Parking Enterprise Management System

Architecture Overview





Unified Development, Inc. & The Net Impact

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DUNCAN SOLUTIONS, INC. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN DUNCAN’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, DUNCAN ASSUMES NO LIABILITY WHATSOEVER, AND DUNCAN DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Duncan products are not intended for use in medical, lifesaving, or life sustaining applications.

Duncan may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Duncan reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Duncan sales office or your distributor to obtain the latest specifications and before placing your product order.

RxV, MX and the Duncan logo are trademarks or registered trademarks of Duncan Solutions or its subsidiaries in the United States and other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2008, Duncan Solutions, Inc. All rights reserved.

Table of Contents

1 Project Overview 7

1.1 Project general description 7

1.2 Solution general description 7

1.3 Guidelines from the functional design (Optional) 7

1.4 Development guidelines 7

1.4.1 Best Practices. 7

1.4.2 Standards and Software 7

1.5 Risks & Assumptions 8

2 Architecture Overview 8

2.1 Architecture Model Goals 8

2.2 Physical architecture 9

2.2.1 Diagram 9

2.3 Logical architecture 9

2.4 Top-Level System Components 9

2.5 Common Page Access Process 10

3 Multi-Tenancy 12

3.1 URL Model 12

3.2 City Routing 13

3.3 Administrative Routing 13

3.4 Maintenance Routing 14

4 Authentication 14

4.1 Authentication 15

4.2 Client-Side Elements 16

4.2.1 Auto-logout 16

4.2.2 Cookies 17

5 Authorization 17

5.1 Page-Based Authorization 18

5.2 NetSqlAzMan Integration 18

5.3 NetSqlAzMan Implementation 18

6 Menu System 19

6.1 NetSqlAzMan Implementation 20

7 Development Model 21

7.1 Duncan.PEMS.Web Project 22

7.1.1 Controllers 23

7.1.2 Areas 23

7.1.3 Sessions and ViewData 23

7.2 Duncan.PEMS.DataAccess 23

7.3 Duncan.PEMS.Entities 23

7.4 Duncan.PEMS.Framework 23

7.4.1 BaseController 23

7.4.2 PemsController 23

7.5 Duncan.PEMS.Security 23

7.6 Duncan.PEMS.Utilities 23

8 Deployment Model 23

9 Database Model 26

9.1 Overview 26

9.2 PEMS RBAC Database 26

9.3 PEMS Databases 26

9.4 Reporting Databases 26

9.5 Maintenance Databases 26

10 PROJECT STATUS REPORT APPROVALS 26

11 APPENDICES 26

11.1 MVC – The Controller, Actions and Authorizations 26

11.2 Convention over Configuration 27

11.3 NetSqlAzMan Management Tool and Menus 27

****Revision**** History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Author | Description | Revision Date |
| 1.0 | Ron Howard | Initial draft with sections 1 – 4 complete | 01/17/2013 |
| 1.1 | Ron Howard | Added Sections 6, 7 and 8. | 01/30/2013 |
| 1.2 | Ron Howard | Updated Section 3.1 to reflect use of a city id vs. a city name | 01/31/2013 |
| 1.3 | Ron Howard | Made a few minor grammatical changes. | 01/31/2013 |

****Document Conventions and Notes****

1. URL used in examples in this document will use host of **dev.pemsportal.com** unless otherwise indicated.
2. References to classes, methods and properties are given with a fully-qualified namespace and shown with a different font and color. This should ensure that the reader of this document will be able to find the referenced code in the project source. For example, Duncan.PEMS.Security.AuthorizationManager, references the class ‘AuthorizationManage’ in the namespace ‘Duncan.PEMS.Security’.

Introduction

# Project Overview

## Project general description

Parking Enterprise Management System (PEMS) is a system to manage client parking resources, including meters, sensors, gateways, cashboxes and spaces. The system is web-based.

## Solution general description

## Guidelines from the functional design (Optional)

## Development guidelines

### Best Practices.

Convention-over-Configuration (CoC) plays a pivotal role in the PEMS system development and operation. It is where coding conventions such as naming drive development and system functionality and process operates upon a pre-defined convention verses configuration. PEMS takes advantage of this methodology to build the system. Throughout this document CoC concepts are applied to PEMS, from authorization to routing to multi-tenancy.

See the Wikipedia article [Convention over configuration](http://en.wikipedia.org/wiki/Convention_over_configuration). There are many resources that discus CoC as applied to MVC 4. For instance [this site](http://www.simple-talk.com/dotnet/asp.net/asp.net-mvc-controllers-and-conventions/) provides insight into how CoC is a central tenant of MVC 4.

Recommend using NuGet (<http://visualstudiogallery.msdn.microsoft.com/27077b70-9dad-4c64-adcf-c7cf6bc9970c>) to manage third-party packages in Visual Studio.

### Standards and Software

The following operating system, DBMS, development environments, and software are used in the development and deployment of PEMS.

Servers

Microsoft Windows Server 2008 R2

Microsoft IIS 7.5

DBMS

Microsoft Sql Server 2008 R2 or better

Development Environment and Languages

Microsoft Visual Studio 2012 (<http://msdn.microsoft.com/en-us/library/dd831853%28v=vs.110%29.aspx>)

Razor Syntax ([tutorial](http://www.asp.net/web-pages/tutorials/basics/2-introduction-to-asp-net-web-programming-using-the-razor-syntax))

C#

Transact-SQL (<http://msdn.microsoft.com/en-us/library/bb510741%28SQL.100%29.aspx>)

Software, frameworks, and packages

Microsoft .NET 4.5 (<http://www.microsoft.com/en-us/download/details.aspx?id=30653>)

ASP.NET MVC 4.0 (<http://www.asp.net/mvc/mvc4>)

Entity Framework 5.0 (<http://www.nuget.org/packages/EntityFramework/5.0.0>)

NetSqlAzMan 3.6.0.15 (<http://netsqlazman.codeplex.com/>)

Other 3rd-Party packages - For up-to-date packages and versions, see the ‘Assemblies’ directory in TFS PEMS Project repository.

## Risks & Assumptions

# Architecture Overview

The architecture model selected is based upon best practices of MVC 4 to implement a multi-tenancy system that is scalable, maintainable and customizable while following the tenants of MVC 4. A key factor in the selection of the architecture is a development model that utilizes the MVC 4 tools available in Visual Studio 2012 to reduce the work-load on the developers and simplify the development model. This will aide in the reduction of code complexity and improve maintainability.

The architecture also leverages the robust capabilities of recent IIS releases to provide a hosting model that is robust, secure and provides capabilities to distribute the PEMS implementation to multiple host machines.

