Kurogo Documentation

Release 1.2

Modo Labs

July 19, 2011

CONTENTS

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

At a high level, the Kurogo Framework includes:

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

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

CHAPTER

ONE

OVERVIEW

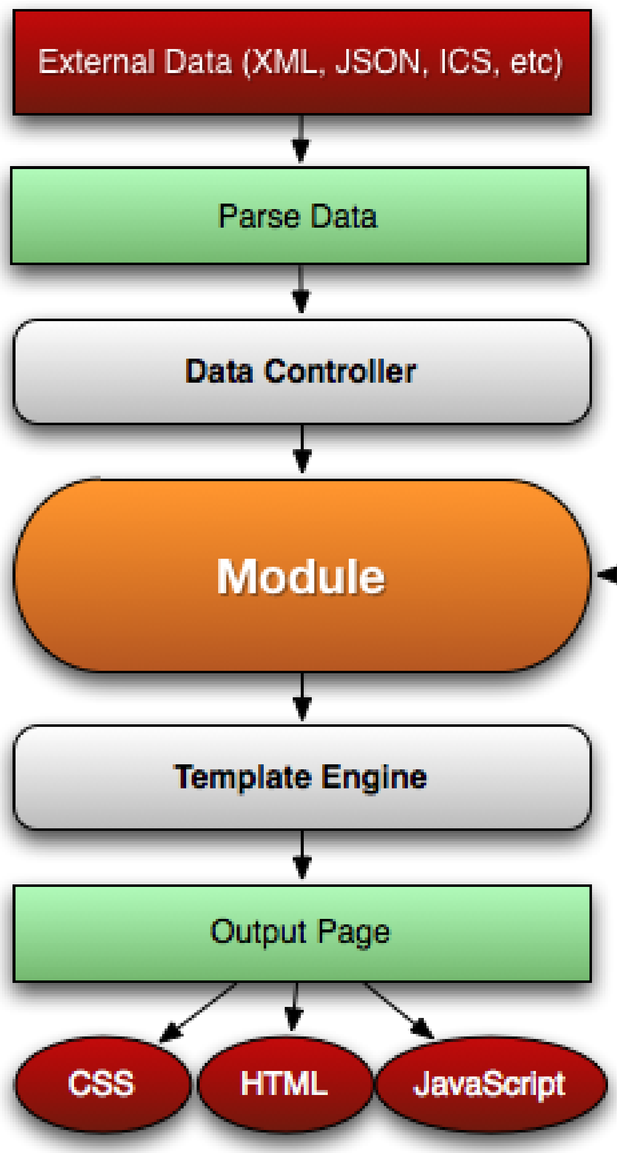
Kurogo is a PHP framework for delivering high quality, data driven customizable content to a wide range of mobile devices. Its strengths lie in the customizable system that allows you to adapt content from a variety of sources, and easily present that to a range of mobile devices from feature phones, 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.

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

1.1 Modules

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

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



1.2 Device Detection

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

Kurogo classifies devices by *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.

1.3 Customization

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

1.3.1 Data Customization

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

1.3.2 Code Customization

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

1.3.3 Output Customization

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

CHAPTER

TWO

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.

2.1 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

2.2 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

2.3 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

2.4 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

CHAPTER

THREE

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.

3.1 Forking and Managing your repository

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

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

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

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

3.2 Creating your site

Because your own project will contain elements that are not part of the master project (i.e. your own site's 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.

3.3 Submitting your changes

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

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

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

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

CHAPTER

FOUR

SETUP AND INSTALLATION

Kurogo is a PHP web application. As such it must be installed on a web server. In version 1.2, there are 2 supported web servers.

4.1 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.
    - Virtual Directories not supported, must be the site root
- PHP 5.2 or higher with the following extensions:
  - xml
  - dom
  - json
  - PDO (Used for *Database Access*)
  - mbstring
- Some PHP modules are optional depending on whether you need their backend functionality
  - LDAP

4.2 Installation

Please note that some of these instructions assume that you have basic system and web server administration knowledge. For more information please see the documentation for your system and 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 the server/port that you specified.

4.3 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. 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.

From the command line, this command would like 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

Note: Currently, Kurogo does NOT support being installed under an alias (Apache) or Virtual Folder (IIS). The method shown above does not work in Windows due to the lack of support for symbolic links.

CHAPTER

FIVE

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.

