Kurogo Developer's Guide

From : http://kurogo.org/guide

Kurogo Developer’s Guide

The Kurogo Framework is a PHP based web application that can help organizations efficiently deliver 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.

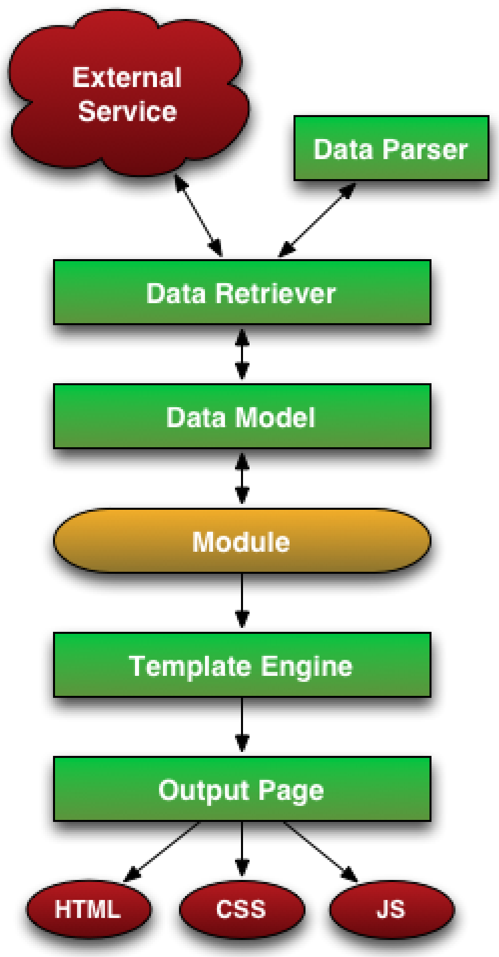
Overview

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 a range of mobile devices from feature phones, early generation smart phones, to modern devices and tablets. The mobile web component exists as a web based application served using PHP. This application is hosted on a web server and users access it using the web browser on their mobile device.

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 Parser*. Then the module would pass the data to the *Template Engine* which outputs the page using appropriate HTML, CSS and Javascript for the device.



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 *pagetype* and *platform*. The pagetype is a generic classification that outlines the device’s basic capabilities: its 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.

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.

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 *DataModel*. Each model uses an object called a *Data Retriever* to retrieve the data and then parse it using a *DataParser* object that takes the resulting data and creates an appropriate PHP structure. Through configuration you can customize which DataRetriever and DataParser are used in a request which can influence the structures that get used. You can also create your own Data Retrievers and Data Parsers to handle the unique qualities of your site's data.

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.

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.

Version History

Kurogo continues to be improved. The following are significant improvements in each version. For more detailed release notes, see the included CHANGELOG file

Version 1.5

- NEW MODULE : *Social*. Show feeds from Facebook and Twitter
- Created *ShellModule* to create modules that run on the command line (UNIX only at this time)
- Kurogo shell modules have been created that can fetch public feed data in the background. See dev guide for setting up automatic fetching
- It is possible to change which module is the home module
- New XML Parser (SimpleXMLDataParser) makes it easier to parse simple XML documents
- Improved the *Calendar* Detail page
- Improved *Map Module* flexibility
- Enabled pagination of people results
- Add *Shibboleth authentication* authority
- Tablet Panes are now loaded Asynchronously
- Improved tablet interface for Photos
- Added ability to retrieve data using the cURL library
- A custom user class can be specified in *Authentication* configurations eliminating the need to create a custom authority
- Support for Co-Ed sports in *Athletics* module
- Updated handling of stats date. See *Migration from Old Versions of Kurogo* for important details on migrating existing stats data
- Many other bug fixes and minor improvements

Version 1.4 (March 5, 2012)

- NEW Module: *Athletics*
- NEW Module: *Photos*
- Updated *Map module*
- Overhauled the *Data Retrieval* classes to better support SOAP and Database retrieval methods and complex data models

- Better client side caching headers
- Added support for in-memory caching of Kurogo data structures using Memcache or APC
- Added developer support for encrypting data on disk (requires mcrypt)
- Added option to locally save user credentials so they can be passed on to external services
- Added support for Google Analytics domain names
- Added support to show emergency notifications on the home screen
- Federated search queries will now happen asynchronously on compliant devices.
- Added support to create copied modules from the admin console.
- Added support to remove modules from the admin console

Version 1.3 (October 13, 2011)

- Support for *localization*
- *MultiSite*
- New *logging* facility
- Updated *Statistics module*
- Updated *Map module*
- Improved support for recurring events in the *calendar module*
- Added support for grouping *content* pages
- If your *news feed* does not have full content, you can add a “read more” link
- Improved method of creating *copied modules*
- Support for YouTube playlists in the *video module*
- Support for Percent Mobile *Analytics*

Version 1.2 (July 19, 2011)

- Added support for grouping *contacts* and *links*
- Added *support for IIS*
- Streamlined *theme* development
- Created protocol for *data sharing between modules*
- Support for alternate methods and custom request headers in *DataController*
- *Admin console* can be used on tablets

Version 1.1 (June 1, 2011)

- Added reordering of feeds in the *admin console*
- Added support for Vimeo in the *Video Module*
- Added bookmarks to the *people module*
- Added HTTP proxy support to *DataController*

Version 1.0 (April 8, 2011)

Initial Release

Getting Support

As an open source project, Kurogo does not include formal support. There are a number of outlets to receive informal support, as well as options for paid support.

Kurogo Google Group

In order to facilite community support, a Google Group named Kurogo-dev has been created. This group includes a mailing list where users can share questions and answers. This list is monitored by members of Modo Labs (maintainers of the Kurogo code), however responses are not guaranteed.

- http://groups.google.com/group/kurogo-dev
- kurogo-dev@googlegroups.com

Github Issues

If you would like to report a bug, please submit an issue on the project's github page:

https://github.com/modolabs/Kurogo-Mobile-Web/issues

Training and Support Through Modo Labs

Users wishing to have more formal support options should contact *sales@modolabs.com*. Support is available in the following areas:

- Initial Developer Training. Learning the framework, module development, theme design.
- Ramp up/implementation support
- Production / incident support

Professional Services

Modo Labs is also pleased to offer professional development and design services for helping users with more advanced needs including new modules, integration, or user experience and interface design. Please contact *sales@modolabs.com* for more information

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.

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`
7. When new changes come down you can run:
    - `git fetch upstream` Get updates to the upstream remote
    - `git merge upstream/master` Merge changes into your 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.

Creating your site

Because your own project will contain elements that are not part of the master project (i.e. your own site's images 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 unique needs, you should not need to edit any files outside of your site's directory.

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

Setup and Installation

Kurogo is a PHP web application. As such it must be installed on a web server. In this version, there are 2 supported web servers.

System Requirements

- Web Servers supported
  - Apache (tested on 2.x)
    - Requires mod_rewrite module
    - Requires .htaccess support (AllowOverride)
    - Subfolder support using symlinks, see *Using Kurogo in a subfolder of a domain*
  - IIS (tested on 7.5)
    - Requires URL Rewrite Module 2.0 - http://www.iis.net/download/URLRewrite
    - Tested using x86 Non Thread Safe version using FastCGI on IIS.
    - Support for virtual folders requires a manual configuration change
    - Experimental subfolder support using Junctions, see *Using Kurogo in a subfolder of a domain*
- PHP 5.2 or higher with the following extensions:
  - zlib
  - xml
  - dom
  - json
  - PDO (Used for *Database Access*)
  - mbstring
  - Zip (needed if parsing KMZ files)
- Some PHP modules are optional depending on whether you need their backend functionality
  - LDAP

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 web server.

1. Extract the files to a location accessible by your web server
2. Set the root of your web server to the *www* folder. (See also *Using Kurogo in a subfolder of a domain*)
3. (Apache Only) Ensure that .htaccess files are enabled. AllowOverride must be set to at least *FileInfo*. (MAMP on OS X has this option enabled by default)
4. (IIS Only) Ensure that the Application Pool has read access to the entire project folder. In IIS 7.5 this is the *IIS AppPoolDefaultAppPool* user
5. 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 (Apache typically: *apache* or *www*, IUSR on IIS) has write access to all the contents *SITE_FOLDER*.
6. In the root *config* directory, make a copy of the *kurogo-default.ini* file called *kurogo.ini*
7. Edit the new kurogo.ini file and change the *ACTIVE_SITE* option to match the name of *SITE_FOLDER*
8. (re)Start your webserver and direct your web browser to http://yourhost/admin to complete the setup.

Using Kurogo in a subfolder of a domain

It is possible under certain circumstances to have Kurogo appear to be installed in a URL location other than the root of a domain. There are several approaches that are supported depending on your environment.

- Using symbolic links in a Unix environment
- Using Virtual Folders in IIS (requires manual configuration change)
- Using Junctions in Windows

Note that using Apache aliases is NOT supported to do the execution order of aliases and mod_rewrite.

Using Symbolic Links in a Unix environment

In a unix environment you can place Kurogo in a subpath by using a symbolic link. Currently this is supported under the following circumstances:

- Using the Apache webserver in a unix based environment (Linux, Mac OS X, etc)
- Apache is enabled to follow symbolic links (Options FollowSymlinks)

If these conditions are true, you can create a symbolic link that points to the *www* folder and place it in your site's root folder (or subfolder).

From the command line, this command would be similar to this:

```
ln -s /path/to/kurogo/www /path/to/documentroot/mobile
```

This would assume you want the subfolder to be named "mobile". You could use any valid folder name you wish. Kurogo is designed to detect this condition automatically and will function without further configuration.

Using Virtual Folders in IIS

If you are using the IIS webserver in the Windows environment, you can install Kurogo in a virtual folder. This permits you to use Kurogo in a path that is not the document root. To use this setup you should:

- Create a virtual folder and point it to the kurogo *www* folder.
- In the Kurogo project folder open *config/kurogo.ini*. If this file does not exist, you should copy kurogo-default.ini to kurogo.ini
- In the *[kurogo]* section, uncomment the *URL_BASE* option and set it to the appropriate path. For example if your site is installed at */kurogo* then you should set*URL_BASE="/kurogo"*

Using Junctions in Windows

The following procedure should work in either IIS or Apache, however it is recommended to use Virtual Folders in IIS

- Ensure you have the Junction program installed on your server. It is distributed by Microsoft, and can be found at the time of this writing athttp://technet.microsoft.com/en-us/sysinternals/bb896768
- The junction program should be located in your PATH, in most circumstances this can be attained by copying the junction.exe file to your System Root folder (C:Windows)
- You can only create junctions between 2 paths *on the same NTFS filesystem*. You cannot create junctions between volumes or on volumes that are formatted FAT32.

Execute something similar to the following in a Command Prompt:

```
junction C:\path\to\documentroot\mobile C:\path\to\kurogo\www
```

This assumes you want the subfolder to be named “mobile”. You could use any valid folder name you wish.

Common Installation Problems

In some cases, you may encounter problems. Most are related to incomplete setup procedure or requirements.

Apache: Viewing any of the modules results in a 404 Not Found error

After viewing your site, you might be redirected to the /info or /home modules but it results in a 404. This indicates that Kurogo has handled your request, however the .htaccess file that routes all requests through Kurogo is not being read properly.

- Ensure that mod_rewrite is enabled. This requires editing your httpd.conf file and ensuring that the line enabling mod_rewrite is present and not commented. Please refer to the instructions for your distribution on the exact location of this file as it varies.
- Ensure that AllowOverride is set to FileInfo or All. AllowOverride is the option that allows apache to read .htaccess files. In certain setups this is disabled by default and must be explicitly enabled in either the httpd.conf file or the configuration file for the virtual host configuration of your site.

Kurogo Error messages are shown as an Apache error message

If you are encountering error messages and they are being shown as Apache messages then it is likely that your server is trapping /error urls

- Remove any “Alias /error” lines from your httpd.conf or virtual host configuration file

MultiSite

MultiSite is a feature of Kurogo. It allows you to host multiple Kurogo site folders from the same webserver. Each site will have its own set of configuration files, cache folders, themes and even custom modules, however they will share the same base Kurogo code. One common application of this would be to host multiple language translations of your site.

MultiSite works by exposing each site as a subpath of your domain. For instance if you had 2 sites named “en” and “es”, then those sites would be accessed in the following manner (using the home module as an example)

- http://example.com/en/home/
- http://example.com/es/home/

Setting up MultiSite

If you wish to use multi site then kurogo *MUST* be installed at the root of your domain. MultiSite will not work properly if you install Kurogo in a subpath of your site.

To setup Multisite, you must first enable it in the config/kurogo.ini file.

- Open up *config/kurogo.ini*. If it does not exists, copy kurogo-default.ini to kurogo.ini
- Set *MULTI_SITE = 1* in the *[kurogo]* section
- Set *DEFAULT_SITE* to the default site you want people to see when they visit your site. For example *DEFAULT_SITE=”en”*
- For added security and performance, you can set a series of *ACTIVE_SITES[]* values for each site that you wish to expose.

```
ACTIVE_SITES[] = "en"

ACTIVE_SITES[] = "es"
```

This would enable only the *en* and *es* sites on this server.

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.

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.

Creating the module folder structure

Note: Please make sure you have followed the *installation steps* and have created a site folder. *SITE_DIR* refers to *site/YOURSITE* where YOURSITE is the name of the site folder you have created.

The Kurogo Framework looks in a variety of locations for module data (see *Source code tour*). You should create your modules in your site's *app/modules* folder. Do**not** place new modules in the root /app/modules folder as these are reserved for included Kurogo modules.

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

Creating the module class file

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

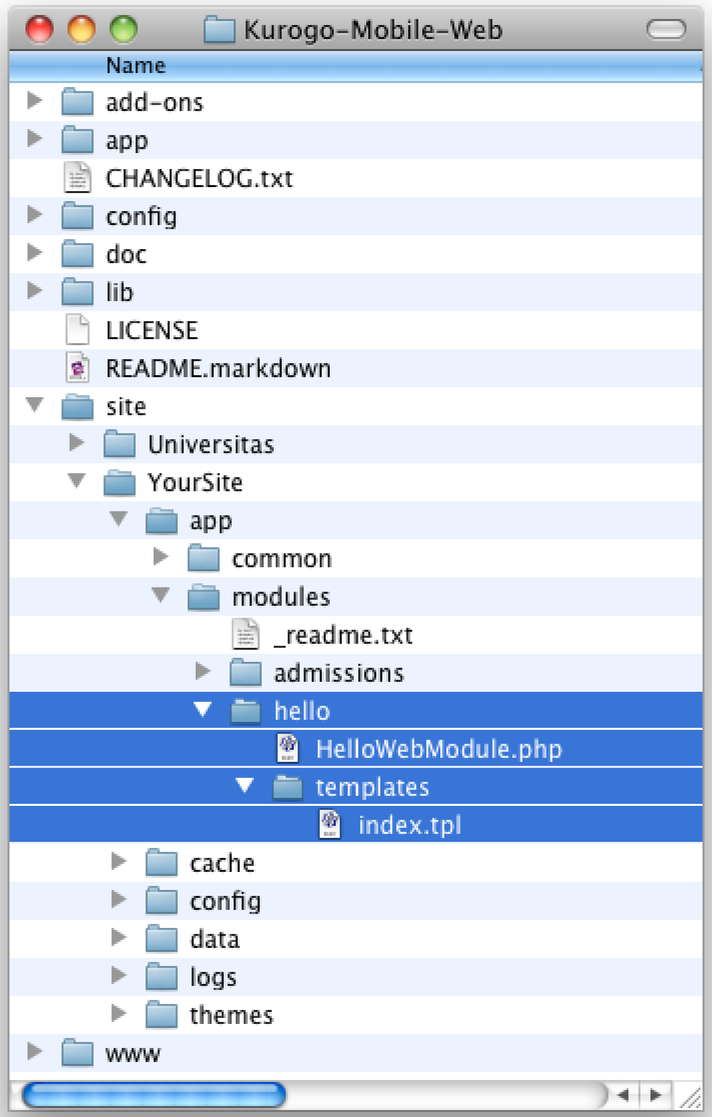
```
<?php



class HelloWebModule extends WebModule

{
  protected $id='hello';

  protected function initializeForPage() {

    $this->assign('message', 'Hello World!');

  }

}
```

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">{$message}</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
```

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*.

Creating the config folder

A configuration folder is required to load the module.

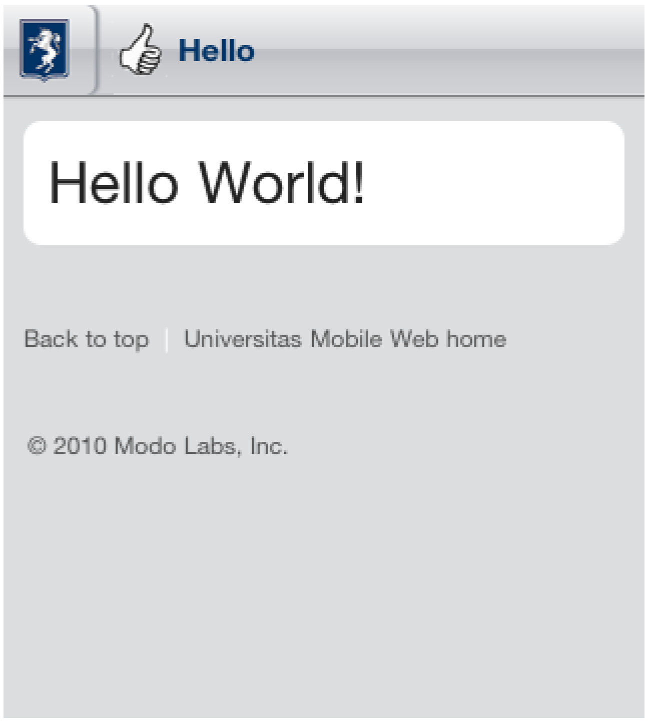
- Create a folder named *hello* in *SITE_FOLDER/config*

- Create a file named *SITE_FOLDER/config/hello/module.ini* with the following contents:

```
[module]

title="Hello"

disabled = 0

protected = 0

search = 0

secure = 0
```

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:



Hello World!

Back to top | Universitas Mobile Web home

© 2010 Modo Labs, Inc.

Congratulations! You've just built a simple module.

Source code tour

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

Basic Layout

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

add-ons

This directory contains additional scripts or code that can be used to interact with other applications

app

This directory contains the code and *templates* for each module provided by Kurogo. 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.

doc (only included in source distribution)

This directory contains various documentation files including this guide. This guide is built using the Sphinx documentation system.

lib

This directory contains libraries that are provided by Kurogo. This includes libraries for data retrieval and parsing, authentication, database access and configuration. Generally speaking, only libraries provided by Kurogo should be in this directory. You should place your libraries in the SITE_FOLDER/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. The .htaccess and web.config files provide the URL redirection support for Apache and IIS.

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.

Provided vs. Site files

As noted in the layout section, there are files provided by Kurogo (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 Kurogo. There are supported 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.

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 root Kurogo 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 important that you keep site specific modules in this folder rather than in the root *app/modules* folder. If you wish to include your work in Kurogo, please see *GitHub Repository*. Also see *Extending an existing module*.
- *cache* - Contains server generated files that are cached for performance. This folder is created as 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.
    - *acls.ini* - Site wide *access control lists*
    - *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. It is also possible that certain deployments would have nothing in the data folder.
- *lib* - an optional folder that contains code libraries used by site modules. The Kurogo *autoloader* will discover and find classes and packages in this folder.
- *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.

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.

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: *Kurogo::includePackage('PackageName')* in your module code). Currently the following packages are available:

- Authentication (included automatically when authentication is enabled)
- Authorization - for connecting to various OAuth based web services
- Cache - classes dealing with in-memory and disk caching
- Calendar - includes classes to deal with calendar data
- Config - classes to deal with configuration files
- DataController - legacy classes dealing with the pre 1.4 DataController class
- DataModel - subclasses of the *DataModel* class
- DataParser - subclasses of the *DataParser* class
- DataResponse - subclasses of the *DataResponse* class
- DataRetriever - subclasses of the *DataRetriever* class
- DateTime - classes for handling date and time
- db - used when you wish to interact with a database
- Emergency - used by the *emergency* module
- Maps - used by the *map* module
- People - used by the *people* module
- RSS - classes for handling RSS data
- Session - Subclasses of the session object, used for session management
- Video - used by the *video* module

Core / Support Files

- compat - defines several functions that normalize behavior throughout PHP versions
- exceptions - defines exception subclasses and sets up exception handling behavior
- *Kurogo* - a singleton class used to consolidate common operations like initialization, site configuration, and administration. *See more*
- minify - interface between the framework and the included open source minify library
- *DeviceClassifier* - An interface between the framework and the *Device Detection Service*

- *deviceData.db* - A SQLite database that contains browser entries used by the internal device detection system.
- *Validator* - A utility class to validate certain types of data

Native API Functions

These functions deal with the API interface that permits access to certain module functions. These interfaces are used primarily by the native applications (i.e. iOS) but is also used by certain modules for AJAX like functionality where supported.

- *APIModule* - The base class for API modules, inherits from Module
- *APIResponse* - A class that encapsulates the common response message for API requests

See *The APIModule Object* for more information.

External Data Retrieval

See *Data Retrieval* for more information

Database Access

Kurogo includes a database connection abstraction library to assist in the configuration of database connections.

- *db* - A database access library based on PDO. It includes abstractions for MySQL, SQLite, PostgreSQL and MS SQL. This support is dependent on support in your PHP installation. The setting up and maintaining of databases and their associated extensions is beyond the scope of this document.
- *SiteDB* - Uses the main database configuration for access.

See *Database Access* for more information

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*.
- *User* - The base class for identifying logged in users
- *UserGroup* - The base class for identifying groups

See *Authentication* for more information

Session Management

- *Session* - Handles the saving and restoration of user state. There are 2 current implementation:
    - *SessionFiles* - Save and restore session data using the built in file handler
    - *SessionDB* - Save and restore session data using a database

Configuration

- *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.

See *Configuration* for more information on configuring Kurogo.

Modules and Templates

- *Module* - The core class that all modules inherit from. Provides a variety of necessary services and behavior to module subclasses. See *Writing Modules*.
- *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

See *Writing Modules* for more information

Other

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

Modules and Templates

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

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

View the *Kitchen Sink* module for examples on using these templates.

Modules

The modules folder contains all the modules that are bundled with Kurogo. 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 *Writing 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). You can create modules at other urls by *copying the module*
- 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*

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 exist, such as min, the minify url). It is important to note that if you create additional files or folders in the www folder that it may interfere with proper operation of the framework.

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 for Apache or the URL Rewrite extension for IIS. 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*

Device Detection

One of the powerful 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.

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 json dataset, located at lib/deviceData.json, that contains a series of patterns and will return the values that match that pattern. In addition, you can define a custom dataset file which contains extensions and/or overrides to the cannonical dataset. This custom file is searched before the canonical dataset, allowing 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.

Data Format

The Kurogo 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*, *compliant* and*tablet*
- *platform* - The specific type of device. Values include *ANDROID*, *BBPLUS*, *BLACKBERRY*, *COMPUTER*, *FEATUREPHONE*, *IPHONE*, *IPAD*, *PALMOS*, *SPIDER*,*SYMBIAN*, *WEBOS*, *WINMO*, *WINPHONE7*
- *description* - a textual description of the device
- *supports_certificates* - Boolean. This property is deprecated and should be ignored

The *pagetype* and *platform* properties are assigned to the *module object* as properties.

Configuration

There are several configuration values that affect the behavior of the device detection service. They are located in *SITE_DIR/config/site.ini*:

- *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_SITE_FILE* - Location of site-specific device detection data if using internal detection. (typically located in *DATA_DIR/deviceData.json*)
- *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

Debugging Options

- *DEVICE_DETECTION_DEBUG* - When you turn this value on, you will see the device detection information on the bottom of the home screen. This is useful if you wish to see how a particular device is classified. If you feel a device is improperly classified, please send a note to kurogo-dev@googlegroups.com with the user agent of the device/browser.
- *DEVICE_DEBUG* - When turned on, this permits you to change the device pagetype and platform used for a given request. This is useful to test behavior and style for other devices that you do not have in your possession using your desktop browser. Simply prepend /device/pagetype-platform/ to your request:
  - http://server/device/basic/home
  - http://server/device/tablet-ipad/news

Customizing your Device Detection

Kurogo 1.3 changed how it accesses device detection files, making it much easier to modify and/or override specific device detections, without sacrificing speed.

Kurogo uses a JSON format to control it’s device detection process. This allows for both fast detection and easy modification/extension of the specified devices. There is a schema defined for this file, located at *LIB_DIR/deviceDataSchema.json*. For more information about JSON Schemas, visit http://json-schema.org/. An example custom file can be found at *DATA_DIR/sampleDeviceData.json*.

The sample deviceData.json file is reproduced below with comments added to help describe the format. Note that comments are not allowed in the actual file, in order to conform to json specifications:

```
{

    // All deviceData files consist of an object containing a
single property, "devices".  devices is an array of device
blocks.

    "devices" : [

        {

            // A device block contains several properties:



            // "groupName" : a machine readable identification of
the device or device group.

            "groupName" : "exampleDevice",



            // "description" : a human readable identification of
the device or device group.

            "description" : "Both revisions of the Example Device
produced by Example Company, Inc.",



            // "classification" : a versioned set of
identifications to be passed to kurogo once a device is matched

            "classification" : {
                // By convention, the highest version number is
placed first in the file.