Convention-over-Configuration (CoC) plays a pivotal role in the PEMS system development and operation. PEMS takes advantage of this methodology to build the system. Throughout this document CoC concepts are applied to PEMS, from authorization to routing to multi-tenancy. See Appendix for additional discussion and resources on CoC.

See the Wikipedia article [Convention over configuration](http://en.wikipedia.org/wiki/Convention_over_configuration). There are many resources that discus CoC as applied to MVC 4. For instance [this site](http://www.simple-talk.com/dotnet/asp.net/asp.net-mvc-controllers-and-conventions/) provides insight into how CoC is a central tenant of MVC 4.

Recommend using NuGet (<http://visualstudiogallery.msdn.microsoft.com/27077b70-9dad-4c64-adcf-c7cf6bc9970c>) to manage third-party packages in Visual Studio.

## Architecture Model Goals

The following constraints and goals drive the architecture.

* The system implementation may be distributed for scalability.
* The architectural model stresses strong interface contracts with separation/isolation of functionality into components that encapsulate operational capabilities and provide a high degree of re-use.
* System component model shall allow for any component to be replaced by like functionality or be deconstructed to sub-components as development unfolds.
* Authentication and authorization will be implemented with a common framework pattern that is transparent to the page implementation pattern.
* NetSqlAzMan provide authorization model and implementation for page and menu access.
* Authentication will be based upon .NET/MVC best-practices.
* Authorization control is at page level.
* Multi-tenancy will be implemented via the MVC Routing pattern. The URL model shall be flexible within the constraints of MVC routing model.
* System components to be independently maintainable and unit-testable.
* Capability to distribute functionality among multiple hosts.

## Physical architecture

### Diagram

## Logical architecture



## Top-Level System Components

****

Figure 1 - Top-Level System Components

There are five main system components. These components represent the main concepts in the PEMS architecture. These components may have additional internal decompositions to achieve design goals, improve testability, maintainability or promote rapid development. The components are:

* **Framework** – Responsible for applying authentication, authorization and resource and page request routing. This is built upon MVC Areas pattern, MVC routing and view engines. Common support functionality for generic operation of PEMS shall reside in this component.
* **Web** – The web page implementations. These are the main user-interactive elements and comprise the bulk of PEMS system work. They operate inside the Framework and leverage the Business components.
* **Security** – All authentication and authorization functions are provided by this component. It also provides the interface to NetSqlAzMan as well as the implementation of MembershipProvider. Alternate methods of authentication and authorization may replace initial implementations without major rework of the Framework.
* **Business** – Responsible for all business data and logic. No business logic should be implemented in other components unless derived from this component.
* **DataAccess** – All PEMS database access is derived from this component. Changes to PEMS database will be isolated to this component.

## Common Page Access Process

There are two types of page resources in PEMS:

* Anonymous Pages
* Authenticated Pages

Anonymous page resources are pages such as the login page where any user on the web can access the page. Anonymous pages have no knowledge about the user or of the user’s city.

Authenticated pages are pages where only authenticated users may access the page and are often constrained further by authorization constraints. The authentication process established the user credentials and the user city. All authenticated users have established these two pieces of information. (See section on Authentication)

For each authenticated page in the PEMS system, the user authentication, city, and authorization rights are established before the execution context is passed to the actual page resource controller and action. The Framework component is responsible for authentication and authorization. The Web component then only needs to produce the requested page as all rights have been established prior to execution of the page creation code.

****

Figure 2 - Framework Authentication and Authorization Flow

Once a user is authenticated, all page resource requests follow a common process flow. This process flow is shown in the figure above.

A page request is first processed by the MVC Routing engine. If the URL cannot be decoded or does not match a known route in the PEMS Routing table, return an error to the user. This is native MVC functionality.

If the URL is decoded and matches an allowable route then the user authentication token is verified. If the authentication token is invalid, redirect user back to login page.

If the authorization token is valid, the active city or admin identifier is extracted from the URL. The user authorization for the associated city or admin page is verified. If the user is not authorized for the requested page, send the user back to the home page.

If the user is authenticated and authorized for the page, pass execution context to the appropriate page resource controller and action. The resulting page is passed back to the user.



# Multi-Tenancy

One of the key goals of this architecture is support of a multi-tenancy system. This support is guided by the requirements and by the routing models that are the centerpiece of the MVC pattern. MVC routing is the mechanism where a requested URL is routed to an action on a controller. This controller action ultimately serves up the page contents. Support for multi-tenancy is based on the MVC pattern concept of ‘Areas’ and MVC routing rules.

## URL Model

The URL model is a key concept in MVC. Page resources are executed based upon the resolution of the URL to a particular controller and action. The MVC routing engine is responsible for parsing the URL and determining which controller and action to call. One of the attributes any URL model must exhibit is that it can uniquely be parsed to extract information and that any URL that meets the parsing model has the same type of information in the same position in the URL. This levies a requirement of a URL to be parsed such that a route (controller and action) can be uniquely determined from the URL. Given these constraints, many URL models are possible.

The URL model that is presently implemented is of the form:

**{host}/{city}/pems/{controller}/{action}/{id}**

Decomposing this example:

http://www.somebasesite.com/boston/pems/home/index/5

**{host}** – The host part of the URL.

[***http://www.somebasesite.com***/127/pems/home/index/5](http://www.somebasesite.com/127/pems/home/index/5)

**{city}** – The client city identifier. This is an internally-assigned id or unique string to make the URL friendly. Note that a {city} identifier can either be the unique numeric id or it can be the unique city name that is hosted at this host.

[*http://www.somebasesite.com*/**127**/pems/home/index/5](http://www.somebasesite.com/127/pems/home/index/5)

--or--

[*http://www.somebasesite.com*/**boston**/pems/home/index/5](http://www.somebasesite.com/boston/pems/home/index/5)

This is used to identify the client in the PEMS system. There will be a “store” entry in NetSqlAzMan with the same **{city}** name and id to control authorizations. Internal to the PEMS system, this identifier indicates which city is being serviced.

**pems** – Serves as a field to give a pattern to the URL such that the **{city}** is separated from the **{controller}/{action}** via a constant.

[*http://www.somebasesite.com/127/****pems****/home/index/5*](http://www.somebasesite.com/127/pems/home/index/5)

**{controller}** – The controller name that will handle the request.

[*http://www.somebasesite.com*/127/pems/**home**/index/5](http://www.somebasesite.com/127/pems/home/index/5)

This controller name, by MVC convention, is implemented in the source code file HomeController.cs. In addition, this controller name is used in NetSqlAzMan as the “Application” under a “store”.