5.1 Case Sensitivity

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

5.2 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*

5.3 Creating the module class file

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

```
<?php

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

5.4 Creating the template file

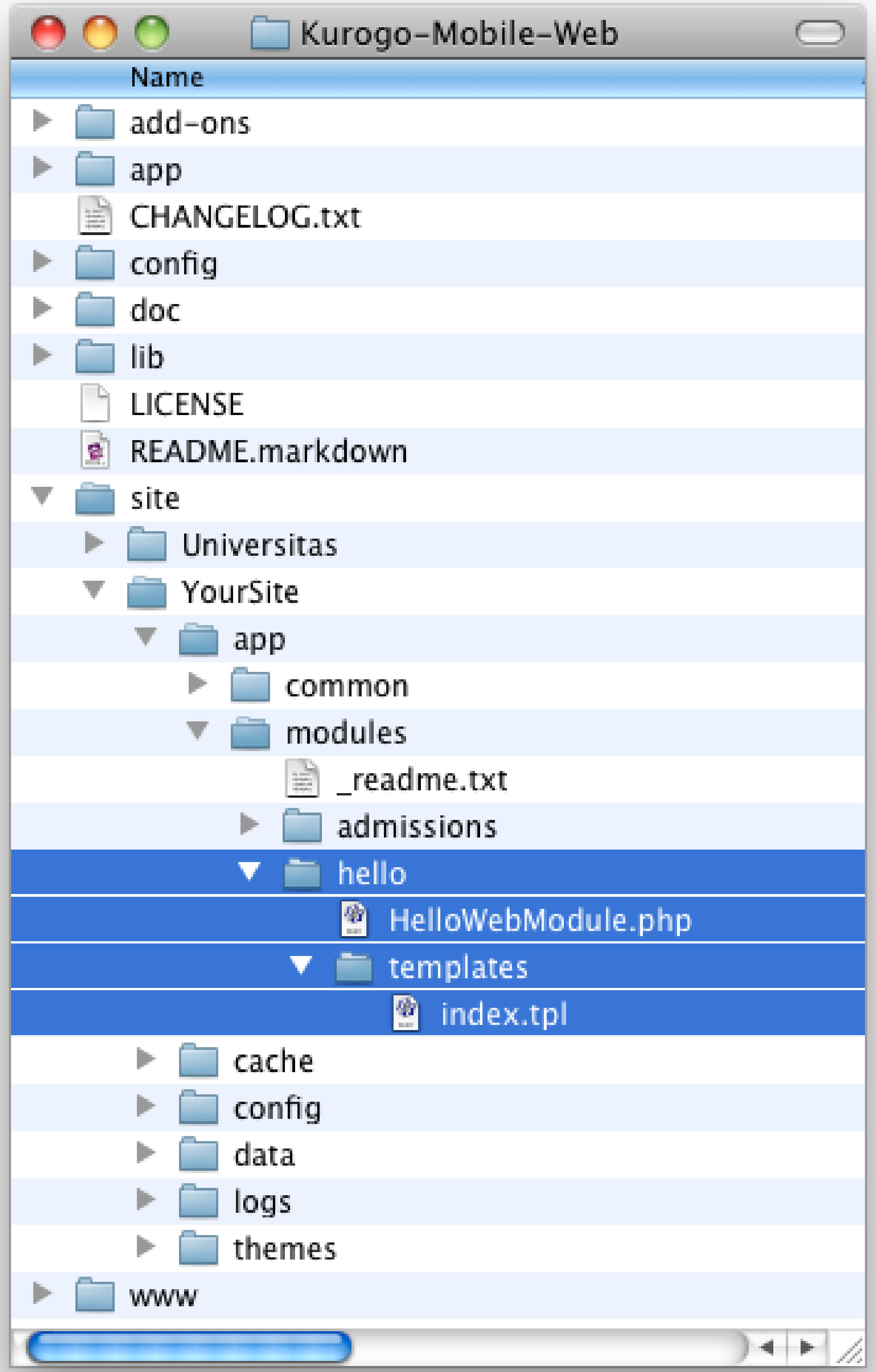
Inside the *hello/templates* directory create a file named *index.tpl* that contains the following contents:

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

<h1 class="focal">{$message}</h1>

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

Your folder structure should look similar to this:

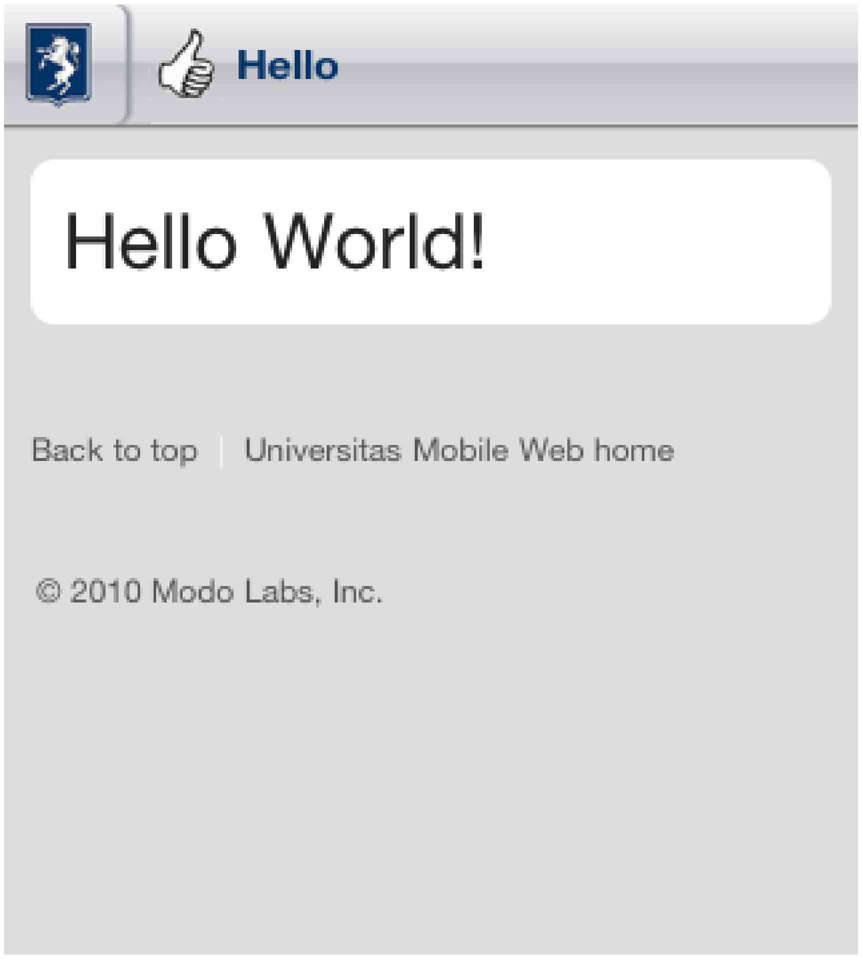

Kurogo-Mobile-Web
Name
add-ons
app
CHANGELOG.txt
config
doc
lib
LICENSE
README.markdown
site
Universitas
YourSite
app
common
modules
_readme.txt
admissions
hello
HelloWebModule.php
templates
index.tpl
cache
config
data
logs
themes
www

5.4.1 Creating the nav bar image

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

5.5 Viewing the module

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



Congratulations! You've just built a simple module.

CHAPTER

SIX

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.

6.1 Basic Layout

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

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

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

6.2 Case Sensitivity

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

6.3 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.

6.4 Libraries

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

6.4.1 Packages

In order to assist developers with including the proper class files, libraries can be grouped into *packages*. This allows you to include necessary functionality without worrying about which files to include in your module (use: *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
- 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

6.4.2 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.
- *PageViews* - A class to log and retrieve page view information for statistics
- *Validator* - A utility class to validate certain types of data

6.4.3 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
- *CoreAPIModule* - Class used to handle site wide API functions (API requests not assigned to a specific module)

6.4.4 External Data Retrieval

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

Included examples of DataControllers/Parsers include:

- *RSSDataController* - retrieves a feed of data in RSS/RDF or Atom formats. The corresponding *RSSDataParser* class takes the resulting data and builds a structure of items located in the feed. Also uses the *RSS* class.
- *CalendarDataController* - retrieves a feed of data in ICS format. The corresponding *ICSDataParser* class takes the resulting data and builds a structure of events in the feed. Also uses the *ICalendar* and *TimeRange* class. The *TrumbaCalendarDataController* is a specific subclass for feeds that utilize the Trumba calendar service.
- *PeopleController* - access directory/person data. Included implementations include the *LDAPPeopleController* and *DatabasePeopleController*. Note this is distinct from authenticating users.
- *HTMLDataController* - retrieves a remote HTML document and optionally extracts a specific HTML ID or element. It uses the *DOMDataParser*.

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

Other included Data Parsers:

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

See *Data Controller* for more information

6.4.5 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

6.4.6 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

6.4.7 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

6.4.8 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.

6.4.9 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

6.4.10 Other

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

6.5 Modules and Templates

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

6.5.1 Common

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

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

6.5.2 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)
- 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*

6.6 Site folder

The site folder contains a series of folders for each *site*. This allows each site to have specific configuration, design and custom code. At any given time there is only one **active site**. You can enable the active site in the *config/kurogo.ini* file found in the 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.

6.7 WWW Folder

The files and folders in the www folder represent the DocumentRoot, the base of the site. To keep the structure clean, all requests are routed through the *index.php* file (the exception is for paths and folders that already 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.

6.7.1 index.php

The index script is the main controller for the framework. All requests are handled through it using an .htaccess override and mod_rewrite 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*

CHAPTER

SEVEN

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.

7.1 Types of Device Detection Servers

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

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

See *Device Detection Configuration* for specific configuration values.

7.2 Data Format

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

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

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

7.3 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_FILE* - Location of device detection SQLite database if using internal detection. (typically located in LIB_DIR/deviceData.db)
- *MOBI_SERVICE_URL* - URL of device detection server if using external detection
  - (Development) https://modolabs-device-test.appspot.com/api/
  - (Production) https://modolabs-device.appspot.com/api/
- *MOBI_SERVICE_CACHE_LIFETIME* - Time (in seconds) to keep cached results from the external device detection service

7.3.1 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

CHAPTER

EIGHT

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.

8.1 Structure of .ini Files

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

8.1.1 Properties

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

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

8.1.2 Sections

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