                "2" : {

                    // The full list of supported pagetypes and
platforms are defined

                    // in the associated schema.

                    // Note: The schema defines a specified set
of pagetypes and

                    //     platforms, in an attempt to catch
misspelling and other user

                    //     input errors.  Kurogo, however, has no
such restrictive list, and

                    //     you can actually specify any
pagetype/platform assuming you also

                    //     have all the necessary
template/configuration files defined.

                    "pagetype"     : "compliant",

                    "platform"     : "android",

                    "supports_certificate" : true

                },



                // In the event that a classification cannot be
found for a given version

                // number, it attempts to search for the next
lowest version.
```

```
                // So if you are requesting version 3, it first
attempts to find version 3.

                // If it cannot find it, it attempts to find
version 2.  If it cannot find

                // that either, it attempts to find version 1,
etc.



                // Because of this, version numbers are a way to
specify when this

                // definition last changed.  This also means that
upgrading to a new

                // version of the detection engine does not
require you to edit every single

                // entry just to continue to use your custom
detection file.

                "1" : {

                    "pagetype"     : "webkit",

                    "platform"     : "android",

                    "supports_certificate" : true

                }

            },



            // "match" : an array of regex and/or string match
blocks which should be matched against the user agent.

            "match" : [
                // You can specify any number of matcher blocks.

                {

                    // There are 4 types of matches.

                    // "prefix" attempts to match the specified
string at the beginning of the user-agent.

                    "prefix" : "Mozilla/5.0 (exampleDevice-v1"

                },

                {

                    // "suffix" attempts to match the specified
string at the end of the user-agent.

                    "suffix" : "exampleDevice-v2)"

                },

                {

                    // "partial" attempts to perform a literal
string match somewhere in the user-agent string.

                    "partial" : "exampleDevice"

                },

                {

                    // "regex" allows you full preg_match syntax
for matching any part of the user-agent string.

                    // Note: Do not include delimiters in your
regex string or escape any potential delimiters.

                    // This is done automatically by Kurogo.

                    "regex" : "exampleDevice-v(1|2)",
                    // Each of the 4 different types of matches
can also include an optional "options" object.

                    // This object can contain various modifiers
for the match functions.  While useable on the

                    // basic string matching operations, it's
most useful in regex, where you can specify case insensitivity
and/or newline matching for the dot token.

                    "options" : {

                        "DOT_ALL" : true,

                        "CASE_INSENSITIVE" : true

                    }

                }

            ]

        }

    ]

}
```

Upgrading from Previous Versions

Kurogo is designed to make upgrades simple and easy. If you follow the prescribed procedure for site creation and extension, you can update your installation to new Kurogo versions without overwriting your customizations.

If you have forked the git repository

Git makes it very easy to implement new changes. You can simply pull down the changes from the master repository and merge it into your repository.

1. In your repository, set up an upstream remote:
   - `git remote add upstream git://github.com/modolabs/Kurogo-Mobile-Web.git`
   - `git fetch upstream`
2. When new changes come down you can run:
   - `git fetch upstream`
   - `git merge upstream/master` Merge changes into your master branch

As long as you have only edited files in your site folder, your merge should apply cleanly.

If you used a downloaded version

If you are not using git to manage your code, you can simply download the new version and simply copy your site folder into the new distribution. Make sure you retain your kurogo.ini file.

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.

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.

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}"
```

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]
```

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
```

Configuration files

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

- *config/kurogo.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.

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/site-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.

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/site-development.ini* and*SITE_DIR/site-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.

Kurogo has included a series of example -production.ini files to indicate recommended values for production servers

Retrieving Configuration Values

There are a variety of methods that are used to retrieve values from the configuration files. Please see *Module Configuration* for more information on how to retrieve these values in your module.

Site Configuration

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

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)

- *DEFAULT_LOGGING_LEVEL* - See *Logging in Kurogo* for more information.
- *LOGGING_LEVEL[area]* - See *Logging in Kurogo* for more information.
- *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 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.

Site settings

- *SITE_DISABLED* - When set to 1 this site is disabled. Useful for *MultiSite*
- *SECURE_REQUIRED* - When set to 1 then the site will require a secure (https) connection
- *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
- *LOCALE* - Locales are used for date/time formatting. If you wish to use a locale other than the server default, then you should set this. Note that valid values are dependent on the operating system of the server.
- *LANGUAGES[]* - A list of language priorities. See *Localization* for more info.
- *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.
- *DEFAULT_CHARSET* - Sets the output encoding. Typically set to UTF-8

Modules

- *HOME_MODULE* - This value sets which module is the home module. When the user taps the home link button (in the top left corner in the default theme), it will return to this module.
- *DYNAMIC_MODULE_NAV_DATA* - This value determines whether modules can present dynamic data on the navigation home screen. This could include dynamic titles, images or other information. If you are not providing dynamic data, then you should turn off this option. It is off by default. See *Dynamic Home Screen Information* for more information
- *CREATE_DEFAULT_CONFIG* - This value determines whether config folders will be automatically created if they don't exist. This can be convienient for development when you want to populate a config folder with the default values, but should be turned off for production to ensure modules that you don't use don't create configuration folders.

Cache

- *MINIFY_CACHE_TIMEOUT* - The timeout for saving the minify cache. This determines how often to look for new templates or css/js files. It should be set high for production sites.

Analytics

- *GOOGLE_ANALYTICS_ID* - set this to your google analytics id and the framework will utilize the google analytics server
- *GOOGLE_ANALYTICS_DOMAIN* - If you use subdomains in your google analytics reporting, set this to the appropriate domain
- *STATS_ENABLED* - if set to 0 then the internal statistics engine will be disabled
- *KUROGO_STATS_TABLE* (kurogo_stats_v1) - The name of the table to use for internal statistics gathering.
- *KUROGO_STATS_SHARDING_TYPE* (month) - To accomodate high volume sites, it is recommended to shard the pageview logs into separate tables. Possible values are: week, day, month.
- *KUROGO_STATS_SUMMARY_TABLE* (kurogo_stats_module_v1) - To improve performance of viewing statistics of large volume sites, a summary table is used.
- *KUROGO_STATS_LOG_FILE* - All access is logged in a file then compiled into database tables
- *KUROGO_VISIT_LIFESPAN* (1800) - The timeout (in seconds) for tracking visits

Temp Directory

- *TMP_DIR* - This should be set to a writable temporary directory. If this entry is blank, it will use the system default temporary directory.

Themes

- *ACTIVE_THEME* - This is set to the active theme. It should be a valid folder inside the *SITE_DIR/themes* directory.
- *TABLET_ENABLED* - If set to 0 then tablet devices will receive the compliant page type templates.
- *TOUCH_ENABLED* - If set to 0 then touch devices will receive the basic page type templates.

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.

Device Detection

See *Device Detection* for more information on configuration values.

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)
- *BOOKMARK_COOKIE_LIFESPAN* - How long to remember the saved bookmarks for a user. In production sites this should be set to a long time, like 15552000 (180 days)

Database

The main database connection can be used by a variety of modules for storing and retrieving values. See the *database* section for specific information on configuration values.

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).

Log Files

- *KUROGO_LOG_FILE* - The location of the Kurogo log file. This is where all Kurogo log statements will be placed depending on the value of DEFAULT_LOG_LEVEL
- *LOG_DATE_FORMAT* - Date format for log files
- *LOG_DATE_PATTERN* - regex pattern of log dates, should match output from LOG_DATE_FORMAT

Module Visibility and protection

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

- *id* - The module id to use. By default this will be the same name as the moduleID. You can change this to create a *copied module* or to use another module's code at this url.
- *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. Use this value for temporarily disabling modules.
- *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.
- *robots* - When set to 0, the module will be excluded from search engines by including a *disallow* entry in robots.txt
- *PREFETCH_DATA* - When set to 1, this module will attempt to prefetch data when the Core shell command prefetchData is run. See *Data Prefetching* for more information.

Optional Common Module Settings

- *SHOW_LOGIN* - By default the login link only appears on the home module. If you wish for it to show up on other modules, you can set this value to 1 on any module you wish to see it. You could also set this to 0 on the home module to suppress its showing.

Permanently disabling modules

When *CREATE_DEFAULT_CONFIG* in *site.ini* is set to 0, if you remove a module's config folder it will be permanently disabled and will not be accessible.

Home Screen

See *Home Module* for information on configuring the look and layout of the home screen. You can also set *HOME_MODULE* equal to a different module.

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.
- *COPYRIGHT_LINK* - Link to copyright notice (optional)
- *COPYRIGHT_NOTICE* - Copyright notice
- *FEEDBACK_EMAIL* - email address where users can send feedback.

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.

Localization

Kurogo has the ability so support different languages and localities. This includes the translation of fixed on-screen values, error messages and other text strings. Currently Kurogo only includes translations for US English, however users can create their own translations and include them in their own site or submit it to Kurogo for inclusion in future versions.

Configuration of Languages

A site can be configured to use a series of languages. When retrieving the proper value, Kurogo will search the string tables in order and return the first match. This means that is not necessary for language string tables to include a translation of every value.

The language order is set in the *[site settings]* section of *SITE_DIR/config/site.ini*. Simply create a series of LANGUAGES[] values in the order of precedence. Kurogo uses ISO country and locality codes for setting language settings. See *list of valid language codes* for language codes supported by Kurogo. For example:

```
LANGUAGES[] = "es_ES"

LANGUAGES[] = "ko_KR"

LANGUAGES[] = "en_US"
```

Would use Spanish, followed by Korean followed by English. Note that it is not necessary to include US English as it is always included at the end of not specified. Language order is specified by the server, and cannot be influenced by the user. If you wish to host a site with different languages, than you can create a duplicate site with different language settings and host them with *MultiSite*. Then you could provide a link that directs the user to the alternative site with a different language.

Note: If you use the admin console, you can only set the primary language, the admin console does not support setting multiple languages. US English is always included last to ensure that a valid value is retrieved.

Using String Tables

There are string tables for the Kurogo core as well as for each module. Each file is located in a *strings* folder and is named for the language/locality it represents. Kurogo will merge site string tables with include string tables so it is always possible to override values included with the project.

Locations include (examples given for US English String tables)

- app/common/strings/en_US.ini
- app/modules/home/strings/en_US.ini
- app/modules/news/stirngs/en_US.ini
- SITE_DIR/app/common/strings/en_US.ini (if you want to override Kurogo string values)
- SITE_DIR/app/modules/home/strings/en_US.ini (if you want to override the home module string values)
- SITE_DIR/app/modules/news/strings/en_US.ini (if you want to override the news module string values)

As with all Kurogo customizations, it is recommended that you create and edit files in your site folder rather than edit the project files to ensure smooth updates.

If you wanted to create a Spanish (es_ES) version you could create

- SITE_DIR/app/common/strings/es_ES.ini
- SITE_DIR/app/modules/home/strings/es_ES.ini
- SITE_DIR/app/modules/news/strings/es_ES.ini

Values

Each string table is a series of keys and values in .ini file format. To update the strings simply replace the value. You do NOT need to include all keys in your file. If you only have a few keys to update, you should simply include those keys in your file.

The names of the keys have been written to permit easy searching to locate where they are used

Format Specifiers

Some values include the replacement of a variable value into the string. Because languages contain different grammar, it is necessary to include a placeholder for this value so it can be properly placed within the sentence it is contained in. An example of this would be:

```
SEARCH_MODULE="Search %s" ; english

SEARCH_MODULE="%s 검색" ; korean
```

the %s would be replaced with the module name.

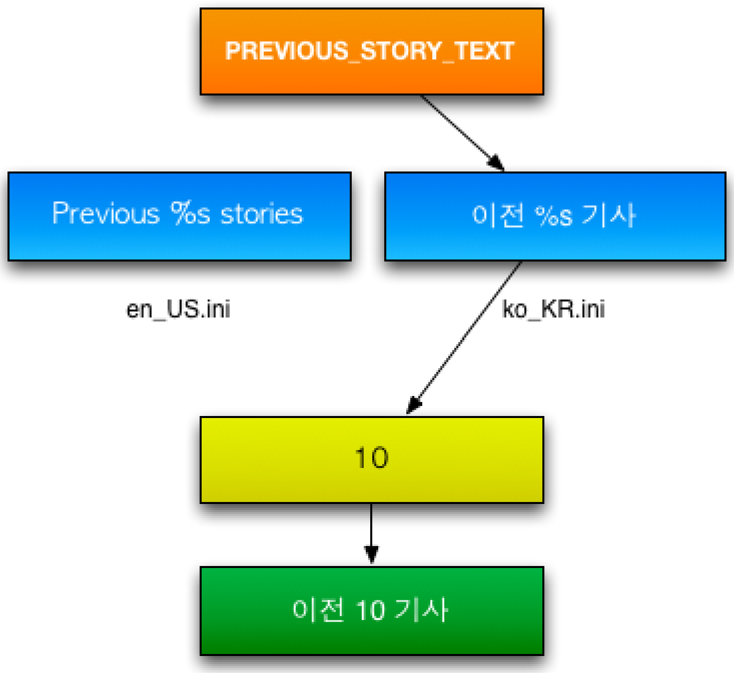
Site Values

Many of the values used by your site are actually standard configuration values and are part of your site. These include things like:

- Site name and organization
- Names of modules and page titles
- Home screen icons
- About text
- Feeds (you may wish to choose feeds that contain content in the appropriate language)

Please see the appropriate *module* or *configuration* documentation for details about setting these values.

Summary



1. String key in template or module
2. Kurogo chooses the appropriate string table
3. If there is a value, it replaces any %s
4. Display string

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. A config folder based on the first path component is looked for and a module is loaded

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
- robots.txt - Kurogo will send a dynamic version of robots.txt depending on which modules have their *robots* option set in their module.ini file.
- 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_MODULE/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 find a folder in the site config folder based on the first path component of the request (/module). The contents before the first “/” will typically refer the *id* of the module. You can override this by adding an *id* property in the module.ini file (see *Copying a Module* for more information). 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 *Writing 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 *Writing Modules* section.

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 *basic*, *touch*, *compliant* or *tablet*.
- 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.

Logging in Kurogo

Kurogo includes a facility for logging normal and exceptional events. The purpose of the logging system is to inform system administrators of important conditions that may need to be fixed, as well as providing a way for developers to debug and trace Kurogo code paths.

By default Kurogo events are logged in *SITE_DIR/logs/kurogo.log*. Each entry has several parts:

```
[date/time] area:priority method URI message
```

The area is a string that represents the Kurogo component responsible for this message. Kurogo defines several standard areas, but developers can use any string to isolate their messages into categories. The priority is one of several values that represents severity of the message. These priorities map to the priorities of the syslog function. The function/method helps you determine where this message originated. The URI is the address the user used to view the page.

Log Settings

There are several settings that affect the behavior of the Kurogo log. All are located in *SITE_DIR/config/site.ini*

- *KUROGO_LOG_FILE* (default LOG_DIR/kurogo.log) - This is the location of Kurogo log file. It must be writable by the webserver. This is where all log messages will be saved.
- *DEFAULT_LOGGING_LEVEL* - Sets the logging level. Only messages at this level or higher will be logged. All other messages will be discarded.
- *LOGGING_LEVEL[area]* - Sets the logging level for a particular area. This is useful for developers that only want to see informational or debugging messages for certain area. This level will override the DEFAULT_LOGGING_LEVEL for this area. You can override any number of areas.

Priority Levels

These are the priority level constants used by Kurogo. Most systems should set this to LOG_WARNING. If you set it to LOG_INFO or LOG_DEBUG you will get a large number of messages. Generally you would only set it lower than warning for debugging purposes, and then only in certain areas.

- LOG_EMERG - system is unusable
- LOG_ALERT - action must be taken immediately

- LOG_CRIT -critical conditions
- LOG_ERR - error conditions
- LOG_WARNING - warning conditions
- LOG_NOTICE -normal, but significant, condition
- LOG_INFO -informational message
- LOG_DEBUG - debug-level message

Standard areas in Kurogo

The following are areas used by internal Kurogo areas to separate logging messages. Developers are free to use any area they wish to log their own messages:

- admin - Admin console
- auth - Authentication and authorization
- config - Configuration
- data - Used by libraries that retrieve external data.
- db - Database
- deviceDetection - Device Detection
- exception - Exceptions
- kurogo - Core functions including initialization
- module - General module events
- session - User session management
- template - HTML templates

Logging in your module or library

The *Kurogo::log($priority, $message, $area)* method is used to send a message to the log. The $priority parameter should be on of the priority level constants, the message should be a string and the area should be a string of the area you wish to log. You could use LOGGING_LEVEL[area] to set the logging for your area as necessary.

Writing Modules

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

The Module Object

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

- Retrieval of configuration and runtime parameters
- Creation of internal URLs
- Authorization

To make a module available to users on the web (regardless of whether it shows up on the home screen), it must subclass the WebModule object. To make its data available as a web service to be read by native (e.g. iOS) apps or AJAX functionality in the web app, the module must subclass the APIModule object.

Modules for the most part provide a subclass of both WebModule and APIModule, though there are many valid reasons a module would subclass one and not the other. The Error module for example, which provides a friendly web interface to inform users of an error, does not have an API counterpart.

Each module's WebModule and APIModule subclass should generally look similar to each other. Details about WebModule and APIModule are in the following pages:

- The WebModule Object
- The APIModule Object
- The ShellModule Object

Module life cycle

Once a *request* has been made, the loading system determines which module to load and creates and instance of it using the *factory* method. The URL determines which module to load, which page to assign and any parameters that are included.

The first path component of the url is the module's *id*, this determines which module is loaded. The factory method will look for a config folder at *SITE_DIR/config/ID/*and load the *module.ini* file.

It will look for an *id* value in that file and load the module that matches that ID. If there is no ID property then it will load module with the id as the config folder.

If there is no config folder for that URL then it will look at the value of the CREATE_DEFAULT_CONFIG value in *site.ini*:

- If it is true then it will attempt to load a module based on that ID and then create the config folder automatically if the module is found.
- If it is false (the default) OR if no module can be found with that ID, then it will fail with a module not found.

The second path component, for WebModule, is the *page*. This will determine the code path and template file to load. If there is no page indicated, then the page will be set to *index*.

For APIModule, it is the *command*. The command must always be present. The single exception to the requisite *id/command* URL format is the request to the CoreAPIModule, which has a single command (*hello*) and no module ID.

After instantiating the object, the *init* method is called. This does several things:

- Assigns the necessary properties including *page*, *args*, *pagetype* and *platform*
- Calls the *initialize()* method that is used for setting up data structures that are used both inside a page and outside (for instance in the federated search)
- In WebModules, the *initializeForPage()* method is called. This method represents the primary entry point for the module's logic. Typically the module would handle different logic based on the value of the *page* property.
- In APIModules, the *initializeForCommand()* method is called. Typically the module would handle different logic based on the value of the *command* property.

Finally the output is generated as follows. WebModule chooses a template to display, based on the value of the *templatePage* property. Initially this is set to the page property, but can be overridden if necessary for more dynamic template display. APIModule creates a JSON response that includes the module id, API version, command requested, and response payload.

Methods to use

There are many methods in the Module object. Many of them are used internally and don't require any discussion. There are several methods that you should be aware of when developing new modules. Be sure to see the respective methods for *WebModule* and *APIModule* as well.

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
- *getOptionalModuleSections($config)* - Like getModuleSections(), but if the config file does not exist it will return false

You can also retrieve values from the site configuration (site.ini). These are for values used by all modules. They are static methods of 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)
- *getSession()* returns the Session object of hte current session.

The WebModule Object

The main additional functionality provided by WebModule is the logic to initialize the template system.

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:

Values the module developer should set in the class declaration:

- *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.

Values set by the parent class:

- *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*

Initialization

- *WebModule::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

Methods to Use

Pages

The following methods handle the templates and titles for pages

- *setTemplatePage($page)* - Sets the name of the page template file to use. Normally the template is derived from the url, but you can use this method to set it dynamically. This will cause $page.tpl to be loaded.
- *setPageTitle($title)* - Sets the page title for this page. Normally this value comes from the *SITE_DIR/config/MODULE/pages.ini* file, but you can use this method to set it dynamically.
- *setBreadcrumbTitle($title)* - Sets the breadcrumb title for this page. Normally this value comes from the *SITE_DIR/config/MODULE/pages.ini* file, but you can use this method to set it dynamically.
- *setBreadcrumbLongTitle($title)* - Sets the breadcrumb long title for this page. Normally this value comes from the *SITE_DIR/config/MODULE/pages.ini* file, but you can use this method to set it dynamically.
- *setPageTitles($title)* - Sets all 3 titles (pageTitle, breadcrumbTitle and breadcrumbLongTitle) to the same value

Actions

- *redirectToModule($id, $page, $args)* - This method will redirect to another module. The *id* parameter is the id of the module to redirect to. The *page* parameter is a string to the destination page. *args* is an associative array of arguments to pass to the page.
- *redirectTo($page, $args, $preserveBreadcrumbs)* - This method will redirect to another page in the module. The *page* parameter is a string to the destination page. *args* is an associative array of arguments to pass to the page. *preserveBreadcrumbs* is a boolean (default false) whether to add the entry to the list of breadcrumbs or start a new series.
- *setRefresh($time)* - Setting this will add a HTTP refresh tag to reload the page after $time seconds.
- *setCacheMaxAge($age)* - Setting this will update the cache headers to allow clients to cache the page after $age seconds. Set to 0 to disable caching. Caching is automatically disabled when authentication is enabled.

URLs

- *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.

Output

- *assign(string $var, mixed $value)* - Assigns a variable to the template. In order to use variable values in your template files, you must assign them in this manner.
- *loadPageConfigFile($name, $keyName)* - Loads a configuration file named *page-{name}.ini* located in the *config/MODULEID/* folder and assigns the values to the template.
- *setAutoPhoneNumberDetection($bool)* - Turns on/off auto phone number detection (for devices that support it). By default phone numbers are automatically detected by certain devices
- *addInlineCSS($inlineCSS)* - Adds a block of inline CSS to the page. This should be used sparingly as CSS files can be cached by the browser. This would be necessary if the css would need to be dynamic
- *addInternalCSS($path)* - Adds a css file that is located on the server. This would typically be used to load css files dynamically. The URL might be in the format "/modules/moduleID/css/cssfile.css". URLs should ALWAYS be referred using a leading slash, even if the site is located in a subfolder. The template engine handles creating the full path
- *addExternalCSS($url)* - Adds a reference to a CSS file located externally use a full http:// url
- *addInlineJavascript($inlineJavascript)* - Similar to addInlineCSS except for javascript
- *addInlineJavascriptFooter($inlineJavascript)* - Similar to addInlineJavascript except that it will load the javascript at the bottom of the page.
- *addInternalJavascript($path)* - Similar to addInternalCSS except for javascript
- *addExternalJavascript($url)* - Similar to addExternalCSS except for javascript

Methods to override

- *initializeForPage* - This method is called when viewing a page. It represents the main logic branch. All modules will have this code.
- *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. It is not needed in all cases.
- *searchItems($searchTerms, $limit=null, $options=null)* - This method is called by other modules (including the default federated search implementation) to retrieve a list of items that meet the included search terms. A limit value will be passed that will include a maximum number of items to return (or null if there is no limit). There is also an optional associative array that is sent that contain options specific to that module. The federated search implementation will add a "federatedSearch"=>true value to allow this method to behave specifically for this situation. This method should return an array of objects the conform to the KurogoObject interface.
- *linkForItem($object, $options=null)* - This method should return an array suitable for showing in a list item. This would include items such as *title* and *url*. The options array may be used to include other information
- *linkForValue($value, Module $callingModule, KurogoObject $otherValue=null)* - This method is used to format a value in another module. It is mostly used by subclasses of the standard module to perform site specific formatting or linking. The call includes the calling module and an optional object that may contain other values. This allows your implementation to consider all values of the object when building the link. This function should return an array that is suitable for a list item, including *title* and *url* values. The default implementation uses the value as the title and uses a url like *moduleID/search?filter=value*.

The Help Page

There is a page called *help* that has special meaning in Kurogo. For each module, you can define a string in the *strings* section of the *module.ini* file named *help* that will allow you to provide a help text for end users. If this value is present then a help link will show up on the page and this will link to the help page containing this text.

```
[module]

title = "Module Name"
```

```
disabled = 0

protected = 0

search = 1

secure = 0



[strings]

help[] = "This module provides services related to lorem ipsum"

help[] = "Additional help entries indicate additional paragraphs"

help[] = "You can have as many paragraphs as you need"
```

Dynamic Home Screen Information

The *Home Module* is used to show the available modules to the users. In the default implementation, the list of modules and their titles and images is specified statically in the {homeModuleID}/module.ini file. In this case the information presented on the home screen is always the same.

In some scenarios it may be necessary to have that information be more dynamic. This would permit custom titles or subtitles, images, and even display based on any conditions that are appropriate. In order to utilize this you must do the following:

- change *DYNAMIC_MODULE_NAV_DATA* to *1*. This option is normally turned off due to increased overhead
- create a subclass of the module you wish to provide dynamic data. I.e. If you wish to have dynamic data for the People module, then create a*SitePeopleWebModule.php* file in *SITE_DIR/app/modules/people* . This step is only necessary if you’re providing this behavior to included modules.

