *kurogo.ini*
6. Edit the new kurogo.ini file and change the *ACTIVE_SITE* option to match the name of *SITE_FOLDER*
7. (re)Start your webserver and direct your web browser to the server/port that you specified.

CHAPTER

FOUR

MAKING YOUR FIRST MODULE - HELLO WORLD

This section will give you an overview of the module creation process. It is meant to be followed along, but most of the in-depth technical knowledge can be found in *Creating a new module*. Most of the content in this section is elaborated in more depth elsewhere.

4.1 Case Sensitivity

It is important to be mindful of case sensitivity issues. Many developers use file systems that are not case sensitive. Most servers (especially Linux based servers), do use case sensitive file systems so it is critical that when defining folder locations and urls that map to files or folders, that the case used is consistent. Typically that means using lower case for urls.

4.2 Creating the module folder structure

Note: Please make sure you have followed the *installation steps* and have created a site folder.

The Kurogo Framework looks in a variety of locations for module data (see *Source code tour*). In most cases you will want to create your module in your site's *app/modules* folder.

- Create a folder named *hello* inside *SITE_DIR/app/modules*
- Inside the *SITE_DIR/app/modules/hello* folder, create a folder named *templates*

4.3 Creating the module class file

Inside the *hello* directory create a file named *HelloWebModule.php* that contains the following contents:

```
<?php

class HelloWebModule extends WebModule
{
  protected $id='hello';
  protected function initializeForPage() {
    $this->assign('message', 'Hello World!');
  }
}
```

4.4 Creating the template file

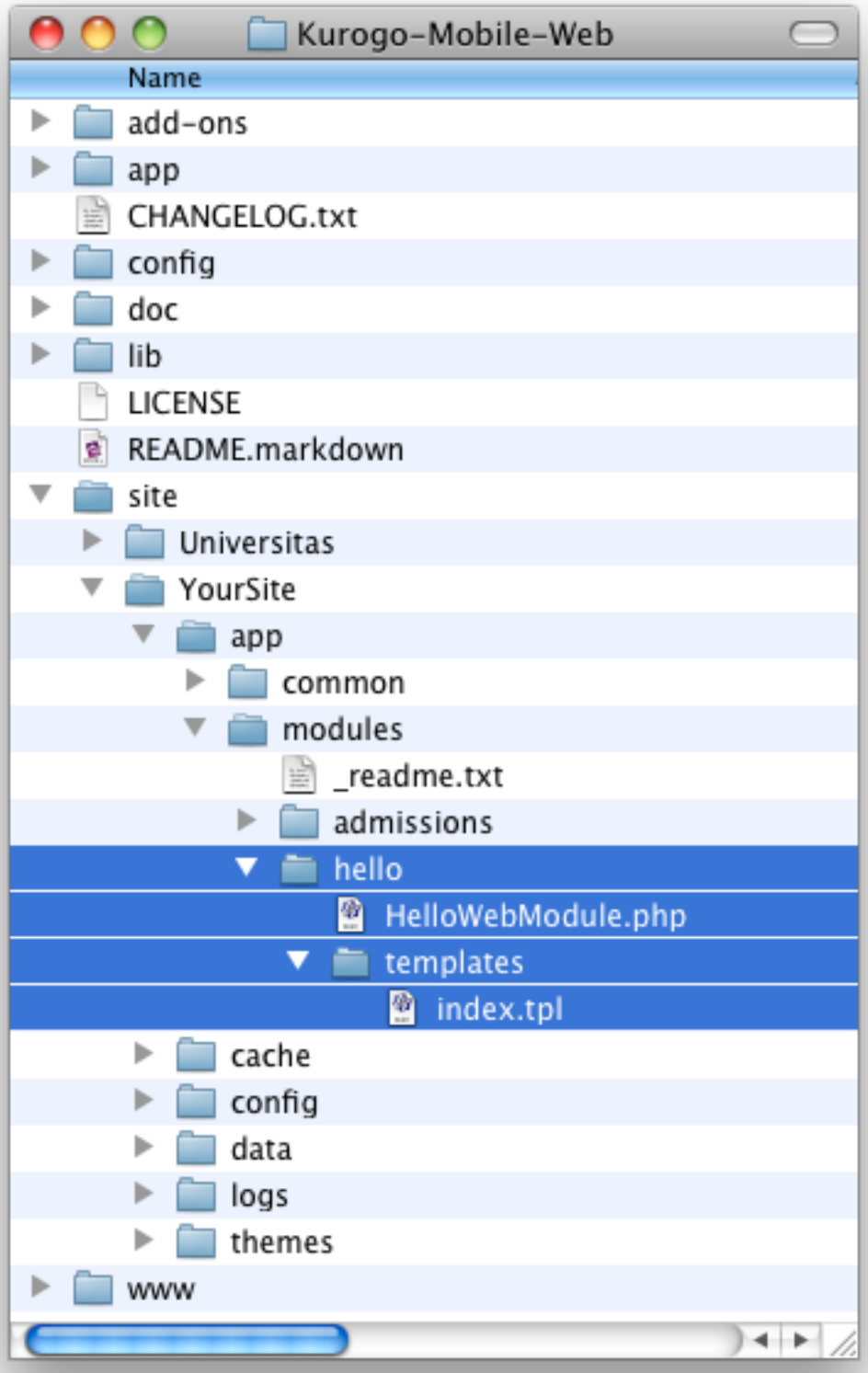
Inside the *hello/templates* directory create a file named *index.tpl* that contains the following contents:

```
{include file="findInclude:common/templates/header.tpl"}

<h1 class="focal">Hello World!</h1>

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

Your folder structure should look similar to this:

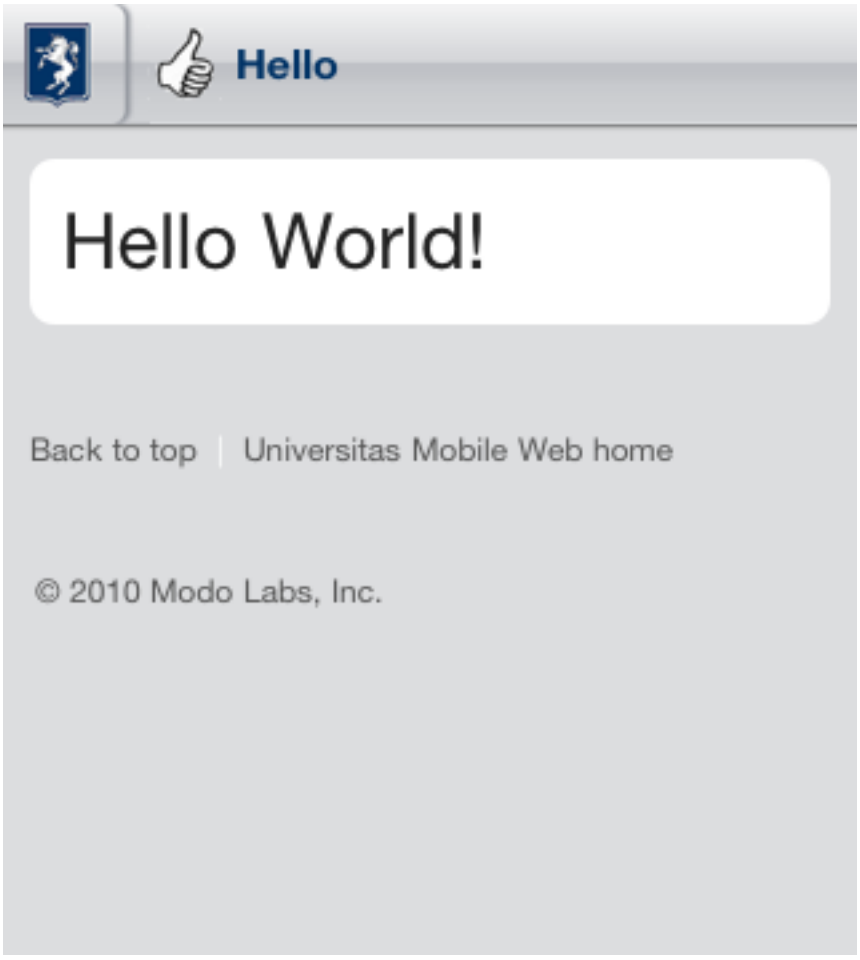

Kurogo-Mobile-Web
Name
add-ons
app
CHANGELOG.txt
config
doc
lib
LICENSE
README.markdown
site
Universitas
YourSite
app
common
modules
_readme.txt
admissions
hello
HelloWebModule.php
templates
index.tpl
cache
config
data
logs
themes
www

4.4.1 Creating the nav bar image

Create a 56 x 56 PNG file named *title-hello.png* and place it in *SITE_FOLDER/themes/default/common/images/compliant*.

4.5 Viewing the module

At this time you should be able to view your new module in your web browser. Assuming your site is on port 8888 on your local machine go to `http://localhost:8888/hello`. If successful you should see your new module:



Congratulations! You've just built a simple module.

See Also:

Creating a new module Detailed explanation of creating a new module.

Extending an existing module Detailed explanation of extending an existing module

CHAPTER

FIVE

SOURCE CODE TOUR

This section will give you a tour of the various files and directories in the Kurogo source code directory. Knowing the layout of the project will help you understand some of the decisions behind code and where to place your own files so upgrading to newer versions is as seamless as possible.

5.1 Basic Layout

There are several directories located in the root of the project folder:

add-ons This directory contains additional scripts or code that be used to interact with other applications

app This directory contains the code and *templates* for each module provided by the project. This also includes shared templates used by every module (including headers and footers). As with the lib folder you should avoid adding or altering these files, but rather put new or altered files in the *Site folder*

config This directory contains the main configuration files for the entire project. Most notably it contains the *kurogo.ini* file which determines the active site.

lib This directory contains libraries that are provided by the project. This includes libraries for data retrieval and parsing, authentication, database access and configuration. Generally speaking only libraries provided by the project should be in this directory. You should place your libraries in the site/lib folder to avoid possible conflict with future project updates.

site This directory contains an entry for each site. See *Site folder* for more detail

www This directory contains the DocumentRoot for the site. It contains the main script *index.php* which handles all incoming requests. It also contains the minify library for delivering optimized css and javascript to clients

5.2 Case Sensitivity

It is important to be mindful of case sensitivity issues when developing and deploying your site. Many developers use file systems that are not case sensitive. Most servers, however, do use case sensitive file systems so it is critical that when defining folder locations and urls that map to files or folders, that the case used is consistent. This guide aims to highlight situations where the framework expects certain files or folders to be in a particular case format. It is critical to test your server in an environment that matches your production environment to discover any case-related problems.

5.3 Provided vs. Site files

As noted in the layout section, there are files provided by the project (app, lib, www) and files for your use (site). As an open source project, you are certainly welcome to alter files in any way that suits your needs. However, be aware that if you alter or add files in the project directories, it may create conflicts when you attempt to update future versions of the project. There are well known methods to *add additional functionality* to existing code while maintaining upgradability.

That being said, if you have improvements that others would benefit from, we encourage you to *submit your changes* to the project.

5.4 Libraries

The framework utilizes a number of code libraries to modularize and abstract certain processes and encourage code reuse. The design goal behind the libraries is to ensure that they operate as generically as possible so that they can function in a variety of uses and contexts (even if, in practice, they are currently used in only one context). Nearly all the libraries exist as PHP classes and currently fall into one of several categories:

5.4.1 Packages

In order to assist developers with including the proper class files, libraries can be grouped into *packages*. This allows you to include necessary functionality without worrying about which files to include in your module (use: *includePackage('PackageName')* in your module code). Currently the following packages are available:

- Authentication (included automatically when authentication is enabled)
- Calendar - includes classes to deal with date and time
- db - used when you wish to interact with a database
- Emergency - used by the emergency module
- Maps - used by the maps module
- Video - used by the video module

5.4.2 Core / Support Files

- autoloader - Defines a function that finds and loads class files on demand
- compat - defines several functions that normalize behavior throughout PHP versions
- exceptions - defines exception subclasses and sets up exception handling behavior
- initialize - called by *index.php* to setup the runtime environment
- minify - interface between the framework and the included open source minify library
- *DeviceClassifier* - An interface between the frame work and the *Device Detection Service*
- *PageViews* - A class to log and retrieve page view information for statistics
- *Validator* - A utility class to validate certain types of data

5.4.3 External Data Retrieval

The main class is *DataController*. It provides functionality to retrieve URL based data (this could include both local and remote data), cache this data, and parse it using a subclass of *DataParser* to prepare it into a structure suitable for use. In its optimal design, a data controller will abstract the details of building the URL, and return a structure that is normalized, allowing the module code to be as generic as possible.

Included examples of DataControllers/Parsers include:

- *RSSDataController* - retrieves a feed of data in RSS/RDF or Atom formats. The corresponding *RSSDataParser* class takes the resulting data and builds a structure of items located in the feed. Also uses the *RSS* class.
- *CalendarDataController* - retrieves a feed of data in ICS format. The corresponding *ICSDataParser* class takes the resulting data and builds a structure of events in the feed. Also uses the *ICalendar* and *TimeRange* class. The *TrumbaCalendarDataController* is a specific subclass for feeds that utilize the Trumba calendar service.
- *PeopleController* - access directory/person data. The only included implementation at this time is the *LDAPPeopleController* which queries information from an LDAP directory. Note this is distinct from authenticating users.

These classes also use the *DiskCache* class to cache the retrieved data.

Other included Data Parsers:

- *PassthroughDataParser* - A no-op parser. Passes the data as is.
- *JSONDataParser* - Parses JSON content into a PHP structure.
- *DOMDataParser* - Parses HTML content into a DOM Object

5.4.4 Database Access

- *db* - A database access library based on PDO. It includes abstractions for MySQL and SQLite
- *SiteDB* - Uses the main database configuration for access.

5.4.5 User Access and Authentication

- *AuthenticationAuthority* - This is the root class for authenticating users, getting user and group data. It is designed to be subclassed so each authority can provide the means of actually authenticating users, but still maintain a consistent interface for the login module. See *Authentication* for more information about the included authorities.
- *AccessControlList* - A class used by the authorization system to restrict access to modules based on user or group membership. This is especially useful for the *Administration Module*.
- *Session* - Handles the saving and restoration of user state. This is currently implemented using PHP session variables.
- *User* - The base class for identifying logged in users
- *UserGroup* - The base class for identifying groups

5.4.6 Configuration

See *Configuration* for more information on configuring Kurogo.

- *Config* - An abstract class that stores key/value data and has logic for handling replacement values (i.e referencing other keys' values within a value)

- *ConfigFile* - Provides an interface for reading and writing an ini configuration file
- *ConfigGroup* - Provides an interface for coalescing multiple configuration files to provide a single key/value store
- *ModuleConfigFile* - A specific config file class to load module config files.
- *SiteConfig* - A specific ConfigGroup that loads the critical site and project-wide configuration files.

5.4.7 Modules and Templates

- *Module* - The core class that all modules inherit from. Provides a variety of necessary services and behavior to module subclasses. See *Modules*.
- *APIModule* - The core class that all api modules inherit from.
- *WebModule* - The core class that all web modules inherit from.
- *HTMLPager* - A support class used to paginate content
- *smarty* - The Smarty Template System
- *TemplateEngine* - An subclass of the smarty object used by the framework

5.4.8 Other

- *ga* - An implementation google analytics for browsers that don't support javascript

5.5 Modules and Templates

Inside the *app* folder you will find folders that contain module and template files

5.5.1 Common

Inside the common folder are template and css files that are used by all modules. Each of these templates may have several variants for different devices. (see *Templates* for detailed information on the template system and file naming) A non-exhaustive list of these templates include:

- **footer.tpl** content placed at the bottom of most pages
- **header.tpl** content placed at the top of most pages
- **help.tpl** template used for displaying help pages
- **formList.tpl** template used for showing a list that enables input
  - **formListItem.tpl** template used for an individual form item in a list
- **navlist.tpl** template used for showing items as a list
  - **listitem.tpl** template used for an individual item in a list
- **pager.tpl** - template for providing pagination for long-form content
- **results.tpl** - template for displaying results in a list
- **search.tpl** - template for presenting a search box
- **share.tpl** - template for presenting a sharing content via social networking

- **springboard** - template for displaying content as a grid of icons
- **tabs.tpl** - template for displaying content in tabs

5.5.2 Modules

The modules folder contains all the modules that are bundled with the project. Each module contains the PHP code and template files needed for its use. It also can include CSS and Javascript files that are specific to that module. For more detailed information on module design, please see *Modules*

The naming conventions are very important (especially for case sensitive file systems):

- The folder **must** be lower case and be the same as the url of the module (/about, /home, /links)
- The folder **must** contain a PHP file named *ModulenameWebModule.php*. If the module is located in the *site* folder *and* it extends an existing module then it should be called *SiteModulenameWebModule.php*.
- The first (and ONLY) letter of the module **must** be capitalized and followed by WebModule.php.
  - **AboutWebModule.php** (NOT aboutwebmodule.php or AboutWebmodule.php)
  - **FullwebWebModule.php** (NOT FullWebModule.php or FullwebWebmodule.php)
  - **SiteNewsWebModule.php** (NOT siteNewsWebModule.php or Sitenewswebmodule.php)
- Template files go into the *templates* folder. There should be a .tpl for each *page* of the module. At minimum there should be an *index.tpl* which represents the default page (unless the module alters that behavior). Each page should be in all lower case.
- If you are overriding a project module you only need to include the pages that you are overriding.
- You may choose to place additional css style sheets in a folder named *css*
- You may choose to place additional javascript scripts in a folder named *javascript*
- You can provide default configuration files in a folder named *config*

It is possible to override an included module's behavior by creating another module in the *site* folder. For more information, please see *Extending an existing module*

5.6 Site folder

The site folder contains a series of folders for each *site*. This allows each site to have specific configuration, design and custom code. At any given time there is only one **active site**. You can enable the active site in the *config/kurogo.ini* file found in the the root of the project directory. It is important the that case used in naming the folder matches the ACTIVE_SITE case in the kurogo.ini file.

Multiple site folders exist to assist developers who might be working on different versions of their site or who want to refer to the reference implementation. Because only one site can be active, you would typically have only one site folder in a production environment.

Each site folder contains the following directories:

- *app* - Site specific templates and modules. Inside this folder you will find 2 folders
  - *common* - Site specific common templates and css
  - *modules* - Site specific modules. To promote ease when updating the framework to new versions, it is usually best if you keep site specific modules in this folder rather than in the root *app/modules* folder. If you wish to include your work in the project, please see *GitHub Repository*. Also see *Extending an existing module*.

- *cache* - Contains server generated files that are cached for performance. This folder is created if needed, but *must* be writable by the web server process.
- *config* - Contains the site specific configuration files in .ini format. Many of these files can be managed using the *Administration Module*
  - *site.ini* - The general configuration file that affects all site behavior such as timezone, log file locations, database configuration, and more.
  - *authentication.ini* - The configuration for user *Authentication*.
  - *strings.ini* - a configuration file containing strings used by the site
  - Each module's configuration is contained a folder named by its module id. There are several standard files for each module:
    - module.ini - Settings for disabling, access control, search and module variables and strings
    - feeds.ini - Specifies external data connections
    - pages.ini - Titles for each page
    - Modules may have other config files as needed
- *data* - a folder that contains data files meant to be used by the server. Unlike cache folders, these files cannot be safely deleted. Examples would include data that is not able to be generated from a web service, SQLite databases, or flat authentication files
- *logs* - Log files
- *themes* - Contains the themes available for this site. Each theme folder contains a *common* and *modules* folder that contains the CSS and image assets for the site. See *Templates* for more information.

5.7 WWW Folder

The files and folders in the www folder represent the DocumentRoot, the base of the site. To keep the structure clean, all requests are routed through the *index.php* file (the exception is for paths and folders that already exists, such as min, the minify url). It is important to note that if create additional files or folders in the www folder that it may interfere with proper operation of the framework.

5.7.1 index.php

The index script is the main controller for the framework. All requests are handled through it using an .htaccess override and mod_rewrite. The .htaccess file rewrites all requests to include a $_GET variable *_path* which includes the path requested. I.e. *http://server/module/page* becomes *http://server/index.php?_page=module/page*. Any additional data in the $_GET or $_POST variables will be available. For greater detail see *Handling Requests*

CHAPTER

SIX

DEVICE DETECTION

One of the strong features of the Kurogo framework is the ability to detect various devices and format content based on that device's capabilities. To support the classification of devices, the framework uses a Device Detection Server that contains a database of devices and outputs a normalized set of properties.

6.1 Types of Device Detection Servers

Kurogo includes an internal device detection server that parses the user agent of the user's device and returns an appropriate series of values. It contains a SQLite database that contains a series of patterns and will return the values that match that pattern. This allows you to control the entire process of detecting devices.

There is also an external device detection service available. The advantage of this service is that it will contain a more up to date database of new devices. There are 2 urls available. One is suitable for development and one for production.

See *Device Detection Configuration* for specific configuration values.

6.2 Data Format

The Kruogo Framework queries the device detection service using the *user agent* of the user's browser. The service will then return a series of properties based on the device:

- *pagetype* - String. One of the device *buckets* that determines which major source of HTML the device will received. Values include *BASIC*, *TOUCH* and *WEBKIT* (aka *COMPLIANT*)
- *platform* - The specific type of device. Values include *ANDROID*, *BBPLUS*, *BLACKBERRY*, *COMPUTER*, *FEATUREPHONE*, *IPHONE*, *PALMOS*, *SPIDER*, *SYMBIAN*, *WEBOS*, *WINMO*, *WINPHONE7*
- *supports_certificates* - Boolean. Whether this devices supports certificate based authentication
- *description* - a textual description of the device

CHAPTER

SEVEN

CONFIGURATION

The Kurogo framework requires very little setup to operate initially. For a production system, however, you are going to want to be familiar with many of the site and module options that can affect file locations, debugging information and module behavior.

All of the site's configuration is controlled using .ini files. You can either edit these files manually or use the *Administration Module* to edit most of these values.

7.1 Structure of .ini Files

Most developers and administrators should find the structure of .ini files familiar. For a complete explanation on ini files, see the documentation for the parse_ini_file() function in the PHP manual.

7.1.1 Properties

The basic element contained in an INI file is the property. Every property has a name and a value, delimited by an equals sign (=). The name appears to the left of the equals sign. Strings should be enclosed in double quote marks. Constants can be included outside quote marks. A unique feature of the framework allows you to reference other values in included ini files by using braces {} around the key name to include.