```
[section]
```

8.1.3 Comments

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

```
; comment text
```

8.2 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.

8.2.1 Local Files

The framework supports overriding configuration files for local server customization. Unless the configuration value *CONFIG_IGNORE_LOCAL* (defined in *config/kurogo.ini*) is set to 1, the framework will also load files with a -local in the file name for each configuration file loaded. I.e. *SITE_DIR/config/site.ini* can be overridden with *SITE_DIR/config/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.

8.2.2 Configuration Mode

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

One use of this would be to create development and production versions of some of your configuration files. You can have *SITE_DIR/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

8.2.3 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.

8.3 Site Configuration

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

8.3.1 Error handling and debugging

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

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

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

8.3.2 Site settings

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

- *AUTODETECT_PHONE_NUMBERS* - Turn this off to prevent the auto detection of telephone numbers in content. This is primarily only supported in iOS devices at this time.

8.3.3 Modules

- *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

8.3.4 Analytics

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

8.3.5 Temp Directory

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

8.3.6 Themes

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

8.3.7 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.

8.3.8 Device Detection

See *Device Detection* for more details

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

8.3.9 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)

8.3.10 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.

8.3.11 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).

8.3.12 Log Files

- *API_LOG_FILE* - Location of the processed API log file
- *API_CURRENT_LOG_FILE* - Location of the active API log file
- *WEB_LOG_FILE* - Location of the processed page view log file
- *WEB_CURRENT_LOG_FILE* - Location of the active page view log file

- *LOG_DATE_FORMAT* - Date format for log files
- *LOG_DATE_PATTERN* - regex pattern of log dates, should match output from LOG_DATE_FORMAT

8.4 Module Visibility and protection

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

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

8.4.1 Permanently disabling modules

If there are modules that you will not use in your site at all, you can completely disable them by editing the *SITE_DIR/config/site.ini* file. In the *[disabled_modules]* section you can add a list of modules that should be disabled. By adding the disabled flag in this location, you can completely remove the configuration folder and it will not get recreated.

```
[disabled_modules]
admin = 1 ; disable the admin module
stats = 1 ; disable the stats module
```

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

8.4.2 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.

8.5 Home Screen

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

8.6 Strings

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

- *SITE_NAME* - The name of the site. Used in the footer and other places.

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

8.7 Administration Module

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

CHAPTER

NINE

HANDLING REQUESTS

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

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

9.1 Path patterns

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

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

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

All other requests will attempt to load a module based on the first path component of the request. The contents before the first "/" will refer the *id* of the module, the contents after the slash will be the page to load. If there is no page specified, the *index* page will be loaded. The script attempts to instantiate a module with the corresponding *id* using the *WebModule::factory* method (see *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.

9.2 Pagetype & Platform Files

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

- pagetype - The basic type of device, is *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.

CHAPTER

TEN

WRITING MODULES

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

10.1 The WebModule Object

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

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

10.1.1 Instantiation and execution

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. If there is no page indicated, then the page will be set to *index*.

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)
- Calls the *initializeForPage()* method. 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.

Finally the template based on the value of the *templatePage* property is displayed. Initially this is set to the page property, but can be overridden if necessary for more dynamic template display.

10.1.2 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.
- *configModule* (string) - This property only needs to be set if you are *copying a module*. It should be set to the url/config folder of the copied module.

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*

10.1.3 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

10.2 Methods to use

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

10.2.1 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*

10.2.2 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

10.2.3 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)

10.2.4 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/page/MODULE.ini* file, but you can use this method to set it dynamically.
- *setBreadcrumbTitle($title)* - Sets the breadcrumb title for this page. Normally this value comes from the *SITE_DIR/config/page/MODULE.ini* file, but you can use this method to set it dynamically.
- *setBreadcrumbLongTitle($title)* - Sets the breadcrumb long title for this page. Normally this value comes from the *SITE_DIR/config/page/MODULE.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

10.2.5 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.

10.2.6 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.

10.2.7 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

10.3 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"
```