- Implement the *getModuleNavigationData($moduleNavData)* method. This method will include an associative array of information for each module suitable for the*springboard* or *list item* templates. It will include keys such as:
    - *title* - The title of the module.
    - *subtitle* - The subtitle of the module. Currently only used in the list view display mode
    - *url* - The url to the module. Defaults to /moduleid. Should only be changed in unusual circumstances
    - *selected* - Whether this module is selected. This is used by the tablet interface since the nav bar is persistent.
    - *img* - A URL to the module icon. The default is /modules/{homeModuleID}/images/{moduleID}.{$this->imageExt}.
    - *class* - The CSS class (space delimited)

  Your implementation should alter the values as necessary and return the updated associative array. If you wish the module to be hidden, return FALSE rather than the array.

The following is an example of a module that shows a different title based on the time of day, and will be invisible during the early morning and nighttime hours.

```php
<?php


class MyWebModule

{

    protected function getModuleNavigationData($moduleNavData) {

        //get the current hour

        $hour = date('H');



        if ($hour >= 9 && $hour < 12) {
            //it's between 9 am and noon

            $moduleNavData['title'] = 'Good Morning';

        } elseif ($hour >=12 && $hour < 18) {

            //it's between noon and 6pm

           $moduleNavData['title'] = 'Good Afternoon';

        } elseif ($hour < 21) {

            //it's between 6pm and 9pm

            $moduleNavData['title'] = 'Good Evening';

        } else {

            //it's in the evening or early morning. make the
module invisible

            return false;

        }



        //you must return the updated array

        return $moduleNavData;

    }

}
```

It is very important that any logic you handle in this method complete quickly as this method is run very frequently and would be run on EVERY page in the tablet interface. It may be useful to cache information if it is based on external data.

The APIModule Object

APIModule provides the logic to initialize and render the JSON contents.

Properties

Values the module developer should set in the class declaration:

- *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.
- *responseVersion* - The version of the API output that will be returned. This **may** be set to a different value depending on the *command*.
- *vmin* - The minimum API version supported by the current implementation.
- *vmax* - The maximum API version supported by the current implementation.
- *response* - The payload to return to the client for their request. This will will be converted into JSON by the superclass.

Values set by the parent class:

- *command* (string) - This property is set when the module initializes and represents the command *requested*.
- *requestedVersion* - The API version requested by the client.
- *requestedVmin* - The minimum API version requested by the client, if the client is able to handle lower API versions.

Methods to Use

Output

- *setResponse($response)* - Sets the *response* property of this object.
- *setResponseVersion($version)* - Sets the *responseVersion* property of the object.
- *setResponseError(KurogoError $error)* - Sets the response’s error field if there an error or exceptional condition occurs anytime during the data handling and output process (for example, if an LDAP search runs over the maximum number of results that can be returned) that the client should know about.

Errors

- *throwError(KurogoError $error)* - If an error occurs such that no useful data can be returned to the client, calling this method will halt the usual process of setting the *response* property and immediately output the error message to the client.
- *invalidCommand()* - Call this method if the client requests a command that cannot be handled. This will result in a *throwError()* call with a standard message.

Configuration

- *getAPIConfigData($name)* - Returns an array from a config file named *page-{name}.ini* located in the *config/MODULEID/* folder.

Methods to override

- *initializeForCommand($command)* - This method represents the module's main logic when constructing the response. It must be overridden by each module.

The CoreAPIModule

The CoreAPIModule is a special subclass of APIModule. It contains general information about the site.

The ShellModule Object

ShellModule provides the logic to execute commands in a Unix shell. It was created primarily to aid in creating tasks that should be performed in the background and used with a tool such as cron.

Executing a Shell

Currently shell module execution is only supported on Unix systems. To execute a shell command you would call:

```
/path/to/kurogo/lib/KurogoShell moduleID command
```

For instance:

```
/path/to/kurogo/lib/KurogoShell core version
```

KurogoShell is a bash script that hands control over to PHP. This requires the PHP binary to be in your path.

Properties

Values the module developer should set in the class declaration:

- *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.

Values set by the parent class:

- *command* (string) - This property is set when the module initializes and represents the command requested.

Methods to Use

The ShellModule is primarily used to perform actions (typically administrative actions).

- *preFetchAllData* - This command will call *getAllControllers* and then attempt to fetch all the data for module. This is used to prime the Kurogo Cache so that users are not waiting for data to be fetched. Not all modules or retrievers will be able to use prefetching due to the nature of their API calls.

Output

- *out* - output data to the console

Methods to override

- *initializeForCommand($command)* - This method represents the module's main logic when executing the shell. It must be overridden by each module.
- *getAllControllers* - In some modules you will need to override this method when implementing prefetch. This would be necessary if you are using a mechanism other than *feeds.ini*

CoreShellModule

The CoreShellModule has several commands:

- *version* - Returns the Kurogo version
- *clearCaches* - Clears the Kurogo cache folder. It is recommended to run this periodically to ensure that caches don't build up
- *fetchAllData* - Cycle through all modules and call fetchAllData for modules that have PREFETCH_DATA = 1 set in their module.ini file.

Data Prefetching

Certain modules can prefetch their data. When run in the background (like a cron job), you can ensure that the data cache is primed. This is especially useful for servers that are slow.

You can run the core shell command *fetchAllData* and it will fetch data for all modules that have PREFETCH_DATA = 1 in their module.ini. Built in modules that support this include:

- News
- Video
- Photos
- Calendar
- Athletics

- Emergency
- Social

It is recommended to run /path/to/kurogo/lib/KurogoShell core prefetchData in a cron job. It should be run at an interval less than the cache time of the modules. 10 minutes is a good default time.

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.

- Home Module
- Info Module
- People Module
- Video Module
- Calendar Module
- News Module
- Emergency Module
- Map Module
- Photos Module
- Athletics Module
- Social Module
- Links Module
- Content Module
- URL Module
- Customize Module
- About Module
- Login Module
- Statistics Module
- Kitchen Sink

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.

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”).

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 titles. 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.

Configuration

The following configuration values are available to customize the behaviour of the home module.

- *SHOW_LOGIN* - (optional) If set to true and *Authentication* is enabled, a link allowing users to login will be displayed. Defaults to true.
- *HIDE_IMAGES* - (optional) Set to true to disable icons when homepage *display_type* is “list”. Defaults to false.

- *SHOW_FEDERATED_SEARCH* - (optional) Enables/disables showing of the federated search bar on the homepage. Defaults to true.

Banner Alert System

Introduced in version 1.4, you can display small notifications on the home screen from other modules. This allows users to know about information without visiting the module. An example of this (and the only included module to support this behavior) is the Emergency Module. Developers can enable their modules to support this behavior by implementing only a few methods.

Configuration

By default the notification is disabled. The following options in the *[notice]* section of the home module's *module.ini* file can be updated to enable notifications.

- *BANNER_ALERT* - Set to 1 to enable notifications
- *BANNER_ALERT_MODULE* - Set this to the module that will provide notifications. The module must conform to the *HomeAlertInterface* object interface
- *BANNER_ALERT_MODULE_LINK* - If set to 1, then the user can tap this notification and it will direct them to the module.

Creating a Notification Module

A module you set using *BANNER_ALERT_MODULE* must conform to the *HomeAlertInterface* interface. In version 1.4 of Kurogo, only the Emergency module supports this behavior. This interface requires the creation of 1 method in your module:

- public function getHomeScreenAlert() - This method should return a dictionary (array) with the following keys: * *title* - A title for the notification * *text* - The text of the notification * *date* - A formatted date string of the notification. Usually the post date of the Notification * *unixtime* - A unix timestamp of the post date of the notification.

If you module implements this interface (and implements the methods) you can use it to provide notifications to your users.

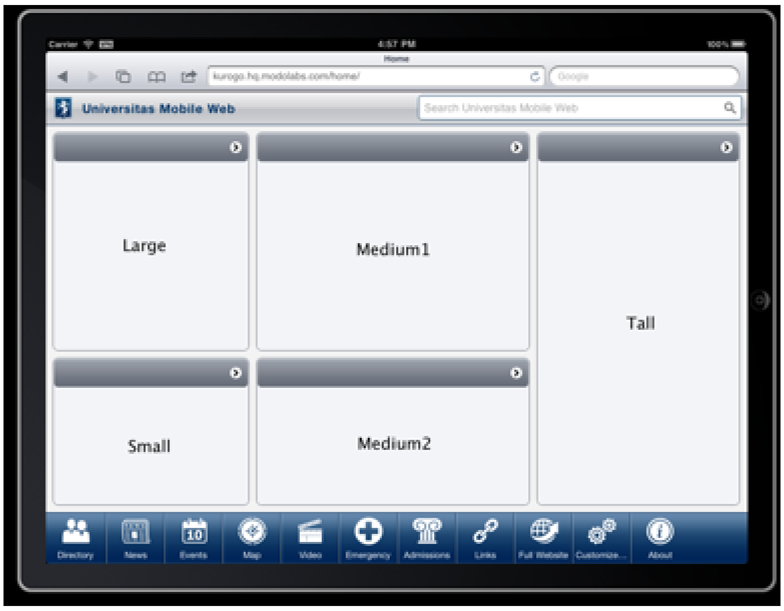
Icons

For compliant browsers, you will need to create icons for each module. These icons should be placed in:*SITE_DIR/themes/[ACTIVE_THEME]/modules/home/images/PAGETYPE*. Each module should have an 72x72 PNG file (for compliant and tablet) or 44x44 GIF file (for touch) named the same as its module id (about.png, news.gif, etc.)

Tablet Interface

Kurogo includes a special layout for tablet devices. The interface includes 2 significant changes from the standard layout. First there is a navigation strip that appears at the bottom of the screen on all pages. This allows easy navigation to any module. in order to support this you will have to include an additional home screen icon for each module with a *-selected* suffix (links-selected.png, calendar-selected.png, etc).

Also, the home screen itself supports showing a reduced set of content on the home screen from a series of modules in *panes*. Currently the layout of the home screen is fixed with 5 different panes. You can choose which module will show up in which pane.



You simply set which modules will appear in which pane by editing the *[tablet_panes]* section of *SITE_DIR/config/home/module.ini*. You would enter the moduleID for the item you want to show in a particular pane:

```
[tablet_panes]

large = "news"

small = "about"

medium1 = "video"

medium2 = "emergency"
```

```
tall = "calendar"
```

If you do not want to show the tablet interface you can change *TABLET_ENABLED* to 0 in *SITE_DIR/config/site.ini*. When the tablet interface has been disabled, tablet devices will receive the *compliant* page type.

Dynamic Home Screen Information

In some scenarios it may be necessary to have the information show on the home screen (or tablet nav bar) to be more dynamic. This would permit custom titles or subtitles, images, and even display based on any conditions that are appropriate. In order to utilize this please read the section on *Dynamic Home Screen Information*.

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 remove the *DEFAULT-COMPLIANT-COMPUTER* value in the *[urls]* section of your site's *site.ini* file and ensure the *DEFAULT* value is set to home.

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 either LDAP based directories (including Active Directory), and *database* backed directories.

Configuring the Server Connection

The configuration for this module is accomplished by using the *Administration Module* or by editing the *SITE_DIR/config/people/feeds.ini* file. There are a variety of values to set in order to connect to your directory system.

- *RETRIEVER_CLASS* allows you to determine the mechanism for retrieving the data. Current options include:
    - LDAPPeopleRetriever - uses a standard LDAP server. You can configure the various fields if your values differ from defaults
    - ADPeopleRetriever - a subclass of the LDAP retriever that has preconfigured mappings for Active Directory
    - DatabasePeopleRetriever - connects to an external database server. This controller assumes that people are mapped to a single row and that the various fields are stored in single (definable) columns
- *PERSON_CLASS* allows you to set a different class name for the returned user objects when searching. This class must be a subclass of the *Person* class. This allows you to write custom behavior to handle the data in your directory service.

Options for LDAPPeopleRetriever and ADPeopleRetriever

- *HOST* - should match the address of your 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.
- *SEARCH_BASE* - 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”
- *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.

The following values inform the controller which attributes to use when searching. These values would only need to be altered if the values differ from the defaults in parentheses.

- *LDAP_USERID_FIELD* (uid, samaccountname for AD)- Stores the unique user name for this user. Do not choose an attribute that is sensitive as they are easily viewed by users
- *LDAP_EMAIL_FIELD* (mail) - The attribute of the user’s email address
- *LDAP_FIRSTNAME_FIELD* (givenname) - The attribute of the user’s first name
- *LDAP_LASTNAME_FIELD* (sn) - The attribute of the user’s last name
- *LDAP_FULLNAME_FIELD* (blank) - If you wish to use a single field for name (like cn), include this here.
- *LDAP_PHONE_FIELD* (telephonenumber) - The attribute of the user’s phone number

Options for DatabasePeopleRetriever

The *DatabasePeopleRetriever* has a number of possible configuration values, all of which are optional. See *Database Access* for a full detail on configuring database connections

If you omit any of the values, it will default to the settings in *Configuring Database Connections* In addition to the connectivity settings, there are several options that tell the controller how to query the database.

The following value inform the controller which table the data is located:

- *DB_USER_TABLE* - (users) The name of the table that stores the user records. This table should at least have fields for userID, name and email. Each row should contain a single user entry.

The following values inform the controller which fields to use for critical fields. These values would only need to be altered if the values differ from the defaults in parentheses.

- *DB_USERID_FIELD* (userID)- stores the userID in the user table. You can use any unique column for the userID field. Do not use sensitive values as they are easily viewed by users.
- *DB_EMAIL_FIELD* (email) - stores the email in the user table
- *DB_FIRSTNAME_FIELD* (firstname) - stores the first name of user.
- *DB_LASTNAME_FIELD* (firstname) - stores the last name of user.

- *DB_PHONE_FIELD* (no default) - stores the user's phone number. If empty then the search will not use the phone number

The other fields are shown by configuring the detail fields below.

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. The default configuration is setup for an LDAP directory. If you use a nonstandard directory, or you utilize the database connector with its own fields, then you will need to customize how this displays.

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" or "phone". Used to format and generate links.
- *module* - (optional) Creates a link to a another module and uses that module's linkForValue method to format the result. See the section on *Module Interaction* for more details.
- *section* - (optional) If this field belongs to a section, the name of that section
- *parse* - (optional) A function which will be run on the value before display. Generated with *create_function*. Gets the argument "$value" and returns the formatted output.

Configuring Display Options

The options listed here relate to the appearence of the people module.

- *BOOKMARKS_ENABLED* - (optional) If set to true, a link to bookmarked entries will appear. Note that if you have not bookmarked any entries, this link will not appear until an entry is bookmarked. Defaults to true.
- *CONTACTS_SUBTITLE_NEWLINE* - (optional) Set to true to display contacts subtitles on a new line. Defaults to false.

- *SEARCH_TIP* - (optional) A string to be shown near the search bar providing a tip about searching. By default this string is empty, and not shown.

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/contacts.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*

Creating groups of contacts

- NOTE - Creation of contact groups is not supported in the admin console at this time.

If you have a number of fixed contacts and need to categorize them you can place them into groups. Creating contact groups involves the following steps:

1. If it does not exist, create a file named *SITE_DIR/config/people/contacts-groups.ini*
2. Add a section to contacts-groups.ini with a short name of your group. This should be a lowercase alpha numeric value without spaces or special characters
3. This section should contain a "title" option that represents the title of the group. Optionally you can include a *description* value that will show at the top of the contacts list for the group
4. Create a file named *SITE_DIR/config/people/contacts-groupname.ini* where *groupname* is the short name of the group you created in *contacts-groups.ini*. This file should be formatted like contacts.ini with each entry being a numerically indexed section
5. To use this group, assign it to a entry in *contacts.ini*. Do not include a url, but rather add a value *group* with a value of the short name of the group. You can optionally add a title that will be used instead of the group title indicated in *contacts-groups.ini*

This is an example *SITE_DIR/config/people/contacts-groups.ini*. Each group is a section that contains title (and optional description). You can have any number of groups:

```
[admissions]

title = "Admissions"
```

SITE_DIR/config/people/contacts-admissions.ini. This is an example file for the *admissions* group. It is formatted like the *contacts.ini* file:

```
[0]

title    = "Admissions Main Number"

subtitle = "(617-555-0001)"

url      = "tel:6175550001"

class    = "phone"



[1]

title    = "Admissions Hotline"

subtitle = "(617-555-0002)"

url      = "tel:6175550002"

class    = "phone"
```

SITE_DIR/config/people/contacts.ini. Include a *group* value to show a group, do not include a *url* value:

```
[0]

title    = "Static Entry 1"

subtitle = "(617-555-0001)"
```

```
url      = "tel:6175550001"

class    = "phone"



[1]

title    = "Admissions"

group    = "admissions"
```

Video Module

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

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:

- *RETRIEVER_CLASS* - The *Data Retriever* to use. Currently supported retrievers include:
  - *YouTubeRetriever*
  - *VimeoRetriever*
  - *BrightcoveRetriever*
- *TITLE* - The textual label used when showing the section list
- *AUTHOR* - optional, used to limit the results by author
- *TAG* - optional, used to limit the results by tag

YouTubeRetriever

There are additional options for YouTube feeds:

- *PLAYLIST* - optional, used to limit the results by a particular playlist

VimeoRetriever

There are additional options for Vimeo feeds:

- *CHANNEL* - optional, used to limit the results by a particular channel

BrightcoveRetriever

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

- token
- playerKey
- playerId

Configuring Display Options

- *MAX_PANE_RESULTS* - (optional) Defines the maximum number of videos to display when viewing the site’s home page on a tablet device. Defaults to 5.
- *MAX_RESULTS* - (optional) Defines the maximum number of results when viewing the index page of the video module. Defaults to 10.
- *BOOKMARKS_ENABLED* - (optional) If set to true, a link to bookmarked entries will appear. Note that if you have not bookmarked any entries, this link will not appear until an entry is bookmarked. Defaults to true.
- *SHARING_ENABLED* - (optional) If set to true a link to share the current video will be display on the detail page. Defaults to true.

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.

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

- *RETRIEVER_CLASS* - allows you to set a different class name for the *Data Retriever*. If your service uses dynamic urls then you should write a custom retriever.
- *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.

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* - Optional value to format the value and/or create a link. Possible values are:

    - datetime - Formats the value as a date/time
    - email - creates a mailto link using the value as the email address
    - phone - creates a telephone link using the value as the phone number
    - url - creates a link, The value is used as the url
- *class* - CSS class added to the field. multiple classes can be added using spaces
- *module* - Creates a link to a another module and uses that module's linkForValue method to format the result. See the section on *Module Interaction* for more details.

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*, *email*

Configuring User Calendars and Resources

There is support for viewing user calendars and resources (such as rooms/equipment). Currently the only supported calendar system is Google Apps for Business or Education. Support for Microsoft Exchange calendars is available through Modo Labs contact *sales@modolabs.com* for information.

To enable User Calendars:

- Setup the *authority* for your Google Apps Domain.
- Ensure that you have entered the required OAuth consumer key and secret

- Ensure that the “http://www.google.com/calendar/feeds” scope is available in your authority.
- Edit *config/calendar/module.ini* and add a *user_calendars* section.
- Set RETRIEVER_CLASS to GoogleAppsCalendarListRetriever
- Set AUTHORITY to the section name of your Google Apps Authority. If this value is not set it will use the first defined GoogleAppsAuthority class

This is an example section from the config/calendar/module.ini file:

```
[user_calendars]

RETRIEVER_CLASS="GoogleAppsCalendarListRetriever"

AUTHORITY="googleapps"
```

To enable Resources:

- Setup the *authority* for your Google Apps Domain.
- Ensure that you have entered the required OAuth consumer key and secret
- Ensure that the “https://apps-apis.google.com/a/feeds/calendar/resource/” scope is available in your authority.
- Edit *config/calendar/module.ini* and add a *resources* section.
- Set RETRIEVER_CLASS to GoogleAppsCalendarListRetriever
- Set AUTHORITY to the section name of your Google Apps Authority

This is an example section from the config/calendar/module.ini file:

```
[resources]

RETRIEVER_CLASS="GoogleAppsCalendarListRetriever"

AUTHORITY="googleapps"
```

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, it can be configured to fetch the content from the URL specified in the article (see FETCH_CONTENT below)

General Options

There are a few options in *SITE_DIR/config/news/module.ini* that can configure basic operations of the news module

- *MAX_RESULTS* (10) - The number of items to show in the news list
- *SHARING_ENABLED* (1) - Whether or not to enable the sharing link on news entries. Set to 0 to disable the sharing link

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

- *RETRIEVER_CLASS* - allows you to set a define an alternate *data retriever* to get the data. The default is URLDataRetriever.
- *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.
- *SHOW_IMAGES* - Show the image thumbnails for this feed. If an image is not available, it will show a placeholder image. If the entire feed does not have images, you may wish to set SHOW_IMAGES=0

- *SHOW_PUBDATE* - Show the publish date in the news list (the published date is always showed in the detail page)
- *SHOW_AUTHOR* - Show the author in the news list (the author is always showed in the detail page)
- *SHOW_LINK* - Show a link to the full article. This is useful for feeds that only contain an introductory paragraph.

Additional Configuration for RSS Feeds

There are certain options that are specific for RSS feeds. Since the default parser is the RSSDataParser, this would apply to most feeds.

- *CHANNEL_CLASS* allows you to set a different class name for the RSS channel. You could create a custom subclass to handle RSS items uniquely. This is rarely necessary
- *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 or parse fields in a custom matter.
- *ENCLOSURE_CLASS* allows you to set a different class name for enclosures. This would allow you to handle custom behavior for enclosures.
- *IMAGE_ENCLOSURE_CLASS* allows you to set a different class name for image enclosures. This would allow you to handle custom behavior for images.
- *REMOVE_DUPLICATES* - This will remove duplicate entries from the feed (i.e. items that have the same GUID). This can be turned on for feeds that suffer from duplication.
- *HTML_ESCAPED_CDATA* - If set to 1 it will decode HTML entities in the content and description fields. This will need to be turned on if feed generators mistakenly encode HTML data in addition to wrapping it in CDATA blocks (this is typically not necessary)
- *USE_DESCRIPTION_FOR_CONTENT* - If this option is set to 1, then the description field will be used as the full content. The description field is typically only used for a short introduction.
- *FETCH_CONTENT* - If this option is set to 1, then Kurogo will attempt to fetch the content found at the URL specified in the item and extract the content.

Emergency Module

The emergency module provides a mobile interface to a site's 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.

Configure Emergency Notice

If you want to display an emergency notice you will need to include a *[notice]* section in *config/emergency/feeds.ini*.

- 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.

Other Options

- *NOTICE_EXPIRATION* - The amount of time (in seconds) to consider a notice to be active. This is useful for feeds where the notices are not removed from the feed. The default is 1 week (604800 seconds)
- *NOTICE_MAX_COUNT* - The maximum number of notices to show from the feed.