```
key1="value"
key2=CONSTANT
key3=ANOTHER_CONSTANT "value"
key4="Using value {key3}"
```

7.1.2 Sections

Properties may be grouped into arbitrarily named sections. The section name appears on a line by itself, in square brackets ([and]). All properties after the section declaration are associated with that section. There is no explicit "end of section" delimiter; sections end at the next section declaration, or the end of the file. Sections may not be nested.

```
[section]
```

7.1.3 Comments

Semicolons (;) indicate the start of a comment. Comments continue to the end of the line. Everything between the semicolon and the End of Line is ignored.

```
; comment text
```

7.2 Configuration files

When running a module, the following config files are loaded automatically:

- *config/site.ini* The framework config file. It's primary role is to indicate the active site and configuration mode
- *SITE_DIR/config/site.ini* - The site configuration file. It contains properties shared by all modules and sets up the basic environment
- *SITE_DIR/config/strings.ini* - Strings table. Includes various strings used throughout the site
- *SITE_DIR/config/MODULEID/module.ini* - Basic configuration file for the current module. Specifies properties regarding the module including disabled status, protected, secure and authorization. Also includes any unique module configurable parameters
- *SITE_DIR/config/MODULEID/pages.ini* - Title/navigation configuration for the current module.

Other modules may also load files from the *SITE_DIR/config/MODULEID* folder for external data configuration, and specific configuration for module output and formatting. Refer to the documentation for a particular module to know the composition of those files.

7.2.1 Local Files

The framework supports overriding configuration files for local server customization. Unless the configuration value *CONFIG_IGNORE_LOCAL* (defined in *config/kurogo.ini*) is set to 1, the framework will also load files with a -local in the file name for each configuration file loaded. I.e. *SITE_DIR/config/site.ini* can be overridden with *SITE_DIR/config/config-local.ini*. *SITE_DIR/config/home/module.ini* can be overridden with *SITE_DIR/config/home/module-local.ini*. It is **not** necessary to duplicate the entire file. Only the values that are different need to be in the -local file. It could also include additional values that are not present in the base config.

These files are ignored by the git version control system and are an excellent place to put sensitive file paths or credentials that should not be part of a public source code repository. It can also aid in deployment since your development machine may use different settings than a production server.

If *CONFIG_IGNORE_LOCAL* is set to 1, then -local files will be ignored. This is useful if you do not use them and may slightly improve performance.

7.2.2 Configuration Mode

In addition to -local files. There is also an option to include configuration override files by specifying a mode string. This string is like -local but can be set to any value. This will allow you to create multiple versions of configuration files, with slightly different versions of certain values and instantly switch between them. This option is set in the *CONFIG_MODE* value of *config/kurogo.ini* These files are not ignored by git.

One use of this would be to create development and production versions of some of your configuration files. You can have *SITE_DIR/config-development.ini* and *SITE_DIR/config-production.ini* with differing values for debugging. Then you can set *CONFIG_MODE* to **development** or **production**. If *CONFIG_MODE* is empty (the default), than no files will be searched. Another example would be to include authorization values for certain modules in a production environment.

Keep in mind that this setting is independent of -local files. -local files will override any option presuming *CONFIG_IGNORE_LOCAL* is not enabled.

7.2.3 Retrieving Configuration Values

There are several methods in the *WebModule object* for retrieving values from configuration files:

- getSiteVar - Retrieves a single value from the main site configuration
- getSiteSection - Retrieves a section (as an array or key=>values) from the main site configuration
- getModuleVar - Retrieves a single value from the module configuration
- getModuleSection - Retrieves a section (as an array or key=>values) from the module configuration

7.3 Site Configuration

The *SITE_DIR/config/site.ini* file configures the basic site configuration. It is broken up into several sections

7.3.1 Error handling and debugging

The properties in this section are used during development. Most of them are boolean values (0 is off, 1 is on)

- *DISPLAY_ERRORS* - Display PHP errors. This can make discovering bugs more easy. You should turn this off on a production site.
- *DEVICE_DEBUG* - When the framework is running in device debugging mode, you can prepend any framework url with *device/[PAGETYPE]-[PLATFORM]/* or *device/[PAGETYPE]/* to see that version of the page in your browser. So for example "/device/basic/about/" will show the basic version of the About module's index page.
- *MODULE_DEBUG* - Enables debugging information provided by each module. The type of information will vary by module. An example of this is showing the LDAP server used by the People module
- *MINIFY_DEBUG* - When Minify debugging is turned on, Minify adds comments to help with locating the actual file associated with a given line.
- *DATA_DEBUG* - Data debugging enables logging and certain output to debug data controller connections. When turned on, it will log url requests in the error log.
- *DEVICE_DETECTION_DEBUG* - Show the device detection info in the footer
- *PRODUCTION_ERROR_HANDLER_ENABLED* - The production error handler will email exceptions to the DEVELOPER_EMAIL address. You should treat exceptions as extraordinary situations that should normally not occur in production environments.
- *DEVELOPER_EMAIL* - an email address to send exception notices. At this time, it uses the php *mail()* function so it may not be compatible with all environments.

You should turn the _DEBUG options to off in a production environment and enable the Production Error Handler with an appropriate developer email address.

7.3.2 Site settings

- *SECURE_HOST* - Alternate hostname to use for secure (https) connections. If not included it will use the same host name.
- *SECURE_PORT* - Alternate port to use for secure connections. Typically you should leave it at the default of 443
- *LOCAL_TIMEZONE* - Set this to your environment's time zone. See http://php.net/manual/en/timezones.php for a list of valid time zones

- *LOCAL_AREA_CODE* - Set this to your environment's primary area code
- *AUTODETECT_PHONE_NUMBERS* - Turn this off to prevent the auto detection of telephone numbers in content. This is primarily only supported in iOS devices at this time.

7.3.3 Analytics

- *GOOGLE_ANALYTICS_ID* - set this to your google analytics id and the framework will utilize the google analytics server
- *PAGE_VIEWS_TABLE* - Used by the stats module to store page view summaries

7.3.4 Temp Directory

- *TMP_DIR* - This should be set to your system's temporary directory (usually /tmp)

7.3.5 Themes

- *ACTIVE_THEME* - This is set to the active theme. It should be a valid folder inside the *SITE_DIR/themes* directory.

7.3.6 URL Rewriting and the default page

In the **[urls]** section you can put a series of values that allow you to map a url to another. Typically this would be if you want to map a module's url to several possible values, perhaps to maintain historical bookmarks. The entered url will be redirected to the value you specify. For example:

- **directory = people** would map the url */directory* to */people* (i.e. the people module)

Take care that you do not create infinite redirect loops.

There is a special case for the *DEFAULT* url. This is the module that is loaded when users enter your site without a module name (i.e. the root of your site). You can configure this to show a different module depending on the type of device/platform. In the initial setting, users browsing your site from a computer will be presented with the **info** module and users browsing your site from a mobile device will be shown the **home** module.

The default option will look for the most specific value when determining which default page to show. You can create entries like such (in uppercase)

- *DEFAULT-PAGETYPE-PLATFORM* - matches the specific pagetype/platform combination. like *DEFAULT-COMPLIANT-COMPUTER* or *DEFAULT-TOUCH-BLACKBERRY*.
- *DEFAULT-PAGETYPE* - matches all the devices from a particular pagetype. Like *DEFAULT-COMPLIANT* or *DEFAULT-BASIC*
- *DEFAULT* will match any device if a more specific entry is not found

This allows you to customize the front door experience for your users.

7.3.7 Device Detection

See *Device Detection* for more details

- *MOBI_SERVICE_VERSION* - Includes the version of device detection to use. Provided for compatibility.

- *MOBI_SERVICE_USE_EXTERNAL* - Boolean. If 0, Kurogo will use the internal device detection server. If 1 it will use an external server
- *MOBI_SERVICE_FILE* - Location of device detection SQLite database if using internal detection. (typically located in LIB_DIR/deviceData.db)
- *MOBI_SERVICE_URL* - Url of device detection server if using external detection
  - (Development) https://modolabs-device-test.appspot.com/api/
  - (Production) https://modolabs-device.appspot.com/api/
- *MOBI_SERVICE_CACHE_LIFETIME* - Time (in seconds) to keep cached results from the external device detection service

7.3.8 Cookies

- *MODULE_ORDER_COOKIE_LIFESPAN* - How long (in seconds) to remember the module order customization. In production sites this should be set to a long time, like 15552000 (180 days)
- *LAYOUT_COOKIE_LIFESPAN* = How long to remember the device detection results for pagetype and platform. In production sites this should be set to a long time, like 1209600 (14 days)

7.3.9 Database

The main database connection can be used by a variety of modules for storing and retrieving values.

- *DB_DEBUG* - When on, queries are logged and errors are shown on the browser. You should turn this off for production sites or you risk exposing SQL queries when there is a database error.
- *DB_TYPE* - The database system currently supports 2 types of connections *mysql* or *sqlite* through PDO
- *DB_HOST* - used by db systems that are hosted on a server
- *DB_USER* - used by db systems that require a user to authenticate
- *DB_PASS* - used by db systems that require a password
- *DB_DBNAME* - used by db systems that require a database
- *DB_FILE* - used by db systems the use a file (i.e. sqlite).

7.3.10 Authentication

- *AUTHENTICATION_ENABLED* - Set to 1 to enable *authentication*
- *AUTHENTICATION_IDLE_TIMEOUT* - Idle Timeout in seconds before users are logged off Use 0 to disable
- *AUTHENTICATION_USE_SESSION_DB* - If 1 then session data will be saved to the site database
- *AUTHENTICATION_REMAIN_LOGGED_IN_TIME* - Time in seconds where users can choose to remain logged in even if closing their browser. If this is set to 0 then user's cannot remain logged in. Typical times are 604800 (7 days) or 1209600 (14 days).

7.3.11 Log Files

- *WEB_LOG_FILE* - Location of the processed page view log file
- *WEB_CURRENT_LOG_FILE* - Location of the active page view log file
- *LOG_DATE_FORMAT* - Date format for log files
- *LOG_DATE_PATTERN* - regex pattern of log dates, should match output from LOG_DATE_FORMAT

7.4 Module Visibility and protection

Each module contains an configuration file in *SITE_DIR/config/modules/MODULEID.ini*. This file contains values common to all modules, as well as module specific values.

- *title* - The module title. Used in the title bar and other locations
- *disabled* - Whether or not the module is disabled. A disabled module cannot be used by anyone
- *search* - Whether or not the module provides search in the federated search feature.
- *secure* - Whether or not the module requires a secure (https) connection. Configuring secure sites is beyond the scope of this document.
- *acl[]* - a series of access control list entries. See *Authentication*.
- *adminacl[]* - a series of access control list entries for administrative tasks. See *Authentication*.

It is important to turn on the disabled flag for any modules you do not wish to use. It is *very* important to make sure that the *admin* module is either disabled or protected appropriately to prevent exposure of critically important data and configuration. If you utilize logins you should make sure the *login* module requires *secure* connections if you have a valid certificate.

7.5 Home Screen

See *Home Module* for information on configuring the look and layout of the home screen.

7.6 Strings

There are a number of strings that are used throughout the framework to identify the site name the organization it is a part of. These include:

- *SITE_NAME* - The name of the site. Used in the footer and other places.
- *ORGANIZATION_NAME* - The name of the organization. Used in the about module.
- *COPYRIGHT_LINK* - Link to copyright notice (optional)
- *COPYRIGHT_NOTICE* - Copyright notice
- *FEEDBACK_EMAIL* - email address where user's can send feedback.

7.7 Administration Module

In addition to editing these files, you can use the administration module to manage the configuration. The admin module is located at */admin* and does not have an icon on the home screen.

7.8 Providing an administration interface to your module

This section is being rewritten...

CHAPTER

EIGHT

HANDLING REQUESTS

This section outlines how the framework processes HTTP requests. Generally speaking, this can be outlined as follows:

1. mod_rewrite sees if the path exists
    - There are only 2 files, and 1 folder in the DocumentRoot: index.php, robots.txt and min.
    - The min folder contains the minify library for handling consolidated versions of css and javascript assets
2. Presuming the file does not exist it will be sent to index.php for processing
3. Certain paths will map to a file in the file system and be returned or a 404 will be returned
4. You can map URLs to other URLs by updating *SITE_DIR/config/site.ini*
5. Otherwise a module based on the path is instantiated and will forward further processing. to that module. An exception is raised if the url maps to a module that does not exist

8.1 Path patterns

The index.php script will analyze the path for several patterns

- favicon.ico if a favicon.ico file exists in the CURRENT_THEME/common/images folder it will be sent to the client
- ga.php will be sent from the lib folder
- requests with a path of *common* or *modules* with a subpath of *images*, *css* or *javascript* are served using the rules according to *Pagetype & Platform Files*. This includes examples such as: /modules/home/images/x.png, /common/css/compliant.css, /modules/admin/javascript/admin.js
- requests with a path of /media will be searched for in the indicated subfolder of the current site folder: i.e. /media/file will map to *SITE_DIR*/media/file

If no pattern has been found, the script will then look at the *[urls]* section of *SITE_DIR/config/site.ini* to see if a url is found. If so, it will redirect to the indicated url.

All other requests will attempt to load a module based on the first path component of the request. The contents before the first "/" will refer the *id* of the module, the contents after the slash will be the page to load. If there is no page specified, the *index* page will be loaded. The script attempts to instantiate a module with the corresponding *id* using the *WebModule::factory* method (see *Modules* for information on how the module files are located) and includes the page and the contents of the $_GET and $_POST variables as parameters. **Note:** the trailing .php for page names is optional.

Examples:

- */home* - will load the *home* module with a page of *index*

- */about/about_site* - will load the *about* module with a page of *about_site*
- */calendars/day?type=events* will load the *calendars* module with a page of *day* and will contain a GET variable named *type* with a value of *events*.
- */news/?section=1* will load the *news* module with a page of *index* and will contain a GET variable named *section* with a value of *1*

Pages are discussed in more detail in the *Modules* section.

8.2 Pagetype & Platform Files

There are a variety of circumstances when you want to have alternate content be delivered based on the characteristics of the device making the request. The *device detection service* will contain 2 important properties that can influence which content is loaded.

- pagetype - The basic type of device, is either *basic* or *compliant*.
- platform - The specific device type. Examples include: *android*, *bbplus*, *blackberry*, *computer*, *featurephone*, *iphone*, *palmos*, *spider*, *symbian*, *webos*, *winmo*

For template files, css files, and javascript files, you can load alternate versions for different device types or platforms. The framework will load the *most specific* file available. For example, if the device is an android device it will look for the "index.tpl" file of a module in the following order:

- index-compliant-android.tpl
- index-compliant.tpl
- index.tpl

The same file from a feature phone would include the following files:

- index-basic-featurephone.tpl
- index-basic.tpl
- index.tpl

This allows you to serve different HTML markup, CSS or Javascript depending on the device. By using CSS `@import` and `{block}` functions in *templates* you can layer utilize common structure or style while providing opportunities for device specific differences as needed.

CHAPTER

NINE

MODULES

The Kurogo framework is based around modules. Each module provides a distinct set of data and services shown to the user.

9.1 The WebModule Object

Each module is a subclass of the WebModule object. Much of the core logic is located within this class including:

- Initialization of the template system
- Retrieval of configuration and runtime parameters
- Creation of internal URLs
- Authorization

9.1.1 Properties

Most of the properties used in the WebModule object exist merely to maintain state and should not be directly referenced, but rather use an accessor method to ensure future compatibility. There are some properties that you will need to use if creating your own module. These include:

- *id* (string) - This property should be set to the same name and capitalization as the module directory. This property **must** be set by all modules. For modules that are duplicates of others, the id should be the id of the parent module.
- *configModule* (string) - This property should be set to the same name and capitalization as the module directory. If not set, it will use the *id* property
- *moduleName* (string) - This property represents the canonical name of the module and is shown at on the nav bar. It can be overridden using the configuration file.
- *page* (string) - This property is set when the module initializes and represents the current page the user is viewing (based on the *request*).
- *pagetype* (string) - contains the pagetype property used in *device detection*
- *platform* (string) - contains the platform property used in *device detection*
- *args* (array) - An associative array of variables (key=>value) imported from the $_GET and $_POST request variables. Use the *getArg($key)* method to retrieve a value in a module rather than access this array directly.

9.1.2 Methods

There are 90 methods in the WebModule object. Many of them are used internally and don't require any discussion. There are several methods that you should be aware of.

Initialization

- *factory* (string $id, string $page, array $args) - This static method is called by *index.php* to setup the module behavior. It will pass the page to load as well as the arguments that part of the request. In order to separate built-in modules from site specific modules, this method will search multiple locations for the module. It is important that the name of the class matches the name of the file.
  - SITE_DIR/app/modules/example/ExampleWebModule.php
  - SITE_DIR/app/modules/example/SiteExampleWebModule.php
  - app/modules/example/ExampleModule.php
- *initialize* - This method is called first when the module is instantiated. It should contain general initialization code. If your module provides federated search capabilities than you can use this method to properly setup any data sources.
- *initializeForPage* - This method is called when viewing a page. It represents the main logic branch.

Accessors

- *getArg($key, $default)* - Retrieves an argument sent via GET/POST, if the *$key* is not present, then it will return the value specified in *$default*

Configuration

There are a number of methods to load configuration data. Configuration allows you to keep certain details such as server locations, urls, and other values out of source code. Each module has a folder of configuration files. The primary configuration data is located in the *module.ini* file. Page data is located in *pages.ini* Modules can use whatever configuration structure that suits their needs. In many cases, complex data structures will need to exist in different files.

You can retrieve values either by key or by entire section (you'll get an array of values). The following methods exist on the Module object.

- *getModuleVar($key, $section=null, $config='module')* - Gets a required module variable $key. If you specify $section it will only look in that section. Will throw an exception if the value is not present
- *getOptionalModuleVar($key, $default='', $section=null, $config='module')* - Gets an optional module variable $key. If you specify $section it will only look in that section. If it is not present, $default will be used (empty string by default)
- *getModuleSection($section, $config='module')* returns an array of values in a module section. Will throw an exception if the section is not present
- *getOptionalModuleSection($section, $config='module')* returns an array of values in a module section. Will return an empty array if the section is not present
- *getModuleSections($config)* - Returns a complete dictionary of sections=>vars=>values for a particular config file. Very handy when you basically want the array structure of an entire file

You can also retrieve values from the site configuration (site.ini). These are for values used by all modules. They are static methods on the Kurogo object.

- *Kurogo::getSiteVar($key, $section=null)* - similar to getModuleVar
- *Kurogo::getOptionalSiteVar($key, $default='', $section=null)* - similar to getOptionalModule Var
- *Kurogo::getSiteSection($section)* - similar to getModuleSection
- *Kurogo::getOptionalSiteSection($section)* similar to getOptionalModuleSection

There are also 2 other methods for getting site strings (strings.ini).

- *Kurogo::getSiteString($key)* - returns a site string. Will throw an exception if not present
- *Kurogo::getOptionalSiteString($key, $default='')* - returns a site string. Will return $default if not present

User Sessions

- *isLoggedIn()* returns whether a user is logged in or not (see *Authentication*)
- *getUser()* returns a User object of the current user (or AnonymousUser if the user is not logged in)

Setters

- *setPageTitle* - Sets the page title for this page. Normally this value comes from the *SITE_DIR/config/page/MODULE.ini* file, but you can use this method to set it dynamically.
- *setBreadcrumbTitle* - Sets the breadcrumb title for this page. Normally this value comes from the *SITE_DIR/config/page/MODULE.ini* file, but you can use this method to set it dynamically.
- *setBreadcrumbLongTitle* - Sets the breadcrumb long title for this page. Normally this value comes from the *SITE_DIR/config/page/MODULE.ini* file, but you can use this method to set it dynamically.
- *setTemplatePage* - Sets the nane of the page template to use. Normally the template is derived from the url, but you can use this method to set it dynamically.

Actions

- *redirectTo($page, $args, $preserveBreadcrumbs)* - This method will redirect to another page in the module. The *page* parameter is a string to the destination page. *args* is an associative array of arguments to pass to the page. *preserveBreadcrumbs* is a boolean (default false) whether to add the entry to the list of breadcrumbs or start a new series.

Template

- *assign(string $var, mixed $value)* - Assigns a variable to the template. In order to use variable values in your template files, you must assign them.
- *loadPageConfigFile($name, $keyName)* - Loads a configuration file named *page-name* located in the *config/MODULEID/* folder and assigns the values to the template.
- *buildBreadcrumbURL($page, $args, $addBreadcrumb)* - This method will return a url to another page in the module. The *page* parameter is a string to the destination page. *args* is an associative array of arguments to pass to the page. *addBreadcrumb* is a boolean (default true) whether to add the entry to the list of breadcrumbs or start a new series.

CHAPTER

TEN

INCLUDED MODULES

There are a number of modules that are included inside the distribution of Kurogo. In this section you can learn more about these modules, what they do and how to configure them.

10.1 Home Module

The home module represents the main portal to your application. It provides a unified list of modules available to users. It can be configured to show the list in a variety of styles.

The *SITE_DIR/config/home/module.ini* file contains the standard module configuration, but also has several other keys for controlling the configuration of the home screen.

10.1.1 Home Screen Type

```
display_type = "springboard"
```

The display type property is a value that controls whether the home screen displays like a grid of icons ("springboard") or a list of items ("list").

10.1.2 Module list and order

There are 2 sections *[primary_modules]* and *[secondary_modules]* that indicate which modules are shown on the home screen.

Each section has a list of values that represent the order of the modules and their labels. The order of these values affects the order of the modules. Each value is the format:

```
moduleID = "Label"
```

Primary modules can be rearranged and hidden by the user using the *Customize* module, secondary modules appear smaller, but cannot be rearranged or removed by the user. Keep in mind that even if the entry is not on the home screen, users can still manually navigate to the url. So if you have a modules that you do not wish to use, ensure they have been *disabled* in their module configuration file.

10.1.3 Icons

For compliant browsers, you will need to create icons for each module. These icons should be placed in: *SITE_DIR/themes/default/modules/home/images/compliant*. Each module should have an 72x72 PNG file named the same as its module id (about.png, news.png, etc.)

10.2 Info Module

The info module is, by default, shown to users who visit the site using a desktop browser. The intent it to provide users with information about your site. The default implementation contains no code, and the design included with the reference site is merely an example. You should create your own page that matches the brand and styling of your site.

If you would prefer to have users see the home page and not the info module from a desktop browser, you can change the *DEFAULT-COMPLIANT-COMPUTER* value in the *[urls]* section of your site's *site.ini* to *home*