10.4 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*.

10.4.1 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 home/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/home/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.

CHAPTER

ELEVEN

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.

11.1 Home Module

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

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

11.1.1 Home Screen Type

```
display_type = "springboard"
```

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

11.1.2 Module list and order

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

Each section has a list of values that represent the order of the modules and their 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.

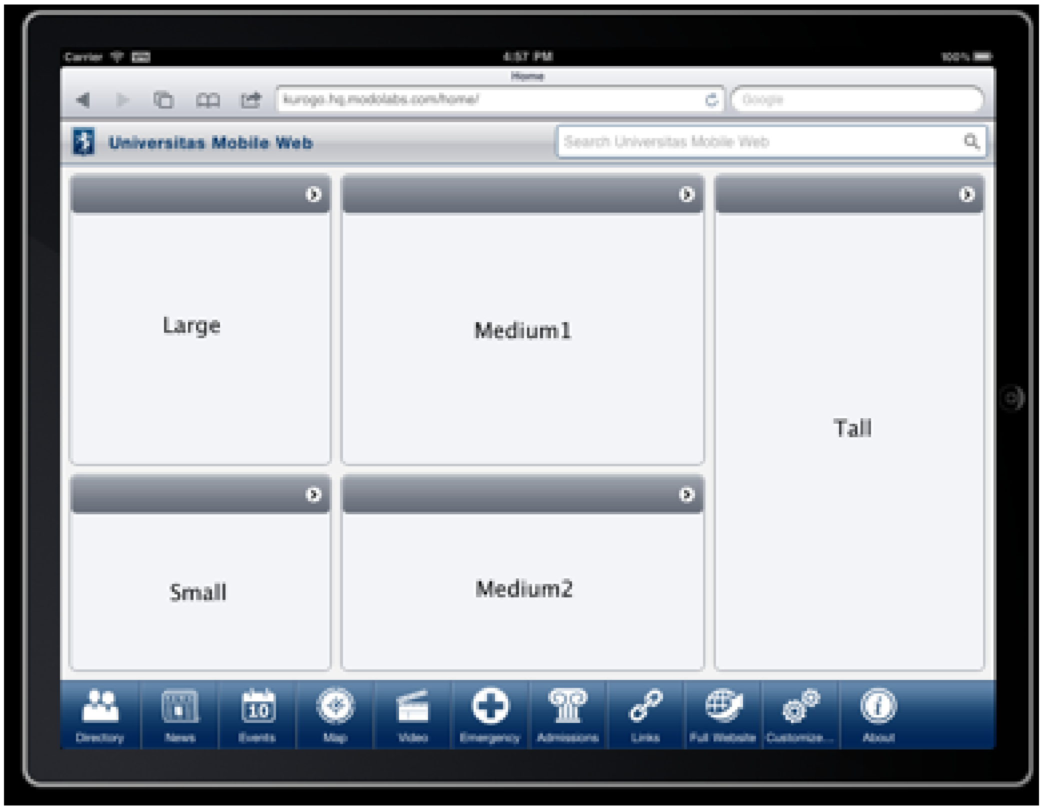
11.1.3 Icons

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.)

11.1.4 Tablet Interface

Kurogo includes a special layout for tablet devices. For version 1.0, it is used for Apple iPad 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.

11.1.5 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*.

11.2 Info Module

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

If you would prefer to have users see the home page and not the info module from a desktop browser, you can 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.

11.3 People Module

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

11.3.1 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.

- *CONTROLLER_CLASS* allows you to set a different class name for the controller. Current options include
  - LDAPPeopleController - uses a standard LDAP server. You can configure the various fields if your values differ from defaults
  - ADPeopleController - a subclass of the LDAP controller that has preconfigured mappings for Active Directory
  - DatabasePeopleController - 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 LDAPPeopleController and ADPeopleController

- *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_PHONE_FIELD* (telephonenumber) - The attribute of the user's phone number

Options for DatabasePeopleController

The *DatabasePeopleController* 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.

11.3.2 Configuring the Detail Fields

Once you have configured the server settings, you need to configure the field mappings between your server and the detail view. 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.

11.3.3 Configuring the Fixed Entries

This module supports the ability to show a list of directory entries on the module index page. You can update the contents of this list by editing the *SITE_DIR/config/people/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"
```

11.4 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

11.4.1 Configuring the Sources

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

Within each feed you use the following options:

- *CONTROLLER_CLASS* - The DataController to use. Currently supported controllers include:
  - *YouTubeVideoControler*
  - *VimeoVideoController*
  - *BrightcoveVideoController*
- *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

BrightcoveVideoController

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

- token
- playerKey

- playerId

11.5 Calendar Module

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

11.5.1 Configuring the Calendar Feed

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

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

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

Optional values

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

11.5.2 Configuring the Detail Fields

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

- *label* - A text label for the field.
- *type* - 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.

11.5.3 Configuring the Initial Screen

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

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

11.5.4 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 CONTROLLER_CLASS to GoogleAppsCalendarListController
- Set AUTHORITY to the section name of your Google Apps Authority

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

```
[user_calendars]
CONTROLLER_CLASS="GoogleAppsCalendarListController"
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 CONTROLLER_CLASS to GoogleAppsCalendarListController
- Set AUTHORITY to the section name of your Google Apps Authority

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

```
[resources]
CONTROLLER_CLASS="GoogleAppsCalendarListController"
AUTHORITY="googleapps"
```

11.6 News Module

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

11.6.1 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

11.6.2 Configuring News Feeds

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

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

- The TITLE value is a label used to name your feed. It will be used in the drop down list to select the current feed
- The BASE_URL is set to the url of your News feed. It can be either a static file or a web service.