**{action}** – The method name in the controller that will handle the request.

[*http://www.somebasesite.com*/127/pems/home/**index**/5](http://www.somebasesite.com/127/pems/home/index/5)

This action name is the method in the controller that is called to handle the page request. In NetSqlAzMan, this action is an “operation”.

**{id}** – An optional id or other URL-encoded data.

[*http://www.somebasesite.com*/127/pems/home/index/**5**](http://www.somebasesite.com/127/pems/home/index/5)

Additional parameter values can be passed via URL-encoding or via web form fields. By convention, a value occupying the {id} position would be passed to the controller action as a parameter named ‘id’.

## City Routing

The present multi-tenancy approach separates the functionality for city clients from functionality that is PEMS administration. The URL is parsed to determine the target city. This city then becomes the active city for the duration of the page request. The city, in conjunction with the user established authorization to page resources. Once a user is authenticated into a city, they cannot jump directly to service a different city without first indicating that city via a change client process and link.

The route entry that supports routing to the city pages is defined to parse out the target city and route the controller and action resolution to a namespace that houses all city functionality.

Based on the example URL above, a URL that would direct to North Sydney functionality, assuming the internal id for North Sydney is 100 and the internal name for North Sydney is “North Sydney”, would look like:

<http://www.somebasesite.com/100/pems/home/index>

--or--

[http://www.somebasesite.com/North Sydney/pems/home/index](http://www.somebasesite.com/North%20Sydney/pems/home/index)

## Administrative Routing

The present multi-tenancy approach treats administrative functionality as a special city. A city defined as ‘admin’ directs requests to the functionality that is PEMS administration. This route uses a MVC attribute on the route entry known as a ‘constraint’. ‘Admin’ becomes the active city for the duration of the page request. The city of ‘admin’, in conjunction with the user established authorization to page resources. Once a user is authenticated into administration, they cannot jump directly to service a different city without first indication that city via a change process and link.

The route entry that supports routing to the city pages is defined to as a constrained city URL where the city is ‘admin’. The controller and action resolution are directed to a namespace that houses all administrative functionality.

Based on the example URL above, a URL that would direct to administrative functionality would look like:

<http://www.somebasesite.com/admin/pems/home/index>

Note that ‘admin’ is spelled out and not indicated by a numeric id. The URL model is quite flexible.

## Maintenance Routing

**Add text on how maintenance routing works.**



# Authentication

Authentication is based upon a user & password in the PEMS system. A user is associated with one or more client cities and/or the administrative system. A user is authenticated against the PEMS system and is then directed directly to a landing page (client city or administrative) or is directed to a page to select the focal client city (or administrative) of the user’s session.

User authentication is based upon the MembershipProvider model available in MVC. This model is considered best practice and will be used by PEMS.

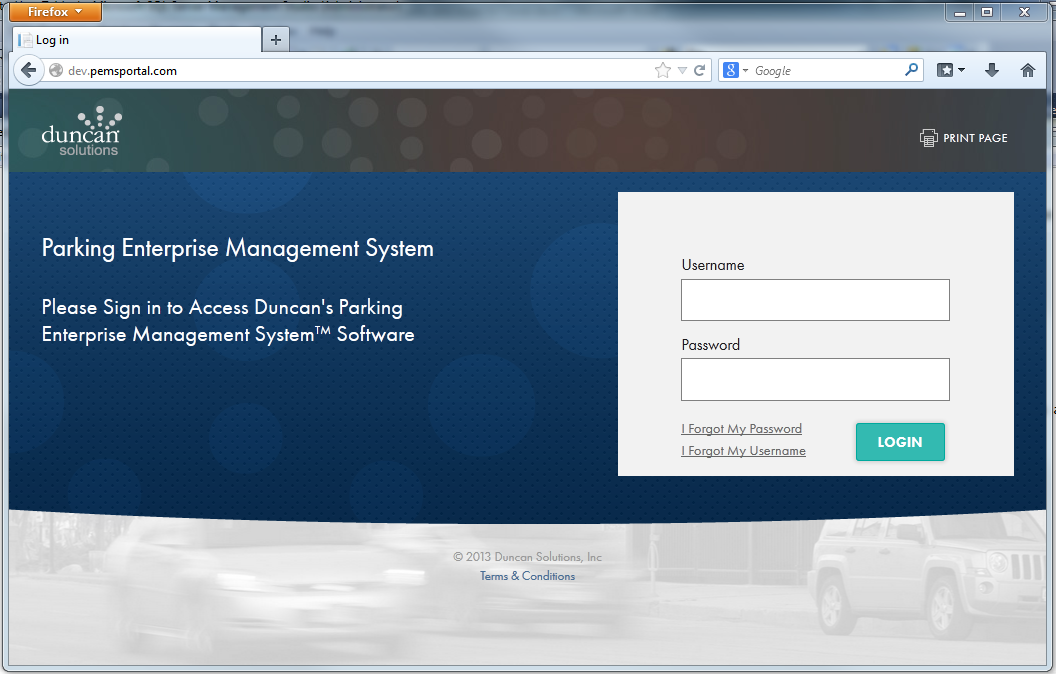


Figure 3 - Basic Login Flow

The process flow indicated above shows the flow of a typical login. The result of this authentication process establishes that a user has rights to access PEMS system and which city, or in the case of a Duncan user, administrative focus of the user’s session in PEMS.

## Authentication

The user authentication process begins with user entering a user name and password on the login page which is the default page when navigating to an instance of PEMS deployed at a given host.



A user is authenticated against tables in the RBAC database via the MVC controller class Duncan.PEMS.Web.Controllers.AccountController which first checks if user is logged out, then check if user exists, then check if user has been disabled. If any of these fail, user is returned to login screen. Finally, the user/password is verified against the RBAC database by calling Duncan.PEMS.Security.SecurityManager.Login(string username, string password). If user fails, return user to the login page and display failure text.

If user name/password are verified, generate an encrypted cookie with the user name, active city set to ‘None’ and a CustomerLoginType set to Unknown. This indicates that the user has successfully logged in but target city and type of user have not yet been determined. Finally, redirect the user to the controller action method Duncan.PEMS.Web.Controllers.HomeController.Landing() which checks which cities user has access right to.

If user only has access to one single city, the user will be again redirected to that city’s home page. If the user has access to two or more cities, a page is displayed where the user can select the city that they wish to enter.

## Client-Side Elements

There are two client-side elements that aid in authentication and security. They are:

auto-logout and a cookie.