```
[urls]
DEFAULT-COMPLIANT-COMPUTER = home
```

10.3 People Module

The people module enables sites to provide mobile access to their directory. With a few short configuration parameters you enable searching and detailed information to users on their mobile device. The built-in module supports connecting to LDAP based directories (including Active Directory)

10.3.1 Configuring the Server Connection

In order to use the people module, you must first setup the connection to your LDAP server. There are 2 required values that must be set and a few optional ones. You can set these values by either using the *Administration Module* or by editing the *SITE_DIR/config/people/feeds.ini* file directly.

- The HOST value should match the address of your LDAP server. Keep in mind that this server must be accessible from the web server the framework is hosted on. Managing network and firewall settings is the responsibility of your network administrator.
- The SEARCH_BASE value should manage the LDAP search base of your directory. You can get this value from the administrator of your LDAP directory. Examples would include "dc=example,dc=com"

In most cases, this will permit you to perform simple search and details views of your data.

Optional values

- *CONTROLLER_CLASS* allows you to set a different class name for the controller. The default is LDAPPeopleController. You could write your own subclass of PeopleController to retrieve the values from a different source, such as a database.
- *PERSON_CLASS* allows you to set a different class name for the returned user objects when searching. This allows you to write custom behavior to handle the data in your directory service.
- *PORT* - Optional (Default 389) The port to connect. Use 636 for SSL connections (recommended if available)
- *ADMIN_DN* - Some servers do not permit anonymous queries. If necessary you will need to provide a full distinguished name for an account that has access to the directory. For security this account should only have read access and be limited to the search bases to which it needs to access.
- *ADMIN_PASSWORD* - The password for the *ADMIN_DN* account.

10.3.2 Configuring the Detail Fields

Once you have configured the server settings, you need to configure the field mappings between your server and the detail view. If your LDAP directory uses standard fields, then most fields should map automatically, however, you may still want to customize how it displays or the order of the fields.

The fields are configured in the *SITE_DIR/config/people/page-detail.ini* file. Each field is configured in a section (the section name should be unique, but it otherwise irrelevant). The order of the sections controls its order in the detail view. Within each section there are several possible values to influence how a field is displayed:

- *label* - (required) A text label for the field. Can include HTML tags.
- *attributes* - (required) Array of fields to put in the contents (should map the the field names in your backend system)
- *format* - (optional) A string for vsprintf to format the attributes. Only needed if more than one attribute is provided.
- *type* - (optional) One of "email", "phone", or "map". Used to format and generate links.
- *section* - (optional) If this field belongs to a section, the name of that section
- *parse* - (optional) A function which will be run on the LDAP results before display. Generated with *create_function*. Gets the argument "$value" and returns the formatted output.

10.3.3 Configuring the Fixed Entries

This module supports the ability to show a list of directory entries on the module index page. You can update the contents of this list by editing the *SITE_DIR/config/people/page-index.ini*. Each entry is a numerically 0-indexed list of sections. Each section has 4 values that map to the the values used by the *listItem* template. Note that because it's displaying a list with URLs, the entries do not have to be phone numbers, but could be any URL.

- *title* - The Name of the entry as it's shown to the user
- *subtitle* - The subtitle, typically the phone number for phone entries.
- *url* - The link it should point to, use *tel:XXXXXXXX* links for phone numbers
- *class* - The CSS class of the item, such as *phone*, *map*, *email*

10.4 Video Module

The video module enables sites to provide mobile access to their video content on 3rd party websites such as Brightcove and YouTube.

10.4.1 Configuring the Sources

The module allows you to organize your videos by section using a distinct feed for each section. Each section contains information on the service provider and can either filter by tag or author, in addition to full textual searches. Depending on the source there are other options to configure. Feeds are configured in the *SITE_DIR/config/video/feeds.ini* file. Each feed is contained in a section. The name of each section is generally not important, but must be unique.

Within each feed you use the following options:

- *CONTROLLER_CLASS* - The DataController to use. Currently supported controllers include the *YouTubeVideoControler* and *BrightcoveVideoController*.
- *TITLE* - The textual label used when showing the section list
- *TAG* - optional, used to limit the results by tag
- *AUTHOR* - optional, used to limit the results by author

BrightcoveVideoController

In order to to use the Brightcove service, you must also include several other parameters. These values are available from Brightcove‘

- token
- playerKey
- playerId

10.5 Calendar Module

The calendar module provides an mobile interface to a series of events. You can browse events by day, category or list, and then view any details of the event. The built in module supports parsing and viewing events in iCalendar (ICS) format.

10.5.1 Configuring the Calendar Feed

In order to use the calendar module, you must first setup the connection to your data. There are 2 required values that must be set and a few optional ones. You can set these values by either using the *Administration Module* or by editing the *SITE_DIR/config/calendar/feeds.ini* file directly.

The module supports multiple calendars. Each calendar is indicated by a section in the configuration file. The name of the section becomes the *type*, used in URLs to indicate which calendar to use. When the type parameter is not indicated in a url, the first calendar is used.

- The TITLE value is a label used to name your calendar feed. It will be used in the heading when browsing and viewing events.
- The BASE_URL is set to the url of your ICS feed. It can be either a static file or a web service.

Optional values

- CONTROLLER_CLASS - allows you to set a different class name for the controller. The default is CalendarDataController. You could write your own subclass to adjust the URL if your source is a web service. The framework also includes an implementation suitable for users who host their calendar data on the Trumba event service.
- PARSER_CLASS (default ICSDataParser) set this to a subclass of *DataParser*. You would only need to change it if your data source returns data in a format other than iCalendar (ICS).
- EVENT_CLASS (default ICalEvent) allows you to set a different class name for the returned event objects when searching. This allows you to write custom behavior to handle custom fields in your feed.

10.5.2 Configuring the Detail Fields

Once you have configured the feed settings, you need to configure how the detail view displays and what values to use. Each field is configured in a section, the section name maps to an event field. The order of the sections controls its order in the detail view. Within each section there are several possible values to influence how a field is displayed. All are optional.

- *label* - A text label for the field.
- *type* - One of “datetime, “email”, “phone”, “category”, “url”. Used to format and generate links.
- *class* - CSS class added to the field

10.5.3 Configuring the Initial Screen

The index page can be configured to show a list of links to show views of the calendars you have configured. You can update the contents of this list by editing the *SITE_DIR/config/calendar/page-index.ini*. Each entry is a section. Each section has values that map to the the values used by the *listItem* template.

- *title* - The Name of the entry as it's shown to the user
- *subtitle* - The subtitle, typically shown below the title
- *url* - The link it should point to. Although you can link to any url, you would typically link to one of the pages within the module. The calendar view pages require you to pass the *type* parameter to indicate which calendar to show:
  - *day* - Shows events for a given day.
  - *year* - Shows all events for a given 12 month period. You can indicate the starting month by passing the month parameter
  - *list* - Shows the next events beginning with the present day. Default limit is 20 events.
  - *categories* - Shows a list of categories. Currently this requires special support to get a list of categories.
- *class* - The CSS class of the item, such as *phone*, *map*, *email*

10.5.4 Configuring User Calendars

There is initial support for viewing user calendars. Currently the only supported calendar system is Google Apps for Business or Education. To enable User Calendars:

- Setup the *authority* for your Google Apps Domain.
- Edit the *config/calendar/module.ini* and set *UserCalendarListController* to *GoogleAppsCalendarListController*.
- If you wish to show resource availability you can set *ResourceListController* to *GoogleAppsCalendarListController*

When this setting is enabled, users who login to their Google Apps account will see their calendars in the calendar module.

10.6 News Module

The news module shows a list of stories/articles from an RSS feed. If the feed provides full textual content, the article is shown to the user in a mobile friendly format. If the feed does not contain full text then the module will redirect the browser to the URL of the article.

10.6.1 Configuring News Feeds

In order to use the news module, you must first setup the connection to your data. There are 2 required values that must be set and a few optional ones. You can set these values by either using the *Administration Module* or by editing the *SITE_DIR/config/news/feeds.ini* file directly.

The module supports multiple feeds. Each feed is indicated by a section in the configuration file. The name of the section should be a 0-indexed number. (i.e. the first feed is 0, the second feed is 1, etc). The following values are required:

- The TITLE value is a label used to name your feed. It will be used in the drop down list to select the current feed

- The BASE_URL is set to the url of your News feed. It can be either a static file or a web service.

Optional values

- CONTROLLER_CLASS - allows you to set a different class name for the controller. The default is RSSDataController. You could write your own subclass to adjust the URL if your source is a web service.
- PARSER_CLASS set this to a subclass of *DataParser*. You would only need to change it if your data source returns data in a format other than RSS/Atom or RDF. The default is RSSDataParser.
- ITEM_CLASS allows you to set a different class name for each item in the feed. This would allow you to handle a feed that has custom fields
- ENCLOSURE_CLASS allows you to set a different class name for enclosures. This would allow you to handle custom behavior for enclosures, including images, video and audio.

10.7 Emergency Module

The emergency module provides a mobile interface to a sites emergency information. The module can display the latest emergency information and a list of emergency contacts. The data source for this module can come from a drupal server, running emergency drupal module which can be found in the add-ons at *add-ons/drupal-modules/emergency*, (Currently only supports Drupal 6). Alternatively, a standard RSS feed can be used for the emergency notice, and the contacts list can be configured with an ini file.

10.7.1 Configuring the Server Connection

In order to use the emergency module, you must first setup the connection to your data. If you want to display an emergency notice you will need to include a *notice* section in *config/emergency/feeds.ini*. In the *notice* section you will need to configure the url for the emergency notice RSS feed.

- If you are using the add-on emergency drupal module, you can set the BASE_URL to "http://YOUR_DRUPAL_SERVER_DOMAIN/emergency-information-v1", where YOUR_DRUPAL_SERVER_DOMAIN. Otherwise just set the BASE_URL to the appropriate RSS feed.

If you also want to include emergency contact phone numbers, you will need to include a *contacts* section in *config/emergency/feeds.ini*

Configure Contacts List

Configure contacts list to connect to the drupal emergency module:

- CONTROLLER_CLASS = "DrupalContactsListDataController"
- DRUPAL_SERVER_URL = "http://YOUR_DRUPAL_SERVER_DOMAIN"
- FEED_VERSION = 1

Otherwise you can configure the contacts list directly in an ini file with:

- CONTROLLER_CLASS = "INIFileContactsListDataController"
- BASE_URL must point to the appropriate ini file

The ini file will need a *primary* section for primary contacts and a *secondary* section for secondary contacts. Each contact is formatted as follows:

```
title[] = "Police"
subtitle[] = ""
phone[] = "6173332893"
```

10.7.2 Using Drupal Emergency Module

Installation

This add on module requires Drupal 6, Drupal 7 is not yet supported. Follow the standard procedure for installing a drupal module, which is:

- In order to install this module you must first install the drupal CCK (Content Creation Kit) module and the drupal Views module
- copy *add-ons/drupal-modules/emergency* into the *sites/all/modules/* directory
- In the drupal administration panel go to modules then select the "Emergency Info" module and click "save configurations".

Usage

To input an emergency notification: create a node of content type "Emergency Notification", the RSS feed will only show the most recently updated Emergency Notification.

To input emergency contacts: create a node of type "Emergency Contacts" and fill out your primary and secondary emergency contacts. Note if you create more than one node of this type the RSS feed will only show the most recently updated, you will probably not want to create more than one node of this type, but instead just update one single node with the most up-to-date contact information.

If the RSS feed generated at *http://YOUR_DRUPAL_SERVER_DOMAIN/emergency-contacts-v1* is missing the contact information you entered you may need to go to */admin/user/permissions* and enable anonymous user for view *field_primary_contacts* and *field_secondary_contacts*

10.8 Map Module

The map module allows you to browse and search for places and view them on a map. Places may be grouped by campus, category, and subcategory.

10.8.1 Configuring Campus Locations

If an institution specifies multiple campuses, the home screen will show a list of campus locations. If there is a single campus, the home screen will show a list of categories to browse by.

Campuses are configured in the *SITE_DIR/config/map/campus.ini* file. Each separate campus has an section as follows::

```
[boston]
title = "Boston Maps"
center = "42.3584308,-71.0597732"
address = "1 Massachusetts Ave., Boston MA 02115"
description = "Description"
```

- The section title is the *id*, a short string to identify the campus.
- *center* is the representative latitude and longitude of the campus.
- You can also include *address* and *description*

10.8.2 Map View Types

The map module currently supports five types of map views generators.

JavaScript based maps (compliant and tablet only)

- Google Maps
- ArcGIS JavaScript

Static image based maps

- Google Static Maps,
- WMS
- ArcGIS *export* API

10.8.3 Configuring Map Data Feeds

The map module currently supports two types of data sources for getting location information, KML and ArcGIS Server (KML is recommended, although support for other data source types will be added in the future). Each data feed is represented as a *category* that a user may browse by from the home screen or within a campus.

The feed configuration file is in *SITE_DIR/config/map/feeds.ini*. Each feed has the following fields:

- *TITLE* is a descriptive name of the category that shows up on the map home screen (for single campuses) or in the campus home screen
- *SUBTITLE* is an optional brief description that appears in small text alongside the title
- *BASE_URL* is the URL location of the data source. This may be a file URL. (i.e. a path)
- *CONTROLLER_CLASS* is the data controller class associated with the type of data source. If the data source is KML, you should use *KMLDataController*. For ArcGIS Server, you should use *ArcGISDataController*.
- *STATIC_MAP_CLASS* is the type of static map image used to display the map. This field is required as lower-end devices can only use static maps. Acceptable values are:
  - GoogleStaticMap
  - ArcGISStaticMap
  - WMSStaticMap
- *STATIC_MAP_BASE_URL* is the base URL of the map image server. This is not required for Google Static Maps.
- *JS_MAP_CLASS* is optional and refers to the type of JavaScript map to use on compliant/tablet devices. Acceptable values:
  - GoogleJSMap
  - ArcGISJSMap
- *DYNAMIC_MAP_BASE_URL* is the base URL of the map image server. This is not required for Google JavaScript Maps.
- *SEARCHABLE* is a boolean value that indicates whether or not this data source should be included in search results.
- *DEFAULT_ZOOM_LEVEL* is the default zoom level of the map image.
- If your insitution has multiple campuses, the *CAMPUS* field specifies which campus this data feed belongs to.
- If you want your data feed to be included in search results, but do not wish to make it a browseable category, you may set the optional *HIDDEN* value to 1

10.8.4 Configuring Map Search

The default map search traverses all KML feeds that have SEARCHABLE set to 1 and finds all Placemarks with a matching title (or location for the "nearby" search that occurs on detail pages). If you have an external search engine, you may override this behavior by subclassing MapSearch in your site lib directory and specifying your class as MAP_SEARCH_CLASS in *SITE_DIR/config/map/module.ini*

10.9 Links Module

The links module presents a list of link items to other pages or sites. You can customize the introductory statement, the manner in which the links are presented and the links themselves.

10.9.1 Configuration

There are several configuration values that affect the display of the links module.

display_type - Similar to the *home module* you can specify either *list* or *springboard*.

Strings

description - This string will show at the top of the page when viewing the list

Links

Links are described using array syntax. There are 3 keys that each link has: *title*, *url* and *icon*. For each link you declare the title, url and icon properties. It is important to include the property even if it is not available (i.e. if there is no icon, just include an empty string)

The title and url represent the link text and url respectively. The icon represents an optional icon that is displayed in the *springboard* display type. These files should be placed in the *SITE_DIR/themes/default/modules/links/images/compliant* folder. You may need to create this folder.

```
[links]
title[] = "This is link 1"
url[]   = "http://example.com/urlforlink1"
icon[]  = "link1_icon.png"
title[] = "This is link 2"
url[]   = "http://example.com/urlforlink2"
icon[]  = "" ; link 2 does not have an icon
title[] = "This is link 3"
url[]   = "http://example.com/urlforlink3"
icon[]  = "link3_icon.png"
```

10.10 Content Module

The content module is a generic module designed to fetch and display freeform content from other sources. It can fetch and display content from another HTML site or display an item from an RSS feed. You can also configure it to display static content configured using the configuration file or administration module.

By default, the content module displays its feeds in a list. Then the user selects a feed (shown using its title) and the content of the feed is shown. If there is only 1 feed then that feed is shown instead of the list.

10.10.1 Configuring Content Feeds

You can specify any number of pages to show in the *SITE_DIR/config/content/feeds.ini* file. Each feed is represented by a section, the name of that section represents the "page" of the module. There are several properties to configure:

- *TITLE* - The title of the feed. This is shown in the list and in the navigation bar
- *CONTENT_TYPE* - the type of content. Values include:
  - *html* - Static html text that is included in the *CONTENT_HTML* property
  - *html_url* - Fetch HTML content from the *BASE_URL* property. Set the *HTML_ID* property to indicate which HTML element to retrieve (good for extracting out content from navigation/headers/footers). If HTML_ID is not specified then the <body> tag will be returned.
  - *rss* - Fetch RSS content from the *BASE_URL* property. Will retrieve the content from the first item in the feed. Good for CMS's that expose their content via RSS. Ensure that this feed contains the full content and not just a link

10.11 URL Module

The url module simply redirects the user to an external url. This is an *abstract* module and in order to use it, you must *Copying a Module*

10.11.1 Configuration

It has one configuration parameter:

url - A url to redirect to.

10.12 Customize Module

The customize module allows users to change the ordering or display of modules on your site's home screen. The module uses *cookies* on the user's browser to save the results, so the effects are limited to the user's device/browser.

10.12.1 Configuration

There are no configurable parameters in this module. If you do not wish for users to be able to adjust the home screen, you can disable this module.

10.13 About Module

The about module provides a standardized way to include information about your site and organization and allow users to contact you.

10.13.1 Configuration

You can configure the values for the menu list as well as the content in the pages.

In the *config/about/page-index.ini* file you can configure the list items that appear in the module. These values map to the *listitem.tpl* template.

There are 2 strings defined in the *[strings]* section of the *config/about/module.ini* file. The *SITE_ABOUT_HTML* value is shown in the *About this website* section. The *ABOUT_HTML* value is shown in the *About {Organization}* section. Each of those values are represented by arrays. Each element of the array represents a paragraph of text.