Optional values

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

11.7 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.

11.7.1 Configuring the Server Connection

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

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

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

Configure Contacts List

Configure contacts list to connect to the drupal emergency module:

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

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

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

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

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

11.7.2 Using Drupal Emergency Module

Installation

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

- In order to install this module you must first install the drupal CCK (Content Creation Kit) module and the drupal Views module

- copy *add-ons/drupal-modules/emergency* into the *sites/all/modules/* directory
- In the drupal administration panel go to modules then select the "Emergency Info" module and click "save configurations".

Usage

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

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

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

11.8 Map Module

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

11.8.1 Configuring Feed Groups

Feed groups are a way to organize map sources logically (i.e. by campus/location)

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

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

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

11.8.2 Base Map Types

The map module currently supports five types of base maps.

JavaScript based maps (compliant and tablet only)

- Google Maps
- ArcGIS JavaScript

Static image based maps

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

11.8.3 Map Data Sources

The map module currently supports three types of geo data formats:

- KML
- ArcGIS Server
- Shapefiles

Any of these sources may be used with any of the base maps listed above.

11.8.4 Configuring Map Data Feeds

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

The feed configuration file is in *SITE_DIR/config/map/feeds-CAMPUS.ini* (where CAMPUS is the name of the campus section in the feedgroups.ini file). Each feed has the following fields:

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

- If your insitution has multiple campuses, the *CAMPUS* field specifies which campus this data feed belongs to.
- If you want your data feed to be included in search results, but do not wish to make it a browseable category, you may set the optional *HIDDEN* value to 1

11.8.5 Example Configurations

Data Sources

KML Data Source

```
TITLE              = "My Placemarks"
BASE_URL           = "http://example.com/feed.kml"
CONTROLLER_CLASS   = KMLDataController
```

ArcGIS Server Data Source

When working with an ArcGIS Server instance with multiple layers, an *ARCGIS_LAYER_ID* may be specified. The default layer is 0.

```
TITLE                = "My ArcGIS Data"
BASE_URL             = "http://path/to/service/MapServer"
ARCGIS_LAYER_ID      = 2
CONTROLLER_CLASS     = ArcGISDataController
```

Shapefile Data Source

Currently, shapefiles must be saved locally on the machine. Due to the multi-file naming scheme of shapefiles, we require that the path be specified without an extension. In the following example, the shapefile from http://www.mass.gov/mgis/biketrails.htm was unarchived and the files biketrails_arc.shp, biketrails_arc.dbf, and biketrails_arc.prj were placed in the directory DATA_DIR"/biketrails". Only the .shp, .dbf, and .prj (if any) files are required.

```
TITLE              = "Massachusetts Bike Trails"
BASE_URL           = DATA_DIR"/biketrails/biketrails_arc"
CONTROLLER_CLASS   = ShapefileDataController
```

Static Base Maps

If a dynamic map is used for compliant/tablet devices, these configurations determine the appearance of maps on touch and basic devices. If no dynamic map is specified, they determine the appearance of maps on all devices.

Google Static Maps

This is the default base map. If you do not specify anything for *STATIC_MAP_CLASS*, this is equivalent to specifying GoogleStaticMap for basic and touch devices. Additionally, if *JS_MAP_CLASS* is also omitted, Google Static Maps will used for compliant/tablet devices.

Web Map Service (WMS)

```
STATIC_MAP_CLASS      = WMSStaticMap
STATIC_MAP_BASE_URL   = "http://path/to/WMS/server"
```

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

ArcGIS Exported Maps

```
STATIC_MAP_CLASS      = ArcGISStaticMap
STATIC_MAP_BASE_URL  = "http://path/to/service/MapServer"
```

Note that it is not possible to add annotations to exported images.

JavaScript Base Maps

If specified, these configurations determine the appearance of maps on tablet and compliant devices.

Google Maps

```
JS_MAP_CLASS       = GoogleJSMap
```

ArcGIS

```
JS_MAP_CLASS          = ArcGISJSMap
DYNAMIC_MAP_BASE_URL = "http://path/to/service/MapServer"
```

11.8.6 Configuring Map Search

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

11.9 Links Module

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

11.9.1 Configuration

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

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

Strings

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

Links

Links are 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"
```

11.10 Content Module

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

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

11.10.1 Abstract

The content module cannot be instantiated directly. It is an *abstract* module. In order to create a module that utilizes its features you should *copy the module*.

Configuring Content Feeds

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

- *TITLE* - The title of the feed. This is shown in the list and in the navigation bar
- *CONTENT_TYPE* - the type of content. Values include:
  - *html* - Static html text that is included in the *CONTENT_HTML* property
  - *html_url* - Fetch HTML content from the *BASE_URL* property.
  - *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

11.10.2 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.

11.11 URL Module

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

11.11.1 Abstract

The URL module cannot be instantiated directly. It is an *abstract* module. In order to create a module that utilizes its features you should *copy the module*.

Configuration

It has one configuration parameter:

url - A url to redirect to.

11.12 Customize Module

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

11.12.1 Configuration

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

11.13 About Module

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

11.13.1 Configuration

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

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

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

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

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

11.14 Login Module

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

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

11.14.1 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.

11.15 Stats Module

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

11.15.1 Configuration

It is important that the *[database]* section of the *site.ini* file is properly configured and a database connection is available. The statistics module relies on a working database to index its data.

CHAPTER

TWELVE

THE KUROGO OBJECT

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

12.1 Static Class Methods

- *Kurogo::sharedInstance()* - Returns the shared Kurogo singleton object. This is typically not necessary to use since all publically 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*

12.2 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*

12.2.1 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

CHAPTER

THIRTEEN

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.

13.1 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
<?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);
      }
  }
}
```

13.2 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.