### Auto-logout

Once a user is authenticated and has selected a city, the subsequent pages monitor the user for activity such as mouse movement or keystrokes. This monitoring is achieved by a JavaScript package called *jquery.idletimer.js* and *jquery.idletimeout.js.* These scripts in conjunction with JavaScript code on the master page */Areas/shared/Views/Shared/\_Layout.cshtml* in the *Duncan.PEMS.Web* project, implement a timeout system when the user becomes idle.

The idle timeout system has three basic times that need to be set:

* Idle Timeout – Number of seconds that a user needs to be idle before timeout process starts. Defaults to Duncan.PEMS.Utilities.Constants.IdleTimeoutDefaults.IdleTimeout seconds.
* Idle Timeout Warning – Number of seconds to display a dialog warning user that they are about to be logged off. Defaults to Duncan.PEMS.Utilities.Constants.IdleTimeoutDefaults.IdleTimeoutWarning seconds.
* Idle Timeout Polling – Number of seconds between AJAX calls to server to keep session alive. Defaults to Duncan.PEMS.Utilities.Constants.IdleTimeoutDefaults.IdleTimeoutPolling seconds.

To set these values, add the following lines to the site’s web.config. Select appropriate values (in seconds).

<appSettings>

<add key="pems.security.timeout" value="3600" /> <!-- In Seconds -->

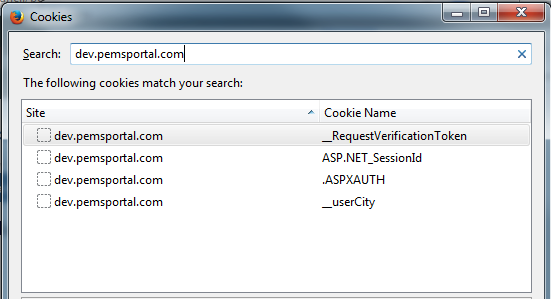
<add key="pems.security.timeout.warning" value="60" /> <!-- In Seconds -->

<add key="pems.security.polling" value="600" /> <!-- In Seconds -->

</appSettings>

### Cookies

There are four cookies used in the PEMS system. Below are the cookies used by PEMS.



* ASP.NET\_SessionId – The cookie containing the unique session id used to associate session variables to this web session.
* .ASPXAUTH – Authentication cookie used by ASP.NET Forms Authentication
* \_\_RequestVerificationToken - .NET C# MVC anti-forgery token
* \_\_userCity – Token used to indicate active user, city and user type. See Duncan.PEMS.Framework.Controller.BaseController for more details on this cookie.



# Authorization

Authorization to a given resource is managed by NetSqlAzman. The authorization model is driven by four factors:

* System requirements
* NetSqlAzMan capabilities
* The MVC event model
* Reducing code complexity

The authorization model is based upon assigning users to roles for a given city or administrative site. These roles are given access to individual page resources. The enforcement of authorization rights is accomplished by injecting the authorization check into the MVC event pipeline and either allowing or denying access to a resource prior to the actual MVC controller method call.

## Page-Based Authorization

Authorization is page-based to reduce the complexity of managing individual field access and the associated code complexity that field-level access authorization creates. This also provides a development aid for the developers where the page is the focus of the action and if a user has access to a page then all functionality they are developing will be available to the user of the page. This lessens the work-load on the developer and the page designer as all functionality is the same for a user so the look and feel as well as the functionality is a single view per page.

## NetSqlAzMan Integration

NetSqlAzMan is used to provide the Authorization functionality for PEMS System. There are core concepts in NetSqlAzMan that have analogies in the MVC model. These analogs are key to the integration of NetSqlAzMan.

Resources in PEMS are accessed via URLs that are described URL Model in section 3.1 above. The key concepts are Controller and Action. A Controller/Action provides the execution element that ultimately serves a URL-based resource. The control of a resource is at the Controller/Action level.

In NetSqlAzMan, a site (city, for instance) maps to a Store. Each Store can have access Roles where users are assigned to a Role. Under each Store there are Applications which represent a store-level grouping of functionality. For instance, Assets would be an Application representing the functionality to manage a city’s assets. Under these Applications, NetSqlAzMan provides for Tasks and Operations. Tasks are being used to provide the menu system. See Menu System in section 6. Operations are discrete functions that are available under an Application and represent a distinct action that a user may take. NetSqlAzMan allows for an Operation to be authorized for a user or a store group. By convention we are only using store groups (Roles) to be authorized to access an Operation.

In NetSqlAzMan an Application is mapped to a Controller in MVC. An Operation is mapped to an Action in MVC. This provides a mechanism where a Controller/Action is authorized via an Application/Operation in NetSqlAzMan

## NetSqlAzMan Implementation

NetSqlAzMan interface is entirely encapsulated in the Duncan.PEMS.Security.AuthorizationManager and Duncan.PEMS.Security.Menu.PemsMenu classes. This approach was chosen to insulate PEMS from the specifics of the NetSqlAzMan API and the integration. If, at a later date, Duncan decides to remove NetSqlAzMan, as long as the Duncan.PEMS.Security.AuthorizationManager and Duncan.PEMS.Security.Menu.PemsMenu class public properties and methods are supported, a new RBAC could be used. See Menu System below for details on the implementation of Duncan.PEMS.Security.Menu.PemsMenu.

Duncan.PEMS.Security.AuthorizationManager implementation is in two source files, *AuthorizationManager.cs* and *AuthorizationManagerEx.cs* in the *Duncan.PEMS.Security* project.

* *AuthorizationManager.cs* – General RBAC authorization access read and write functions.
* *AuthorizationManagerEx.cs* – Methods used to process and write XML configuration files for the RBAC system. This XML and the associated processing are covered under the “PEMS Authentication and Menus Users Guide.docx” document.

The Duncan.PEMS.Security.AuthorizationManager class has three different constructors used in three distinct roles:

Duncan.PEMS.Security.AuthorizationManager.AuthorizationManager() - Used to provide Authorization functionality for the anonymous pages during login and for the Change Client page. This type of instance will also be used to create a new city in the Authorization system.

Duncan.PEMS.Security.AuthorizationManager.AuthorizationManager(PemsCity city) – Used specifically in the

Duncan.PEMS.Security.AuthorizationManager.AuthorizationManager(string city) -

associated directly with a city via the city name, or with no city.

An instance associated with no city (default constructor) is used to provide Authorization functionality for the anonymous pages during login and for the Change Client page. This type of instance will also be used to create a new city in the authorization system.