Configure Contacts List

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 to connect to the Drupal emergency module:

- *RETRIEVER_CLASS* = "DrupalContactsDataRetriever"
- *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* = "INIFileContactsListRetriever"
- *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"
```

In the case that you have added a *secondary* contacts section you may change the title of the secondary contacts link by changing the module variable*MORE_CONTACTS*. By default this variable is set to “More Emergency Contacts”.

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*

Map Module

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

Basic Configuration

Feed Groups

Each feed group corresponds to a base map, characterized by center location and background appearance. For organizations with multiple campuses, feed groups are a logical way to split up maps by campus.

Groups are configured in the *SITE_DIR/config/map/feedgroups.ini* file. There must be at least one entry that looks like the following:

```
[groupID]

title                = "Title of my map"

subtitle             = "Subtitle of my map"

center               = "42.3584308,-71.0597732"

DEFAULT_ZOOM_LEVEL   = 10
```

- *[groupID]* is a short, alphanumeric identifier for the group, e.g. “boston”, “newyork”, “daytime”, etc.
- *title* (required) - a short description of this group, e.g. “Boston Campus”
- *subtitle* (optional) - more explanatory text if title is not sufficient.
- *center* (required) - the latitude, longitude point that represents the center of this group. *No spaces* are allowed before and after the comma.
- *DEFAULT_ZOOM_LEVEL* (recommended) - initial zoom level to display the map at. These are numbers generally between 0 and 21. Very roughly, each zoom level covers the following extents:
    - 0: earth
    - 1: hemisphere
    - 2: multiple continents
    - 3: one continent
    - 4: large country

- 5: medium-sized country
- 6: small country or large state
- 7: small state
- 8: tiny state
- 9: metropolitan area + vicinity
- 10: metropolitan area
- 11: large city
- 12: small city
- 13: town
- 14: several neighborhoods
- 15: a few neighborhood
- 16: neighborhood
- 17: several blocks
- 18: a few blocks
- 19: several buildings
- 20: a few buildings
- 21: a building

The base map used by default is Google Maps on compliant/tablet devices, and Google Static Maps on basic/touch devices. Kurogo also supports ArcGIS JavaScript maps on compliant/tablet devices, and ArcGIS static maps and WMS on basic devices. See the Advanced Configuration section for details.

Data Feeds

Each data feed is represented as a *category* that a user may browse by from the home screen or within a campus.

For people who do not have any existing mapping infrastructure, we generally recommend creating feeds using Google Earth. Google Earth allows you to add pins to the map and export the result as a KML file.

To add feeds to a feed group, create an entry for it in the file *SITE_DIR/config/map/feeds-GROUP.ini* (where GROUP is the id of the group from feedgroups.ini). Each entry should look like the following:

```
[index]

TITLE                = "Fun Places in Washington"
```

```
SUBTITLE          = "These are places I like to check out on
vacation"

BASE_URL          = DATA_DIR"/washington.kml"

SEARCHABLE        = 1
```

- *index* is a number (starting from 0) indicating the sort order for this feed.
- *TITLE* is a descriptive name of the category that shows up in the list of categories
- *SUBTITLE* (optional) is brief description that appears in small text alongside the title
- *BASE_URL* (required) is the URL or file location of the data source.
- *SEARCHABLE* - just set this to 1.

Note: if you use a feed in .kmz instead of .kml format, you *must* also specify the following in the entry:

```
RETRIEVER_CLASS        = "KMZDataRetriever"
```

Kurogo also supports data from the ArcGIS REST API and the ESRI Shapefile format. If you use any of these formats, or are looking to change other aspects of the data feeds and base maps, see the Advanced Configuration section.

User Interface Options

The following options appear by default in *SITE_DIR/config/map/module.ini*:

```
BOOKMARKS_ENABLED          = 1

MAP_SHOWS_USER_LOCATION    = 1

SHOW_DISTANCES             = 1

DISTANCE_MEASUREMENT_UNITS = "Imperial"

SHOW_LISTVIEW_BY_DEFAULT   = 0
```

- *BOOKMARKS_ENABLED* - whether or not the user can bookmark locations.

- *MAP_SHOWS_USER_LOCATION* - whether or not the user can display the location marker on the map. This does not turn off geolocation in general.
- *SHOW_DISTANCES* - whether or not the list of nearby places displays distance from the searched location.
- *DISTANCE_MEASUREMENT_UNITS* - either “Metric” or “Imperial”.
- *SHOW_LISTVIEW_BY_DEFAULT* - if there is only one campus, the index page of the map module will show a full screen map view on compliant devices. If this value is set to 1, the index page of the map module shows a list view. If there are multiple campuses, the index page is a list view regardless of this parameter. multip

Advanced Configuration

Feed Groups

In addition to *title*, *subtitle*, and *center*, each group may also specify the following:

- *JS_MAP_CLASS* (optional) - the type of base map to use for devices that support JavaScript maps, see *Base Maps*.
- *DYNAMIC_MAP_BASE_URL* (required if *JS_MAP_CLASS* is ArcGISJSMap) - the base URL where the base map JavaScript API is hosted.
- *STATIC_MAP_CLASS* (optional) - the type of base map to use for devices that do not support JavaScript maps, see *Base Maps*.
- *STATIC_MAP_BASE_URL* (required if *STATIC_MAP_CLASS* is ArcGISStaticMap or WMSStaticMap) - the base URL where the static base map service is hosted.
- *NEARBY_THRESHOLD* (optional, defaults to 1000) - distance threshold in meters to use when performing searches for nearby locations
- *NEARBY_ITEMS* (optional, defaults to 0) - maximum number of items to return from a nearby search. If the value is 0, there is no limit.

Example configuration:

```
[honolulu]

title                    = "Honolulu Campus"

subtitle                 = "Our new satellite office that nobody
knows about"

center                   = "21.3069444,-157.8583333"
```

```
JS_MAP_CLASS          = "ArcGISJSMap"

DYNAMIC_MAP_BASE_URL = "http://myhost/MapServer"

STATIC_MAP_CLASS      = ArcGISStaticMap

STATIC_MAP_BASE_URL  = "http://myhost/MapServer"

NEARBY_THRESHOLD      = 1609

NEARBY_ITEMS          = 12
```

Data Feeds

In addition to *TITLE*, *SUBTITLE*, and *BASE_URL*, each feed may also specify the following:

- *MODEL_CLASS* - data model class associated with the type of data source. The default is MapDataModel.
- *RETRIEVER_CLASS* - data retriever class to use for the feed, if not the default. The default depends on the MODEL_CLASS. If you are not using a custom model class, this should only be necessary for KMZ files (which need KMZDataRetriever).
- *SEARCHABLE* - boolean value that indicates whether or not this data source should be included in internal search results. This value is irrelevant if you use an external search engine. The default is false.
- *HIDDEN* (optional) - if true, this feed will not show up in the list of browsable categories. This may be used if a site wants to have a different set of placemarks show up in search results from the ones users can browse.

Some config values set for individual feeds can override the values in the associated feed group. For example, the “honolulu” feed group may use a nearby threshold of 1000 meters when searching, but we have a dense feed in where we only want items within 200 meters. In this case set NEARBY_THRESHOLD can be set on the individual feed. The overridable config parameters are DEFAULT_ZOOM_LEVEL, JS_MAP_CLASS, DYNAMIC_MAP_BASE_URL, STATIC_MAP_CLASS, STATIC_MAP_BASE_URL, NEARBY_THRESHOLD, and NEARBY_ITEMS.

KML/KMZ

KML is the default feed type in the map module. In other words, if the feed config does not specify MODEL_CLASS or RETRIEVER_CLASS, Kurogo will assume the feed is in KML format.

Kurogo only supports a subset of KML tags. Kurogo will ignore all unsupported tags except <MultiGeometry>, <Model>, <gx:Track>, <gx:Multitrack> – these tags will cause Kurogo to throw exceptions. Also, several tags are parsed but never shown in the UI.

The following tags are parsed and affect the UI:

```
<Folder>

    <name>

    <description>

<StyleMap>

    <Pair>

        <key>

        <styleURL>

<Style>

    <iconStyle>

        <href>

        <w>

        <h>

    <balloonStyle>

        <bgColor>

        <textColor>

    <lineStyle>

        <color>

        <weight>
    <polyStyle>

        <fill>

        <color>

<Placemark>

    <address>

    <name>

    <description>

    <Snippet>

    <Point>

        <coordinates>

    <Polygon>

        <outerBoundaryIs>

        <innerBoundaryIs>

    <LineString>

        <coordinates>

    <LinearRing>
```

The following tags are parsed but currently have no effect on the UI:

```
<Document>

    <name>

    <description>
    <scale> (under iconStyle)
```

See Google’s KML documentation for more information.

ArcGIS Server

To use ArcGIS Server, specify the following in feeds-<group>.ini:

```
MODEL_CLASS = "ArcGISDataModel"
```

If the service has multiple layers, Kurogo only uses one layer at a time. You may specify different layers for different feeds by specifying

ARCGIS_LAYER_ID = <number>

where <number> is the numeric ID of the layer. Sublayers are not currently supported.

See Esri’s ArcGIS Server documentation for more information.

Shapefile

To use shapefiles, specify the following in feeds-<group>.ini:

```
MODEL_CLASS = "ShapefileDataModel"
```

Shapefiles located across the network must be in a zip folder containing no directories (i.e. the contents are all .shp, .dbf, .shx, and .prj files). Note that to use zipped shapefiles, the ZipArchive extension must be enabled in PHP.

Larger shapefiles may be unzipped and stored locally in a subdirectory of DATA_DIR. In this case, the BASE_URL must be specified without the extension, e.g. the shapefile consisting of DATA_DIR”/myshapefile.shp” and DATA_DIR”/myshapefile.dbf” must be specified as:

```
BASE_URL = DATA_DIR"/myshapefile"
```

See Wikipedia’s entry on the Shapefile specification for more information.

Configuring Display Options

- *SHOW_DISTANCES* - (optional, defaults to true) Whether or not to show the current distance to an item.
- *DISTANCE_MEASUREMENT_UNITS* - (optional, defaults to ‘Metric’) Changes the distance units, valid values are *Imperial* or *Metric*.
- *MAP_SHOWS_USER_LOCATION* - (optional, defaults to false) Whether or not to show the user’s location on the map.
- *BOOKMARKS_ENABLED* - (optional, defaults to true) If set to true, a link to bookmarked entries will appear. Note that if you have not bookmarked any entries, this link will not appear until an entry is bookmarked.

Base Maps

Kurogo selects the base map following the configuration and these default rules:

If both JS_MAP_CLASS and STATIC_MAP_CLASS are left unspecified, Kurogo by default will select Google Static Maps for basic/touch devices and Google Maps for compliant/tablet devices. If both are specified, JS_MAP_CLASS will be used for compliant/tablet and STATIC_MAP_CLASS for touch/basic.

If **only** STATIC_MAP_CLASS is specified, both compliant/tablet and basic/touch devices will use the base map specified by STATIC_MAP_CLASS. If **only**JS_MAP_CLASS is specified, Google Static Maps will be chosen for basic/touch devices.

JavaScript base maps (compliant and tablet only)

Acceptable options for JS_MAP_CLASS are as follows.

Google Maps

To explictly use Google Maps (rather than rely on it showing up by default), enter the configuration:

```
JS_MAP_CLASS = "GoogleJSMap"
```

See Google’s Maps documentation for more information.

ArcGIS Tiled Service Maps

To use tiles from an ArcGIS tile server, enter the configuration:

```
JS_MAP_CLASS = "ArcGISJSMap"
```

```
DYNAMIC_MAP_BASE_URL = "http://..."
```

Additional dynamic layers from an ArcGIS Dynamic Service Map may be added on top of the base map by specifying DYNAMIC_MAP_BASE_URL as an array, e.g.

```
DYNAMIC_MAP_BASE_URL[] = "http://my.tiled.service/MapServer"

DYNAMIC_MAP_BASE_URL[] = "http://my.dynamic.service/MapServer"
```

The first element of DYNAMIC_MAP_BASE_URL must be a tiled service. There must be one and only one tiled service.

See Esri's ArcGIS JavaScript documentation for more information.

Static image base maps

Acceptable options for STATIC_MAP_CLASS are as follows.

Google Static Maps

To explicitly use Google Static Maps (rather than rely on it being the default), enter the configuration:

```
STATIC_MAP_CLASS = "GoogleStaticMap"
```

Google Static Maps does not currently have support for polygon overlays.

See Google's Static Maps documentation for more information

Web Map Service (WMS)

To use images from a WMS service, enter the configuration:

```
STATIC_MAP_CLASS = "WMSStaticMap"

STATIC_MAP_BASE_URL = "http://..."
```

Note that it is not possible to add overlays to WMS maps.

See the Open Geospatial Consortium's WMS documentation for more information.

ArcGIS exported images

To use exported images from an ArcGIS server, enter the configuration:

```
STATIC_MAP_CLASS = "ArcGISStaticMap"

STATIC_MAP_BASE_URL = "http://..."
```

Note that it is not possible to add overlays to an exported image.

See Esri's export API documentation for more information.

Map Search

Map search is configured in module.ini. If this is not configured, Kurogo's default behavior is to use the class MapSearch, which walks through all feeds in the selected feed group.

Optionally, the following parameters may be configured:

```
MAP_SEARCH_CLASS          = "MyMapSearchSubclass"

MAP_EXTERNAL_SEARCH_CLASS = "GoogleMapSearch"
```

Searches initiated within the map module use the MAP_SEARCH_CLASS, which defaults to "MapSearch". Searches initiated by modules other than the map module *may*use a different search class if the optional config parameter MAP_EXTERNAL_SEARCH_CLASS is configured to a different class.

The included class GoogleMapSearch uses uses either Google Places or the Google Geocoding service. Geocoding is selected by default. To use Places (assuming you have an API key from Google), add the following configurations to *SITE_DIR/config/maps.ini*:

```
[maps]

USE_GOOGLE_PLACES     = 1

GOOGLE_PLACES_API_KEY = AbCDeFGH123789zzzzzzzzzzzzzzzzzzxwycbA
```

Terms of Use for External Providers

Users of Google Maps and related products (which includes the majority of Kurogo installations) need to be aware that usage restrictions apply on all these products.

The Google Maps/Earth API terms of service is here.

Sites with heavy traffic should be aware of recent changes to usage limits on embedded Google Maps.

Photos Module

The photos module enables sites to provide mobile access to photos hosted by 3rd party websites such as flickr and Picassa

General Options

There are a few options in *SITE_DIR/config/photos/module.ini* that can configure basic operations of the photos module:

[module]

- *MAX_RESULTS* - The number of photos to show per page.
- *BOOKMARKS_ENABLED* - Set to 1 to enable or 2 to disable bookmarking photos
- *SHARING_ENABLED* - Set to 1 to enable or 2 to disable sharing photos

[strings]

- *description* - This text will show at the top of the index page

Configuring the Sources

The module allows you to organize your photos by album using a distinct feed for each album. Each section contains information on the service provider and can retrieve photos from a particular user's account. Feeds are configured in the *SITE_DIR/config/photos/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:

- *RETRIEVER_CLASS* - The *Data Retriever* to use. Currently supported retrievers include:
  - *FlickrFeedRetriever* - Retrieves photos using the flickr Feed service
  - *FlickrAPIRetriever* - Retrieves photos using the flickr API service. This requires an API key from flickr
  - *PicasaRetriever* - Retrieves photos from the Picassa service.
- *TITLE* - The textual label used when showing the album list

FlickrFeedRetriever

The flickr Feed retriever can retrieve public user photosets and group photos. This feed, however, is limited to viewing 20 photos (This is a flickr limit). If you would like access to all your photos, then you will need to use the FlickrAPIRetriever with an API key. Use the following options to configure the FlickrFeedRetriever:

- *USER* - The flickr User ID. Use a tool like http://idgettr.com/ to find your user id
- *PHOTOSET* - The photoset id of the album to show. This id can typically be found in the URL address bar when viewing the photoset.
- *GROUP* - The group ID of the group photos to retrieve. This id can typically be found in the URL address bar when viewing the group.

FlickrAPIRetriever

The flickr API retriever can retrieve public user photosets and group photos. It has the same options as the FlickrFeedRetriever, but requires an API key from flickr. Unlike the FlickrFeedRetriever, it does not limit the number of photos returned. In addition to the options found in the FlickrFeedRetriever, it has the follow option:

- *API_KEY* - The API key you received from flickr.
  See http://www.flickr.com/services/api/misc.api_keys.html for information on getting a key.

PicasaRetriever

The Picasa retriever views photos from a public Picassa album. It has the following required options:

- *USER* - The user account associated with the album. This is typically the email address of the owner (user@gmail.com)
- *ALBUM* - The album ID of the album.

Athletics Module

The athletics module provides and interface to view news and event information for athletic events. It utilizes a mixed feed system where data comes from multiple independent sources. You can configure a list of sports for either gender and each sport has its own news and schedule/results feed.

General Options

There are a few options in *SITE_DIR/config/photos/module.ini* that can configure basic operations of the athletics module:

- *BOOKMARKS_ENABLED* - (optional) If set to true, a link to bookmarked entries will appear. Note that if you have not bookmarked any entries, this link will not appear until an entry is bookmarked. Defaults to true.
- *SHARING_ENABLED* - Set to true to enable or false to disable sharing photos

Sport Configuration

The first step to configuring the athletics module is to specify the sports for each gender. The list of sports will come from the *config/athletics/feeds.ini* file. Each sport will have a section in this file. If a sport is competed by both genders, then it will have multiple sections with unique section keys ([m-basketball], [w-basketball]). The*feeds.ini* file will also contain the configuration for that sport's news feed (if present). A corresponding entry in *schedule.ini* will contain the information regarding the schedule and results for that sport. Currently there is support for ICS or CSTV feeds for schedule and result information.

Sport and News Configuration

For the sport and news configuration, each sport will have a section in *config/athletics/feeds.ini*. The section name must be unique and will also be used by the schedule information. The feeds config includes the following options:

- *TITLE* - The title of the sport as its shown in the list
- *GENDER* - The gender of the sport. Should be *men* or *women*.
- *BASE_URL* - Optional - The URL of the news feed. By default it will assume RSS. If this is in a different format you can use the same news feed options found in the *News Module*

For the schedule information, you will need to create a section in *config/athletics/schedule.ini* with the same name as you created in *feeds.ini*. This section will contain the schedule and results information. Currently it supports ICS calendar feeds or CSTV

XML feeds. It uses standard DataRetriever configuration options. If you just include a *BASE_URL* then it will attempt to parse the data using the ICS Data Parser.

CSTV Configuration

If your institution has a CSTV site (CBS Sports) then there is an easy way to retrieve event data.

- *RETRIEVER_CLASS* = Set this to *CSTVDataRetriever*
- *BASE_URL* - Should be set to the event XML feed. It is typically in a format such as: http://goteam.com/data/ABBR-event-info-YEAR.xml where ABBR is an abbreviation of your school and YEAR is a 4 digit academic year of the events to retrieve. This will typically be the same for all sports.
- *SPORT* - Set this to the value of the <sport> key in the XML file. This will filter the results for this sprot

ICS Configuration

If you are using an ICS feed for sporting events and results, then it is recommended that you create a distinct feed/calendar for each sport. Typically you would only need to configure the *BASE_URL* option to indicate the source of the event data.

Custom Data Types

It is simple to create a custom parser for your own event information. The Athletics module expects event information to be returned as an array of AthleticEvent objects.

Social Module

The social module shows a list of posts from a variety of social network service accounts. Currently this includes public feeds from Facebook and Twitter.

Configuring the Sources

The module allows you to show the recent posts from any number of social network accounts. Accounts are configured in the *SITE_DIR/config/social/feeds.ini* file. Each account is contained in a section. The name of each section is generally not important, but must be unique. Each

Within each section you use the following options:

- *RETRIEVER_CLASS* - The *Data Retriever* to use. Currently supported retrievers include:
    - *TwitterDataRetriever* - Retrieves tweets from a user's public timeline
    - *FacebookDataRetriever* - Retrieves posts from a user or page's public wall.
- *TITLE* - A textual label for this account
- *ACCOUNT* - The user name / page name for the account to show

Other options are specific to those services.

TwitterDataRetriever

The Twitter retriever retrieves tweets from a user's public timeline. It is not required, but it is recommended that you sign up for a Twitter developer account and create a Twitter application. Once you have done so you should add the necessary oauth information:

- *OAUTH_CONSUMER_KEY* - The OAuth consumer key found in your application's configuration page
- *OAUTH_CONSUMER_SECRET* - The OAuth consumer secret found in your application's configuration page

These values must be present in the account's section in *feeds.ini*, however you can simply put a value such as "anon". By not including your OAuth key, you will be limited to a certain number of requests per hour, but due to Kurogo caching the data, you will likely not reach this limit. It is not necessary to have different keys for each account if you are including multiple Twitter accounts. They can share the same values.

FacebookDataRetriever

The Facebook retriever retrieves posts on a user, group or page's *public* wall.
The *ACCOUNT* portion of the feed configuration should include either the user's name, group name or page name. For pages that have not applied for a user name, you should include the

number found in the URL (i.e. facebook.com/###########). It is*required* that you register your application with Facebook. The Facebook API requires an app key and secret. Go to https://developers.facebook.com/apps to add a new application.

- *OAUTH_CONSUMER_KEY* - The App ID of your application
- *OAUTH_CONSUMER_SECRET* - The App secret of your application

It is not necessary to have different app ids for each account if you are including multiple Facebook accounts. They can share the same values.

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.

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 *description_footer* - This string will show at the bottom of the page when viewing the list

Links

Links are editing in the *links.ini* configuration file. Each link is represented by a configuration section. Within each section, there are 3 possible keys:

- *title* - The title of the link
- *url* - The url of the link
- *icon* - a 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.

```
[0]

title = "This is link 1"

url   = "http://example.com/urlforlink1"

icon  = "link1_icon.png"



[1]

title = "This is link 2"

url   = "http://example.com/urlforlink2"

icon  = "" ; link 2 does not have an icon
```

```
[2]

title = "This is link 3"

url   = "http://example.com/urlforlink3"

icon  = "link3_icon.png"
```

Creating groups of links

- NOTE - Creation of link groups is not supported in the admin console at this time.

You can create a group of links in order to organize large amounts of links into categories. Creating link groups involves the following steps:

1. If it does not exist, create a file named *SITE_DIR/config/links/links-groups.ini*
2. Add a section to links-groups.ini with a short name of your group. This should be a lowercase alpha numeric value without spaces or special characters
3. This section should contain a “title” option that represents the title of the group. Optionally you can include a *description* value that will show at the top of the links list for the group. You can also include a display_type to include a different link display type than the main link list.
4. Create a file named *SITE_DIR/config/people/links-groupname.ini* where *groupname* is the short name of the group you created in *links-groups.ini*. This file should be formatted like links.ini with each entry being a numerically indexed section
5. To use this group, assign it to a entry in *links.ini*. Do not include a url, but rather add a value *group* with a value of the short name of the group. You can optionally add a title that will be used instead of the group title indicated in *links-groups.ini*

This is an example *SITE_DIR/config/people/links-groups.ini*. Each group is a section that contains title (and optional description). You can have any number of groups:

```
[admissions]

title = "Admissions"
```

SITE_DIR/config/people/contacts-admissions.ini. This is an example file for the *admissions* group. It is formatted like the *contacts.ini* file:

```
[0]

title    = "Admissions Main Number"

subtitle = "(617-555-0001)"

url      = "tel:6175550001"

class    = "phone"



[1]

title    = "Admissions Hotline"

subtitle = "(617-555-0002)"

url      = "tel:6175550002"

class    = "phone"
```

SITE_DIR/config/people/contacts.ini. Include a *group* value to show a group, do not include a *url* value:

```
[0]

title    = "Static Entry 1"

subtitle = "(617-555-0001)"

url      = "tel:6175550001"

class    = "phone"
```

```
[1]

title    = "Admissions"

group    = "admissions"
```

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. Optional grouping may be configured.

Configuring Content Feeds

You can specify any number of pages to show in the *SITE_DIR/config/MODULE/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.
    - *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

Creating Groups of Content

- NOTE - Creation of content groups is not supported in the admin console at this time.

You can create groups of content in order to organize similar content into categories. Creating content groups involves the following steps:

1. If it does not exist, create a file named *SITE_DIR/config/MODULE/feedgroups.ini*
2. Add a section to feedgroups.ini with a short name of your group. This should be a lowercase alpha numeric value without spaces or special characters.
3. This section should contain a *TITLE* option that represents the title of the group. Optionally you can include a *DESCRIPTION* value that will show at the top of the content list for the group.
4. Create a file named *SITE_DIR/config/MODULE/feeds-groupname.ini* where *groupname* is the short name of the group you created

in *feedgroups.ini*. This file should be formatted like feeds.ini with each entry being a uniquely indexed section.

5. To use this group, assign it to a entry in *feeds.ini*. Add a value *GROUP* with a value of the short name of the group. You can optionally add a *TITLE* that will be used instead of the group title indicated in *feedgroups.ini*

The *feeds.ini* file may contain both groups and content entries. They will be displayed in the order the sections appear in the *feeds.ini* file. If only one group is added, that group will be displayed. If only one content entry exists in either a group or in *feeds.ini* it will be displayed.

This is an example *SITE_DIR/config/MODULE/feedgroups.ini*. Each group is a section that contains title (and optional description). You can have any number of groups:

```
[applying]

TITLE = "Applying to Universitas"

DESCRIPTION = ""



[visiting]

TITLE = "Visiting"

DESCRIPTION = ""
```

SITE_DIR/config/MODULE/feeds-applying.ini. This is an example file for the *applying* group. It is formatted like the *feeds.ini* file:

```
[admissions]

TITLE = "Admissions"

SUBTITLE = ""

SHOW_TITLE = 0

CONTENT_TYPE = "html_url"
```

```
BASE_URL = "http://universitas.modolabs.com/admissions"

HTML_ID = "node-2"
```

SITE_DIR/config/MODULE/feeds.ini. Include a *group* value to show a group:

```
[applying]

TITLE = "Applying to Universitas"

GROUP = "applying"



[visiting]

TITLE = "Visiting"

GROUP = "visiting"



[othercontent]

TITLE = "Other Content"

SUBTITLE = ""

SHOW_TITLE = 0

CONTENT_TYPE = "html_url"

BASE_URL = "http://www.example.com/othercontent"

HTML_ID = "html-id"
```

Options for HTML Content

There are a few options to handle the extraction of data from an HTML document. In most cases you only want to include a fragment of the document and strip away things like HTML and HEAD tags and remove headers and footers. There are two ways to indicate which content to include:

- *HTML_ID* - Use this option to include only a single element (and its child elements) based on its HTML id attribute. This is the simplest, and most recommended option if it is available. The value for this option is case sensitive.
- *HTML_TAG* - Use this to include all elements of a certain tag. For instance set it to “table” to include all table elements or “p” to include all paragraph elements.
  Do **not** include the surrounding brackets (<, >)

If you do not include either of these options then the entire contents of the body tag will be extracted.

URL Module

The url module simply redirects the user to an external url.

Configuration

It has one configuration parameter in its module.ini file

url - A url to redirect to.

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.

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.

About Module

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

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.

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.

In order to use the login module you must first configure *Authentication*.

Configuration

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

- *LOGIN_INDEX_MESSAGE* - A message shown at the top of the authority choose screen (index). Not shown if there is only 1 direct authority.
- *LOGIN_INDIRECT_MESSAGE* - A message shown at the heading of the indirect authorities. Typically explains that the user will be redirected to another site and then returned.
- *LOGIN_DIRECT_MESSAGE* - A message shown at the top of the login form for direct authorities
- *LOGIN_LABEL* - The label used for the user name field of the login form. This only shows up for logins to direct authorities.
- *PASSWORD_LABEL* - The label used for the password field of the login form. This only shows up for logins to direct authorities.
- *FORGET_PASSWORD_URL* - If specified, a url that is included in the footer of the login form. This only shows up for logins to direct authorities.
- *FORGET_PASSWORD_TEXT* - If specified, text that is included in the footer of the login form. This only shows up for logins to direct authorities.

Statistics Module

The statistics module (/stats) provides an interface to view your site's local analytics. Each page view is recorded with enough detail to create a variety of reports and the module has configurable options to indicate the types of reports and charts to display.

Configuration

Version 1.5 uses a log file to log page views to Kurogo.

View Logging

After configured, Kurogo will log every page view from each module. Each request will include the following data:

- date/time stamp
- site
- service (web or api)
- the request URI
- the referrer
- the user agent
- the ip address
- the logged in user and authority (if using authentication)
- a visit id
- pagetype (from device detection)
- platform (from device detection)
- module
- page
- additional data set by the module (for instance, which news article was read or the search term)
- the size of the response
- the elapsed time generating the response

Reporting

The statistics module includes a series of reports for viewing the overall views for the site as well as views grouped by module, pagetype and platform.

Updating the data

To ensure best performance, the stats data is stored in a log file and then ingested into a database table.

Migration from Old Versions of Kurogo

Previous versions of Kurogo used different manner of stats recording. All requests were immediately added to the database table. In Kurogo 1.5, requests are first logged to a log file and then ingested into the database.

In order to improve performance, the database tables are sharded into different tables based on time. The default option of *KUROGO_STATS_SHARDING_TYPE* is to have 1 table per calendar month. If you want to maintain your previous statistics when upgrading you will need to run a Kurogo shell command to migrate this data:

```
/path/to/kurogo/lib/KurogoShell stats migrate
```

This command can be run on a running server, however keep in mind that it will take a considerable amount of time (in some cases an hour or longer).

Kitchen Sink

The Kitchen Sink module has no user facing functionality. Its sole purpose is to demonstrate the various templates of Kurogo and give developers examples on how standard Kurogo templates and classes are rendered. It can be used to assist developers and designers with theming their site.

The Kurogo Object

The Kurogo object is a singleton instance that contains several methods that consolidate common tasks when developing modules.

Static Class Methods

- *Kurogo::sharedInstance()* - Returns the shared Kurogo singleton object. This is typically not necessary to use since all publicly documented methods are static methods on the Kurogo class.
- *Kurogo::tempDirectory()* - Returns the configured temporary directory
- *Kurogo::siteTimezone()* - Returns a DateTimeZone object set to the site's configured time zone
- *Kurogo::includePackage($packageName)* - Adds a library package to the autoloading path. See *The AutoLoader*
- *Kurogo::getSiteVar($key, $section=null)* - See *Configuration*
- *Kurogo::getOptionalSiteVar($key, $default='', $section=null)* - See *Configuration*
- *Kurogo::getSiteSection($section)* - See *Configuration*
- *Kurogo::getOptionalSiteSection($section)* See *Configuration*
- *Kurogo::getSiteString($key)* - See *Configuration*
- *Kurogo::getOptionalSiteString($key, $default='')* - See *Configuration*
- *Kurogo::getCache($key)* - Retrieves a value from the memory cache - See *Caching*
- *Kurogo::setCache($key, $value, $ttl = null)* - Sets a value to the memory cache - See *Caching*
- *Kurogo::deleteCache($key)* -Removes a value from the memory cache - See *Caching*
- *Kurogo::log($priority, $message, $area)* - Logs a value to the Kurogo log - See *Logging in Kurogo*
- *Kurogo::encrypt($value, key)* - Encrypts a value (requires the mcrypt extension). See *Encryption*
- *Kurogo::decrypt($value, key)* - Decrypts a value (requires the mcrypt extension). See *Encryption*

The AutoLoader

Before using a PHP class, it is necessary to ensure that the class declaration has been loaded. Kurogo follows the pattern to include each class as a distinct PHP file with the same name as its class. It is not necessary, however, to use the PHP *require* or *include* statements to utilize these files Kurogo includes an autoloading mechanism that will automatically include the file when the class is requested. In most cases you can simply utilize a method or constructor of a class and its definition will be loaded. The autoloader will search the site lib folder (if present) and the base Kurogo lib folder for a file with the same name as the class you are attempting to load. So if you make a call to *SomeClass::method()* it will look for the following files:

- *SITE_DIR/lib/SomeClass.php*
- *KurogoRoot/lib/SomeClass.php*

Packages

In order to promote organization of class files, a concept of Packages was created. This allows the grouping of files with similar functionality together. Including a package simply adds that subfolder to the list of paths the autoloader will search. It will also attempt to load a file named *Package.php* in the lib folder. This gives you an opportunity to load global constants or function declarations (not part of a class). This file is optional.

For example, if the *Maps* package is loaded then the following paths will be added to the autoloader search paths:

- *SITE_DIR/lib/Maps/*
- *KurogoRoot/lib/Maps/*

And the autoloader will attempt to load *lib/Maps.php*.

You can create your own packages by simply creating a folder in your site's lib folder. The following packages are part of the Kurogo distribution:

- Authentication (included automatically when authentication is enabled)
- Authorization - for connecting to various OAuth based web services
- 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
- People - used by the people module
- Video - used by the video module

Encryption

Version 1.4 adds methods to store and retrieve encrypted data. This is primarily useful for saving sensitive data from remote servers in a secure fashion. These methods require the mcrypt extension. Like any encryption system it is only secure as the keys used to encrypt the data. The default behavior is to use the SITE_KEY constant which is generated using the install path of the server software. You can set this key by updating the SITE_KEY value in *site.ini*

Caching

Version 1.4 also adds methods to improve the performance of Kurogo by utilizing in-memory caching. If the server contains certain extensions, you can greatly improve the performance of production servers by caching information used by Kurogo such as configuration values, search paths for templates, and remote data values.

Configuration

Kurogo supports caching using 2 different systems:

- APC - The Alternative PHP cache
- Memcache

Configuration for this system is accomplished through the *[cache]* section of *kurogo.ini*. There are a few options used by all caching types:

- *CACHE_CLASS* - The type of caching system to use. Current options include *APCCache* and *MemcacheCache*
- *CACHE_TTL* - The default time-to-live (in seconds) for cache values. This will keep the values in the cache for the specified time. A time of 10-15 minutes (600-900) is usually adequate for most sites

APC

There are no additional options available or needed for APCCache

Memcache

These options only apply to MemcacheCache

- *CACHE_HOST* - The hostname or IP address of your memcache server. If this value is an array (CACHE_HOST[]) then Kurogo will utilize connection failover
- *CACHE_PORT* - The port used to connect to the memcache server. By default this is 11211
- *CACHE_PERSISTENT* - If set to 1 a persistent connection will be used, default is false
- *CACHE_TIMEOUT* - Timeout in seconds to connect to the servers. This should rarely be altered.
- *CACHE_COMPRESSED* - If set to 1, compression will be used. Default is true.

Module Interaction

In many cases it is useful to share data and create links between modules. In order to promote a consistent interface, a series of conventions has been established. These conventions deal primarily with:

- The creation of links and formatting of values to one module based on data values from another (i.e. a link to the map module from values in the people directory)
- The creation of links and formatting of values based on the model object of a module (i.e. the formatting and retrieval of calendar events)
- The retrieval of data from another module based on criteria (used in the federated search feature)

If you are writing new modules or want to format data from your various data sources than you should read this section carefully.

Formatting Data from Existing Modules

There are many examples where you have data that exists in one data source (an LDAP directory, a calendaring system) that references other data (map locations, people). Unfortunately there may not be a strong link between those 2 systems and linking them together requires a bridge.

There are 2 modules in Kurogo–the People and Calendar modules–that have built in support for showing and linking to data in other modules. The *module=xxx*parameter of the detail configuration creates a link to another module. The default implementation simply uses the same value in the directory and links to the *search*page of the target module using the value as a *filter* parameter. So a link to the map module would look like this:

- *map/search?filter=value*

This works in many cases, but sometimes you can provide a more specific link that includes a better formatted text and a more specific link. The overall steps are as follows:

- Create a subclass of the target module (i.e. map)
    - Create a file named *SiteMapWebModule.php* in *SITE_DIR/app/modules/map*. You may have to create the enclosing folders

    - The file should be a subclass of the MapWebModule. You do not need to include any additional properties since this is merely an *extension* of the map module
- Implement a *linkForValue($value, Module $callingModule, KurogoObject $otherValue=null)* method in your subclass. This method will receive:
    - a value from the other system
    - a reference to the calling module (so you can determine which module is making the call)
    - optionally the underlying object provided by the module so you can consider all the values when generating the response

This method should return an array suitable for a *list item*. At very least you must include a *title* value. Normally you would also include a *url* value unless you do not wish to include a link.

```
<?php



class SiteMapWebModule extends MapWebModule

{

  public function linkForValue($value, Module $callingModule,
KurogoObject $otherValue=null) {



      switch ($callingModule->getID())

      {

          case 'people':

            //look at the location field to see which office they
are from.

            //This assumes the relevant location is in the
"location" field.

            $person = $otherValue;
            switch ($person->getField('location'))
            {
                case 'New York':
                    // New York office is first entry
                    return array(
                        'title'=>'New York Office',
                        'url'=>buildURLForModule($this->id,
'detail', array(
                                'featureindex'=>0,
                                'category'=>'group:0'
                            ))
                    );
                    break;

                case 'Boston':
                    // Boston office is the 2nd entry
                    return array(
                        'title'=>'Boston Office',
                        'url'=>buildURLForModule($this->id,
'detail', array(
                                'featureindex'=>0,
                                'category'=>'group:1'

                            ))

                    );

                    break;

                default:

                    // don't include link

                    return array(

                        'title'=>$value

                    );

            }

            break;

        default:

            //return the default implementation for other modules

            return parent::linkForValue($value, $callingModule,
$otherValue);

    }

}

}
```

Enabling interaction from new modules

If you are writing a new module then there are several methods needed to allow full interaction.

- *searchItems($searchTerms, $limit=null, $options=null)* - This method should return an array of objects that conform to the *KurogoObject* interface using the*searchTerms* as a filter. Your implementation should call the necessary methods to perform a simple search using this criteria. You can also utilize the options array to perform more structured queries. If you utilize the default implementation of the federated search method, it will include a *federatedSearch=>true* value in order to handle that case in a unique way if you wish.
- *linkForItem(KurogoObject $object, $options=null)* - This method should return an array suitable for a *list item* based on the object included. This would typically be an object that is returned from the *searchItems* method. An options array is included to permit further customization of the link. For examples, see the People, News, Calendar and Video modules.

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 same name as the current page based on the rules of *Pagetype & Platform Files*.

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.

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

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 specify 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.

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.

{foreach}

Iterates through an array:

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

  {$key} = {$value}

{foreach}
```

See more in the Smarty Documentation

{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

Standard Template Fragments

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

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"}
```

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

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

AppQ

What is AppQ?

AppQ is a component of the Kurogo Mobile Framework that allows mobile web modules to be quickly ported to iOS and Android native applications. AppQ is a hybrid app system where Kurogo Mobile Web modules are embedded in web views inside the native app.

AppQ Features

AppQ uses a native navigation stack

- On iOS (and to a lesser extent Android 4.0+) page transitions are very distinctive and difficult to reproduce with web technologies, so AppQ modules will feel more native than other hybrid solutions, providing a better user experience.
- Because page transitions are handled natively, AppQ modules do not need libraries like jQuery Mobile or Sencha Touch. These large javascript libraries increase page load time and can make hybrid apps feel more sluggish.

AppQ modules can co-exist with native Kurogo-iOS and Kurogo Android modules

- By allowing easy mixing and matching of native and AppQ modules, you can concentrate resources on functionality which gets the greatest benefit from a fully native implementation.

AppQ modules use templates customized for each native platform

- AppQ leverages the Kurogo Mobile Web's template inheritance system to allow you to customize your UI elements for each native platform.
- Stock Kurogo Mobile Web templates are already customized for the native platforms.

AppQ modules store CSS, Javascript and images inside the native app

- This decreases page load time and network bandwidth use
- By including these files in your native app, the user's "first launch" experience will be faster than a normal mobile web experience
- When you update your mobile web UI, the native apps will download new versions of the module files

AppQ 1.0 Technical Capabilities and Limitations

Any Kurogo Mobile Web module can become an AppQ module. If you can write it as a web module, it can be an AppQ module. Like web modules, AppQ modules display live content from the Kurogo Mobile Web server. AppQ is currently not a good fit if your native app needs to store or cache content for offline viewing.

For version 1.0 the following capabilities and limitations are outlined.

What AppQ 1.0 can do:

- Anything a Kurogo Mobile Web module can do, including:
  - Module development with HTML, CSS and Javascript technologies
  - Kurogo Mobile Web theming support
- Ability to update module look and feel without releasing new versions of the native apps
- Use custom “native look and feel” assets for common UI elements and built-in Kurogo mobile web templates
- Access to native UI elements:
  - Navigation bar: can set title, back button title and add a refresh content button
  - Alert dialogs
  - Action dialogs (cancel, optional destructive button, and up to 10 other optional buttons)
  - Share dialog (automatically via Smarty share.tpl template)
  - Native mail composition dialog when user taps on a mailto: link
- Basic GPS location as provided by HTML5 web engine (navigator.geolocation)

What AppQ 1.0 can’t do:

- Offline storage
  - Providing AppQ offline storage similar to what is available on fully native apps is a difficult technical and architectural problem
  - HTML5 offline storage is too small for many use cases
  - Access to native storage mechanisms is a difficult technical and architectural problem on Android
    - On Android the APIs necessary for this (web view cache) were introduced in Ice Cream Sandwich (ICS)
      - Currently most devices are on Gingerbread so we cannot require ICS

        - All supported iOS versions can provide this functionality via NSURLCache and CoreData
- Camera access
- NFC access
- Other direct sensor access

Preparing a Web Module for AppQ

Most modules will work with AppQ with a minimal amount of effort. However there are a few modifications which may be necessary, especially for complex modules with lots of javascript or conditional UI display.

Telling AppQ about your module pages

One of the things AppQ needs to know about your module is which pages it supports. AppQ pulls this information from your module’s pages.ini configuration file. In order for AppQ to function correctly you need to make sure each page your module supports is listed in pages.ini, even if the page title for that page is the same as the page name.

pages.ini

```
[index]



[detail]

pageTitle = "Detail"



[search]

pageTitle = "Search Results"

breadcrumbTitle = "Search"
```

Locating images and other static assets

AppQ builds asset zip files of the javascript, css and images used by a web module. When it searches for these files it loads the templates with all the template variables unset. However, if your web module only shows certain UI elements when the template variables are set, AppQ will not be able to find those elements. To work around this, each module can define a custom version of **initializeForPage()** called **initializeForNativeTemplatePage()** which is used by AppQ. When defining this function you should set your template variables so that all UI elements involving images, CSS and Javascript files are visible.

For example, here is a version of the function for a module which uses the ellipsizer javascript module on its index and search pages and has a share button on its**detail** page:

MyWebModule.php

```
<?php
protected function initializeForNativeTemplatePage() {
    // Native template support
    // specify anything that goes into the header or footer here
    // and force the appearance of assets so they get loaded into the template
    switch ($this->page) {
        case 'index':
            // force appearance of section select button
            $this->assign('sections', array(1, 2));
        case 'search':
            $this-
>addInternalJavascript('/common/javascript/lib/ellipsizer.js');
                break;

            case 'story':
                $this->assign('shareTitle', $this->getLocalizedString('SHARE_THIS_STORY'));
                $this->assign('shareEmailURL', 'dummy');
                $this->assign('shareRemark',    'dummy');
                $this->assign('storyURL',       'dummy');
        }
    }
```

A second option is to specify the files in an array. This option is most useful when getting the page to display some assets would require a lot of code:

MyWebModule.php

```php
<?php
protected function nativeWebTemplateAssets() {
    return array(
'/min/g=file:/common/javascript/lib/ellipsizer.js',
            '/common/images/share.png',
            '/common/images/button-email.png',
            '/common/images/button-facebook.png',
            '/common/images/button-twitter.png'
    );
}
```

A third option is to specify the needed files in the module.ini config file. This option is best when you have added images to your custom module theme and don't want to subclass the module:

module.ini

```ini
[module]
title = "AppQ Test"
disabled = 0
protected = 0
search = 1
secure = 0
```

```
MAX_RESULTS = 10

SHARING_ENABLED = 1



[native_template]

additional_assets[] =
"/min/g=file:/common/javascript/lib/ellipsizer.js"

additional_assets[] = "/common/images/share.png"

additional_assets[] = "/common/images/button-email.png"

additional_assets[] = "/common/images/button-facebook.png"

additional_assets[] = "/common/images/button-twitter.png"



[strings]

help[] = "The news home screen features most recent news across
all categories. Click on an individual news item to read the full
story. Note that clicking a link within the story will launch
your browser. You can share each article using email, Facebook,
or Twitter by clicking on the gray arrow button top right."
```

Javascript global variables

Occasionally web modules will set variables in the global namespace which are subsequently referenced inside loaded javascript files.

AppQ-incompatible global variable use

In the following example, the site has defined a custom header.tpl which defines a global javascript variable myGlobals based on the array contents of a template variable:

header.tpl

```
{extends file="findExtends:common/templates/header.tpl"}

{block name="javascript"}

  <script type="text/javascript">

    var myGlobals = {json_encode($globalsArray)};

  </script>

  {$smarty.block.parent}

{/block}
```

Then in the site's *common.js* the *myGlobals* variable is referenced at the top level of the file:

common.js

```
var firstGlobal = myGlobals[0];
```

Techniques like this are incompatible with AppQ because an empty html wrapper with <head> tag and javascript files are generated beforehand and the per-page content is loaded via AJAX. In the case above the value of myGlobals will always be null because globalsArray won't be set when the html wrapper is generated.

AppQ-compatible global variable use

AppQ doesn't prevent all use of global variables. The key is to use the built-in module functions *WebModule::addInlineJavascript()* and*WebModule::addInlineJavascriptFooter()* to define per-page global javascript variables and then use *WebModule::addOnLoad()* to trigger a function in your common.js to reference them. AppQ will automatically move these javascript blocks around to ensure that they are loaded after the AJAX call and declared in the global namespace.

For example the AppQ-safe way of implementing the above myGlobals variable is instead of overriding header.tpl, we move the extra javascript into the module's php file:

MyWebModule.php

```
<?php

protected function initializeForPage() {

    parent::initializeForPage();

    $this->addInlineJavascript('var myGlobals =
'.json_encode($this->globalsArray).';');

    $this->addOnLoad('myOnLoad();');

}
```

And then in common.js we initialize firstGlobal within the load function:

common.js

```
var firstGlobal = null;

function myOnLoad() {

    firstGlobal = myGlobals[0];

