Ignite ChMS  
complete Developer

Reference

Version: 1.0.15

Nick Airdo

David Turner

Last updated: 3/16/2011

# Table of Contents

Table of Contents 2

System Structure 3

Ignite.Framework Project 3

Model - Cms Entities 4

Ignite.Web Project 5

Caching 7

Blocks 8

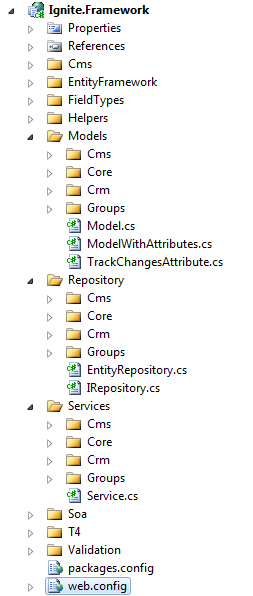
Themes 9

# **System Structure**

There are two projects in the Ignite ChMS solution: Ignite.Framework and Ignite.Web.

## Ignite.Framework Project

This project has all the EF (entity framework) Models, Repository, Services, etc.



**Models** – Auto generated using the T4 template, each class under the folders (Cms, Core, Crm, etc.) represents an EF (entity framework) entity whose data is persisted to a particular database table using a corresponding repository class described next.

While some classes will inherit from Model, most all custom and core entities will inherit from either the ModelWithAttributes class

**Repository** – These classes handle fetching/persisting the entity data to the database. Using the Repository Pattern allows us to perform some testing using a mock database and not the actual database. These classes are also auto-generated using the T4 template.

**Services** – These classes hold the “business logic” for the Ignite application and are also auto-generated using the T4 template. Generally speaking, most everything outside of the Ignite Framework will/should access Ignite core entities/objects via the services layer.

### Model - Cms Entities

The CMS entities are the parts that make up the Content Management System of Ignite. These are primarily Sites, Pages, and Blocks.

**Sites** – These typically correspond to a unique website or domain and are comprised of a collection of pages.

**Page** – A page belongs to a site and also has a layout which defines its structure or zones (ie, header, footer, main, etc.). A page can have a parent page and can also have one or more child pages. A page can also be configured to cache its rendered output for performance considerations (when appropriate) by setting our custom OutputCacheDuration property to anything greater than 0 (seconds).

**Blocks** – These “building blocks” represent reusable pieces of functionality (ASP.NET UserControls). Blocks can be added to a page by adding them a zone on a page or by adding them to a zone in a layout. Adding a block to a zone in a layout will cause all pages which use that layout to automatically have an instance of that block.

Blocks can also control how long they are cached by using the OutputCacheDuration property.

Blocks can also use the ThemePath property as a prefix for any theme-specific things (such as images, css, etc.).

The cache methods (AddCacheItem(), GetCacheItem(), FlushCacheItem()) can be used to cache custom data across requests. By default the item’s cache key will be unique to the block instance, but if caching more than one item in your block, you can specify a different key for each item.

The AttributeValue(attributeName) method can be used to get the value of any attribute associated to the instance of the block

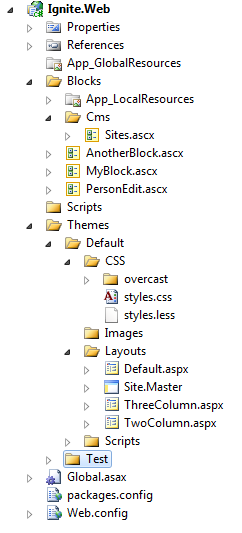
The UserAuthorized(actionName) method can be used to test whether the current user (if there is one) is allowed to perform the requested action

If a block needs data from the page routing/path information (such as the action value or site ID) it can use the PageParameter() method to fetch the value.

**Layout** – For now, these are represented by physical files that are defined in a Theme (found in the Ignite.Web project) and define one or more zones. For example, the Ignite default theme has a default layout that defines two zones: head and main. Additionally, Layouts can also use ASP.NET Master Pages to further control layout.

## Ignite.Web Project

This project holds all the reusable building Blocks and Themes.



# Caching

# Blocks

# Themes