CHAPTER

FOURTEEN

TEMPLATES

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

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

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

14.1 Variables and Modifiers

Including values from variables is a two step process:

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

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

14.2 Including and Extending Templates

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

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

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

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

You can also assign variables when including the template:

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

This will assign the value 42 to the variable thing

14.2.1 Blocks

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

Consider a base template template.tpl:

```
This template is the base template.

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

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

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

An example for template-compliant.tpl:

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

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

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

An example for template-basic.tpl:

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

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

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

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

14.3 Control Structures

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

14.3.1 {foreach}

Iterates through an array:

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

See more in the Smarty Documentation

14.3.2 {if} / {else}

Conditionally displays content:

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

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

14.4 Standard Template Fragments

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

14.4.1 header.tpl / footer.tpl

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

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

Content goes here....

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

14.4.2 navlist.tpl

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

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

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

14.4.3 listitem.tpl

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

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

Optional keys

- *label* - A textual label for the item

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

CHAPTER

FIFTEEN

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 single CSS file. 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 understanding of CSS, and skill with an image editor such as Photoshop or GIMP.

15.1 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 site changes are isolated from the framework sources and can be more easily moved to a new version of the framework.

The core visual interface of Kurogo lives in *app/*. It is made up of HTML templates, CSS and Javascript files. All HTML, CSS and Javascript in the core interface can be overridden by a theme. While it's possible to directly edit the files in *app/*, doing so will increase the probability 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.

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.

15.2 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 Kurogo v1.2 there are significantly fewer CSS 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.

15.2.1 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 high-end devices. 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 3+), open a few test views of your site for different device classes:

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

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.

15.2.2 2. Modify the basic theme CSS

In your theme directory (which we'll refer to from now on as *THEME_DIR*), open *common/css/common.css*. This is the base theme CSS file. The essential rules you'll need to edit include:

- *body*: Body background color (and tiling image, if you so desire) and base text size, line height, and font family. 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 (preferred) or pixels. (lines 8-9)
- *body, th, td, li, option, select, input*: Primary text color (line 14)
- *a, a:visited, .focal a, .focal a:visited, .tabbody a, .tabbody a:visited*: Default link text color (line 19)
- *.nonfocal a, .nonfocal a:visited*: Link text color in areas that use the body background color; for themes with a light background, this may be the same as (or close to) line 19, but for themes with dark backgrounds it should be reversed out for contrast and legibility (line 24)
- Header styles: relative font sizes and colors for *h1* through *h4* in focal content areas, which usually have white or light backgrounds (lines 27-54)
- *dt, .label, .legend, .legend h2, .legend h3, .searchlegend*: Accent/highlight text color used in focal content areas, which usually have white or light backgrounds (line 60)

- *.address, .smallprint, .fineprint, .dek*: Secondary text color used for less important text (line 64)
- *.springboard a, .springboard a:visited*: Text color of labels below icons in springboards (grids of icons), e.g., the homescreen (line 73)
- *.nonfocal, .nonfocal .legend, .formlabel* and *.nonfocal h1, .nonfocal h2, .nonfocal h3, .nonfocal h4*: Text color and heading color, respectively, in areas that use the body background color; for themes with a light background, these may be the same as (or close to) the primary text color set in line 14, but for themes with dark backgrounds they should be reversed out for contrast and legibility. Note that the heading styles can be broken apart if you want to style them separately.
- *.shaded, .HomeModule .blockborder*: Shaded containers used to contain the tabs in many tabbed screens and in homescreen portlets on tablets (lines 99-100)
- *.shaded h1, .shaded h2, .shaded h3, .shaded h4*: Text color for headings that sit on the shaded containers. Should have legible contrast from the background color set in line 99.
- *#navbar*: Size, background color/image, and base font size for the navigation bar at the top of every screen other than the home screen. Does not apply to Basic device class. It is recommended that the height not be modified. (lines 114-115)
- *.breadcrumbs, .breadcrumbs a, .breadcrumbs a:visited, .pagetitle*: Breadcrumbs and page titles at the top of every screen other than the home screen. Does not apply to Basic device class. This text color should contrast with the background color or image specified in *#navbar* (line 114) for legibility. (line 120)
- *#footer, #footerlinks a, #footerlinks a:visited, a.copyright, a.copyright:visited*: Text and links that appear in the sitewide footers. Usually slightly less contrast than the primary text and link colors, but should still be legible against the body background set in line 8.
- *.sidenav a, .sidenav a:visited, .paging a:visited*: Color of paging navigation links used in the Calendar module (to move from day to day) and certain paged content displays; for themes with a light background, these may be the same as (or close to) the primary text color set in line 14, but for themes with dark backgrounds they should be reversed out for contrast and legibility. Note that the heading styles can be broken apart if you want to style them separately.

Other styles may be modified as well, but the ones listed above are essential for any theme.

15.2.3 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]

[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.

- Basic and Touch device classes: *THEME_DIR/modules/home/images/logo-home.gif* must be a GIF image [3]. This image will be centered horizontally within the screen. The default size is 208x35px, cropped tight to the actual artwork.
- Compliant device class: *THEME_DIR/modules/home/images/logo-home.png* must be a PNG image [3]. The default size is 280x60px, cropped tight the actual artwork. 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.

15.2.4 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:

[3] **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.

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/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/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/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

[4] **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

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]

15.2.5 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)

15.3 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, 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), 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/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/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.

15.3.1 Home-screen logo

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

15.3.2 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*).

15.3.3 Homepage module icons

Assuming you've created your standard-resolution module icons at *THEME_DIR/modules/home/images/compliant/[MODULE_ID].png*, make duplicates of all of them into *THEME_DIR/modules/home/images/compliant-bbplus* and *THEME_DIR/modules/home/images/compliant-blackberry* directories. Then replace the module icons in *THEME_DIR/modules/home/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.

15.3.4 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].

15.4 Technical Notes about Theming

15.4.1 CSS and Javascript

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

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.

15.4.2 Images

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

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

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

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

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

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

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

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

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

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

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

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

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

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

CHAPTER

SIXTEEN

STANDARD LIBRARIES

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

16.1 Remote Data Gathering and Parsing

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

The hierarchy looks like this:

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

16.1.1 DataController

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

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

16.1.2 DataParser

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

Currently there are 4 included DataParser implementations

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

16.2 Data Validation

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

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

CHAPTER

SEVENTEEN

DATA CONTROLLER

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

17.1 Usage

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

17.1.1 Instantiation

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