An instance associated with a city has a very specific role. That role is to provide



# Menu System

A menu item in PEMS represents an authorized resource that the user can access. That resource may be a page served by MVC via the Controller/Action model or it may be an external resource such as IZenda reporting site. Users will see only menu selections that represent resources they have been authorized to use.

NetSqlAzMan provides the backing for the menu system. Authorization is used to generate the allowable menu entries presented to a user of a specific role. In order to understand the capabilities of NetSqlAzMan and how its structure maps to resources exposed by the MVC model, the terminology of NetSqlAzMan needs to be mapped to MVC as well as the PEMS system.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **NetSqlAzMan** | **MVC** | **PEMS System** |
| A. | Store | Web App | City |
| B. | Application | Controller | Functional Area |
| C. | Task |  | Menu Item |
| D. | Operation | Action | Menu Action |
| E. | DBUser |  | User |
| F. | Store Group |  | User Role Assignment |

1. NetSqlAzMan Store represents a client city in the PEMS system. This also represents the active web application in MVC. For instance, this may be a client city such as North Sydney or the PEMS Administration site. For each active city, there will be a store created in NetSqlAzMan.
2. NetSqlAzMan Application represents a general functional area such as Asset Management in the PEMS System. It also maps to a Controller class in MVC. The Application effectively represents a menu group that is at the top level of the menu.
3. NetSqlAzMan Task represents the PEMS Menu Item that provides the menu detail such as the menu text, the tool tip text and other attributes of a menu item. A Task does not necessarily map to a concept in MVC but may subsequently point to an Action in MVC via an Operation in NetSqlAzMan. A Task is the fundamental representation of a menu item. A Task may have an Operation assigned to it. It also may have alternative menu actions defined for it by Task Attributes. Access is granted to a Task or alternatively to the Task’s Operation for a particular user role in order to make that menu item appear in the PEMS menu.
4. NetSqlAzMan Operation represents an Action in an MVC Controller. This can be thought of as a page in the web application. Operations are mapped to Tasks in order to link a menu item to a particular page provided by MVC. A user is authorized to use an operation when the Operation has access rights granted to a role or which a user is a member. If an Operation has been assigned to a Task then users authorized on that Operation will see the menu item represented by the Task.
5. NetSqlAzMan DBUser represents a user n the PEMS system. There is a one-to-one mapping between a PEMS user and a NetSqlAzMan DBUser. These DBUsers are assigned to a Store Group for purposes of granting authorization to a menu item.
6. NetSqlAzMan Store Group represents a role in the PEMS System. A Store or city can have multiple Store Groups created for it. An example could be Administrators and Clerks. These groups are at the city level and are used to categorize users into roles for the PEMS system for a city. Menus are authorized for a particular Store Group(s). These Store Groups control which menus are visible to a DBUser based on their membership.

A menu item can have many attributes that control what is displayed for text, tool tip text, target URLs and other attributes as needed. These attributes are also part of the NetSqlAzMan capability and provide a convenient mechanism by which a menu item can be customized while following the NetSqlAzMan application capabilities.

PEMS Administrative functions available via the web application are ultimately going to be used to create the menu items, assign users to groups and authorize the groups to menu items. Alternatively the native NetSqlAzMan management tool may be used to define these various elements to create menu items. Appendix 10.3 provides some example screen shots and descriptions of the use of the management tool to aid in understanding of the use of NetSqlAzMan functions to generate the menus..

## NetSqlAzMan Implementation

NetSqlAzMan interface is entirely encapsulated in the Duncan.PEMS.Security.AuthorizationManager and Duncan.PEMS.Security.Menu.PemsMenu classes. This approach was chosen to insulate PEMS from the specifics of the NetSqlAzMan API and the integration. If, at a later date, Duncan decides to remove NetSqlAzMan, as long as the Duncan.PEMS.Security.AuthorizationManager and Duncan.PEMS.Security.Menu.PemsMenu class public properties and methods are supported, a new RBAC could be used. See Authorization above for details on the implementation of Duncan.PEMS.Security.AuthorizationManager.



# Development Model

The development model has been chosen to promote team development and leverage the MVC 4 project template model and tools in Visual Studio 2012 with minimum customization. MVC 4 templates for VS 2012 provides the basis for the development environment.

The PEMS solution is composed of seven projects that encapsulate related functionality.

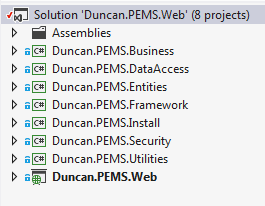


Figure 4 - PEMS Solution and Projects

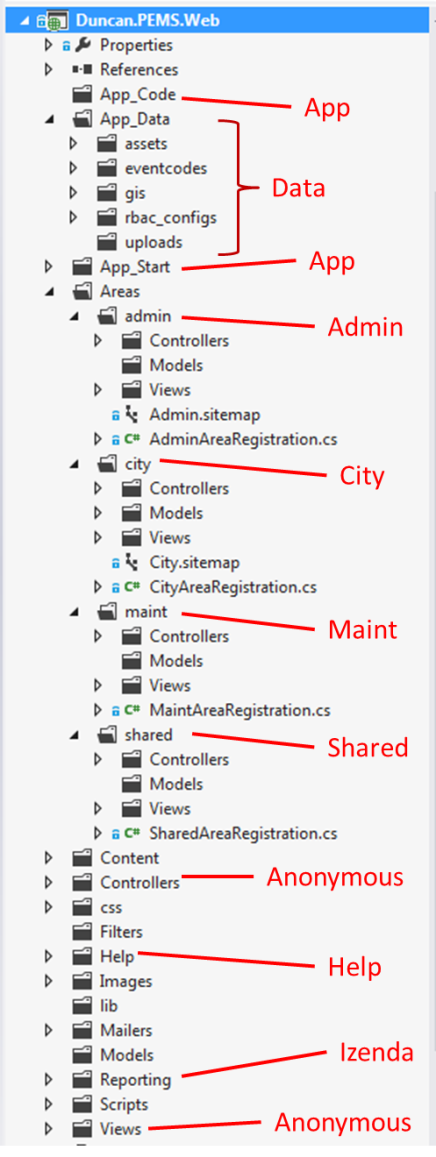
The projects are:

* Duncan.PEMS.Business – Project that contains PEMS business logic functionality. All business process and business logic will be contained in this project.
* Duncan.PEMS.DataAccess – Project that contains the data access components. All database and file access will be homed in this project. This project contains Entity Framework models that provide access to the RBAC and the PEMS databases.
* Duncan.PEMS.Entities – Project will contain the Models for the MVC pattern. This is one customization of the MVC project template that has been made. Normally the Models in MVC would be located in the same project as the View and Controllers.
* Duncan.PEMS.Framework – Project contains the base framework of the PEMS system. This framework is common to all pages served via MVC pattern. This project contains the page authentication and authorization process that is applied to all web application pages. This project also contains the MVC Routing Engine and View Engine customizations used by the PEMS System. These functions are common to the entire MVC implementation of PEMS.
* Duncan.PEMS.Install – Project contains the console application, database scripts and PEMS documentation.
* Duncan.PEMS.Security – Project contains the elements that comprise the authorization and authentication functionality. This project provides abstractions from the underlying implementations of the authorization and authentication systems. For instance, all NetSqlAzMan access functions are constrained to only this project such that, if need be, NetSqlAzMan can be deprecated and replaced by an alternative role-based access system.
* Duncan.PEMS.Utilities – Project contains shared utility resources such as common enumerations and constants as well as non-specific utility functions shared by all of the projects.
* Duncan.PEMS.Web – Project contains the MVC project that comprises the PEMS web application. This project has been designed to support multi-tenancy and anonymous page resources and is implemented with a native MVC project template. The following section discusses the internal project structure in more detail.

## Duncan.PEMS.Web Project

This project has a distinct structure that leverages MVC Areas and reflects the model of page access in PEMS System. PEMS has three main areas and two shared area that have common and distinct functional roles.

* Anonymous – Pages such as Log In and Forgot Password that are accessed by users that have not been authenticated.
* City – Resources and functionality that city client users accesses to transact business for a city.
* Maint – Resources and functionality for the mobile Field Maintenance site.
* Admin – Resources and functionality that a Duncan administrative user access to administer the PEMS system.
* Shared – Shared resources that are common functions used in both the City and Admin roles.
* App – Shared functions that are used by the web app during startup and core operations.
* Help – Context-sensitive help content and general help content of PEMS.
* Izenda – Files used for constructing and displaying Izenda reports. These work in conjunction with resources in the City area.
* Data – Operational data used by PEMS. This includes configuration XML, sample CSVs and other data. This does not include config files.



These areas are represented in the project structure. This can be seen in the adjoining figure.

The project uses the MVC Areas paradigm to support multi-tenancy development such that the web application supports all users. This reduces duplication of functionality and provides a functional base model by which the PEMS System is being developed.

The two project area that are shared, App and Shared, are applicable at different levels.

* App – Common functionality to the entire web application such as route table initializations and view engine creation.
* Shared – Resources that are shared between the two Areas of Admin and City

The Anonymous resources are found in both the Controllers and Views folder and are applicable at the root level of the web application.

Under both the Maint, City and Admin areas there are Model, View and Controller folders. As discussed previously, the Model folder is replaced by the project called Duncan.PEMS.Entities.

This structure was chosen as it supports both the PEMS System model and is natively supported by the MVC project tools and templates.

Figure 5 - Project Folders

The development model is designed to be expanded as the project scope changes and is an initial starting model that meets the present goals of PEMS.

### Controllers

### Areas

### Sessions and ViewData

## Duncan.PEMS.DataAccess

This is the data access layer project. It contains the Entity Framework models and the associated generated files. Changes to the database schemas that are accessed by Entity Framework contexts must be reflected here. This project also contains Views, Stored Procedures and other database objects created for PEMS. These objects are included in the individual databases but the authoritative versions are found here.

## Duncan.PEMS.Entities

This project contains the business entities and MVC models used by PEMS.

## Duncan.PEMS.Framework

### BaseController

### PemsController

## Duncan.PEMS.Security

## Duncan.PEMS.Utilities



# Database Model

## Overview

PEMS system database model is comprised of four logical databases. These databases are:

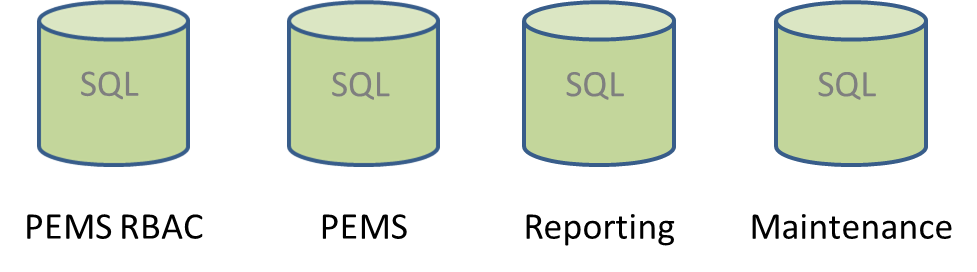


Figure 6 PEMS System Logical Databases

* PEMS RBAC – The Role-based access control, user and customer properties, and PEMS settings database
* PEMS – The transactional database for meters and asset, events, transactions and other operational data for one or more customers.
* Reporting – A duplicate schema of the PEMS database used for the Izenda reporting functions.
* Maintenance – The database used for field maintenance data and transactions.

## PEMS RBAC Database

### General description

The PEMS RBAC database contains the tables for the RBAC implementation, customer information and settings for all customers, user information and settings for all users, user authentication and PEMS system configuration data.

### Tables list

|  |  |
| --- | --- |
| **Table Name** | **Description** |
| AccessLog |  |
| AccessLogMembershipEvents |  |
| CustomerGrids |  |
| CustomerGridTemplate |  |
| CustomerGridTemplateCol |  |
| CustomerGridTemplateMap |  |
| CustomerProfile |  |
| CustomerReports |  |
| CustomerReportsCategory |  |
| CustomerSettings |  |
| CustomerSettingsGroup |  |
| CustomerSettingsGroupMember |  |
| CustomerSettingType |  |
| CustomerSettingTypeList |  |
| CustomerType |  |
| ErrorMessages |  |
| HelpMap |  |
| LocaleResources |  |
| LocaleResourcesCustom |  |
| LoginAttemptHistory |  |
| MaintenanceGroupCustomer |  |
| MembershipEventTypes |  |
| netsqlazman\_ApplicationAttributesTable |  |
| netsqlazman\_ApplicationGroupMembersTable |  |
| netsqlazman\_ApplicationGroupsTable |  |
| netsqlazman\_ApplicationPermissionsTable |  |
| netsqlazman\_ApplicationsTable |  |
| netsqlazman\_AuthorizationAttributesTable |  |
| netsqlazman\_AuthorizationsTable |  |
| netsqlazman\_BizRulesTable |  |
| netsqlazman\_ItemAttributesTable |  |
| netsqlazman\_ItemsHierarchyTable |  |
| netsqlazman\_ItemsTable |  |
| netsqlazman\_LogTable |  |
| netsqlazman\_Settings |  |
| netsqlazman\_StoreAttributesTable |  |
| netsqlazman\_StoreGroupMembersTable |  |
| netsqlazman\_StoreGroupsTable |  |
| netsqlazman\_StorePermissionsTable |  |
| netsqlazman\_StoresTable |  |
| News |  |
| NewsContent |  |
| ResourceEvents |  |
| ResourceEventTypes |  |
| SystemInformation |  |
| TimeZones |  |
| UserCustomerAccess |  |
| UserPasswordHistory |  |
| UserPasswordQuestion |  |
| UserProfile |  |
| UserSettings |  |
| Version |  |
| webpages\_Membership |  |
| webpages\_OAuthMembership |  |
| webpages\_Roles |  |
| webpages\_UsersInRoles |  |