```
[strings]
SITE_BLOCK_HTML[] = "This is a paragraph"
SITE_BLOCK_HTML[] = "This is another paragraph with <i>html</i>"
```

Take care to ensure your HTML is valid markup. This module does not attempt to adjust any HTML in the configuration.

10.14 Login Module

The login module is used by sites that provide protected or personalized experience for their modules. It provides a unified interface to log into the site.

10.14.1 Configuration

There are several configuration values that affect the display of the login module. They are all in the *strings* section of *module.ini*

- *LOGIN_MESSAGE* - A message shown at the header of the login page
- *LOGIN_LABEL* - The label used for the login field of the login form
- *PASSWORD_LABEL* - The label used for the password field of the login form
- *LOGIN_FOOTER_URL* - If specified, a url that is included in the footer of the login form
- *LOGIN_FOOTER_TEXT* - If specified, text that is included in the footer of the login form

10.15 Stats Module

The statistics module (/stats) provides an interface to view your site's local analytics. You can view page views by week, 12 week, year and 3 year increments. It will summarize traffic by platform and module. This gives you an idea of what content your visitors are viewing.

CHAPTER

ELEVEN

TEMPLATES

In addition to the logic parts of the module, pages use templates to output the content to the browser.

The framework utilizes the Smarty template engine. This engine has a variety of features including template inheritance, inclusions, looping constructs, and variables. The framework wraps around the Smarty engine using it to display the HTML. Smarty uses a variety of special tags enclosed in braces {} to handle variables, functions and control structures. The tags most often used in the framework are explained below. For a quick reference see the Smarty Crash Course.

As part of the display process, the template engine is initialized, and a template file is displayed based on the current page. The engine will search the module folder for a template file with the name name as the current page based on the rules of *Pagetype & Platform Files*.

11.1 Variables and Modifiers

Including values from variables is a two step process:

1. In your module PHP file, call *$this->assign(string $var, mixed $value)* to assign a value to a template variable. `$this->assign('thing', 42)` will assign the value 42 to the template variable "thing"
2. In your template you can refer to this variable as *{$thing}*

You can also use modifiers to alter the presentation of a value.

11.2 Including and Extending Templates

Templates can include other templates. This allows the creation of reusable blocks for common fragments of HTML. The framework heavily utilizes this feature.

To include a template use the {include} tag. Use it in the framework like this:

```
{include file="findInclude:template.tpl"}
```

Using the *findInclude* ensures that if there are multiple versions of the template, the proper one will be used based on the pagetype and platform.

You can also assign variables when including the template:

```
{include file="findInclude:template.tpl" thing=42}
```

This will assign the value 42 to the variable thing

11.2.1 Blocks

When designing templates for multiple device types, often the case is that you only need to change a certain part of the template and leave the rest as is. Smarty has this capability to *extend* templates replacing only what's needed. It uses 2 tags, {block} and {extend} to provide this feature

Consider a base template template.tpl:

```
This template is the base template.

{block name="content1"}
This is some content
{/block}

{block name="content2"}
This is some more content
{/block}
```

Notice that the {block} tag has an opening and closing part. We can use the {extends} tag on the specific types. For a template that extends another, you simply provide alternate content for whatever blocks you wish to replace. If you wish to eliminate a block, simply include a blank block pair. If you do not specific a block, it will be included as is.

An example for template-compliant.tpl:

```
{extends file="findExtends:template.tpl"}

{block name="content1"}
  This content will be shown to compliant browsers
{/block}
```

In this case, the *content1* block will have alternate content and the *content2* block will be displayed as is.

An example for template-basic.tpl:

```
{extends file="findExtends:template.tpl"}

{block name="content2"}{/block}
```

In this case, the *content2* block will not be shown at all and the *content1* block will be displayed as is.

This technique will permit you to create layered content that has exceptions or alternative versions for different device types. Keep in mind that in child templates, you can only define content inside {block}s, any content outside the blocks will be ignore. See the included module templates for more examples and the section on template inheritance in the Smarty Documentation.

11.3 Control Structures

You can include some basic logic inside your templates to affect flow and conditionally present content. Most of these structures utilize syntax that is identical to the corresponding PHP structures.

11.3.1 {foreach}

Iterates through an array:

```
{foreach $arr as $key=>$value}
  {$key} = {$value}
{foreach}
```

See more in the Smarty Documentation

11.3.2 {if} / {else}

Conditionally displays content:

```
{if $test}
This will be displayed if test is true
{/if}
```

Smarty uses the same conventions as PHP to determine the truth value of an expression. See more in the Smarty Documentation

11.4 Standard Template Fragments

There are a variety of template fragments included that will allow you to include common interface elements in your own templates.

11.4.1 header.tpl / footer.tpl

The header and footer files should generally appear at the top and bottom respectively of your main template files. This ensures that the site navigation and other wrapper content:

```
{include file="findInclude:common/templates/header.tpl" scalable=false}

Content goes here....

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

11.4.2 navlist.tpl

One of the most important elements is the navigation list. It renders an HTML list based on an array of list items. This list is formatted appropriately for the device.

There are several variables you can pass to affect how it is displayed:

- *navlistID* this will assign the value to the id of the list. This would allow custom CSS rules to be applied to this list
- *navlistItems* an array of list items (each of which is an array). See *listitem* for a list of keys each list item should have
- *secondary* adds the *secondary* class to the navlist

11.4.3 listitem.tpl

Used by the navlist template for each list item. When passing the values to the navlist each item in the array is a list item. Each item should be an array. There are a variety of keys used for each item:

- *title* - The text shown on the list item

Optional keys

- *label* - A textual label for the item

- *boldLabels* - if true, the label will be bolded
- *labelColon* - If false, the colon following the label will be suppressed
- *url* - A url that this item links to when clicked/tapped.
- *img* - A url to an image icon that is displayed next to the item
- *imgWidth* - The width of the image
- *imgHeight* - The height of the image
- *imgAlt* - The alt text for the image
- *class* - CSS class for the item
- *subtitle* - Subtitle for the item
- *badge* - Content (typically numerical) that will appear in a badge

CHAPTER

TWELVE

STYLE AND THEMES

The Kurogo Framework has a theming layer which allows sites to make most stylistic changes to the web application without modifying the core libraries. The advantage of using the theming layer is that site changes are isolated from the framework sources and can be more easily moved to a new version of the framework.

The core visual interface of Kurogo lives in "app/". It is made up of HTML templates, CSS and Javascript files. All HTML, CSS and Javascript in the core interface can be overridden by a theme.

Each theme is contained within a directory inside the *SITE_DIR/themes* folder. By convention the default theme is named *default*.

Themes have the same directory structure as the core visual interface directory (app/). This allows paths in the CSS and HTML to be the same for the core interface and the theme interface.

12.1 CSS and Javascript

All CSS and Javascript files are loaded automatically using Minify. Rather than having to specify each CSS and Javascript file per page, Minify locates the files based on their names. The naming scheme is similar to that of the templates, except there is a special file name "common" which indicates the file should be included for all devices:

12.1.1 CSS Search Paths:

CSS search paths from least specific to most specific. All matching CSS files are concatenated together from least specific to most specific. This allows you to override styles for specific pages or devices.

Check common core files in */app/common/css/* for:

- common.css
- [PAGETYPE].css
- [PAGETYPE]-[PLATFORM].css
- [PAGE]-common.css
- [PAGE]-[PAGETYPE].css
- [PAGE]-[PAGETYPE]-[PLATFORM].css

Check module core files in */app/modules/[current module]/css/* for:

- common.css
- [PAGETYPE].css

- [PAGETYPE]-[PLATFORM].css
- [PAGE]-common.css
- [PAGE]-[PAGETYPE].css
- [PAGE]-[PAGETYPE]-[PLATFORM].css

Check common theme files in *SITE_DIR/themes/[ACTIVE_THEME]/common/css/* for:

- common.css
- [PAGETYPE].css
- [PAGETYPE]-[PLATFORM].css
- [PAGE]-common.css
- [PAGE]-[PAGETYPE].css
- [PAGE]-[PAGETYPE]-[PLATFORM].css

Check module theme files in *SITE_DIR/themes/[ACTIVE_THEME]/modules/[current module]/css/* for:

- common.css
- [PAGETYPE].css
- [PAGETYPE]-[PLATFORM].css
- [PAGE]-common.css
- [PAGE]-[PAGETYPE].css
- [PAGE]-[PAGETYPE]-[PLATFORM].css

12.1.2 Javascript Search Paths:

Because Javascript does not allow overriding of functions, only the most device specific file in each directory is included, and theme files completely override core files. When overriding be aware that you may need to duplicate code or move it into a common file to get it included on multiple pagetypes or platforms.

Check common theme files in *SITE_DIR/themes/[ACTIVE_THEME]/common/javascript/* for:

- common.js
- [PAGETYPE]-[PLATFORM].js or if not check [PAGETYPE].js
- [PAGE]-common.js
- [PAGE]-[PAGETYPE]-[PLATFORM].js or if not check [PAGE]-[PAGETYPE].js

If there are no common theme files, check common core files in /app/common/javascript/* for:

- common.js
- [PAGETYPE]-[PLATFORM].js or if not check [PAGETYPE].js
- [PAGE]-common.js
- [PAGE]-[PAGETYPE]-[PLATFORM].js or if not check [PAGE]-[PAGETYPE].js

Check module theme files in *SITE_DIR/themes/[ACTIVE_THEME]/modules/[current module]/javascript/* for:

- common.js
- [PAGETYPE]-[PLATFORM].js or if not check [PAGETYPE].js
- [PAGE]-common.js

- [PAGE]-[PAGETYPE]-[PLATFORM].js or if not check [PAGE]-[PAGETYPE].js

If there are no module theme files, check module core files in */app/modules/[current module]/javascript/* for:

- common.js
- [PAGETYPE]-[PLATFORM].js or if not check [PAGETYPE].js
- [PAGE]-common.js
- [PAGE]-[PAGETYPE]-[PLATFORM].js or if not check [PAGE]-[PAGETYPE].js

Because Minify combines all files into a single file, it can be hard to tell where an given line of CSS or Javascript actually comes from. When Minify debugging is turned on (MINIFY_DEBUG == 1), Minify adds comments to help with locating the actual file associated with a given line.

Note that the framework caches which files exist so it doesn't have to check all the possible files on every page load. If you add a new file you may need to empty the minify cache to pick up the new file.

12.2 Images

Because images can live in either the core templates folder or the theme folder, image paths have the theme and platform directories added automatically. Images are either common to all modules or belong to a specific module. In order to allow flexible image naming, the device the image is for is specified by folder name rather than file name.

Images are searched across paths and the first image file present is returned.

Common Image Search Paths: (ie: /common/images/[IMAGE_NAME].[EXT])

Check theme images in *SITE_DIR/themes/[ACTIVE_THEME]/common/images/* for:

- [PAGETYPE]-[PLATFORM]/[IMAGE_NAME].[EXT]
- [PAGETYPE]/[IMAGE_NAME].[EXT]
- [IMAGE_NAME].[EXT]

Check core images in */app/common/images/* for:

- [PAGETYPE]-[PLATFORM]/[IMAGE_NAME].[EXT]
- [PAGETYPE]/[IMAGE_NAME].[EXT]
- [IMAGE_NAME].[EXT]

Module Image Search Paths: (ie: /modules/[MODULE_ID]/[IMAGE_NAME].[EXT])

Check theme images in *SITE_DIR/themes/[ACTIVE_THEME]/modules/links/images/* for:

- [PAGETYPE]-[PLATFORM]/[IMAGE_NAME].[EXT]
- [PAGETYPE]/[IMAGE_NAME].[EXT]
- [IMAGE_NAME].[EXT]

Check core images in */app/modules/[MODULE_ID]/images/[PAGETYPE]-[PLATFORM]/* for:

- [PAGETYPE]-[PLATFORM]/[IMAGE_NAME].[EXT]
- [PAGETYPE]/[IMAGE_NAME].[EXT]
- [IMAGE_NAME].[EXT]

The rationale for searching for images rather than just specifying the full path is so that themes don't have to override a template just to replace an image being referenced inside it with an IMG tag. By dropping their own version of the image in the theme folder, the theme image will automatically be selected. The device selection aspect of the image search algorithm is mostly just for convenience and to make the templates and CSS files more terse.

Note that image paths in CSS and templates should always be specified by an absolute path (ie: start with a /) but not contain the protocol, server, port, etc. Any url base or device path will be prepended automatically by the framework.

12.3 Important Assets to customize

With the understanding of how assets are loaded, there are several file locations you should be aware of.

12.3.1 Background and Text

- background
- text classes

12.3.2 Navigation

- home icon
- module icons
- nav background

12.3.3 Home Screen

- header graphic
- module icons

CHAPTER

THIRTEEN

STANDARD LIBRARIES

There are a number of included libraries that can provide various services.

13.1 Remote Data Gathering and Parsing

Retrieving and parsing remote data is an important task of web applications. In order to provide a consistent interface, a series of abstract classes and libraries have been provided.

The hierarchy looks like this:

url -> DataController (getData/items) -> DataParser (calls parseData) -> Returns Objects

13.1.1 DataController

A class that handles the retrieval of data from a data source. You set a URL, and a parser (a subclass of DataParser). The class will retrieve the data, cache it based on a provided cache lifetime, and then run the data through the parser to generate a PHP data structure. Typically the DataController is the public interface to a web service.

This class is discussed in further depth in *Data Controller*

13.1.2 DataParser

This class that handles the parsing of the data retrieved from a DataController. It generally uses just one method *parseData* which is passed a string of the data. It is the responsibility of the parser to return an appropriate PHP structure that can be used in the application. In some cases, it might benefit to have layer cache the results of the parsing if this is an expensive operation that might be repeated often. The responsibility of caching these results is up to subclasses since implementations needs may vary.

Currently there are 4 included DataParser implementations

- *ICSDataParser* - Parses data within an iCalendar (ICS) file. Will return an iCalendar object. (see lib/iCalendar.php)
- *JSONDataParser* - Parses the string as JSON. Will return the string decoded into its PHP equivalent data type.
- *PassthroughDataParser* - Does no processing, simply returns the string as is. This is useful when you want to use the data controller, but no parsing is necessary.
- *RSSDataParser* - Parses data within a RSS/Atom/RDF feed using the XML parser. Will return an array of RSSItem objects (see lib/RSS.php)

13.2 Data Validation

There are several utility methods available to perform validation of input. They are implemented as static methods of the *Validator* object. Each return a boolean of true or false depending on whether the value is valid for the particular

- *isValidEmail* - returns true if the value is a valid email address
- *isValidPhone* - returns true if the value is a valid phone number (currently only works for US/Canada numbers)
- *isValidURL* - returns true if the value is a valid url

CHAPTER

FOURTEEN

DATA CONTROLLER

One of the most important classes to understand in Kurogo is the DataController class. This powerful class is designed to abstract the process of gathering data from a web service. Controllers should be designed to provide an interface to the data in a data centric way rather than a service centric way.

14.1 Usage

Using the existing data controllers in modules is very straight forward.

14.1.1 Instantiation

The controller is instantiated using the *DataController::factory($id, $args)* method. The $id parameter is a string representing the subclass to instantiate, the $args parameter is an array of options:

```
<?php

$controller = DataController::factory('MyDataController', $args);
```

Typically the args will come from a *configuration* file.

After instantiation, the DataController class will call the *init($args)* method. This creates a DataParser and sets the properties of the controller based on the options. There are several common options you can include (some may be required by the class you are creating):

- *PARSER_CASS* - The class name of the parser to use. If not present it will use the value of the Controller class' DEFAULT_PARSER_CLASS property.
- *BASE_URL* - Sets the base url of the request
- *TITLE* - Sets the title of the controller
- *CACHE_LIFETIME* - Sets the cache lifetime, in seconds

Each subclass can define its own set of options and handle those in its *init* method. Just make sure to call *parent::init($args)* first. These arguments are also sent to the *init* method of the controllers's DataParser class. Subclasses might also set default filters that should be sent in all requests.

14.1.2 Setting the properties

Set appropriate parameters for the request. This might include:

- Setting a range of dates to return for a calendar controller

- Setting a category of items to return

Each controller has a specific interface of methods to manage the filters for the request. There is an internal method *addFilter($var, $value)* that will add parameters to the url that will be sent to the service. Modules should generally not set these filters, but instead use logical methods to abstract the options being set.

14.1.3 Retrieving the Items

Once the properties have been set, you can request the items that meet the request using the *items($start ,$limit)* method. The *$start* parameter is a 0-indexed integer to represent which item in the sequence to start. *$limit* represents how many entries to return. If those parameters are omitted, it will return all items present in the request.

14.2 Subclassing

You may wish to override the default value of several properties:

- $DEFAULT_PARSER_CLASS to the name of the default parser class you wish to use. Should be subclass of DataParser
- $cacheFolder - the name of the folder within the CACHE_DIR where downloaded files will be cached
- $cacheFileSuffix - a suffix to use for the cached files

There are several methods that you should be familiar with to use this class appropriately:

- *addFilter($filter,$value)* / *removeFilter($filter)* - Maintains a internal array of key/value filters that your controller can use to generate a filtered result set. The default implementation uses these filters as parameters in your url request.
- *setBaseURL($url)* - Sets the base url to use. You will have the opportunity to manipulate the url that gets used if you subclass the *url()* method.