}
```

Using Native Callbacks

AppQ comes with a small number of built-in hooks to native features.

Page Load

On page load, AppQ sets the page title and the title of the back button which will be show on the page immediately after this one on the navigation stack. By default AppQ uses the page title and breadcrumb title used by the mobile web. If you wish custom titles specifically for AppQ you can specify them in your module's *pages.ini*with the keys *nativePageTitle* and *nativeBreadcrumbTitle*.

In addition, AppQ also supports adding a reload button to the navigation bar which will allow the user to reload the content of the page. You can specify the refresh button per page in your module’s pages.ini with the key nativePageRefresh or programmatically with the function *WebModule::setWebBridgePageRefresh()*.

Dialogs

One of the ways users can spot a hybrid app is through its dialogs. Either the dialog is a javascript popup and contains a URL at the top or it is a floating div and only mostly looks like a native dialog. To make AppQ modules feel more native, AppQ provides javascript functions to generate native dialogs for common operations.

Alert Dialogs

Alert dialogs are used when you want to notify the user of an unexpected situation and possibly ask them to choose between doing nothing and one or two actions.

kgoBridge.alertDialog(title, message, cancelButtonTitle, mainButtonTitle, alternateButtonTitle, statusCallback, buttonCallback)

- *title* - (required) A short human readable title (shown in bold on the dialog)
- *message* - (optional) A human-readable message (show in regular text below the title)
- *cancelButtonTitle* - (required) Title of the button which dismisses the alert and cancels any actions the alert refers to
- *mainButtonTitle* - (optional) Title of the primary button
- *alternateButtonTitle* - (optional) Title of an alternate button
- *statusCallback* - (optional) A callback function which will return an error if the dialog fails to display. The callback should have the following signature:
    - function statusCallback(error, params)
        - *error* - If there is no error, this will be null. If there is an error, the error object will contain the following properties:
            - *code* - a numeric code indicating what error occurred
            - *title* - a short string categorizing the error
            - *message* - a string describing the error
        - *params* - (ignored) always null
- *buttonCallback* - (optional) A callback function which is called when one of the buttons is clicked. The callback should have the following signature:
    - *function buttonCallback(error, params)*
        - *error* - If there is no error, this will be null. If there is an error, the error object will contain the following properties:
        - *code* - a numeric code indicating what error occurred

        - *title* - a short string categorizing the error
        - *message* - a string describing the error
    - *params* - If there is no error, the params object will contain the following property:
        - *button* - with a string value indicating which button was tapped. This string may be one of:
            - *cancel*
            - *main*
            - *alternate*

To simplify common cases, the following two functions are also available:

kgoBridge.alert(message, responseCallback)

- *message* - (required) A human-readable message
- *responseCallback* - (optional) A callback function which will be called when the dialog is dismissed. The callback should have the following signature:
    - function responseCallback()

kgoBridge.confirm(question, responseCallback)

- *question* - (required) A human-readable message
- *responseCallback* - (optional) A callback function which will be called when the dialog is dismissed. The callback should have the following signature:
    - *function responseCallback(confirmed)*
        - *confirmed* - true if the user clicked “OK” and false if they clicked “Cancel”.

Action Dialogs

kgoBridge.actionDialog(title, cancelButtonTitle, destructiveButtonTitle, alternateButtonTitles, statusCallback, buttonCallback)

- *title* - (required) A short human readable title
- *cancelButtonTitle* - (required) Title of the button which dismisses the dialog and cancels the action the alert refers to
- *destructiveButtonTitle* - (optional) Title of a destructive action if there is one (e.g. delete data). Button is emphasized or shown in red to warn user.
- *alternateButtonTitles* - (required) An array of titles of additional buttons to display. Each button should correspond to a possible non-destructive action the user can take.

- *statusCallback* - (optional) A callback function which will return an error if the dialog fails to display. The callback should have the following signature:
    - *function statusCallback(error, params)*
        - *error* - If there is no error, this will be null. If there is an error, the error object will contain the following properties:
            - *code* - a numeric code indicating what error occurred
            - *title* - a short string categorizing the error
            - *message* - a string describing the error
        - *params* - (ignored) always null
- *buttonCallback* - (optional) A callback function which is called when one of the buttons is clicked. The callback should have the following signature:
    - *function buttonClickedCallback(error, params)*
        - *error* - If there is no error, this will be null. If there is an error, the error object will contain the following properties:
            - *code* - a numeric code indicating what error occurred
            - *title* - a short string categorizing the error
            - *message* - a string describing the error
        - *params* - If there is no error, the params object will contain the following property:
            - *button* - with a string value indicating which button was tapped. This string may be one of:
                - *cancel*
                - *destructive*
                - *alternateN* - where N is a number between 0 and the number of alternate buttons minus 1

To simplify common cases, the following function is also available:

kgoBridge.shareDialog(buttonConfig)

- *buttonConfig* - (required) An object with the following properties
    - *mail* - (optional) a string containing a URL to share something via email (mailto:user@example.com)
    - *facebook* - (optional) a string containing a URL to share something on Facebook
    - *twitter* - (optional) a string containing a URL to share something on Twitter

Normally you should not need to call the *kgobridge.shareDialog()* function. Just include the share.tpl template and this function will be called for you.

Debug Logging

On iOS it is difficult to get console logging message out of a UIWebView. To make this easier, AppQ provides a logging function which will send messages to the Xcode console via NSLog() when the native app is run in debug mode. This function also works on Android and should be used there as well for consistency.

kgoBridge.log(message)

- message - (required) A human-readable message to log to the native console

Theming a Module for AppQ

Kurogo Mobile Web comes with native app images and styles for stock templates. However if you have custom UI you may also wish to theme your UI specifically for AppQ. You can target AppQ on all native platforms or specific platforms using the same pagetype/platform mechanism used for identifying different types of phone browsers on the mobile web.

Targeting AppQ devices

In order to help you target native apps specifically, AppQ adds a third classification type called “browser”. It also adds a new value “common” for pagetype and platform so that you can target specific browser values for all pagetypes and platforms.

For example, here are the new possible CSS file names and the devices they impact:

pagetype	**-**	**platform**	**-**	**browser**		**Devices Impacted**
common					.css	all devices
compliant					.css	all phones
tablet					.css	all phones
common	-	android			.css	all Android Devices
compliant	-	android			.css	all Android phones
compliant	-	iphone			.css	all iPhones
common	-	common	-	native	.css	AppQ on all devices
compliant	-	common	-	native	.css	AppQ on all phones

pagetype	**-**	**platform**	**-**	**browser**		**Devices Impacted**
tablet	-	common	-	native	.css	AppQ on all tablets
common	-	android	-	native	.css	AppQ on Android devices
common	-	iphone	-	native	.css	AppQ on iOS devices
compliant	-	android	-	native	.css	AppQ on Android phones
compliant	-	iphone	-	native	.css	AppQ on iOS phones
tablet	-	android	-	native	.css	AppQ on Android tablets
tablet	-	iphone	-	native	.css	AppQ on iPad

Javascript files, templates and image directories follow a similar naming scheme.

Debugging AppQ theme problems

Debugging problems with AppQ theming can be difficult on a device due to the lack of a full featured DOM inspector. Fortunately you can use the device debugging feature of the Kurogo Mobile Web server to also debug AppQ modules.

First, make sure *DEVICE_DEBUG=1* in your site.ini file. Then go to http://localhost/device/compliant-iphone-native/mymodule/ to see the AppQ iPhone module version of your module. Similarly http://localhost/device/tablet-android-native/mymodule/ will show you the AppQ Android tablet version of your module. You should use a web browser similar to the native app's browser. For example iOS and Android use a Webkit browser so you will want to use Safari or Chrome when using the debugging mode.

Because AppQ provides a native navigation stack, the AppQ device debugging mode will not show the breadcrumb bar. As a result you will need to use the browser navigation arrows to go back.

This debugging mode simulates the AJAX page loading used by AppQ. Because it is running in a web browser, it cannot implement any of the native hooks normally available to AppQ. When you attempt to trigger one of these hooks, the debug mode will attempt to *console.log()* the trigger URL so you can use the Web Inspector see what parameters would have been passed to the back end.

The AppQ Native Asset Zip File

Once your module has been modified to support AppQ you can build your first asset zip files. These archives contain all the images, CSS and javascript which will be cached locally inside the native app to improve performance.

Building the Asset Zip File

To build the asset zip files, go to the Admin panel and in the navigation menu on the left side, select Module Configuration and then your module. On the right-hand side you will need three tabs underneath the module's name. Select the AppQ tab. You will see buttons to build iPhone and Android templates. Click the buttons which correspond to the native platforms you support. AppQ asset zip files may take up to a minute to generate, depending on how many pages your module supports.

Once the asset zip files have finished building, you will see links to download the images below each button. These links will look likehttp://www.example.com/media/web_bridge/iphone/mymodule.zip. If you have enabled the tablet theme for your site, AppQ will also build tablet versions of the same asset zip files, such as http://www.example.com/media/web_bridge/iphone/mymodule-tablet.zip.

Every time you build a new copy of the asset zip files, the Kurogo Mobile Web server will tell each native app to download the new files. As a result you should try to minimize the number of times you rebuild the files on a production server.

Preloading the Asset Zip File

AppQ asset zip files live on the Kurogo Mobile Web server in your site's media folder. The Kurogo Mobile Web server tells native apps which modules support AppQ, and the apps will then download the asset files from the media folder. While this only happens each time the asset zip files change, you may wish to include the current version of the asset zip files inside your native apps so that the user's "first launch" experience is fast.

The Admin panel provides download links to obtain copies of the asset zip files for inclusion in your native app sources. If you have tablet versions of the assets, you should also obtain those at this time.

iOS (iPhone and iPad)

AppQ asset files live inside the project's resource folder on iOS. Here are example asset zip file paths for the iOS app "MyApp" and AppQ module "mymodule".

- Kurogo-iOS/Projects/MyApp/Resources/modules/mymodule/mymodule.zip
- Kurogo-iOS/Projects/MyApp/Resources/modules/mymodule/mymodule-tablet.zip

These files will be automatically added to your project through Xcode folder references.

Android

AppQ asset files live inside the project's resource folder on iOS. Here are example asset zip file paths for the Android app "MyApp" and AppQ module "mymodule".

- Kurogo-Android/site/MyApp/config/modules/mymodule/assets/web_bridge.zip
- Kurogo-Android/site/MyApp/config/modules/mymodule/assets/web_bridge-tablet.zip

Once the asset zip files are copied into those locations, clean your project and rebuild and the assets will be built into your app.

Updating an AppQ Module

Every time you update your module and deploy it to your Kurogo Mobile Web server, you will need to build new asset zip files so that the zip files match the mobile web version of the module. Existing native apps will download the new zip files so you do not need to worry about your existing users. However after you deploy to your Kurogo Mobile Web server, you may wish to release a new version of your native app with the new zip files included so that the "first launch" experience does not involve an immediate download of the updated asset zip files. Unless your module has large numbers of images, its zip files will be small, but the download time may still be noticeable on a slow network.

Style and Themes

The Kurogo Framework supports both simple and deep visual customization to reflect your organization’s visual brand identity. Basic visual properties (such as colors, content and header backgrounds, fonts, and more) are very easy to customize across all device classes with changes to a theme configuration file and a handful of CSS files. Logos, module icons, header and body backgrounds, and other images can be easily replaced to complete your visual branding. Kurogo also gives you the flexibility to deliver advanced CSS and high-resolution images to devices and browsers that can support them, or use a simpler set of theme assets to simplify theme creation and maintenance.

Beyond straightforward visual branding, Kurogo theming can also extend deeper into application-level styling, templates, and images. Just about anything your users can see or interact with can be customized, depending on your institution’s needs and your development team’s technical abilities. This document covers the basics of visual theming; functional customization through module extensions and template overrides is covered in *Extending an existing module*.

Visual theming requires a working knowledge of CSS, and skill with an image editor such as Photoshop or GIMP.

Theming Overview

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 changes to your site’s specific theme are isolated from the underlying framework 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, and a core set of application images. All HTML, CSS and Javascript in the core interface can be overridden by a theme. While it’s possible to directly edit the files in *app/*, doing so will make it likely that future upgrades to Kurogo will break your site. As with everything else you build with Kurogo, it is highly recommended that you **not** directly edit any contents of this directory. If there are CSS rules or image files you wish to replace in your theme, it is recommended that you create new versions of those rules and images in your theme directory.

Each theme is contained within a directory inside the *SITE_DIR/themes* folder. By convention the default theme is named *default*. Each site can have multiple themes, but only one theme can be active at any time. You can easily switch between active themes from the *Site Configuration > Theme* screen in the Kurogo administration console.

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.

Tutorial: Implement a Simple Theme

Because of Kurogo's breadth and depth, implementing a simple theme is a multi-step process. However, each step can be broken down into fairly discrete tasks, and with each version of Kurogo we have made efforts to reduce the number of files and image assets that need to be revised or replaced to create a workable theme. Of course, theming can be as deep and extensive as you desire. This is part of Kurogo's underlying philosophy of flexibility and scalability – making it easy to get up and running while supporting potentially limitless customization and extension to meet your organization's specific mobile needs now and in the future.

1. Create a working theme directory

It's recommended that you build a new theme by duplicating the default theme, editing its theme CSS, and replacing key image files. This allows you to quickly switch back to the default theme to check the effect of changes you're making in your new theme, or to revert to a working theme if you run into trouble.

The first decision to make is whether you want to make the extra effort to create high-resolution assets as part of *Optimizing for High-Density Displays*. This extra effort (consisting of several extra versions of up to dozens of image files) can yield noticeably sharper-looking images on devices with high-resolution screens. Don't worry if you're not sure or change your mind; you can always start with the default (simpler) theme structure and refine later with as much or as little high-density optimization as you like.

In *SITE_DIR/themes*, duplicate either the *default* (simpler) or *hi-def* (optimized for high-density displays) directory and give the new directory a descriptive name.

In your site's Kurogo administration console, go to the *Site Configuration > Theme* page and select your new theme, and click the "Save" button.

In a modern web browser (e.g., Chrome, Firefox 4+, Safari 4+), open a few test views of your site for different device classes:

- *http://[SITE_PATH]/device/compliant/[HOME_MODULE]/*
- *http://[SITE_PATH]/device/compliant-bbplus/[HOME_MODULE]/*
- *http://[SITE_PATH]/device/touch/[HOME_MODULE]/*
- *http://[SITE_PATH]/device/basic/[HOME_MODULE]/*
- *http://[SITE_PATH]/device/tablet/[HOME_MODULE]/*

As you make the changes detailed below, come back to your browser and refresh the relevant test views to make sure that the changes have the intended effect.

2. Modify the basic theme values

Kurogo 1.4 introduces support for setting theme values in a single configuration file, for theme attributes such as colors and font styles that get applied across many theme CSS files. This theme configuration file dramatically simplifies the definition of most if not all of the core theming that was previously done in CSS.

In your theme directory (which we'll refer to from now on as *THEME_DIR*), open *config.ini*. This is the theme configuration file.

At runtime, Kurogo parses all of the CSS which has been concatenated for the current page and device class and looks for a specific text pattern that indicates the use of theme constants. You can see this in your *THEME_DIR/common/common.css* file; look for any value defined in the format *@@@[constant_name]@@@*. Kurogo then replaces this pattern with the value of that named constant as defined in the theme configuration file. For instance, every instance of *@@@primary_text_color@@@* will be replaced by the actual value of the constant *primary_text_color* as defined in your theme configuration file; in the case of the standard Kurogo reference theme ("Universitas" in blue), this value is #036, or a dark blue. So, for example, line 15 of common.css would be delivered to the user's browser as *color: #036*.

In *THEME_DIR/config.ini*, the global theme style constants are in lines 14-39. These constants are:

- *primary_font_family*: Font family definition for the entire site. This can be defined as a single value (e.g., "Arial" or "sans-serif"), but given the non-overlapping range of default fonts available on different mobile platforms, an array of families is recommended, with a fallback of “sans-serif” or “serif”.

- *base_font_size*: Base font size. Almost all of the other font sizes throughout your web app will be calculated as percentages of this base font size, which can be specified in points or pixels.
- *primary_text_color*: Primary text color for most text and navigation elements throughout the site
- *secondary_text_color*: Text color for secondary text such as smallprint, fineprint, legends, labels, nonselected tabs, and text in secondary nav lists
- *tertiary_text_color*: Text color for the small-print detail text that sometimes shows up in secondary nav list items
- *nonfocal_text_color*: Color of text items (including text links) that appear on the body background, e.g., within nonfocal elements
- *strong_text_color*: Text color for certain emphasized elements, such as active tabs
- *heading_font_family*: Font family for <h1> and <h2> elements, including content item titles
- *primary_heading_color*: Text color for <h1> and <h2> elements, including content item titles
- *secondary_heading_color*: Color for <h3> and <h4> elements, including section headings within grouped tables and subheadings within content areas
- *nonfocal_heading_color*: Color for all <h1>, <h2>, <h3>, and <h4> elements that appear on the body background, e.g., within nonfocal elements
- *focal_link_color*: Color of text links within focal content areas
- *nonfocal_link_color*: Color of text links that appear on the body background, e.g., within nonfocal elements
- *body_background_color*: Overall body background color. Often largely or even entirely hidden or overridden by focal content areas.
- *body_background_image*: Overall body background image. Optional; use “none” if no image is required, or CSS format e.g., “url(/common/images/bodyback.jpg)”
- *focal_background_color*: Background color for focal content areas and navigation and results lists
- *subfocal_background_color*: Tinted background color for subfocal content areas and nested navigation lists. Usually a tint between body_background_color and focal_background_color
- *focal_border_color*: Color of borders around focal content areas and navigation and results lists, and hairlines between list items

- *search_border_color*: Color of borders around search fields and behind text search buttons
- *navbar_background_color*: Background color behind navbar (usually not seen if there's a navbar background image)
- *navbar_text_color*: Color of navigation-bar text (page title and breadcrumb text) in Compliant and Tablet device classes

In addition to modifying these theme configuration values, you can always modify or override individual rules in specific theme CSS files. However, simply changing these theme configuration values will take care of most, if not all, of the theming that previously required editing actual CSS.

You can also extend the use of these global theme values by adding your own constants to your *THEME_DIR/config.ini*. Any value that can be expressed as a text string can be defined in the theme config file and used in any of your theme CSS files, whether at the common level or at the level of individual modules.

3. Add your logo or other branding artwork

Your organization's logo (or other identifying/branding image to be used in your mobile web app) will typically appear in several places:

Homepage

You'll need to create a version of the logo to appear on the homepage: [1] [2]

- Basic and Touch device classes: *THEME_DIR/modules/[HOME_MODULE]/images/logo-home.gif* must be a GIF image [3]. This image will be centered horizontally within the screen. The default size is 208x35px.
- Compliant device class: *THEME_DIR/modules/[HOME_MODULE]/images/logo-home.png* must be a PNG image [3]. The default size is 280x60px. The Compliant home logo/banner image is one that benefits noticeably from *Optimizing for High-Density Displays*.

Header logos

The top left corner of every screen for every device class includes a logo/branding image. This image appears to the left of the page title on the Basic device class, and as the leftmost part of the header/navigation bar on all other device classes.

- Basic device class: *THEME_DIR/common/images/basic/logo.gif* must be a GIF image [3]. The default size is 35x35px.
- Compliant device class: *THEME_DIR/common/images/compliant/homelink.png* must be a PNG image [3]. The default size is 57x45px. It should be designed in such a way that it appears seamlessly on top of the header/navigation bar background (*navback.png*, in, in the same directory). The Compliant header logo is one item that benefits from *Optimizing for High-Density Displays*.
- Touch device class: *THEME_DIR/common/images/touch/homelink.gif* must be a GIF image [3]. The default size is 40x30px. It should be designed in such a way that it appears seamlessly on top of the header/navigation bar background (*navback.jpg*, in, in the same directory). Typically it should incorporate some visual indication of a drilldown (e.g., right-facing arrow) to the right of the actual logo.
- Tablet device class: *THEME_DIR/common/images/tablet/homelink.png* must be a PNG image [3]. The default size is 66x52px. This is designed in such a way that it appears seamlessly on top of the header/navigation bar background (*navback.png*, in the same directory).

Favicon and bookmark icons

- *THEME_DIR/common/images/favicon.ico* must be a 16x16px ICO file, which is variously used by different browsers as the favicon, bookmarks and history icon, and in the screen title bar.
- *THEME_DIR/common/images/icon.png* must be a 57x57 (or pixel-doubled 114x114px; see *Optimizing for High-Density Displays* [3]) PNG, used as the homescreen shortcut icon for iOS devices and some Android devices.

4. Customize or replace the module icons

Each module is visually represented by an icon on all device classes other than Basic. Kurogo’s default theme includes a full set of professionally-created module icons, including many for modules not actually included in Kurogo. You are free to use and modify these icons, or replace some or all of them with ones that you create or license. If you’re creating or licensing your own module icons, it’s highly recommended that you start with vector images (e.g., Illustrator or EPS), which can be scaled to any size at full quality. If you can’t create or purchase vector icon images, at least make every effort to start with bitmap (e.g., Photoshop) images at a large size such as 200x200px before scaling down to the actual sizes and formats you’ll need for your web app.

The module icons need to be saved in the following sizes and formats:

Homepage module icons

These appear on the homepage, as well as the Customize Homescreen module and the desktop-oriented Info module.

- Compliant device class: The module icons in *THEME_DIR/modules/[HOME_MODULE]/images/complaint/[MODULE_ID].png* must be PNG images [3]. They should be the same size as the springboard images for modern BlackBerry devices (as set in *THEME_DIR/common/css/compliant-bbplus.css*, lines 26-27, and*THEME_DIR/common/css/compliant-blackberry.css*, lines 17-18). By default this is 64x64px, which is slightly larger than the default size for other Compliant devices. The file names must be exactly in the format *[MODULE_ID].png* (e.g., calendar.png, map.png, news.png, etc.)[#f4]_. For Compliant devices, the homepage icons may notably benefit from *Optimizing for High-Density Displays*.
- Touch device class: The module icons in *THEME_DIR/modules/[HOME_MODULE]/images/touch/[MODULE_ID].gif* must be GIF images [3]. The default size is 44x44px. The file names must be exactly *[MODULE_ID].gif* (e.g., calendar.gif, map.gif, news.gif, etc.) [4]

Breadcrumb module icons

These appear in the header/navigation bar at the top of every module page in all device classes other than Basic. On each module's main screen, the icon is used to identify the module but is not tappable; in all subsequent drilldown screens, the icon is incorporated into a tappable/clickable breadcrumb by which the user can navigate back to the module home screen.

- Compliant device class: The icons in *THEME_DIR/common/images/complaint/title_[MODULE_ID].png* must be PNG images [3], generally transparent, colored and styled to look good on the background color/image for the navigation bar (this background is specified in the *#navbar* rule in*THEME_DIR/common/css/compliant.css*). The default size is 28x28px. For Compliant devices, the breadcrumb module icons may notably benefit from *Optimizing for High-Density Displays*.
- Touch device class: The icons in *THEME_DIR/common/images/touch/title_[MODULE_ID].gif* must be GIF

images [3], generally transparent, colored and styled to look good on the background color/image for the navigation bar (this background is specified in the *#navbar* rule in *THEME_DIR/common/css/touch.css*).. The default size is 28x28px.
- Tablet device class: The icons in *THEME_DIR/common/images/tablet/title_[MODULE_ID].png* must be PNG images [3], generally transparent, colored and styled to look good on the background color/image for the navigation bar (this background is specified in the *#navbar* rule in *THEME_DIR/common/css/tablet.css*).. The default size is 28x28px.

Tablet tab-bar module icons

The Tablet device class uses a site-wide tab bar at the bottom of the screen to provide quick navigation between modules. Though not technically part of the Tablet homepage, these images are in the *THEME_DIR/modules/[HOME_MODULE]/images/tablet/* directory, to keep them grouped with the other module icons of similar size and format. The Tablet's tab bar uses two variations of the module icons. Both variations must be transparent PNGs [3] at 45x45px. Larger sizes will work fine, but with no visible benefit..

- Normal/unselected: Should be colored and styled for good contrast and legibility against the background for the Tablet tab bar. This background is specified in the *#footernav* rule in *THEME_DIR/common/css/tablet.css*. The file names must be exactly *[MODULE_ID].png* (e.g., calendar.png, map.png, news.png, etc.) [4]
- Selected: Should be colored and styled for good contrast and legibility against the background for the selected state of the Tablet tab bar. This background is specified in the *#footernav .selected a* rule in *THEME_DIR/common/css/tablet.css*. The file names must be exactly *[MODULE_ID]-selected.png* (e.g., calendar.png, map.png, news.png, etc.) [4]

5. Customize or replace supporting graphics

The following secondary and support graphics should be color-adjusted or replaced to match your overall theme design:

Help buttons

Buttons in the top right of the screen for Compliant and Tablet device classes:

- Compliant device class: *THEME_DIR/common/images/compliant/help.png* must be a PNG image, typically 24-bit with transparency, for use on Compliant-class devices. The default size is 46x45px. It should be designed in such a way that it appears seamlessly on top of the header/navigation bar background (navback.png, in the same directory).

- Tablet device class: *THEME_DIR/common/images/tablet/help.png* must be a PNG image, typically 24-bit with transparency, for use on Compliant-class devices. The default size is 52x52px. It should be designed in such a way that it appears seamlessly on top of the header/navigation bar background (navback.png, in the same directory).

Header bar backgrounds

Tiling background image for the header bar (navigation and breadrcrumbs) at the top of every screen in most device classes:

- Compliant device class: *THEME_DIR/common/images/compliant/navback.png* must be a PNG image, typically 24-bit with transparency, for use on Compliant-class devices. The default size is any width by 48px tall, of which the bottom 3px is typically a drop shadow fading to transparent.
- Touch device class: *THEME_DIR/common/images/touch/navback.jpg* must be a JPG image, for use on Touch-class devices. The default size is any width by 48px tall, of which the bottom 3px is typically a drop shadow fading to the body background color.
- Tablet device class: *THEME_DIR/common/images/tablet/navback.png* must be a PNG image, typically 24-bit with transparency, for use on Tablet-class devices. The default size is any width by 50px tall.

Breadcrumb separator images

Separator image between elements of the breadcrumb (drill-up) links in the header bar for Compliant and Tablet device classes:

- Compliant device class: *THEME_DIR/common/images/compliant/drillup-r.png* must be a PNG image, typically 24-bit with or without transparency, for use on Compliant-class devices. The default size is 18x45px, and it should be designed to sit seamlessly on top of the header bar background (*THEME_DIR/common/images/compliant/navback.png*).
- Compliant device class: *THEME_DIR/common/images/tablet/drillup-r.png* must be a PNG image, typically 24-bit with transparency, for use on Compliant-class devices. The default size is 18x50px, and it should be designed to sit seamlessly on top of the header bar background (*THEME_DIR/common/images/tablet/navback.png*).

Other graphics

Color-adjust or replace any or all of the following with images of the same size and format:

- Bullet images: *THEME_DIR/common/images/compliant/bullet.png* and *THEME_DIR/common/images/tablet/bullet.png* (identical), and*THEME_DIR/common/images/touch/bullet.gif*
- Search buttons: *THEME_DIR/common/images/compliant/search_button.png* and *THEME_DIR/common/images/tablet/bullet.png* (identical)

Optimizing for High-Density Displays

All modern smartphones have displays with a pixel density (number of pixels per physical inch) higher than a typical desktop or laptop computer. For example, the first three generations of iPhones and iPod Touches, and the first generation of Android and webOS devices, all had displays with 150-170 pixels per inch (ppi).

A growing number of high-end devices have significantly higher-density displays, to further improve clarity and legibility. iOS devices with Retina Displays (iPhone 4 and 4S, iPod Touch 4) have twice the pixel density of older iOS devices. Android devices with HDPI displays (e.g., with the common 480x800px or 480x854px screens) and XHDPI displays (e.g., the 720p displays on the latest flagship Android phones), Windows Phone 7 devices, and some recent webOS devices have 1.5 times (or more) the pixel density of earlier/lower-end smartphones. Because these devices have more physical screen pixels in the same space, text and images can look sharper and more legible, especially for small text and detailed graphics.

On such devices, web pages that provide a higher-resolution image while retaining the display size (through HTML attributes or CSS) can yield images that are visibly sharper and more legible on-screen. For instance, substituting a pixel-doubled homescreen logo (*THEME_DIR/modules/[HOME_MODULE]/images/logo-home.png*) at 560x120px (twice the default 280x60px size) while retaining the *width=280, height=60* attributes in HTML will make that image have maximum possible visual quality on high-density displays. However, this comes at the cost of larger file size. You need to evaluate whether the increased visual quality and legibility are worth the tradeoff. In many cases, 1.5x assets (e.g., 420x90px version of *THEME_DIR/modules/[HOME_MODULE]/images/logo-home.png*) will offer a good tradeoff between increased visual quality and file-size. You may want to experiment with different multipliers, viewing the results on different devices, to find the best tradeoff on an image-by-image basis.

Generally, logos, highly detailed images, and images incorporating text will benefit most from using high-density versions. Note that BlackBerry devices running any OS prior to 6.0 do not scale images well, so it's best to use images sized exactly for them. Currently there are no tablet devices that take advantage of high-density images.

Kurogo ships with two reference themes: default (simple, standard-resolution) and "hi-def" (with optimizations for high-density displays). By switching between these themes in your site admin console and viewing it on a high-density device (e.g., iPhone 4, iPod Touch 4, high-end Android device, Pre3, etc.), you can see for yourself the difference that such optimizations make, and decide for yourself the degree to which you want to make such optimizations for your own site.

The following items will benefit the most from using higher-resolution images. The general technique is the add the higher-than-default-resolution images to the*[IMAGE_DIR]/compliant/* directory, and default-resolution images to the *[IMAGE_DIR/compliant-blackberry]* and *[IMAGE_DIR/compliant-bbplus]* directories.

Home-screen logo

Assuming you've created your standard-resolution *THEME_DIR/modules/[HOME_MODULE]/images/logo-home.png* image, make duplicates of it into*THEME_DIR/modules/[HOME_MODULE]/images/compliant-bbplus* and *THEME_DIR/modules/[HOME_MODULE]/images/compliant-blackberry* directories. Then replace*THEME_DIR/modules/[HOME_MODULE]/images/logo-home.png* with a higher-resolution version.

Header logo images

Assuming you've created your standard-resolution *THEME_DIR/common/images/compliant/homelink.png* image, make duplicates of it into*THEME_DIR/common/images/compliant-bbplus* and *THEME_DIR/common/images/compliant-blackberry* directories. Then replace*THEME_DIR/common/images/compliant/homelink.png* with a higher-resolution version, making sure that this higher-resolution version mates well with the navbar background image (*THEME_DIR/common/images/compliant/navback.png*).

Homepage module icons

Assuming you’ve created your standard-resolution module icons at *THEME_DIR/modules/[HOME_MODULE]/images/compliant/[MODULE_ID].png*, make duplicates of all of them into *THEME_DIR/modules/[HOME_MODULE]/images/compliant-bbplus* and *THEME_DIR/modules/[HOME_MODULE]/images/compliant-blackberry*directories. Then replace the module icons in *THEME_DIR/modules/[HOME_MODULE]/images/compliant* with higher-resolution versions, being sure to name them exactly *[MODULE_ID].png* [4]. **Caution:** This can quickly make the total filesize of your homepage quite large, especially if you have a lot of modules. Try 1.5x versions of these images first, rather than 2x (Retina Display) versions.

Breadcrumb module icons

Assuming you’ve created your standard-resolution breadcrumb module icons at *THEME_DIR/common/images/compliant/title_[MODULE_ID].png*, make duplicates of all of them into *THEME_DIR/common/images/compliant/compliant-bbplus* and *THEME_DIR/common/images/compliant-blackberry* directories. Then replace the module icons in *THEME_DIR/common/images/compliant* with higher-resolution versions, being sure to name them exactly *title_[MODULE_ID].png* [4].

Technical Notes about Theming

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:

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

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.

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.

Footnotes

[1] **Custom homepage logo/banner image sizes:** *THEME_DIR/config.ini* stores the height and width of the homescreen logo/banner image for different device classes. The values defined in this config file are written into the actual HTML as attributes on the <img> tag. The reason these image dimensions are handled this way, rather than in CSS, is that many browsers will not apply a CSS height and width until the image is loaded, but will always reserve the space defined in the <img> object's *height* and *width* attributes. The CSS-driven approach will cause the items on the home screen to jump vertically as soon as the logo image finishes loading, causing a usability problem, especially on touchscreen devices.

[2] **Homepage with full-bleed banner image:** If you create a home-page design a full-bleed focal image at the top of the page (e.g., a large photograph with your logo superimposed on it), you can set the image dimensions in *THEME_DIR/config.ini* to *banner-width = 100%* and *banner-height = auto*. You should create the artwork at a minimum width of 320px, with a recommended maximum height of 240px. Note that this approach is only recommended for the Compliant device class, as the GIF image(s) used for the Basic and Touch device classes will render very poorly when scaled.

[3] *(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)* **Transparent GIFs and PNGs:** Assets for the Basic and Touch device classes are often GIFs. These should typically be transparent with a transparency matte color matching your homepage background color (except for images that are meant exclusively to sit on focal content areas, in which case the transparency matte color should be white). Assets for the Compliant and Tablet device classes are often PNGs. When tranparent PNGs are used, 24-bit with transparency will work best; 8-bit with transparency can be used to minimize file-size, but the background matte color will need to be set similarly to that of the transparent GIFs.

[4] *(1, 2, 3, 4, 5)* **Module IDs:** All of the variations of the module icons need to have filenames based on the relevant module ID. Generally, you'll be safe just replacing existing files with new ones with the same name. If you want to be sure of the module ID, you can go to you r

Standard Libraries

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

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:

Module -> Data Model -> Data Retriever -> Data Parser -> Returns Objects

Data retrieval is discussed in depth in *Data Retrieval*.

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

Data Retrieval

Kurogo’s foundation is built on the concept of accessing remote data. Nearly every module connects to remote servers, using a variety of protocols, and retrieves this data, and normalizes it in a format that the module can use. In Kurogo versions 1.3 and earlier, this was typically accomplished using the DataController Class. In version 1.4, the system has been refactored and improved to introduce more flexibility for developers and more easily permit additional protocols beyond simple HTTP to access the remote data. This includes SOAP and Database access in addition to more flexible HTTP access. It also permits more complicated data modeling scenarios to deal with situations where the necessary data cannot be retrieved in a single request.

- Data Model
- Data Retriever
- Data Parser
- Data Response

Data Model

The Data Model class is the interface the module uses to access the data. Each module is typically paired with a Data Model Class (NewsDataModel, PeopleDataModel). This permits the module to be written more simply by using easy to understand methods and a common interface regardless of the mechanism to retrieve the data.

Many modules use a common subclass, the *ItemListDataModel*. This subclass has the ability to grab a list of items, manage things like paging and limiting options, search, and single item retrieval. It is designed to be used with data that presents a collection of items and allows the user to view a detail of an individual item.

Developers should not need to write any subclass of a Data Model for included modules. If you want to connect to one of your services, then you will either set the appropriate configuration, or write a DataRetriever to connect to this service.

ItemListDataModel

This subclass of Data Model has methods to manage a collection of items and retrieve individual items. It also permits searching within the collection.

Overview

This model will retrieve the values from the retriever and expects to receive an array of items. The *getItem* method will allow you to retrieve a specific item from the collection. For retrievers that have specific methods for retrieving a single item they should implements the *ItemDataRetriever* interface. Otherwise the model will return the item that matches the id. If a module has a method for searching collections, then it should implement the *SearchDataRetriever* interface. If it does, the model will call the search method on the retriever. Otherwise it will iterate through the collection and return items that match a *filterEvent* method.

Subclassing

If you are writing a module that processes a collection of similar items, then creating a subclass is appropriate. It is not not necessary to create a subclass if you are simply adding another service to an existing module. You should instead create a new data retriever. Refer to the documentation for the module to see details on the values and options sent to each retriever.

Public Methods

These methods can be called by the module to set options and retrieve the data.

- *setStart($start)* - Sets the starting item to retrieve in the collection. This is used by modules to implement paging of the collection. The first item in the collection is 0. The 'start' option will be set so it can be used by the retriever if the service supports limiting records.
- *setLimit($limit)* - Sets the maximum number of items to return. This is used by modules to implement paging of the collection. If this value is not set then all the items in the collection will be returned. The first item in the collection is 0. The 'limit' option will be set so it can be used by the retriever if the service supports limiting records.
- *items()* - Return the collection of items based on the current options
- *search($keyword)* - Performs a search and returns the items that match the search.
- *getItem($id)* - Returns a single item based on its id
- *getTotalItems()* - Returns the total number of items in the collection. Used to implement paging.

Relationship with the Retriever

As part of the initialization process, the Data Model will instantiate a *Data Retriever*. It will pass the initialization parameters to the retriever.

Options

The Data model uses a dictionary of options to configure the retriever. The specific options set are indicated in each module's documentation. Examples include:

- *startDate* - Set by the calendar data model to indicate the start date of events to retrieve
- *author*- Set by the video data model to indicate the author to retrieve
- *limit* - Set by the ItemListDataModel to limit the number of items to return
- *action* - used by several data models to set the mode of retrieval. (i.e. individual item vs. collection vs search)

Refer to the documentation for each module for information on writing retrievers for that particular module.

Data Retriever

The Data Retriever class actually retrieves the data from the remote service. It accepts a series of initialization arguments and contacts the service using the appropriate protocol. Most custom DataRetrievers will be a subclass of one of the standard retrievers:

- *URLDataRetriever* - Retrieves data using URL or a local file. *OAuthDataRetriever* is a standard subclass that using OAuth to sign the request.
- *SOAPDataRetriever* - Retrieves data using a SOAP request using a WSDL file, method and parameters.
- *DatabaseDataRetriever* - Retrieves data using a SQL query.

The DataRetriever class also handles caching and sending the response data to the *parser* for processing.

Developers would need to write a custom retriever if the service does not use a fixed URL, method (SOAP) or SQL query (database).

Public Methods

There are 2 critical methods that are used by any interface to the data retriever:

- *getData(&$response)* - This method returns the parsed data. The *response* should also be set in the response variable. This method will retrieve the data and response from the cache if available.
- *getResponse()* - Retrieves the data and returns just the DataResponse object.
- *setOption($var, $value)* - Sets an option on the retriever.

These methods generally don’t need to be overridden if you are using one of the standard subclasses.

Internal Methods

- *retrieveResponse()* - This method should return a DataResponse object. If you are not using one of the standard subclasses, then you must implement this method to actually retrieve the data.
- *setContext($var, $value)* - This value is passed to the DataParser. It can be used to set options that affect the parsing of the data, as well as pass important state information.
- *initResponse* - Call this method if you are implementing your own *retrieveResponse()* method. It will return a initialized response object ready for use.

Special Properties

There are several properties that can affect the behavior of all retrievers:

- *DEFAULT_RESPONSE_CLASS* - The subclass of DataResponse that is used for the response.
- *DEFAULT_PARSER_CLASS* - The default parser that that will be used if the *PARSER_CLASS* initialization value is not set.
- *PARSER_INTERFACE* - The required interface for a DataParser. Some retrievers require their parsers to conform to a particular interface.
- *DEFAULT_CACHE_LIFETIME* - The default cache timeout if the *CACHE_LIFETIME* initialization value is not set.

Initialization Values

Retrievers typically are sent the values of a configuration file. There are several values that used by all retrievers to configure the behavior of a particular instance:

- *PARSER_CLASS* - Sets the data parser class to be used.
- *AUTHORITY* - Sets the *authentication authority* to be used. This is useful if your retriever requires authentication information
- *OPTIONS* - Can set a series of options to the retriever. Options and their values are specific to the retriever. Since options are defined in this fashion as arrays, they can only be used in PHP 5.3 (PHP 5.2 does not support array syntax in ini files)

Certain subclasses have additional configuration values.

Writing A Retriever

For most cases, you want to subclass an existing standard retriever type (URL, SOAP, Database). But if you want to create your own, then you must implement*retrieveResponse* which should return a Data Response object. You should consult the documentation for the module to learn what options are set and what additional interfaces need to be supported.