### Database diagram























### Views

### Stored procedures

## PEMS Databases

## Reporting Databases

## Maintenance Databases

|  |  |
| --- | --- |
|  |  |
| |  | | --- | | WorkOrderEvent | | WorkOrderImage | | WorkOrderPart | | WorkOrders | | WorkOrdersAudit | | WorkOrderStatus | | Parts | |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |





# Infrastructure

## General

## Logging handling (Log4Dot Net)

## Auditing and tracing handling

## Exception handling

### General

### Custom Exceptions

## Monitoring

### Database monitoring for performance

## Data Access

### General

### Typed Datasets

## Common

### Common Enums

### Common Objects

### Common functions

## Multi-Language Support

## Printing Mechanism

## Cache management

### Web site cache

### Application cache

### Database cache

## Properties files based Configuration

### Global configuration

### Web site configuration

## Unit Testing

### Method

### Tested Components

## Transactions

## Validations

### Application validations

### Web site validations



# Business entities

## General

## Entities diagram

## Entities description



# Class Design

## General Structure

## State management

### Cookies

### Application State

### Session State

## Master pages and themes

## Pages

## User controls

## Custom controls

## Resources

## Common functions

## Error Pages

## UI graphic guidelines

## Class diagrams

## Description

### Features

### Web parts

### Templates

### Event handlers

### Http handlers.

### Lists



# APPENDICES

## MVC – The Controller, Actions and Authorizations

MVC applications are composed of Models, Views and Controllers. Models and Views perform the normal roles in PEMS as they do in other MVC implementations. The Controllers is where PEMS implements the authorization scheme.

A Controller provides access to a resource via an Action that is executed based on a request derived from the parsing of a URL and the routing engine native to MVC. This Routing is discussed in URL Model section above. The outcome of the routing is that a particular Action in a particular Controller will be executed to serve the resource to the user.

PEMS authorization model is based on this model where rights to a resource (Controller/ Action) are controlled for a user group. This model is the basis of the authorization model in NetSqlAzMan.

* NetSqlAzMan Application is analogous to a Controller in MVC
* NetSqlAzMan Operation is analogous to an Action in a Controller in MVC

Therefore, the resource in MVC is authorized by controlling authorization to an Application’s Operations in NetSqlAzMan.

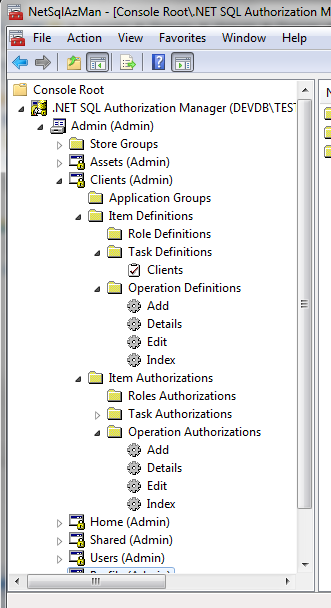
## Convention over Configuration

~~Convention-over-Configuration (CoC) is a key concept in MVC 4. It is where coding conventions such as naming drive development and system functionality and process operates upon a pre-defined convention verses configuration.~~

~~See the Wikipedia article~~ [~~Convention over configuration~~](http://en.wikipedia.org/wiki/Convention_over_configuration)~~. There are many resources that discus CoC as applied to MVC 4. For instance~~ [~~this site~~](http://www.simple-talk.com/dotnet/asp.net/asp.net-mvc-controllers-and-conventions/) ~~provides insight into how CoC is a central tenant of MVC 4.~~

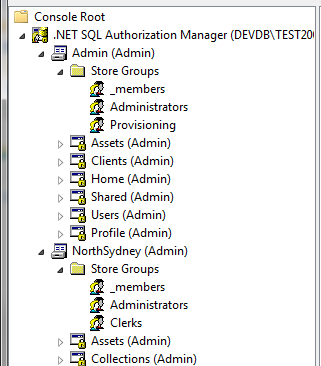
## NetSqlAzMan Management Tool and Menus

The following screen shots are provided along with some descriptive text to provide additional insight into the NetSqlAzMan-based menu system. This is not intended to be an exhaustive users guide but more of a short tutorial.



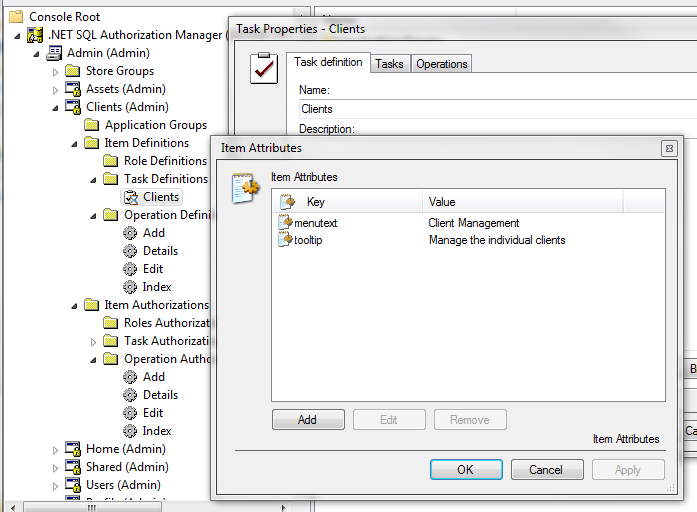
This is a typical Store in NetSqlAzMan. This Store is called ‘Admin’ and represents the authorization rights of resources and menu items under various functional groups (Applications). For Menu system, these groups need not directly represent menu items. These groups must represent Controllers from the MVC model.

The Application that is shown expanded, ‘Clients’, represents a set of functionality and a Controller in MVC called ClientController. This Application (ClientController in MVC) has constituent Operations (Actions in MVC) that will contain, at a minimum, Actions called Add, Details, Edit and Index. These Operations are where authorizations are ultimately applied.