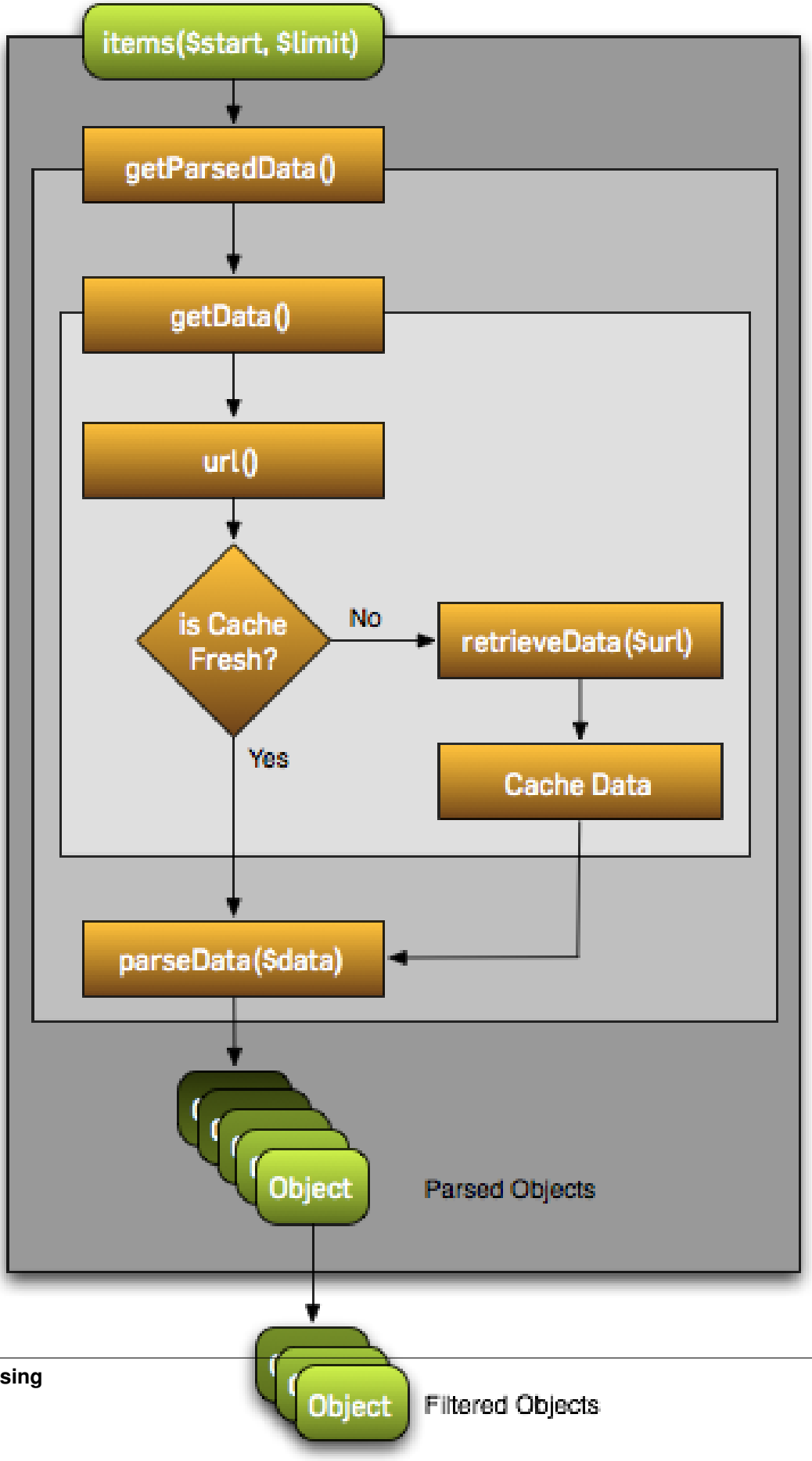
14.2.1 The Retrieval Process

The data retrieval process has been abstracted in a way that allows subclasses to customize as few or many steps as the situation warrants.

The *items* method calls *getParsedData()*, which consists of retrieving the raw data with *getData* followed by calling *parseData* to generate an array of objects.

14.2.2 Override:

You can override any part of the process. Generally you want to override the most specific piece so you gain the benefits of caching and reduce the amount of code you need to implement.


items($start, $limit)
getParsedData()
getData()
url()
is Cache Fresh?
No
retrieveData($url)
Yes
Cache Data
parseData($data)
Object
Parsed Objects
Object
Filtered Objects

- *items($start, $limit)* if you want complete control of the data retrieval process. You will be expected to return an array of item objects. It is also important for you to call *setTotalItems($totalItems)* to return the number of items in the collection, and only return *$limit* items.
- *getParsedData()* if you need to control the number of items in a specific request. The *items* method will then return the correct filtered result based on the $start and $limit parameters. This method should return an array of objects. You will be responsible for caching.
- *getData()* if you only need to control the raw data from the service. This might be necessary if the nature of the data is not cacheable or if it does not use a url (i.e. it uses direct access PHP functions). This method should return a value (typically a string) that can be sent to the parseData() method of the DataParser object. You will be responsible for caching.
- *url()* if you only need to control the url that gets requested. This is necessary if the correct url must be determined at the point of retrieval. The default implementation combines the base url with the filters as query string parameters. This method should return a full URL as a string. By implementing this method you still benefit from caching.
- *retrieveData($url)* if the data cannot be retrieved using the PHP *file_get_contents()* function then you'll need to override this method. This would primarily be in cases where you cannot use a GET method or if specific HTTP headers must be set. You still will benefit from caching.

14.2.3 Internal methods

- *setBaseURL($url, $clearFilters=true)* - Sets the base url for the request. If $clearFilters is true (default) then the query string filters will be cleared. Set this parameter to false to maintain any filters.
- *addFilter($var, $value)* - Adds a parameter to the url query string. Currently only one value per parameter is supported.
- *removeFilter($var)* - Removes a filter from the query string
- *removeAllFilters()* - Removes all filters from the query string
- *setTotalItems($total)* - This value is typically set by the parseData() method by querying the DataParser for the total number of items.

CHAPTER

FIFTEEN

CREATING A NEW MODULE

The framework is built to make adding new functionality easy. The goal is to allow you to focus on creating the logic and design that is unique to your module rather than worry about basic functionality.

This chapter will describe the creation of a simple module and gradually add more features. This module will parse the data from a video feed using the Google YouTube Web service.

15.1 Creating the initial files

In order to ensure that your module does not conflict with current or future modules in the framework, you will want to create your files in the *SITE_DIR/app/modules/* folder.

Inside this folder is the module class file as well as a folders for templates, css and javascript. Each template file is placed in the *templates* named according to which *page* you are on, with the default page named *index*. The class file follows the format (ModuleID)WebModule.php. This file should be a sublcass of the *WebModule* class and at very least must contain a property named *id* that indicates the module id and a implementation of *initializeForPage()*

15.1.1 Steps

- Create a folder named *video* in the SITE_DIR/app/modules folder
- Create a templates folder inside the SITE_DIR/app/modules/video folders
- Create *VideoWebModule.php* with the following contents:

```
<?php

class VideoWebModule extends WebModule
{
  protected $id='video';
  protected function initializeForPage() {
  }
}
```

- Create *templates/index.tpl* with the following contents:

```
{include file="findInclude:common/templates/header.tpl"}

<h1 class="focal">Video</h1>

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

- Create a 56x56 PNG image named *title-video.png* and place it in *SITE_DIR/themes/default/common/images/compliant*. This will be the image that will show up in the nav bar for this module

You can now access this module by going to */video* on your server

15.2 Retrieving and Parsing Data

Now it's time to get some data. Most web services provide their data by making HTTP requests with certain parameters. We will use the YouTube Data API as the source of our data. It can return results in a variety of formats, but for simplicity we will choose JSON.

15.2.1 Creating a Data Library

We will utilize the *DataController* class to deal with the retrieval, parsing and caching of this data. Our first step is to create a subclass of DataController and add the appropriate methods to search for videos based on our topic. It will have a public method called *search* that will accept a string to search in YouTube and then return an array of videos based on that query. In your own implementation you could have a fixed query that returns videos from a specific channel or user. Consult the YouTube API reference for more information.

Libraries should be placed in the *SITE_DIR/lib* folder. You should create this folder if it does not exist.

15.2.2 Steps

- Create *YouTubeDataController.php* in the SITE_DIR/lib folder with the following contents:

```
<?php

class YouTubeDataController extends DataController
{
    protected $cacheFolder = "Videos"; // set the cache folder
    protected $cacheSuffix = "json";   // set the suffix for cache files
    protected $DEFAULT_PARSER_CLASS='JSONDataParser'; // the default parser

    public function search($q)
    {
        // set the base url to YouTube
        $this->setBaseUrl('http://gdata.youtube.com/feeds/mobile/videos');
        $this->addFilter('alt', 'json'); //set the output format to json
        $this->addFilter('q', $q); //set the query
        $this->addFilter('format', 6); //only return mobile videos
        $this->addFilter('v', 2); // version 2

        $data = $this->getParsedData();
        $results = $data['feed']['entry'];

        return $results;
    }

    // not used yet
    public function getItem($id){}

}
```

Some notes on this listing:

- The *cacheFolder* and *cacheSuffix* properties set the cache settings.
- The *DEFAULT_PARSER_CLASS* property sets which parser will be used (it can be overridden by setting the *PARSER_CLASS* key when using the factory method.
- The *search* method sets the base URL and adds filters. Filters work as parameters that are added to the url's query string. The *getParsedData* method is called which will retrieve that data (using the cache if necessary) and run the data through the parser (a JSON parser in this case). In the case of the YouTube feed, the entries are present in the *entry* field of the *feed* field. You can use the print_r() or vardump() functions to output the contents of the data to understand its structure
- Note that to keep this entry short, we are not utilizing any error control. This should not be considered a robust solution

Now that we have a controller, we can utilize it in our module. Here is an updated *VideoWebModule.php*

```
<?php

class VideoWebModule extends WebModule
{
  protected $id='video';
  protected function initializeForPage() {
    //instantiate controller
    $controller = DataController::factory('YouTubeDataController');

    switch ($this->page)
    {
       case 'index':
            //search for videos
            $items = $controller->search('mobile web');
            $videos = array();

            //prepare the list
            foreach ($items as $video) {
                $videos[] = array(
                    'title'=>$video['title']['$t'],
                    'img'=>$video['media$group']['media$thumbnail'][0]['url']
                );
            }

            $this->assign('videos', $videos);
            break;
    }
  }
}
```

Some notes on this listing:

- We instantiate our controller using the DataController factory method with the name of the class as the first parameter. Any options can be specified in an associative array in the second parameter.
- Using a *switch* statement allows us to have different logic depending on which page we are on. We can add logic for other pages shortly
- Then we use our search method and search for a fixed phrase. The method returns an array of entries
- We iterate through the array and assign values for each item. We're using the video title for the item title and grabbing a thumbnail to use as our image
- We then assign the videos array to the template

Finally we update the *index.tpl* file to utilize a results list to show the list of videos:

```
{include file="findInclude:common/templates/header.tpl"}

{include file="findInclude:common/templates/results.tpl" results=$videos resultsID="videoList" titleT

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

- We include the results.tpl file which expects an array of items set in the results variable. We set a titleTruncate value to cut off lengthy video titles
- We also set the resultsID variable to assist in styling

You should now be able to view the list of videos by going to */video*. There are two things we will need to add.

1. Showing the movie details
2. Styling the list to look better

We will address the first item next.

15.3 Detail Page

Most modules will have more than one page to show content. In this module we will allow the user to drill down and see more detail for a video and then play it in the browser. In order to maintain the breadcrumb navigation properly, we use the *buildBreadcrumbURL($page, $args, $addBreadcrumb)* method which is part of the WebModule object. This method takes 3 parameters, the page name we wish to link to (within the same module), and an array of arguments that get passed. The $addBreadcrumb parameter is a boolean to determine whether breadcrumbs should be generated. The default is true and this is typically what we want. Adding the url to the list is simple by adding another key to our item array in *VideoWebModule.php*:

```
<?php

//prepare the list
foreach ($items as $video) {
    $videos[] = array(
        'title'=>$video['title']['$t'],
        'img'=>$video['media$group']['media$thumbnail'][0]['url'],
        'url'=>$this->buildBreadcrumbURL('detail', array(
            'videoid'=>$video['media$group']['yt$videoid']['$t']
            ))
    );
}
```

- We simply add a *url* key to our array and use the *buildBreadcrumbURL* method to build an appropriate url. We set the page to *detail*. The *args* parameter is set to an array that has one key: *videoid* which we will pass the videoid of our video. We will use that parameter when loading the detail.

15.3.1 Retrieving an Entry

We will now need to update the *YouTubeDataController* to implement the *getItem($id)* method. This method is used to retrieve a single item from the collection based on its id. The concept of what makes an id is dependent on the context and should be documented to assist others on how to retrieve values. It can be any value as long as it is unique. Some systems have the ability to retrieve details on specific items. We will use YouTube's API to retrieve a specific item.

Update the *getItem* method in *YouTubeDataController.php*

```
<?php

// retrieves a YouTube Video based on its video id
public function getItem($id)
{
    $this->setBaseUrl("http://gdata.youtube.com/feeds/mobile/videos/$id");
    $this->addFilter('alt', 'json'); //set the output format to json
    $this->addFilter('format', 6); //only return mobile videos
    $this->addFilter('v', 2); // version 2

    $data = $this->getParsedData();
    return isset($data['entry']) ? $data['entry'] : false;
}
```

- We first set the base url to add the video id
- We add the appropriate filters to use the correct API in JSON format
- After gettings the parsed result, we return the *entry* key which contains the details of the video
- You should return FALSE if the entry could not be found
- In a more generic controller, we would return a video object that would abstract all the field details and provide an interface to these details. We will leave that exercise to you.