```
<?php



class MyDataRetriever extends DataRetriever

{
    protected function init($args) {

        parent::init($args); // you MUST call parent::init()


        /* handle a certain config value */

        if (isset($args['SOME_CONFIG_VALUE'])) {

            $this->doSomething();

        }


    }


    protected function retrieveResponse() {

        $response = $this->initResponse();


        // do the work to actually get the data

        $data = doSomethingToGetData();


        $response->setResponse($data);


        return $response;

    }
```

```
}
```

This is an example for a generic retriever. Subclasses of standard retrievers should be implemented differently. Consult the documentation for each common subclass for more information.

Common Subclasses

- URLDataRetriever
- SOAPDataRetriever
- DatabaseDataRetriever

URLDataRetriever

The URLDataRetriever is the most common retriever used. It retrieves data from a HTTP server (or local file). You can either configure it with static configuration data as well implementing methods to determine the values dynamically.

There are several values that ultimately make up the request:

- The base url - This could be an http url or a local file reference
- The query string parameters
- HTTP method - The default is GET
- Custom HTTP headers
- POST data - when using a POST method

Static Configuration

When retrieving the data from a configuration file, the URLDataretriever will set the following values, if present:

- *BASE_URL* - The initial URL. It should include http/https as necessary or can reference a local file. You can use one of the directory constants such as*DATA_DIR*. It can include the query string
- *METHOD* - The HTTP method. By default, it will use GET
- *HEADERS* - An array of headers. Each header should be expressed as: HEADERS[header] = “value”. Since headers are defined in this fashion as arrays, they can only be used in PHP 5.3 (PHP 5.2 does not support array syntax in ini files)
- *DATA* - Data to be sent if using POST.

Dynamic Values

In some cases the URL cannot be determined until runtime. This is because it relies on data input from the user or other information. Subclasses of URLDataRetriever have the opportunity to set these values at a variety of times depending on when it is appropriate.

Internal Methods

These methods could be called in the *init*, *setOption* or *initRequest* methods to set the values.

- *setBaseURL($url, $resetParameters=true)* - Sets the url. If $resetParameteters is true (the default) then the array of parameters will be reset. This value can include the query string, but it is typically better to use addParameter to ensure values are escaped properly.

- *addParameter($var, $value)* - Adds a value to the query string.
- *setMethod($method)* - Sets the HTTP method (GET, POST)
- *setHeader($header, $value)* - Sets an HTTP header
- *setData($data)* - Sets the data used in a POST request

Callback methods

There are a variety of methods that are called when the request is prepared. You can subclass these methods to return your own values.

- *initRequest* - This method is called before the request is made. This is an opportunity to set the various values using the above internal methods based on the current options and settings. You should call parent::initRequest(). No return value is necessary

cURL

In certain cases you may wish to utilize the cURL library for retrieving data. You can include a *USE_CURL = 1* option to enable cURL. The configuration will also recognize valid CURLOPT_XXXX options set in the configuration file. i.e. CURLOPT_PROXY, CURLOPT_SSLCERT, CURLOPT_USERPWD, etc.

SOAPDataRetriever

The SOAPDataRetriever allows you to easily retrieve data from a SOAP web service. This retriever uses a subclass of the built-in PHP SOAPClient class. You can either configure it with static configuration data as well implementing methods to determine the values dynamically.

There are several values that ultimately make up the request:

- The base url. This typically comes from a WSDL file, but can be set manually
- The target URI. This value typically comes from a WSDL file but can be set manually.
- The SOAP Method
- The parameters for the request. The format of these parameters is dependent on the SOAP method being called

Static Configuration

When retrieving the data from a configuration file, the SOAPDataretriever will set the following values, if present:

- *WSDL* - The location (URL or local location) of the WSDL file. This is typically the best way to initialize the SOAP service.

The following values are only valid if you are not using a WSDL file:

- *BASE_URL* - The HTTP endpoint of this service. This is only necessary when not providing a WSDL location.
- *URI* - The target URI. This is only necessary when not providing a WSDL location.

SOAP Actions

- *METHOD* - The SOAP method to call
- *PARAMETERS* - An array of parameters to send. Note that if the SOAP method requires a complex value for any of the paramaters, you cannot define this in configuration since it cannot be syntactically expressed in the INI file

Dynamic Values

In many cases the method cannot be determined until runtime. This is because it relies on data input from the user or other information. Subclasses of SOAPDataRetriever have the opportunity to set these values at a variety of times depending on when it is appropriate.

Internal Methods

These methods could be called in the *init*, *setOption* or *initRequest* methods to set the values.

- *setMethod($method)* - the SOAP method to be called
- *setParameters($parameters)* - an array of parameters to be sent with the request. Note that some services use a single parameter that is a complex value (i.e. a single element array whose sole value is an array of values), and some services use an array of parameters (a multi-element array of single values).

Callback methods

There are a variety of methods that are called when the request is prepared. You can subclass these methods to return your own values.

- *initRequest* - This method is called before the request is made. This is an opportunity to set the various values using the above internal methods based on the current options and settings. You should call parent::initRequest(). No return value is necessary

DatabaseDataRetriever

The DatabaseDataRetriever allows you to easily retrieve data from a relational database. This will utilize the Kurogo *database* classes and configuration styles to connect and retrieve data using standard SQL queries. You can either configure it with static configuration data as well implementing methods to determine the values dynamically.

There are several values that ultimately make up the request:

- The connection configuration. This will either use the site database, or you can configure a different database server
- The SQL query to execute. This should be a SELECT statement
- An array of parameters. These parameters are sent to the underlying PDO database engine and used as bound parameters for the request. You should use bound parameters to assist against SQL injection attacks.

Static Configuration

In addition to the standard *Configuring Database Connections*, when retrieving the data from a configuration file, the DatabaseDataRetriever will set the following values, if present:

- *SQL* - The SQL query to execute.
- *PARAMETERS* - An array of parameters to use as bound parameters. Since parameters are defined in this fashion as arrays, they can only be used in PHP 5.3 (PHP 5.2 does not support array syntax in ini files)

Dynamic Values

In many cases the method cannot be determined until runtime. This is because it relies on data input from the user or other information. Subclasses of SOAPDataRetriever have the opportunity to set these values at a variety of times depending on when it is appropriate.

Internal Methods

These methods could be called in the *init*, *setOption* or *initRequest* methods to set the values.

- *setSQL($sql)* - Set the SQL to be called. Note that Kurogo does not provide any SQL validation, translation or abstraction for different database systems. This must be valid SQL for the target database engine.
- *setParameters($parameters)* - an array of parameters to be used as bound parameters (they typically replace ? in the query) The use of bound parameters is strongly recommended to protect against SQL injection attacks.

Callback methods

There are a variety of methods that are called when the request is prepared. You can subclass these methods to return your own values.

- *initRequest* - This method is called before the request is made. This is an opportunity to set the various values using the above internal methods based on the current options and settings. You should call parent::initRequest(). No return value is necessary

Data Parser

The Data Parser class is responsible for parsing the server response into a common format. This format is determined by the module and typically includes returning objects that conform to a particular interface.

There are several standard parsers included to handle common data types:

- *INIFileParser* - Parses .ini files into sections and key/value pairs
- *JSONDataParser* - Parses JSON data. Typically you would extend this to convert the data into objects.
- *PassthroughDataParser* - Does not parse the data. Useful in situations where the data needs to be returned as is.
- *RSSDataParser* - Parses RSS data into individual RSS items
- *XMLDataParser* - Parses XML data. This class needs to be extended to handle the elements appropriately.

Methods to Implement

- *parseData($data)* - This method is called to actually parse the data from the request. Note that for some retrievers (i.e. DatabaseRetrievers or LDAPRetrievers) the value will actually be a resource rather than a string. It should return the data parsed into it's appropriate value. The correct value type depends on the underlying data model. Typically this would be an object or an array of objects.

Internal Methods

- *setTotalItems($total)* - This should be called if the service provides a field to indicate the total number of items in the request (and the number of items actually returned has been limited). This assists with paging.
- *getContext($val)* - Retrieves a context value that was set by the retriever to assist with parsing the data. Examples include resource variables and state information.

Special Properties

To customize the behavior of the parser, you can override certain properties:

- *parseMode* - should be one of the class constants:

  - DataParser::PARSE_MODE_RESPONSE (default) - this parser will parse a DataResponse object. The DataParser class implements the parseResponse method that will call the parseData method to parse the contents of the response
  - DataParser::PARSE_MODE_FILE - this parser expects to receive a file name that points to a file to parse. This is useful for parsers that utilize functions that act on files rather than strings.

Initialization Values

Retrievers typically are sent the values of a configuration file. There are several values that used by all retrievers to configure the behavior of a particular instance:

- *HALT_ON_PARSE_ERRORS* - If false then the parser should catch any exceptions while parsing and continue.

Certain subclasses have additional configuration values.

Writing A Parser

```
<?php

class MyDataParser extends DataParser
{

    protected function parseData($data) {

        // Take the data and parse it into objects or an array of
objects

        return $parsedData;
    }
}
```

Data Response

The data response object (and its subclasses) are the objects returned by *DataRetrievers*. These objects encapsulate the request and response so information such as status codes, headers and other data can be organized.

As a developer you would have 2 uses for this class:

1. Extracting the contents of the response if you are writing a parser or similar class that requires handling a reponse
2. Creating a response when writing a custom retriever. Typically this is not needed if you are simply subclassing one of the included retriever classes.

Public Methods

- *getResponse()* - Returns the string response
- *getResponseError()* - Returns any error message
- *getCode()* - Returns a status code (In HTTPDataReponse this will be the HTTP code)

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 Twitter feed.

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 subclass of the *WebModule* class and at very least must contain a property named *id* that indicates the module id and a implementation of *initializeForPage()*

Steps

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

```
<?php

class TwitterWebModule extends WebModule
{
  protected $id='twitter';
```

```
    protected function initializeForPage() {
    }
}
```

- Create *SITE_DIR/app/modules/twitter/templates/index.tpl* with the following contents:

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

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

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

- Create *SITE_DIR/config/twitter/module.ini* with the following contents:

```
[module]
title="Twitter"
disabled = 0
protected = 0
search = 0
secure = 0
```

- Create a 56x56 PNG image named *title-twitter.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 */twitter* on your server. At this point, it has no useful functionality. These steps can be repeated for any future module you wish to create.

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 Twitter Data API as the source of our data. It returns data in JSON format.

Creating a Data Library

We will utilize the *DataRetriever* class to deal with the retrieval, parsing and caching of this data. Our first step is to create a subclass of *URLDataRetriever* and add the appropriate methods to return tweets from a particular users' public timeline. It will have a public method called *tweets* that will accept a username show in Twitter and then return an array of tweets based on that query.

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

Steps

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

```php
<?php

class TwitterDataRetriever extends URLDataRetriever
{
    protected $DEFAULT_PARSER_CLASS =
'JSONDataParser';

```

```
    public function tweets($user) {
        $this->setBaseURL('http://api.twitter.com/1/statuses/user_timeline.json');
        $this->addParameter('screen_name', $user);
        $data = $this->getData();
        return $data;
    }
}
```

Some notes on this listing:

- 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. See *Data Retriever* for more information.
- The *tweets* method sets the base URL and adds filters. Filters work as parameters that are added to the URL's query string. The *getData* 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).
- 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 retriever, we can utilize it in our module. Here is an updated *TwitterWebModule.php*

```
<?php

class TwitterWebModule extends WebModule
{
  protected $id='twitter';
  protected function initializeForPage() {

      //instantiate controller
      $this->$controller =
DataRetriever::factory('TwitterDataRetriever',
array());

      switch ($this->page)
      {
         case 'index':
              $user = 'kurogofwk';

              //get the tweets
              $tweets = $this->controller-
>tweets($user);

```

```
                //prepare the list
                $tweetList = array();
                foreach ($tweets as $tweetData) {
                    $tweet = array(
                        'title'=> $tweetData['text'],
                        'subtitle'=> $tweetData['created_at']
                    );
                    $tweetList[] = $tweet;
                }

                //assign the list to the template
                $this->assign('tweetList', $tweetList);
                break;
            }
        }
    }
```

Some notes on this listing:

- We instantiate our controller using the DataRetriever 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 will add logic for other pages shortly

- Then we use our tweets method and send it a string value. The method returns an array of tweets
- *Note:* When debugging the contents of a web service call, it can be useful to output its contents. You may find it useful to use the *KurogoDebug::debug($var, $halt=false)* method. The first parameter is a variable (typically an array or object), the second parameter is a boolean. If true, then script execution will stop. It will also contain a function trace to assist in code path debugging.
- We iterate through the array and assign values for each item. We're using the text value for the item title and the post date as our subtitle. In this example, the value is not formatted, but you could use the DateFormatter class to format the value.
- We then assign the tweetList array to the template

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

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



{include file="findInclude:common/templates/results.tpl"
results=$tweetList}



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

- We include the results.tpl file which expects an array of items set in the results variable.

You should now be able to view the list of tweets by going to */twitter*.

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 tweet. 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 *TwitterWebModule.php*:

```php
<?php



//prepare the list

foreach ($tweets as $tweetData) {

    $tweet = array(

        'title'=> $tweetData['text'],

        'subtitle'=> $tweetData['created_at'],

        'url'=> $this->buildBreadcrumbURL('detail',
array('id'=>$tweetData['id_str']))

    );

    $tweetList[] = $tweet;

}
```

- 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: *id* which we will pass the id of our tweet. We will use that parameter when loading the detail.

Retrieving an Entry

We will now need to update the *TwitterDataRetriever* 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 Twitter's API to retrieve a specific item.

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

```
<?php



// retrieves a tweet based on its id

public function getItem($id) {

    $this-
>setBaseURL('http://api.twitter.com/1/statuses/show.json');

    $this->addParameter('id', $id);

    $data = $this->getData();

    return $data;

}
```

- We set the base url to the show JSON method
- The getData() method will retrieve the data and return it parsed

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 *TwitterWebModule.php*:

```
<?php

    case 'detail':

        $id = $this->getArg('id');

        if ($tweet = $this->controller->getItem($id)) {

            $this->assign('tweetText', $tweet['text']);
            $this->assign('tweetPost', $tweet['created_at']);

        } else {

            $this->redirectTo('index');

        }

        break;
```

- Use the *getArg()* method to retrieve the *id* 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 a tweet for that id.
- We then assign a few variables to use in our template.
- If the tweet 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"}

<div class="focal">

<p>{$tweetText}</p>



<p class="smallprint">{$tweetPost}</p>

</div>

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

- This template uses simple variable substitution to create a few elements for the tweet text and post date.

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.

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/home/module.ini* file.

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

This will create a link to the twitter module with a label that says Twitter.

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/twitter/* with the following contents:

```
[index]

pageTitle = "Twitter"



[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)

Module Configuration

The first implementation used a fixed string to search for twitter. 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/twitter/* with the following contents:

```
title = "Twitter"

disabled = 0

protected = 0

search = 0

secure = 0

TWITTER_USER = "kurogofwk"
```

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 *TWITTER_USER* parameter that will hold the query to use for the list.

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

```
<?php
```

```
$user = $this->getModuleVar('TWITTER_USER');

$items = $controller->tweets($user);
```

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

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

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

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* or *linkForValue* methods. 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.

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
<?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*.

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*. The process is simple:

- Create a new folder in the *SITE_DIR/config* folder named with the url
- Create a module.ini file that has an *id* property that matches the name of the module you wish to load. (news, content, url, etc)

Here is an example of the *URL Module* for the address */fullweb*. This file would be located at *config/fullweb/module.ini*

```ini
[module]

id = "url"

title = "Full Website"
```

```
disabled = 0

protected = 0

search = 0

secure = 0

url = "http://example.com"
```

This module would use the same logic and templates as the module indicated by *id*, but it would use its own set of configuration files, in this case in the*SITE_DIR/config/fullweb* folder.

Database Access

There are several situations where utilizing a database may be necessary. Kurogo has created a standard database access abstraction system that utilizes the PDO php library. This section outlines both the configuration of database connections as well as instructions on how to utilize database calls when writing modules or libraries.

Supported Database Backends

Kurogo includes support for the following database backends. Keep in mind that utilizing these systems requires the appropriate PHP extension. Installing and configuring the database server and the PHP extensions is beyond the scope of this document.

- MySQL
- SQLite
- PostgreSQL
- Microsoft SQLServer. Note that support for SQL Server is limited to servers running Microsoft Windows and requires the Microsoft Library found at:http://msdn.microsoft.com/en-us/sqlserver/ff657782.aspx

An important note about SQL statements

The Kurogo database library is merely a connection abstraction library. It is not a SQL abstraction library. Therefore it is important to make sure that different backend systems do not support the same SQL language and dialects and you must write your statements accordingly. See *Using the database access library* for information on targeting back ends if the SQL statements must be different.

Kurogo Features that use Database connections

- The internal device detection system uses an included SQLite database to store data on browsers
- The *Statistics Modules* uses a database to index the access logs and prepare the reports. If you are in a load balanced environment, you would want to use a centralized database.
- You can optionally store session data in a database rather than on the server file system by using the SessionDB class.

- The DatabasePeopleController uses a database to get directory information (rather than an LDAP server)
- The DatabaseAuthentication authority uses a database for authentication

Only the features that you require and configure would require a database. It is possible to run Kurogo without using a database.

Configuring Database Connections

There is a primary set of database connection settings in the *database* section of the *site.ini* file. All database connections (with the exception of the internal device detection database) will use that series of settings by default. You can also override those settings by providing the appropriate values in each particular service's configuration. Regardless of where the settings are set, the keys and values are similar.