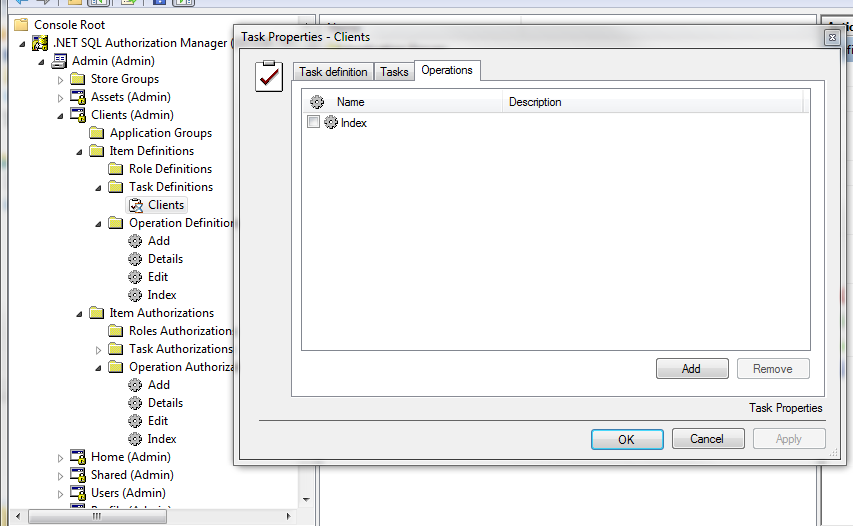
This shot shows Store Groups under two stores. These groups represent the Roles available for a user to join. Additional Roles can be created for each Store as needed. Each Store has one base group called ‘\_members’. All users created under this Store will be found in that Group. It is used for membership and not for authorization.

In the above example ‘Admin’ has two Roles created for it: ‘Administration’ and ‘Provisioning’. Users would be assigned to one or more groups and Operations would be authorized for these groups.

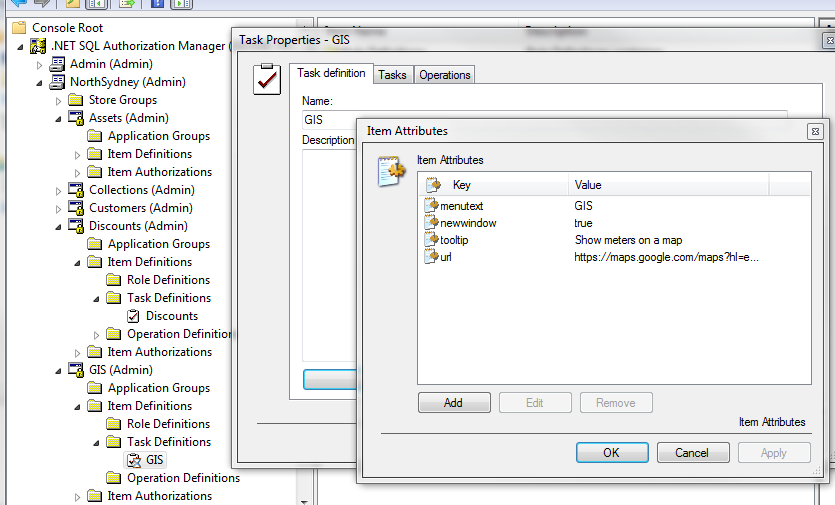


Tasks represent the actual menu item. In this shot, the Task called ‘Clients’ has several attributes that further provide configuration for the menu item. In this shot both the menu text attribute and the tool tip attribute can be seen.

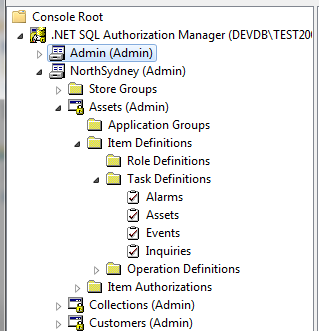
This task is also the one special condition task where it has the identical name to the Application. That indicates that this is a top-level menu item.



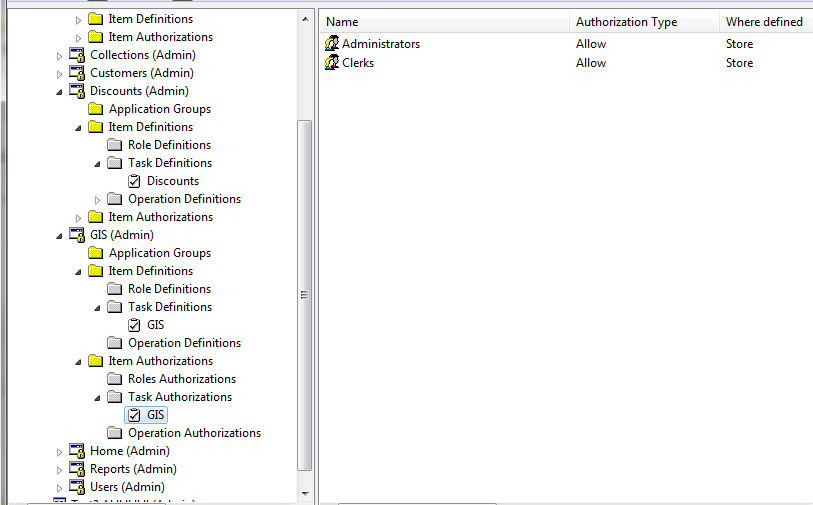
Continuing to examine this Task, this shot shows that there is an Operation assigned to the Task. This operation provided the information to create the URL that is ultimately selected on the menu. In this example, the menu would ultimately call a MVC Controller called ClientsController and request the Action called Index.



A Task (menu item) is not required to only access a MVC-based resource. This shot illustrates how additional Task attributes can point the menu to an external resource with attribute that indicates how to open the resource. The attribute system of NetSqlAzMan provides great extensibility to the menu system.



A menu item can contain sub-menus. In this shot the Assets Application for North Sydney has a top-level menu item which is indicated by the Task named the same as the application. There are several other Tasks seen in this shot which represent sub-menus. What is not shown is that the Task called Assets has sub-Tasks, Events Inquiries and Alarms that appear as child menu items. These relationship are configured in NetSqlAzMan and require no additional programming to implement for each Role.



This screen shot shows an example of Authorizations for a menu item. It is seen in the shot that the authorizations are at a Task level. This is important to note in this case since the Task does not have a child Operation where authorizations can be determined. This example is a menu item that points to an external resource.