```
<?php

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

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

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

- *PARSER_CASS* - The class name of the parser to use. If not present it will use the value of the Controller class' DEFAULT_PARSER_CLASS property.
- *BASE_URL* - Sets the base url of the request
- *TITLE* - Sets the title of the controller
- *CACHE_LIFETIME* - Sets the cache lifetime, in seconds
- *HTTP_PROXY_URL* - Use an HTTP proxy for the request. Should be in URI format. I.e: *tcp://user:password@proxy.example.com:port*

These options can also be defined in the *[data_controller]* section of *SITE_DIR/config/site.ini* any options specified here will be used by all connections (unless they are overridden in their specific configuration) This would be a good place to include a global HTTP proxy.

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

17.1.2 Setting the properties

Set appropriate parameters for the request. This might include:

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

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

17.1.3 Retrieving the Items

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

17.2 Subclassing

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

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

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

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

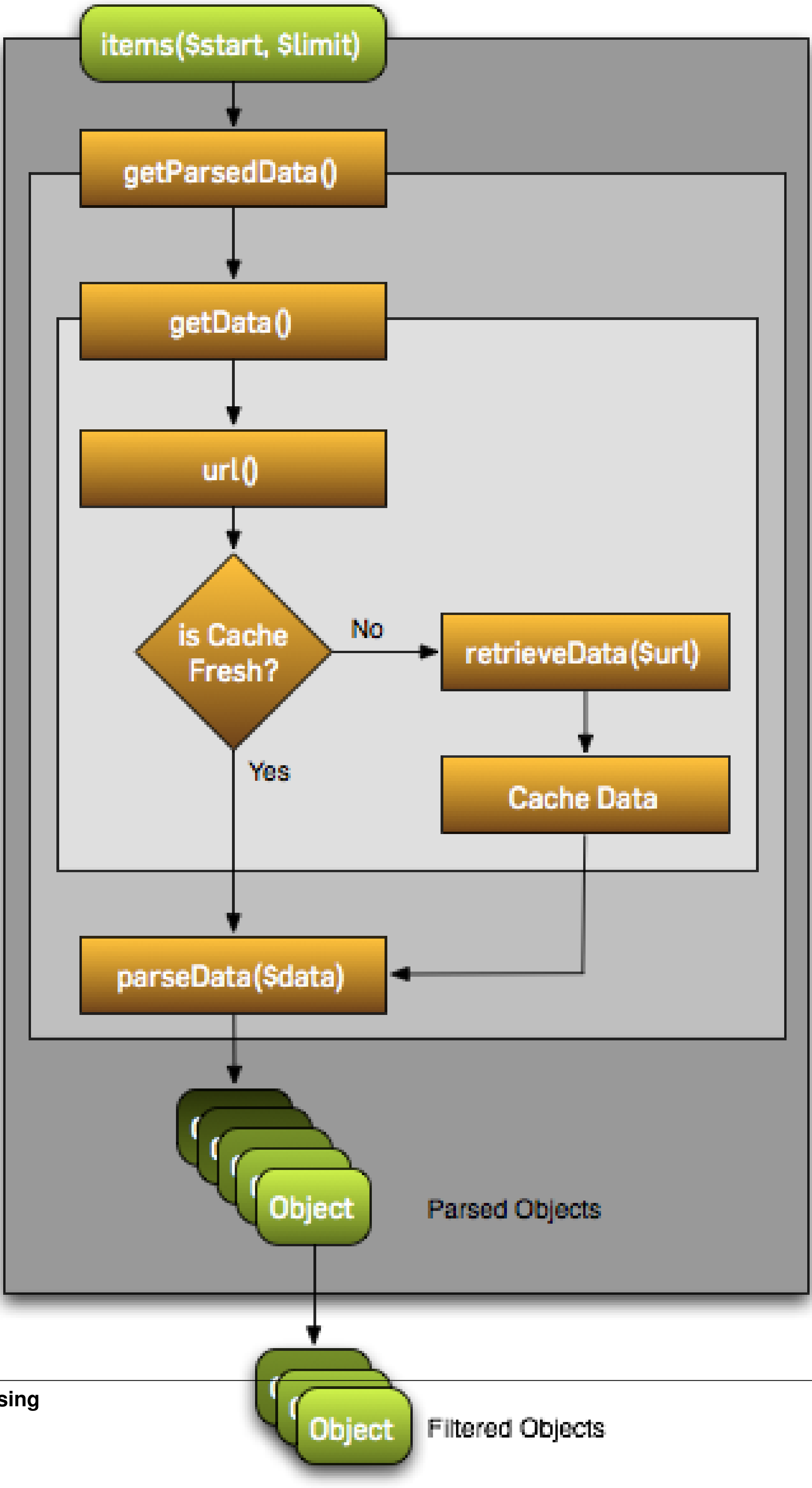
17.2.1 The Retrieval Process

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

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

17.2.2 Override:

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


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

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

17.2.3 Getting response data

- *getResponse()* - Returns the (unparsed) response body from the request
- *getResponseHeaders()* - Returns an array of response headers after the request has been made
- *getResponseHeader($header)* - Returns a specific response header based on its name ('Content-Length', etc). This is currently case sensitive
- *getResponseStatus()* - Returns the HTTP response message ('OK', etc)
- *getResponseCode()* - Returns the most recent HTTP code from the response (200, 404, etc)