15.3.2 Preparing and displaying the detail view

Now that we have this method, we can use it in our module. We extract the fields we need and assign them to our template. We simply add another entry to the our *switch* branch for our *detail* page in *VideoWebModule.php*:

```
<?php
case 'detail':
   $videoid = $this->getArg('videoid');
   if ($video = $controller->getItem($videoid)) {
      $this->assign('videoid', $videoid);
      $this->assign('videoTitle', $video['title']['$t']);
      $this->assign('videoDescription', $video['media$group']['media$description']['$t']);
   } else {
      $this->redirectTo('index');
   }
   break;
```

- Use the *getArg()* method to retrieve the *videoid* parameter. It is important in any implementation to ensure that you handle cases where this value may not be present.
- You then use the *getItem* method to retrieve an entry for that id.
- We then assign a few variables to use in our template.
- If the video is not available (i.e. *getItem* returns false), we use the *redirectTo* method to redirect to the index page

Now it is time to write our *detail.tpl* template

```
{include file="findInclude:common/templates/header.tpl"}

<h1 class="focal videoTitle">{$videoTitle}</h1>
<p class="nonfocal">
    <iframe class="youtube-player" type="text/html" width="298" height="200" src="http://www.youtube
    </iframe>
```

```
</p>
<p class="focal">{$videoDescription}</p>

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

- This template uses simple variable substitution to create a few elements for the title and description. We then use an iframe to embed the YouTube player Keep in mind that some videos will not play on all devices due to difference in encoding methods.

15.4 Adding some Style

Although the module already has some formatting due to built in styles, there is some additional css styling that can be done to improve the look.

- Create a *css* folder inside the *video* module folder

Create *compliant.css* in the css folder with the following contents:

```
#videoList li {
 height: 75px;
 padding: 0 10px 0 0;
 overflow: hidden;
}

#videoList a {
  margin-left: 100px;
  padding: 5px 18px 5px 10px;
  height: 65px;
  line-height: 22px;
}

#videoList img {
 height: 75px;
 width: 100px;
 left: -100px;
 top: 0;
}

.videoTitle {
    font-size: 20px;
    line-height: auto;
}
```

- We fix the height of the results row to 75 pixels and reset the padding. A 10px padding on the right ensures that the arrow is offset appropriately from the right side.
- All of the list item content is wrapped in an anchor tag. We move the margin to the left to make room for the image and then reset the padding, and adjusted the height and line-height to accommodate longer titles
- The image is fixed to a 75x100 size and moved 100 pixels from the left.
- The video title on the detail page is shrunk to accommodate longer titles

This could be improved further, but with a few simple rules we have made the output look better.

15.5 Configuration

Now we will explore some possibilities with using configuration files to add the module to the home screen, refine the experience and make the module more flexible.

15.5.1 Home Screen

Adding the module to the home screen is simple. You can either use the *Administration Module* or by editing the *SITE_DIR/config/module/home.ini* file.

1. In the *[primary_modules]* section, add an entry that says `video="Video"`
2. Create a 72x72 PNG image named *video.png* and place it in the *SITE_DIR/themes/default/modules/home/images/compliant*

This will create a link to the video module with a label that says Video.

15.5.2 Page configuration

Each module should have a configuration file that determines the name of each page. These names are used in the title and navigation bar.

Create a file named *pages.ini* in *SITE_DIR/config/video/* with the following contents:

```
[index]
pageTitle = "Video"

[detail]
pageTitle = "Detail"
```

Each section of a page ini file is the name of the page (i.e. the url). It has a series of values (all are optional)

- *pageTitle* - Used to set the value used in the title tag (uses module name by default)
- *breadcrumbTitle* - Used to set the name of the page in the navigation bar (uses pageTitle by default)
- *breadcrumbLongTitle* - Used to set the name of the page in the footer of basic pages (uses pageTitle by default)

15.5.3 Module Configuration

The first implementation used a fixed string to search for videos. In order to include a more flexible solution, you can utilize a configuration parameter to set the string to search.

Create (or edit) a file named *module.ini* in *SITE_DIR/config/video/* with the following contents:

```
title = "Video"
disabled = 0
protected = 0
search = 0
secure = 0
SEARCH_QUERY = "mobile web"
```

The module configuration file contains some fields used by all modules, and also can contain values unique to that module. The common values include:

- *title* - The module title. Used in the title bar and other locations
- *disabled* - Whether or not the module is disabled. A disabled module cannot be used by anyone

- *protected* - Protected modules require the user to be logged in. See *Authentication*.
- *search* - Whether or not the module provides search in the federated search feature.
- *secure* - Whether or not the module requires a secure (https) connection.

You can also add your own values to use in your module. In this case we have added a *SEARCH_QUERY* parameter that will hold the query to use for the list.

We can now use it in our *VideoWebModule.php* file when we call the search method:

```
<?php

//search for videos
$items = $controller->search($this->getModuleVar('SEARCH_QUERY'));
```

The method *getModuleVar* will attempt to retrieve a value from the *config/MODULEID/module.ini* file. You can also use the *getSiteVar* method to retrive a value from *config/site.ini* which is used by all modules

CHAPTER

SIXTEEN

EXTENDING AN EXISTING MODULE

Sometimes a module exists but doesn't quite provide the behavior or look that you want. As an open source project, you can freely edit any file you want to alter the behavior, but there are supported ways to extend or alter a module while still maintaining the ability to cleanly upgrade your project when new versions come around.

There are several ways you can alter a module.

- Adjusting a page template file
- Providing alternate logic
- Replacing a module completely

16.1 Altering an existing template

Overriding a template is a very simple process. You simply provide an alternate template in your site folder and that file will be loaded instead.

For example, if you want to extend the *story.tpl* of the news module you would create *story.tpl* in *SITE_DIR/app/modules/news/templates*.

There are two approaches to updating a template.

- You can completely replace it. This will rewrite the entire template
- You can extend it. If the template provides {blocks} you can use the {extends} tag to replace only certain parts of the template

16.2 Providing alternative logic (extension)

If you want to replace some of the PHP logic you can provide a subclass of the module. This allows you to override a method or property. It is important to understand the consequences of the method you override. In some cases you will want to call the *parent::* method to ensure that the base logic is executed. An example of this would be the *initializeForPage* method. If you wanted to override the people module you would create *SitePeopleModule.php* in *SITE_DIR/app/modules/people*:

```
<?php

class SitePeopleWebModule extends PeopleWebModule
{
    protected function initializeForPage() {
        switch ($this->page)
```

```
        {
            case 'index':
                // insert new logic for index page.....
                break;
            default:
                parent::initializeForPage();
        }
    }
}
```

This would allow you to override the logic for the index page, but keep the other pages the same. You can include alternate page templates for whatever pages you need to replace.

16.3 Replacing a module completely

This process is similar to extending the module except that you extend from the *Module* class rather than the original module. This is useful if you want to have a module that has a URL that is the same as an existing module. For instance, if you want to write a completely new *about* module you will create a *AboutModule.php* file in the *SITE_DIR/app/modules/about* folder. It would look like this:

```
<?php

class AboutWebModule extends WebModule
{
    protected $id='about';
    protected function initializeForPage() {
        // insert logic
    }
}
```

It is important to include the *$id* property like you would with a *new module*.

16.4 Copying a Module

In some cases you may want to have multiple modules that exist under different URLs that share the same logic, but have different configurations. An example of this would be the *Content Module* or *URL Module*. In this case you simply subclass the parent module and provide a different *$configModule* property:

```
<?php

class SomethingWebModule extends ContentModule
{
    protected $configModule = 'something';
}
```

This module would use the same logic and templates as its parent module, but it would use its own set of configuration files, in this case in the *SITE_DIR/config/something* folder. Make sure that the class name prefix matches the configModule value.

CHAPTER

SEVENTEEN

AUTHENTICATION AND AUTHORIZATION

While many services are suitable for a public audience, there are instances where you want to restrict access to certain modules or data to authenticated users. You may also want to provide personalized content or allow users to participate in feedback.

The Kurogo framework provides a robust system for authenticating users and authorizing access to content. You can provide the ability to authenticate against private or public services and authorize access or administration based on the user's identity or membership in a particular group.

Kurogo is designed to integrate with existing identity systems such as Active Directory, MySQL databases, Twitter, Facebook and Google. You can supplement this information by creating groups managed by the framework independent of the user's original identity.

17.1 Authentication

Authentication is the process that establishes the users' identity. Typically this occurs when the user provides a username and password. The framework then tests those credentials against a central authority. If the authority validates the credentials, the user is logged in and can now consume authorized services or personalized content.

Authentication by the Kurogo framework is provided through one or more *authentication authorities*. Each authority is configured to connect to an existing authentication system. Depending on your site's needs you can utilize a private self-hosted authentication system (like an LDAP/Active Directory or database system) or a public system (Twitter, Facebook, Google). There are also hybrid approaches that utilize external services that expose standard authentication services (Google Apps). For simple deployments, you can also utilize a flat-file based system that requires no external service.

Each authority can provide various services including:

- User authentication - either through a direct login/password form or through an external system based on OpenID or OAuth
- User attributes - At minimum authorities should supply id, name and email information. Some authorities can provide this information to any user in their system, however others can only provide this information on the logged in user
- Group information and membership - Some authorities will also contain information on groups which all you to logically organize users. Some authorities are design to only contain users from their own domains, while others have the ability to utilize users from other authorities in their membership

It is not necessary for an authority to provide all services. It is possible to have one authority provide user authentication and information, and another provide group information. If you do not utilize groups in your authorization schemes, you may not need any group information at all.

17.1.1 Enabling Authentication

In order to support authenticating users you must set the *AUTHENTICATION_ENABLED* setting in *SITE_DIR/config/site.ini* to 1. This setting is disabled by default because if all your modules are public there is no need to involve the overhead of PHP session management to determine if a user is logged in or not.

17.1.2 Configuring Authorities

Authorities are defined in the *authentication.ini* file in the *SITE_DIR/config* folder. Each authority is represented by its own section. The section name is referred to as the *authority index*, programmatic value used by the framework. This value can be whatever value you wish, but you should take care to not rename this section after you have deployed your site, otherwise it may cause problems if you refer to it in any module authorization settings.

Each authority has a number of required attributes and depending on the type of authority it will have several others. See *Configuration* for more information on configuration files.

The following values are *required* for all authorities:

- *TITLE* - This value is used when referencing the framework to users. If there is more than one authority available to users to choose the title will direct them to the correct one.
- *CONTROLLER_CLASS* - This value should map to a valid subclass of *AuthenticationAuthority*. This defines the core behavior of the authority.
- *USER_LOGIN* - There are 3 possible values:
  - FORM - Use the login form
  - LINK - Use a login link. The authority should handle this using their login method
  - NONE - This authority does not provide authentication services (i.e. just group services)

Included Authorities

To allow the Kurogo framework to operate in a wide variety of settings, the project has included several classes that can connect to various types of authentication and authorization services. Each one has its own unique instructions for setup and use. Please read these documents carefully and be aware of important requirements for development and deployment.

17.2 Flat File Authentication

The *PasswdAuthentication* class provides authentication and user/group information in a locally hosted flat file structure. It represents the simplest form of authentication an group management and can be run without any external dependancies or services.

17.2.1 Configuration

The *PasswdAuthentication* authority has only 2 additional values beyond the standard:

- *USER_FILE* - a path to the user file. This file can be placed anywhere, but it is recommended to place it in the DATA_DIR folder which is mapped to *SITE_DIR/data*. i.e. If you use the DATA_DIR constant it should not be in quotes: *DATA_DIR"/users"*
- *GROUP_FILE* - a path to the group file. This file can be placed anywhere, but it is recommended to place it in the DATA_DIR folder which is mapped to *SITE_DIR/data*. i.e. If you use the DATA_DIR constant it should not be in quotes: *DATA_DIR"/groups"*

17.2.2 Format of the user file

The user file is formatted very similar to a typical unix passwd file, with a few modifications.

Each line represents a single user. Blank lines, or lines that begin with a pound (#) symbol will be ignored. For each line there are a series of fields separated by colons (:) The field order is as follow:

1. *userID* - a short name for the user
2. *password* - an md5 hash of the user's password (unix users can use md5 -s "password" to generate a hash)
3. *email* - the email address of the user
4. *full name* - the full name of the user

17.2.3 Format of the group file

The group file is formatted very similar to a typical unix groups file

Each line represents a single group. Blank lines, or lines that begin with a pound (#) symbol will be ignored. For each line there are a series of fields separated by colons (:) The field order is as follow:

1. *group* - a short name for the group
2. *gid* - a numerical id for the group
3. *members* - a comma separated list of users. Each user is represented by their username/email. If the user is from another authority you should enter it as authority|userID

17.2.4 Usage

Using this authority is useful when you want to only setup a few accounts that are not connected to an existing directory system. For instance, if you want to protect a few modules with a simple username and password.

This is also a good system to use when you want to manage groups of users from an authority that does not natively support groups. You can create groups made up of users from a variety of authorities to make it easier to manage authorization.

17.3 DatabaseAuthentication

The *DatabaseAuthentication* class provides authentication and user/group information in a relational database. It can either use the default database setup by the main configuration file or use a different system with its own credentials and settings. You can customize the tables used to lookup authentication data, and the fields to use in those tables. You can also specify the algorithm used to hash the password. The authority also presents a sensible set of default settings to allow easy setup of a new series of tables with minimal configuration.

Note that this authority does not currently support the ability to actually create and manage users or groups. It is assumed that you are connecting this to an existing system or you are managing the users and groups directly or through another utility. This is a read only system.

17.3.1 Configuration

The *DatabaseAuthentication* authority has a number of possible configuration values, all of which are optional. The following values affect the connectivity to the database system:

- DB_TYPE - The database system currently supports 2 types of connections *mysql* or *sqlite* through PDO
- DB_HOST - used by db systems that are hosted on a server
- DB_USER - used by db systems that require a user to authenticate
- DB_PASS - used by db systems that require a password
- DB_DBNAME - used by db systems that require a database
- DB_FILE - used by db systems the use a file (i.e. sqlite).

If you omit any of the above values, it will default to the settings in *SITE_DIR/config/site.ini* In addition to the connectivity settings, there are several options that tell the authority how to query the database. It is not necessary to include both user and group information if you only need one.

The following values inform the authority which database tables the data is located:

- *USER_TABLE* - (users) The name of the table that stores the user records. This table should at least have fields for userID, password and email. It can also have fields for first/last name or full name. Each row should contain a single user entry
- *GROUP_TABLE* (groups) The name of the table that stores group information. It should have fields for shortname and group id. Each row should contain a single group entry.
- *GROUPMEMBERS_TABLE* - (groupmembers) The name of the table that stores the members of each group, it should have a field for the group name/id and the userID of the user. Each row should contain an entry that contains the group name and userID. The system will search for members that match the group name.

The following values inform the authority which fields to use:

- *USER_USERID_FIELD* (userID)- stores the userID in the user table. For systems that use the email address as the key, you should include the email field
- *USER_PASSWORD_FIELD* (password) -stores a hashed value of the user's password. See USER_PASSWORD_HASH for possible hashing algorithms (Default is md5)
- *USER_EMAIL_FIELD* (email) - stores the email in the user table
- *USER_FIRSTNAME_FIELD* (empty) - stores the first name of user. Won't be used unless it is specified
- *USER_LASTNAME_FIELD* (empty) - stores the last name of user. Won't be used unless it is specified
- *USER_FULLNAME_FIELD* (empty) - stores the full name of user. Won't be used unless it is specified
- *GROUP_GROUPNAME_FIELD* (group) - stores the short name of the group in the group table
- *GROUP_GID_FIELD* - (gid) - stores the group id of the group in the group table. Should be numerical
- *GROUPMEMBER_GROUP_FIELD* (gid) - which field to use when looking up groups in the group member table. This is typically the same value as the group name or gid field
- *GROUPMEMBER_USER_FIELD* (userID) - which field to use when looking up user in the group member table. This is typically either the userID or the email

- *GROUPMEMBER_AUTHORITY_FIELD* - If present you can store the authority index in this field. This allows the system to map group members to other authorities.

Other values affect how the group membership is keyed

- *GROUP_GROUPMEMBER_PROPERTY* - (gid) - This is not stored in the database, but refers to which field will be used to look up group information in the group member table. Valid values are *gid* or *group* (i.e. the shortname) gid is the default.

There are other values that affect the method of password hashing

- ***USER_PASSWORD_HASH* (md5) - This is a string that represents a valid hashing function. It indicates what** hashing algorithm is used to store the password. See hash_algos() for a list of valid hashing algorithms. Keep in mind that available algorithms may differ by PHP version and platform.
- *USER_PASSWORD_SALT* (empty) - If present this string will be *prepended* to any string as a salt value before hashing.

17.3.2 How it Works

User Authentication

If you support user authentication (by setting the USER_LOGIN option to FORM) the authority will look in the *USER_TABLE* and look for a record with the *USER_USERID_FIELD* or *USER_EMAIL_FIELD* matching the login typed in. If that record is found it will see if the value in the *USER_PASSWORD_FIELD* matches the hashed value of the password typed in (using the *USER_PASSWORD_HASH* algorithm). The hash methods used in the database and in the configuration must match.

User Lookup

Users are looked up in the *USER_TABLE* and look for a record with the *USER_USERID_FIELD* or *USER_EMAIL_FIELD* matching the value requested. If found, a user object is populated with *USER_USERID_FIELD*, *USER_EMAIL_FIELD* and *USER_FIRSTNAME_FIELD*,*USER_LASTNAME_FIELD* and *USER_FULLNAME_FIELD* if present.

Group Lookup

Groups are looked up in the *GROUP_TABLE* and look for a record with the *GROUP_GROUPNAME_FIELD* or *GROUP_GID_FIELD* matching the value requested. If found, a group object is populated with the *GROUP_GROUPNAME_FIELD* and *GROUP_GID_FIELD* values.

Group Membership Lookup

Group membership is queried in the *GROUPMEMBERS_TABLE*. The getMembers() method will construct an array of user objects using the *GROUPMEMBER_USER_FIELD*. All users that match the *GROUPMEMBER_GROUP_FIELD* will be returned (using the value of the groups *GROUP_GROUPMEMBER_PROPERTY*, i.e. gid or short name) The user objects are created from the authority referenced by the *GROUPMEMBER_AUTHORITY_FIELD*. If there is no authority field it will use the same authority as the group (i.e. it will use the *USER_TABLE*). Because of the ability to include the authority field. You can reference users from other authorities in this table (i.e. ldap users, google users, etc)

17.3.3 Using Default Values

If you simply wish to include your own reference database, you can use all the default values with tables defined as such:

```
CREATE TABLE users (userID varchar(64), password varchar(32), email varchar(64), firstname varchar(5
CREATE TABLE groups (`group` varchar(16), gid int);
CREATE TABLE groupmembers (gid int, authority varchar(32), userID varchar(64));
```

This will give you a table structure compatible with the default values.

17.4 LDAP Authentication

The *LDAPAuthentication* class provides authentication and user/group information from an LDAP server. You specify a server, LDAP search base, and other optional parameters, and your LDAP users can authenticate to your mobile site.

You can provide both user authentication as well as group membership information, if available.

17.4.1 Configuration

The *LDAPAuthentication* authority has a number of possible configuration values:

- *HOST* - Required. The dns/ip address of the LDAP server.
- *PORT* - Optional (Default 389) The port to connect. Use 636 for SSL connections (recommended if available)
- *USER_SEARCH_BASE* - Required if providing user authentication. Set this to the LDAP base dn where your user objects are located. It can be as specific as you wish. If necessary you could specify a specific container or OU.
- *USER_UID_FIELD* - Optional (default uid) - Specifies the ldap field to use that stores the user's login id
- *USER_EMAIL_FIELD* - Optional (default mail) - Specifies the ldap field to use that stores the user's email address
- *USER_FIRSTNAME_FIELD* - Optional (default givenname) - Specifies the ldap field to use that stores the user's first name
- *USER_LASTNAME_FIELD* - Optional (default sn) - Specifies the ldap field to use that stores the user's last name
- *GROUP_SEARCH_BASE* - Required if providing group information. Set this to the LDAP base dn where your group objects are located. It can be as specific as you wish. If necessary you could specify a specific container or OU.
- *GROUP_GROUPNAME_FIELD* - Optional (default cn). Specifies the ldap field to use that stores the group short name.
- *GROUP_GID_FIELD* - Optional (default gid), Specifies the ldap field to use that stores the numerical group id.
- *GROUP_MEMBERS_FIELD* - Required if providing group information. Specifies the ldap field that indicates the members in the group. This authority assumes that this is a multi-value field in the group object.
- *ADMIN_DN* - Some servers do not permit anonymous queries. If necessary you will need to provide a full distinguished name for an account that has access to the directory. For security this account should only have read access and be limited to the search bases to which it needs to access.

- *ADMIN_PASSWORD* - The password for the *ADMIN_DN* account for systems that do not permit anonymous access. If you use an admin account with passwords, you should ensure you are connecting to your server using SSL (set *PORT* to port 636)

17.4.2 How it Works

User Authentication

If you support user authentication (by setting the USER_LOGIN option to FORM) the authority will search for a user with the *USER_UID_FIELD* or *USER_EMAIL_FIELD* matching the login typed in. If that record is found it will attempt to bind to the ldap server using the password found. If the bind is successful the user is authenticated. It is *strongly* recommended that you use SSL (by setting *PORT* to 636) to protect the security of passwords.

User Lookup

Users are looked by executing an LDAP search beginning at *USER_SEARCH_BASE* in either the *USER_UID_FIELD* or *USER_EMAIL_FIELD*. If found, a user object is populated with *USER_USERID_FIELD*, *USER_EMAIL_FIELD* and *USER_FIRSTNAME_FIELD*,*USER_LASTNAME_FIELD* and *USER_FULLNAME_FIELD* if present. Other values are placed in the *attributes* property of the user object.

Group Lookup

Groups are looked by executing an LDAP search beginning at *GROUP_SEARCH_BASE* in*GROUP_GROUPNAME_FIELD*. If found A group object is populated with *GROUP_GROUPNAME_FIELD* and *GROUP_GID_FIELD*

Group Membership Lookup

Group membership is looked up by executing an LDAP search for the *GROUP_GROUPNAME_FIELD* and retrieving the *GROUP_MEMBERS_FIELD* values. Each value should be a valid user id. It will return an array of user objects. LDAP authorities can only return user objects from the same authority.

17.5 Active Directory Authentication

The active directory authority allows you to easily configure your site to authenticate against an Active Directory domain. It is a subclass of *LDAP Authentication* and provides simpler configuration settings since AD domains share similar basic characteristics.

17.5.1 Configuration

Because attributes in active directory are standardized, there are only a few parameters necessary:

- *HOST* - Required. The DNS address of your active directory domain. You could include a specific domain controller if desired.
- *PORT* - Optional (Default 389) The port to connect. Use 636 for SSL connections (recommended if available)
- *SEARCH_BASE* - The LDAP search base of your active directory domain.

- *ADMIN_DN* - Most active directory domains do not permit anonymous queries. If necessary you will need to provide a full distinguished name for an account that has access to the directory. For security this account should only have read access and be limited to the search bases to which it needs to access.
- *ADMIN_PASSWORD* - The password for the *ADMIN_DN* account.

Once configured, users can login with their short name (samaccountname) or their email address (if present).

17.6 Facebook Authentication

The Facebook authority allows you to authenticate users by using their Facebook account. This is useful if you have modules that contain personalization and you don't want to maintain a separate account system. Because the Facebook system includes users that are not part of your organization, it is not suitable for restricting access.

Facebook uses a form of *OAuth*. Instead of authenticating directly to Facebook, the user gets redirected to the Facebook login page. Then they must authenticate and then authorize access to the application. Your application has no access to the user's login or password.

In order to successfully authenticate users using Facebook, you must first register your application.

- Go to http://www.facebook.com/developers/createapp.php
- Enter an App Name
- Once the Application is created, click the *Web Site* tab.
- Take note of the Application ID and Application Secret.
- You need to set the Site URL and Site Domain to match your domain. Facebook will only accept authentication requests from a subdomain of the domain you enter.
- Make sure to save your changes.

17.6.1 Configuration

There are a few parameters you need to configure:

- *USER_LOGIN* - Must be set to LINK, which will show a link to the facebook login page.
- *API_KEY* - Set this to the *Application ID* provided by Facebook
- *API_SECRET* - Set this to the *Application Secret* provided by Facebook

You should keep your API key and secret protected since they represent your application's identity. Do not place these values in a public source code repository.

17.6.2 How it Works

OAuth systems work by redirecting the user to an authentication page hosted by the service. The application sends a series of values including a URL callback with the request. Once the request is complete, the service redirects back to the callback URL.

The callback URL is generated by Kurogo based on the domain of server you are connecting to. Facebook requires that the callback url must be in the same domain (or subdomain) as the Site Domain value specified when registering your application. This presents a problem during development since many developers will use *localhost* to test their applications while coding. The best way to combat this is to edit your *hosts* file that allows you to create static DNS->IP mappings. Most unix systems will have this file located at */etc/hosts*. You should edit this file by adding a test subdomain entry that maps to 127.0.0.1 (which is localhost). For instance:

127.0.0.1 dev.example.com

Now, instead of testing your site using localhost, you would direct your browser to *dev.example.com* (or whatever your domain is). You can also include the port if your server is listening on a port other than 80. Thus when Facebook redirects back to *dev.example.com* your computer will use the local ip address of 127.0.0.1.

17.7 Twitter Authentication

The Twitter authority allows you to authenticate users by using their Twitter account. This is useful if you have modules that contain personalization and you don't want to maintain a separate account system. Because the Twitter system includes users that are not part of your organization, it is not suitable for restricting access.

Twitter uses a form of *OAuth*. Instead of authenticating directly to Twitter, the user gets redirected to the Twitter login page. Then they must authenticate and then authorize access to the application. Your application has no access to the user's login or password.

In order to successfully authenticate users using Twitter, you must first register your application.

- Go to http://dev.twitter.com/apps/new
- Enter an Application Name, description and URL. The Application type should be browser. The callback URL should match the domain of your site
- Click to register the application
- In the OAuth 1.0a settings section, take note of the Consumer Key and Consumer secret

17.7.1 Configuration

There are a few parameters you need to configure:

- *USER_LOGIN* - Must be set to LINK, which will show a link to the Twitter login page.
- *CONSUMER_KEY* - Set this to the *Consumer Key* provided by Twitter
- *CONSUMER_SECRET* - Set this to the *Consumer Secret* provided by Twitter

You should keep your consumer key and secret protected since they represent your application's identity. Do not place these values in a public source code repository.

17.7.2 How it Works

OAuth systems work by redirecting the user to an authentication page hosted by the service. The application sends a series of values including a URL callback with the request. Once the request is complete, the service redirects back to the callback URL.

17.8 Google Authentication

The Google authority allows you to authenticate users by using their Google account. This is useful if you have modules that contain personalization and you don't want to maintain a separate account system. Because the Google system includes users that are not part of your organization, it is not suitable for restricting access.

Google uses a form of *OpenID*. Instead of authenticating directly to Google, the user gets redirected to the Google login page. Then they must authenticate and then authorize access to the application. Your application has no access to the user's login or password.

This Authority is suitable for authenticating people with *any* google account. If you wish to limit logins to people from your Google Apps domain, then please use the *Google Apps Authentication* authority.

17.8.1 Configuration

There is very little to configure for this authority. You simply include a *USER_LOGIN = LINK* value along with the title and *GoogleAuthentication* controller class.

17.8.2 How it Works

OpenID systems work by redirecting the user to an authentication page hosted by the service. The application sends a series of values including a URL callback with the request. Once the request is complete, the service redirects back to the callback URL and the user is logged in. Google requires that the user authorize the ability for the application to view the user's email address.

17.9 Google Apps Authentication

The Google Apps authority allows you to authenticate users in your Google Apps Domain. Because it is a limited access system, it is well suited to control access to modules to people in your organization.

Google Apps uses a hybrid form of *OpenID* and *OAuth*. Instead of authenticating directly to Google, the user gets redirected to the Google Apps login page. Then they must authenticate and then authorize access to the application. Your application has no access to the user's login or password.

This authority also has the ability through OAuth to access protected data in your google apps domain. (Only for paid Google Apps for Business or Google Apps for Education accounts). This allows you to use Kurogo to display your domain's calendars, documents or other protected files in the mobile browser. This is handled without requiring you to divulge sensitive usernames or passwords. All access is handled through revokable keys. Implementations for accessing calendar and other data is forthcoming.

17.9.1 Configuration

To configure authentication, you only need to add a few parameters:

- *USER_LOGIN* - Should be set to *LINK*
- *DOMAIN* - should be set to your Google Apps domain (example.com)

To allow access to domain data (like calendars)

- *CONSUMER_KEY* - Consumer key provided by google (typically your domain example.com)
- *CONSUMER_SECRET* - Consumer secret provided by google (see below on how to obtain this value)

17.9.2 How it Works

OpenID systems work by redirecting the user to an authentication page hosted by the service. The application sends a series of values including a URL callback with the request. Once the request is complete, the service redirects back to the callback URL and the user is logged in.

The callback URL is generated by Kurogo based on the domain of server you are connecting to. Google *requires* that the callback url must be in the same domain (or subdomain) as your Google Apps domain. This presents a problem during development since many developers will use *localhost* to test their applications while coding. The best way to

combat this is to edit your *hosts* file that allows you to create static DNS->IP mappings. Most unix systems will have this file located at */etc/hosts*. You should edit this file by adding a test subdomain entry that maps to 127.0.0.1 (which is localhost). For instance:

127.0.0.1 dev.example.com

Now, instead of testing your site using localhost, you would direct your browser to *dev.example.com* (or whatever your domain is). You can also include the port if your server is listening on a port other than 80. Thus when Google redirects back to *dev.example.com* your computer will use the local ip address of 127.0.0.1.

17.9.3 Accessing Domain data (Alpha)

Google Apps for Business and Education have the ability to perform retrieval of domain data using *OAuth*. This includes retrieving calendars and other data from your organization's users. This feature, however is not automatically enabled. To enable this feature, you must enable *2-legged OAuth access control*.

- Log into your domain's administration panel (you must be an administrator for your domain to accomplish this task)
  - Go to http://google.com/a
  - Click sign in.
  - Enter your domain and choose Domain Management.
  - Log in with a domain administrator's account
- Click on the *Advanced tools* section of the domain management application
- In the authentication section choose *Manage OAuth domain key*
- In the OAuth consumer key section, ensure that *Enable this consumer key* is selected
- Take note of the consumer key and consumer secret
- In the *Two-legged OAuth access control* section, ensure that *Allow access to all APIs is selected*

This permits Kurogo to use the consumer key and secret to retrieve data for your organization. It is VERY important to keep this consumer secret in a protected location. If it has been compromised, you can click *Regenerate OAuth consumer secret*.

Once you have this data you can utilize the features of the *Calendar Module* and enable user calendars

17.10 Authorization

Once a user's identity has been established, it is possible to authorize use of protected modules and tasks based on their identity. Authorization is accomplished through *access control lists*. Developers are likely familiar with this concept in other contexts (eg. file systems).

17.10.1 Access Control Lists

An access control list is a series of rules that defines who is permitted to access the resource (ALLOW rules), and who is expressly denied to access the resource (DENY rules). Rules can be defined for users, groups or entire authorities. You can mix and match rules to tune the authorization to meet your site's needs.

Access control lists are defined in the module's configuration file *SITE_DIR/config/modules/MODULE.ini* Each entry is entered as a series of *acl[]* entries. The brackets indicates to PHP that the acl attribute is an array of values.

If a module has an access control list entry it will be protected and only users matching the acl rules will be granted access. A user will only be granted access if:

- They match an ALLOW rule, AND
- They do NOT match a DENY rule

If a user is part of a DENY rule, they will be immediately be denied access.

Syntax of Access Control Lists

Each access control list contains 3 parts, separated by a colon ":". The parts represent:

1. The action of the rule. This is either *A* (allow) or *D* (deny).
2. The rule type. Current types include *U* (user), *G* group or *A* authority
3. The rule value. If you have multiple authorities use "AUTHORITY|value". The default authority is the one defined first in the *SITE_DIR/config/authentication.ini* file
    - For users: use the userID or email address.
    - For groups: use the short name or gid for the group.
    - For authorities: use the *authority index*

To better illustrate the syntax, consider the following examples:

- *A:U:admin* - Allow the user with the userID of *admin* from the default authority
- *A:G:staff* - Allow the group with the short name of *admin* from the default authority
- *A:A:ldap* - Allow all users from the authority with the index of *ldap*
- *D:G:ldap|students* - Deny users from the group *students* from the *ldap* authority
- *A:U:google|user@gmail.com* - Allow a user with the email user@gmail.com from the *google* authority

A typical configuration file for the *admin* module might look like this:

```
title = "Admin"
disabled = 0
search = 0
secure = 0
acl[] = "A:G:ldap|admin"
acl[] = "A:G:ad|domainadmins"
acl[] = "D:U:ad|Administrator"
```

This would allow members of the group *admin* of the ldap authority and members of the *domainadmins* group in the ad authority to access this module, but specifically deny the *Administrator* user in the ad authority.

17.10.2 Using the Flat-file Authority to extend other authorities

The flat file authority *PasswdAuthentication* allows you to specify users and groups using a flat-file structure stored on the web server rather than on another system. this is useful in situations where you do not want the burden of maintain an authority system because your user base is small.

Another use of this is to use a central system for user authentication, but use the flat files for groups management. This technique is useful when you do not have direct control over the administration of the authority and therefore cannot create groups (or the authority does not inherently support groups)

You can also use the *DatabaseAuthentication* authority to store just group information if you have a database server (or use a SQLite file) under your control.

View the instructions for those authorities for more information on using them. If you do not wish to use authorities for user logins, set the *USER_LOGIN* value to NONE. This will allow the authority to be referenced for group information but will not attempt to authenticate users.