- *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 type of database system. Current values include:
  - mysql
  - sqlite
  - pgsql
  - mssql

The following values are valid for host based systems (mysql, pgsql and mssql)

- *DB_HOST* - The hostname/ip address of the database server.
- *DB_USER* - The username needed to connect to the server
- *DB_PASS* - The password needed to connect to the server
- *DB_DBNAME* - The database where the tables are located
- *DB_PORT* - The port used to connect to the server. If empty it will use the default (mysql and pgsql only)

The following values are valid for file based systems (sqlite)

- *DB_FILE* - The location of the database file. Use the DATA_DIR constant to save the file in the site data dir. This folder is well suited for these files.

Using the database access library

If you are writing a module that requires database access, you can utilize the database classes to simplify your code and use the same database connections easily.

- Include the db package: *Kurogo::includePackage('db');*
- Instantiate a db object with arguments, the arguments should be an associative array that contains the appropriate configuration parameters. If the argument is blank then it will use the default settings found in the *database* section of site.ini
- Use the *query($sql, $arguments)* method to execute a query. The arguments array is sent as prepared statement bound parameters. In order to prevent SQL injection attacks you should utilize bound parameters rather than including values in the SQL statement itself
- The query method will return a PDOStatement object. You can use the *fetch* method to return row data.
- The *lastInsertID* method of the db object will return the ID of the last inserted row.

```
<?php

Kurogo::includePackage('db');

class MyClass
{
    function myMethod() {

        $db = new db();

        $sql = "SELECT * FROM sometable where somefield=? and 
someotherfield=?";
        $result = $db->query($sql, array('value1','value2'));
        while ($row = $result->fetch()) {
            // do something
        }
    }
}
```

Authentication

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.

Setting up 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 allow you to logically organize users. Some authorities are designed 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.

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.

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.

- Flat File Authentication
- Database Authentication
- LDAP Authentication
- Active Directory Authentication
- CAS Authentication
- Shibboleth Authentication
- Google Apps Authentication
- Google Authentication
- Facebook Authentication
- Twitter Authentication

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.

Configuration

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

- *PASSWD_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"*
- *PASSWD_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"*

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

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

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.

Database Authentication

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.

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:

- *DB_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
- *DB_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.
- *DB_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:

- *DB_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
- *DB_USER_PASSWORD_FIELD* (password) -stores a hashed value of the user's password. See DB_USER_PASSWORD_HASH for possible hashing algorithms (Default is md5)
- *DB_USER_EMAIL_FIELD* (email) - stores the email in the user table
- *DB_USER_FIRSTNAME_FIELD* (empty) - stores the first name of user. Won't be used unless it is specified
- *DB_USER_LASTNAME_FIELD* (empty) - stores the last name of user. Won't be used unless it is specified
- *DB_USER_FULLNAME_FIELD* (empty) - stores the full name of user. Won't be used unless it is specified
- *DB_GROUP_GROUPNAME_FIELD* (group) - stores the short name of the group in the group table
- *DB_GROUP_GID_FIELD* - (gid) - stores the group id of the group in the group table. Should be numerical
- *DB_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
- *DB_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
- *DB_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

- *DB_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

- *DB_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. You can also use the *hmac_* prefix to use the hmac signing method (i.e. hmac_sha1). This requires setting the*DB_USER_PASSWORD_KEY* value.

- *DB_USER_PASSWORD_KEY* (empty) - Necessary if you use the more secure *hmac* variant hashing algorithms. This uses a shared key to sign the value using*hash_hmac*
- *DB_USER_PASSWORD_SALT_BEFORE* (empty) - If present this string will be *prepended* to any string as a salt value before hashing. This is useful if you are using fixed salts.
- *DB_USER_PASSWORD_SALT_AFTER* (empty) - If present this string will be *appended* to any string as a salt value before hashing. This is useful if you are using fixed salts.
- *DB_USER_PASSWORD_SALT_FIELD_BEFORE* (empty) - If present the value of this field for the user will be *prepended* to any string as a salt value before hashing. This is useful if you are using variable salts.
- *DB_USER_PASSWORD_SALT_FIELD_AFTER* (empty) - If present the value of this field for the user will be *appended* to any string as a salt value before hashing. This is useful if you are using variable salts.

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*DB_USER_USERID_FIELD* or *DB_USER_EMAIL_FIELD* matching the login typed in. If that record is found it will see if the value in the *DB_USER_PASSWORD_FIELD*matches the hashed value of the password typed in (using the *DB_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 *DB_USER_TABLE* and look for a record with the *DB_USER_USERID_FIELD* or *DB_USER_EMAIL_FIELD* matching the value requested. If found, a user object is populated with *DB_USER_USERID_FIELD*, *DB_USER_EMAIL_FIELD* and *DB_USER_FIRSTNAME_FIELD*,*DB_USER_LASTNAME_FIELD* and*DB_USER_FULLNAME_FIELD* if present.

Group Lookup

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

Group Membership Lookup

Group membership is queried in the *DB_GROUPMEMBERS_TABLE*. The getMembers() method will construct an array of user objects using the*DB_GROUPMEMBER_USER_FIELD*. All users that match the *DB_GROUPMEMBER_GROUP_FIELD* will be returned (using the value of the groups*DB_GROUP_GROUPMEMBER_PROPERTY*, i.e. gid or short name) The user objects are created from the authority referenced by the*DB_GROUPMEMBER_AUTHORITY_FIELD*. If there is no authority field it will use the same authority as the group (i.e. it will use the *DB_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)

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(50), lastname varchar(50));

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.

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.

Configuration

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

- *LDAP_HOST* - Required. The dns/ip address of the LDAP server.
- *LDAP_PORT* - Optional (Default 389) The port to connect. Use 636 for SSL connections (recommended if available)
- *LDAP_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.
- *LDAP_USER_UID_FIELD* - Optional (default uid) - Specifies the ldap field to use that stores the user’s login id
- *LDAP_USER_EMAIL_FIELD* - Optional (default mail) - Specifies the ldap field to use that stores the user’s email address
- *LDAP_USER_FIRSTNAME_FIELD* - Optional (default givenname) - Specifies the ldap field to use that stores the user’s first name
- *LDAP_USER_LASTNAME_FIELD* - Optional (default sn) - Specifies the ldap field to use that stores the user’s last name
- *LDAP_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.
- *LDAP_GROUP_GROUPNAME_FIELD* - Optional (default cn). Specifies the ldap field to use that stores the group short name.
- *LDAP_GROUP_GID_FIELD* - Optional (default gid), Specifies the ldap field to use that stores the numerical group id.
- *LDAP_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.
- *LDAP_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.

- *LDAP_ADMIN_PASSWORD* - The password for the *LDAP_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 *LDAP_PORT* to port 636)

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 *LDAP_USER_UID_FIELD* or*LDAP_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 *LDAP_PORT* to 636) to protect the security of passwords.

User Lookup

Users are looked by executing an LDAP search beginning at *LDAP_USER_SEARCH_BASE* in either the *LDAP_USER_UID_FIELD* or *LDAP_USER_EMAIL_FIELD*. If found, a user object is populated
with *LDAP_USER_USERID_FIELD*, *LDAP_USER_EMAIL_FIELD* and*LDAP_USER_FIRSTNAME_FIELD*,*LDAP_USER_LASTNAME_FIELD* and *LDAP_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 *LDAP_GROUP_SEARCH_BASE* in *LDAP_GROUP_GROUPNAME_FIELD*. If found A group object is populated with *LDAP_GROUP_GROUPNAME_FIELD* and *LDAP_GROUP_GID_FIELD*

Group Membership Lookup

Group membership is looked up by executing an LDAP search for
the *LDAP_GROUP_GROUPNAME_FIELD* and retrieving
the *LDAP_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.

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.

Configuration

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

- *LDAP_HOST* - Required. The DNS address of your active directory domain. You could include a specific domain controller if desired.
- *LDAP_PORT* - Optional (Default 389) The port to connect. Use 636 for SSL connections (recommended if available)
- *LDAP_SEARCH_BASE* - The LDAP search base of your active directory domain.
- *LDAP_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.
- *LDAP_ADMIN_PASSWORD* - The password for the *LDAP_ADMIN_DN* account.

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

CAS Authentication

The CAS authority allows you to authenticate users via a Central Authentication Service (CAS). Because it is a limited access system, it is well suited to control access to modules to people in your organization.

Information about the CAS server and the CAS Protocol is available from jasig.org.

The CAS Protocol is a web-based single-sign-on protocol. Instead of entering authentication credentials directly into a login form, the user gets redirected to the CAS login page. Then they must authenticate there and will be subsequently redirected to your application. Your application has no access to the user's password.

Configuration

The CASAuthentication authority relies on the phpCAS library to handle communication with the CAS server. If you don't already have phpCAS, download the latest release.

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

- *USER_LOGIN* - Should be set to *LINK*
- *CAS_PHPCAS_PATH* - If phpCAS is not in your include-path, specify the filesystem path that contains CAS.php.
- *CAS_PROTOCOL* - The protocol to use. Should be equivalent to one of the phpCAS constants, e.g. `"2.0"`. *CAS_VERSION_1_0* => `"1.0"`, *CAS_VERSION_2_0*=> `"2.0"`, *SAML_VERSION_1_1* => `"S1"`
- *CAS_HOST* - The host name of the CAS server, e.g. `"cas.example.edu"`
- *CAS_PORT* - The port the CAS server is listening on, e.g. `"443"`
- *CAS_PATH* - The path of the CAS application, e.g. `"/cas/"`
- *CAS_CA_CERT* - The filesystem path to a CA certificate that will be used to validate the authenticity of the CAS server, e.g.`"/etc/tls/pki/certs/my_ca_cert.crt"`. If empty, no certificate validation will be performed (not recommended for production).

If you wish to authenticate as a proxy (which will allow Kurogo to may proxy-authenticated requests to back-end data sources) then you can provide the following attributes to configure proxy authentication.

- *CAS_PROXY_INIT* - Set to `1` to initialize as a proxy.

- *CAS_PROXY_TICKET_PATH* - The path in which phpCAS should look for proxy-granting-tickets. E.g. `"/tmp"`
- *CAS_PROXY_FIXED_CALLBACK_URL* - If your Kurogo installation isn't running under SSL but shares a filesystem with an SSL-enabled host, then you can provide an alternate proxy-granting-ticket storage URL. E.g. `"https://host.domain.edu/storePGT.php"`

If your CAS server returns user attributes in a SAML-1.1 or CAS-2.0 response, you can provide these attributes to Kurogo to display full names and support group-based *Access Control and Authorization*.

- *ATTRA_EMAIL* - Attribute name for the user's email adress, e.g. `"email"`.
- *ATTRA_FIRST_NAME* - Attribute name for the user's first name, e.g. `"givename"`.
- *ATTRA_LAST_NAME* - Attribute name for the user's last name, e.g. `"surname"`.
- *ATTRA_FULL_NAME* - Attribute name for the user's full name, e.g. `"displayname"`.
- *ATTRA_MEMBER_OF* - Attribute name for the user's groups, e.g. `"memberof"`.

```
[cas]

CONTROLLER_CLASS        = "CASAuthentication"

TITLE                   = "Central Authentication Service (CAS)"

USER_LOGIN              = "LINK"

CAS_PROTOCOL            = "2.0"

CAS_HOST                = "login.example.edu"

CAS_PORT                = 443

CAS_PATH                = "/cas/"

CAS_CA_CERT             = ""

ATTRA_EMAIL             = "EMail"

ATTRA_FIRST_NAME        = "FirstName"

ATTRA_LAST_NAME         = "LastName"
```

```
ATTRA_FULL_NAME          = "DisplayName"

ATTRA_MEMBER_OF          = "MemberOf"
```

Shibboleth Authentication

The Shibboleth authority allows you to authenticate users using the Shibboleth federated identity system. In a properly configured environment it can provide a single sign on experience for users without exposing credentials to the web server.

Requirements

The Shibboleth authority requires a functioning Shibboleth Service Provider (SP) on the Kurogo server. This typically includes the Shibboleth daemon and an apache module. Kurogo does not include nor attempt to simulate a Shibboleth SP. You should ensure your SP has been setup and is properly recognized by your IDP before attempting to configure Kurogo.

Installation

It is recommended that Shibboleth be configured to either protect the entire site (by adding the appropriate apache directives in the configuration) or by protecting the*/login* page by including a *<Location /login>* directive in the Apache configuration. A typical Apache configuration might look like this:

```
<Directory /var/www/> #this should be set to the root of your
Kurogo Installation

      AuthType shibboleth

      require shibboleth

</Directory>



<Location /login>

  AuthType shibboleth

  ShibRequestSetting requireSession 1

  require valid-user

</Location>
```

The *<Directory>* section ensures that Shibboleth will override anything in Kurogo (this is necessary to ensure that Shibboleth handles the /SSO and other endpoints it handles instead of Kurogo). The *<Location>* section ensures that a user is logged in to view the */login* page. If a user is not logged in they will be redirected to the IDP login page.

Configuration

Enabling the Shibboleth SP and adding the ShibbolethAuthentication authority is sufficient to authenticate users to Kurogo. There are several configuration options that will link shibboleth attributes to user data. All are optional:

- *SHIB_EMAIL_FIELD* - The field containing the user's email address
- *SHIB_FIRSTNAME_FIELD* - The field containing the user's first name
- *SHIB_LASTNAME_FIELD* - The field containing the user's last name
- *SHIB_FUlLNAME_FIELD* - The field containing the user's full name
- *SHIB_ATTRIBUTES[]* - An array of values that will be added as attributes to the user object. They can be retrieved using $user->getAttribute($key)

How it Works

The Shibboleth SP will redirect the user to the IDP login page. This process is handled by Shibboleth completely independent of Kurogo. The Kurogo server never receives any credentials. The IDP will redirect back to the Kurogo server and return to the login page. The ShibbolethAuthentication authority will inspect the PHP*$_SERVER* variable for the *REMOTE_USER* property and set the user's ID accordingly. It will also populate the user object with any additional attributes defined by the configuration.

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 form of *OpenID*. 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.

Configuration

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

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

```
[googleapps]

CONTROLLER_CLASS         = "GoogleAppsAuthentication"

TITLE                    = "Our Domain"

USER_LOGIN               = "LINK"

GOOGLEAPPS_DOMAIN        = "example.com"
```

How it Works

The Google Apps Authentication system uses OpenID by redirecting the user to an authentication page provided by Google. 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.

Accessing Domain data

If you have the need to access Google Apps Data on behalf of the user, then you will need to provide a consumer key, secret, and scope. These are values that identify the application to Google (and the user as a result) and identify the types of data you wish to access. If you have no need to access data, then you do not need to enter these values. You must first register your application's domain. The domain *must* match exactly the fully qualified domain name of the web application's host name. OAuth requires the registered domain and the callback URL (which is built using the domain of the kurogo instance) to match exactly. If you need to have an alternate domain name for development, you will need to register that domain as well. Once you have completed that process you will have an OAuth consumer key and secret you can use. This data will be included in the authentication declaration:

- *OAUTH_CONSUMER_KEY* - Consumer key provided by google
- *OAUTH_CONSUMER_SECRET* - Consumer secret provided by google
- *GOOGLE_SCOPE[]* - A repeatable list of scope URLs that indicate which services you wish to access. These are defined by Google and shown athttp://code.google.com/apis/gdata/faq.html#AuthScopes. Common examples used by Kurogo include:
  - http://www.google.com/calendar/feeds - User calendar data
  - https://apps-apis.google.com/a/feeds/calendar/resource/ - Calendar resources (Google Apps for Business/Edu only)
  - http://www.google.com/m8/feeds - Contacts
  - http://docs.google.com/feeds, http://spreadsheets.google.com/feeds, http://docs.googleusercontent.com - Google docs

```
[googleapps]

CONTROLLER_CLASS        = "GoogleAppsAuthentication"

TITLE                   = "Our Domain"

USER_LOGIN              = "LINK"

GOOGLEAPPS_DOMAIN       = "example.com"

OAUTH_CONSUMER_KEY      = "mobile.example.com"
```

```
OAUTH_CONSUMER_SECRET    = "abcdABCD1234ABCD"   ; provided by
google

GOOGLE_SCOPE[]           = "http://www.google.com/calendar/feeds"

GOOGLE_SCOPE[]           = "https://apps-
apis.google.com/a/feeds/calendar/resource/"

GOOGLE_SCOPE[]           = "http://www.google.com/m8/feeds"

GOOGLE_SCOPE[]           = "http://docs.google.com/feeds"

GOOGLE_SCOPE[]           = "http://spreadsheets.google.com/feeds"

GOOGLE_SCOPE[]           = "http://docs.googleusercontent.com"
```

You should only include scopes for data that you plan on accessing. If you have no need to access data, then you do not need to enter these values.

The Callback URL and development

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.

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.

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.

```
[google]

CONTROLLER_CLASS          = "GoogleAuthentication"

TITLE                     = "Google"

USER_LOGIN                = "LINK"
```

How it Works

The Google Authentication system uses OpenID by redirecting the user to an authentication page provided by Google. 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.

Accessing Google Data

If you have the need to access Google Data on behalf of the user, then you will need to provide a consumer key, secret, and scope. These are values that identify the application to Google (and the user as a result) and identify the types of data you wish to access. If you have no need to access data, then you do not need to enter these values.

The following values must be included if you wish to access Google Data (

- *OAUTH_CONSUMER_KEY* - Consumer key provided by google. You can provide “anonymous”
- *OAUTH_CONSUMER_SECRET* - Consumer secret provided by google
- *GOOGLE_SCOPE[]* - A repeatable list of scope URLs that indicate which services you wish to access. These are defined by Google and shown athttp://code.google.com/apis/gdata/faq.html#AuthScopes. Common examples used by Kurogo include:
    - http://www.google.com/calendar/feeds - User calendar data
    - https://apps-apis.google.com/a/feeds/calendar/resource/ - Calendar resources (Google Apps for Business/Edu only)
    - http://www.google.com/m8/feeds - Contacts
    - http://docs.google.com/feeds, http://spreadsheets.google.com/feeds, http://docs.googleusercontent.com - Google docs

```
[google]

CONTROLLER_CLASS        = "GoogleAuthentication"

TITLE                   = "Google"

USER_LOGIN              = "LINK"

OAUTH_CONSUMER_KEY      = "anonymous"

OAUTH_CONSUMER_SECRET   = "anonymous"

GOOGLE_SCOPE[]          = "http://www.google.com/calendar/feeds"

GOOGLE_SCOPE[]          = "https://apps-
apis.google.com/a/feeds/calendar/resource/"

GOOGLE_SCOPE[]          = "http://www.google.com/m8/feeds"

GOOGLE_SCOPE[]          = "http://docs.google.com/feeds"
```

```
GOOGLE_SCOPE[]          = "http://spreadsheets.google.com/feeds"

GOOGLE_SCOPE[]          = "http://docs.googleusercontent.com"
```

You should only include scopes for data that you plan on accessing. If you have no need to access data, then you do not need to enter these values.

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.

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.
- *FACEBOOK_API_KEY* - Set this to the *Application ID* provided by Facebook
- *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.

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.

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

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.
- *OAUTH_CONSUMER_KEY* - Set this to the *Consumer Key* provided by Twitter
- *OAUTH_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.

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.

Access Control and 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).

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 and groups. You can mix and match rules to tune the authorization to meet your site’s needs.

Access control lists are defined in either *SITE_DIR/config/acls.ini* (for site authorization) or *SITE_DIR/config/MODULE/acls.ini* (for module authorization). Each entry is entered as a numerically indexed section.

If the *site* has an access control list entry (in SITE_DIR/config/acls.ini), then it will be used for ALL modules. This would protect the entire site. If a *module* has an access control list entry (in SITE_DIR/config/MODULE/acls.ini) it will be protected and only users matching the acl rules will be granted access.

For 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.

Access control lists can also be edited in the Administration Console.

Syntax of Access Control Lists

Each ACL is a section in an acls.ini file. The first ACL will be section 0, the second will be section 1 and so on. Each section has a number of keys:

- *type*: Either U (for user access, i.e. who can use the module), or A (for admin access, who can administer the module).
- *action*: Either A (allow) or D (deny). Deny rules always take precedence.
- *scope*: Either U (user), G (group) or E (everyone, i.e. ALL users, including anonymous users),

- *authority*: The index of the authority. For user scope this can be blank and would match a user from any authority.
- *value*: The particular user/group to match. For user scope this can be "*" which would match all authenticated users.

To better illustrate the syntax, consider the following examples:

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

```
[0]

type = "U"

action = "A"

scope = "G"

authority = "ldap"

value = "admin"



[1]

type = "U"

action = "A"

scope = "G"

authority = "ad"

value = "domainadmins"



[2]

type = "U"
```

```
action = "D"

scope = "U"

authority = "ad"

value = "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.

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.

Credits and License

Kurogo is distributed under the *MIT License* by Modo Labs, Inc.

Modo Labs, Inc.

100 Cambridgepark Drive, Suite 302

Cambridge, MA 02140

kurogo@modolabs.com

Credits

Modo Labs Team

Developers

- Pete Akins (Lead Developer, Mobile Web)
- Muhammad Amjad
- Dario Baldoni
- Alexis Ellwood
- Ian Gavalakis
- Wesley Hirsch
- Sonya Huang (Lead Developer, iOS)
- Jim Kang
- Edward Liu
- Newstar Liu
- Ebrahim Kobeissi
- David Ormsbee
- Brian Patt (Lead Developer, Android)
- Brent Wang
- Jeffrey You
- Saturn Zhang

Designers

- James Choi
- Kate Dobroth
- Eric Kim (Lead Designer)

- Hoon Lee
- Ilona Moreland
- Brian Tepfenhart

Project Management & QA

- Yuki Nagatoshi
- Marshall Vale
- Andrew Yu

Special Thanks

- Justin Anderson
- Massachusetts Institute of Technology

The Kurogo Framework is based on the MIT Mobile Framework, and incorporates contributions from:

- Harvard University
- Doug Beck (University of Central Florida)
- Jared Lang (University of Central Florida)
- Chris Conover (University of Central Florida)
- University of Vermont
- Adam Franco (Middlebury College)

Third Party Libraries (Mobile Web)

Minify

License: BSD Copyright: Steve Clay, Ryan Grove Link: http://code.google.com/p/minify/

Smarty

License: LGPL Copyright: Monte Ohrt, Uwe Tews Link: http://www.smarty.net/

Third Party Libraries (iOS)

Facebook iOS SDK

License: Apache 2.0

Copyright: Facebook

Link: https://github.com/facebook/facebook-ios-sdk

JSON

License: BSD

Copyright: Stig Brautaset

Link: http://stig.github.com/json-framework

Third Party Libraries (Android)

License

List of Languages Supported by Kurogo

The following is a list of language codes accepted by Kurogo for the LANGUAGES value in *SITE_DIR/config/site.ini* Note that this does not meant that a translated string table exists.

Code	**Language Name**	**Native Name**
af_ZA	Afrikaans	Afrikaans
am_ET	Amheric	አማርኛ
be_BY	Belarusian	Беларуская
bg_BG	Bulgarian	български език
ca_ES	Catalan	Català
cs_CZ	Czech	čeština
da_DK	Danie	Dansk
de_AT	German (Austria)	Deutsch (Österreich)
de_CH	German (Swiss)	Deutsch (Schweiz)
de_DE	German	Deutsch (Deutschland)
el_GR	Greek	Ελληνικά
en_AU	English (Australia)	English (Australia)
en_CA	English (Canada)	English (Canada)
en_GB	English (United Kingdom)	English (United Kingdom)
en_IE	English (Ireland)	English (Ireland)
en_NZ	English (New Zealand)	English (New Zealand)
en_US	English (United States)	English (United States)
es_ES	Spanish	Español
et_EE	Estonian	Eesti
eu_ES	Basque	Euskara
fi_FI	Finnish	Suomi

Code	**Language Name**	**Native Name**
fr_BE	French (Belgium)	Français (Belgique)
fr_CA	French (Canada)	Français (Canada)
fr_CH	Frensh (Swiss)	Français (Suisse)
fr_FR	French	Français (France)
he_IL	Hebrew	עברית
hr_HR	Croatian	Hrvatski
hu_HU	Hungarian	Magyar
hy_AM	Armenian	Հայերեն
is_IS	Icelandic	Íslenska
it_CH	Italian (Swiss)	Italiano (Svizzera)
it_IT	Italian	Italiano (Italia)
ja_JP	Japanese	日本語
kk_KZ	Kazakh	Қазақ тілі
ko_KR	Korean	한국어
lt_LT	Lithuanian	Lietuvių
nl_BE	Dutch (Belgium)	Vlaams
nl_NL	Dutch (Netherlands)	Nederlands
no_NO	Norwegian	Norsk
pl_PL	Polish	Polski
pt_BR	Portuguese (Brazil)	Português (Brasil)
pt_PT	Portuguese	Português
ro_RO	Romanian	Română
ru_RU	Russian	Русский
sk_SK	Slovak	Slovenčina
sl_SI	Slovene	Slovenščina

Code	Language Name	Native Name
sr_YU	Serbian	Српски
sv_SE	Swedish	Svenska
tr_TR	Turkish	Türkçe
uk_UA	Ukrainian	Yкраїнська
zh_CN	Chinese (Simplified)	简体中文
zh_TW	Chinese (Traditional)	繁體中文