17.2.4 Internal methods

- *setBaseURL($url, $clearFilters=true)* - Sets the base url for the request. If $clearFilters is true (default) then the query string filters will be cleared. Set this parameter to false to maintain any filters.
- *addFilter($var, $value)* - Adds a parameter to the url query string. Currently only one value per parameter is supported.
- *removeFilter($var)* - Removes a filter from the query string
- *removeAllFilters()* - Removes all filters from the query string
- *setTotalItems($total)* - This value is typically set by the parseData() method by querying the DataParser for the total number of items.
- *addHeader($header, $value)* - sets a header to be included in the request.
- *setMethod($method)* - Sets the HTTP method to use in the request. Values include *GET*, *POST*, *PUT*, *DELETE*. Support for other methods requires support from the remote server. The default method is *GET*
- *setTimeout($timeout)* - sets the timeout (in seconds) for the remote request.

17.3 KurogoObject Interface

In order to present a common interface for retrieving data, the KurogoObject interface has been created. Currently this interface contains no methods. It's presence exists to ensure that any module that exposes a search mechanism and participates in *module interaction* uses objects. In the future, this interface may have required methods to promote certain object oriented design principles.

CHAPTER

EIGHTEEN

CREATING A NEW MODULE

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

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

18.1 Creating the initial files

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

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

18.1.1 Steps

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

```php
<?php

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

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

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

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

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

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

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

18.2 Retrieving and Parsing Data

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

18.2.1 Creating a Data Library

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

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

18.2.2 Steps

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

```php
<?php

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

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

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

        return $results;
    }

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

}
```

Some notes on this listing:

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

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

```php
<?php

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

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

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

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

Some notes on this listing:

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

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

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

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

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

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

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

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

We will address the first item next.

18.3 Detail Page

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

```php
<?php

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

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

18.3.1 Retrieving an Entry

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

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

```
<?php

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

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

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

18.3.2 Preparing and displaying the detail view

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

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

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

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

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

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

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

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

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

18.4 Adding some Style

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

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

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

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

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

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

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

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

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

18.5 Configuration

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

18.5.1 Home Screen

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

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

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

18.5.2 Page configuration

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

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

```
[index]
pageTitle = "Video"

[detail]
pageTitle = "Detail"
```

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

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

18.5.3 Module Configuration

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

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

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

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

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

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

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

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

```
<?php

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

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

CHAPTER

NINETEEN

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

19.1 Altering an existing template

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

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

There are two approaches to updating a template.

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

19.2 Providing alternative logic (extension)

If you want to replace some of the PHP logic you can provide a subclass of the module. This allows you to override a method or property. It is important to understand the consequences of the method you override. In some cases you will want to call the *parent::* method to ensure that the base logic is executed. An example of this would be the *initializeForPage* or *linkForValue* methods. If you wanted to override the people module you would create *SitePeople-Module.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.

19.3 Replacing a module completely

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

```
<?php

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

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

19.4 Copying a Module

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

```
<?php

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

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

CHAPTER

TWENTY

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.

20.1 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

20.1.1 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.

20.2 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.

20.3 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

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.

20.4 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
<?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
        }
    }
}
```

CHAPTER

TWENTYONE

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.

21.1 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.

21.1.1 Enabling Authentication

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

21.1.2 Configuring Authorities

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

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

The following values are *required* for all authorities:

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

Included Authorities

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

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.
- *DB_USER_PASSWORD_SALT* (empty) - If present this string will be *prepended* to any string as a salt value before hashing.

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

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

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).

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.

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 at http://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.

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 at http://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.

CHAPTER

TWENTYTWO

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).

22.1 Access Control Lists

An access control list is a series of rules that defines who is permitted to access the resource (ALLOW rules), and who is expressly denied to access the resource (DENY rules). Rules can be defined for users 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.

22.1.1 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.

22.2 Using the Flat-file Authority to extend other authorities

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

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

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

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

CHAPTER

TWENTYTHREE

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

23.1 Credits

23.1.1 Modo Team

Developers

- Pete Akins (Lead Developer, Mobile Web)
- Muhammad Amjad
- Alexis Ellwood
- Ian Gavalakis
- Sonya Huang (Lead Developer, iOS)
- Jim Kang
- David Ormsbee
- Brian Patt

Designers

- Eric Kim (Lead Designer)
- Hoon Lee
- Ilona Moreland
- Brian Tepfenhart

Project Management & QA

- Yuki Nagatoshi
- Andrew Yu

23.1.2 Special Thanks

- Justin Anderson
- Massachusetts Institute of Technology

The Kurogo Framework is based on the MIT Mobile Framework, and incorporates contributions from:

- Harvard University
- University of Central Florida

23.1.3 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/

23.1.4 Third Party Libraries (iOS)

Facebook iOS SDK

License: Apache 2.0
Copyright: Facebook
Link: https://github.com/facebook/facebook-ios-sdk

Google Analytics iPhone SDK

Copyright: Google
Link: http://code.google.com/mobile/analytics/docs/iphone/

JSON

License: BSD
Copyright: Stig Brautaset
Link: http://stig.github.com/json-framework

MGTwitterEngine

License: Custom (see source in Kurogo-iOS/Contrib/MGTwitterEngine)
Copyright: Matt Gemmell
Link: https://github.com/mattgemmell/MGTwitterEngine

Sci-Fi Hi-Fi iPhone

License: MIT
Copyright: Buzz Anderson (Based partly on code by Jonathan Wight, Jon Crosby, and Mike Malone)
Link: https://github.com/ldandersen/scifihifi-iphone

23.2 License