

Visual Access Controls

Developer's Guide

Table of Content

1	<i>Introduction</i>	<i>4</i>
2	<i>Supported Technologies.....</i>	<i>4</i>
3	<i>How to add Visual Access Controls to your application</i>	<i>4</i>
3.1	Prerequisite	4
3.2	Installation	4
4	<i>Components</i>	<i>5</i>
4.1	Web.Config	5
4.2	Database Objects	5
5	<i>Sample Application</i>	<i>6</i>
5.1	Installation	6
5.2	Default administrator's login to sample application	6
5.3	Web User Controls	6
5.3.1	Data Access Security.....	6
5.3.2	Module Access Security.....	6
6	<i>Data Access Security</i>	<i>7</i>
6.1	How does Data Access Security work.....	7
6.2	Functionalities.....	7
6.2.1	Maintain the data hierarchies.....	7
6.2.2	Assign data access permissions to ASP.net Roles	8
6.2.3	Add dynamic data access security constraint to "Select" statements.....	8
6.2.4	Import Data Hierarchy.....	8
6.3	Data Access Security -Object library.....	9
6.3.1	Display and maintain data hierarchy tree.....	9
1)	GetRootNode.....	9
2)	GetChildNodes.....	9
3)	GetNodeCode	9
4)	InsertNode	9
5)	UpdateNode	9
6)	DeleteNodeRecursive.....	10
7)	UndeleteNode.....	10
8)	UndeleteNodeRecursive.....	10
9)	InitNodeSecurityCode	10
10)	RebuildSecurityCode	10
6.3.2	Maintain Role permissions	10
1)	InsertRolePermission.....	10
2)	DeleteRolePermission	10
3)	DeleteAllRolePermission	10

4)	DeleteAllTablePermissions	11
5)	isNodeInRoleDasPermissions.....	11
6.3.3	SQL Builder	11
1)	Properties to add the Select query	11
2)	Properties to join a data hierarchy.....	11
3)	Properties to add the node name	12
4)	Function to build the query	12
7	Module Access Security.....	13
7.1	Developer's UI.....	13
7.1.1	Setup	13
7.1.2	Workflow and Functionalities	13
7.2	Administrator's UI.....	14
7.3	Module Access Security Objects Library.....	15
1)	AddFunction (Used in Developer's UI)	15
2)	AddFunctionToRole.....	15
3)	DeleteFunction (Used in Developer's UI)	15
4)	GetChildNodes.....	15
5)	GetFunctionPath	15
6)	GetRootNodes	15
7)	isFunctionAuthorized	15
8)	isFunctionAssignedToRole	16
9)	ModifyFunctionName (Used in Developer's UI)	16
10)	RemoveAllFunctionsFromRoleByRoleID.....	16
11)	RemoveAllFunctionsFromRoleByRoleName.....	16
12)	RemoveFunctionFromRole	16

1 Introduction

The Visual Access Controls enables you to deliver custom web applications with higher level of manageability.

- With **Module Access Security** you can easily map each function in the application to security switches that can be dynamically assigned to roles by the system administrator.
- With **Data Access Security** you can segregate the data between users belonging to different organizations while sharing the same database. The site administrator can create a standard ASP.Net Role then grant the Role access to part (or parts) of the Data Hierarchy. All users belonging to the Role will dynamically have access to a subset of the data based on the Role's Data Access Security permissions.

2 Supported Technologies

Visual Access Controls work hand-in-hand with the standard ASP.Net Membership and Role Manager providers. It provides tools that enable the system administrator to dynamically assign roles to system functionalities and access to specific data sets without the need to hardcode authorizations.

Visual Access Controls supports for VB.Net and C# custom web application development with SQL Server 2008 or higher.

3 How to add Visual Access Controls to your application

3.1 Prerequisite

- SQL Server 2008 or later, SQL Server Express 2008 or later
- Existing website application running in IIS and using ASP.Net authentication and authorization (i.e. ASP.Net Membership and Role Manager providers)

3.2 Installation

1. Database setup:
 - a. Open and execute **CreateVisualAccessControlsTable.sql** in SQL Server Management Studio (see section 4.2 for more info about the database tables)
2. Website setup:
 - a. Add a reference to **VisualAccessControls.dll** in your web application
 - b. Add **VisualAccessControlsLicenseKey** key and connection string name "**VacSqlServer**" to the Web.Config file (see section 4.1 for more info on how to modify Web.Config)
3. Copy Web User Controls from the Sample Apps (*optional*) :

The sample application comes with web user controls that include all the functionalities provided by Visual Access Controls for Module Access Security and Data Access Security. The code is available in C# and VB for ASP.Net Web Application project and ASP.Net Website project. For a quick start just select the sample application for the type of project and language you are using and copy **\App_VisualAccessControl** directory into to your root directory then use the available web user controls in your web pages

4 Components

4.1 Web.Config

The Web.Config must include the following elements:

1. License Key: Copy and paste your license key in this section

```
<appSettings>
  <add key="VisualAccessControlsLicenseKey" value="blablablablablablabla==" />
</appSettings>
```

2. Connection string with name = "**VacSqlServer**". For example:

```
<connectionStrings>
<add name="VacSqlServer"
  connectionString="Server=My Server;Database=VisualAccessControl;uid=Admin;pwd=123456;"
  providerName="System.Data.SqlClient"/>
</connectionStrings>
```

4.2 Database Objects

The Visual Access Controls are accompanied by 9 system tables prefixed by "vac_". The objects are created when you execute the script **CreateVisualAccessControlsTable.sql**

1. Module Access Security tables:

- a. **vac_ModuleAccessSecurityFunctions**: Holds the module hierarchy of the custom web application with the set of functions for each module
- b. **vac_ModuleAccessSecurity**: Used to store the relationship between the various roles and the assigned functions

2. Data Access Security tables:

- a. **vac_DataAccessSecurityTables**: Stores reference records to the four available data hierarchy tables namely:
 - i. Organizational Hierarchy
 - ii. Client Hierarchy
 - iii. Territory Hierarchy
 - iv. Entity Hierarchy
- b. **vac_DataAccessSecurity**: Used to store the relationship between the various roles and the assigned nodes
- c. **vac_DataAccessSecurityImport**: It is a staging table used for data import into the hierarchy tables
- d. Data hierarchy tables
 - i. **vac_DH_Organization**
 - ii. **vac_DH_Clients**
 - iii. **vac_DH_Territories**
 - iv. **vac_DH_Entities**

5 Sample Application

The sample application is an open source code intended for the developers to easily learn how to use the object library, copy snippets or even reuse the web user controls in their web applications.

5.1 Installation

Create a new empty database in SQL Server 2008 (or SQL Server 2008 Express edition) and execute the script **VisualAccessControlSampleDB_SQL2008.sql** to create the necessary database objects.

The sample application comes in 4 different project-type/language combinations:

- csWebbApp folder is an ASP.Net Web Application project in C#
- csWebsite folder in an AP.Net WebSite project in C#
- vbWebbApp folder is an ASP.Net Web Application project in VB.Net
- vbWebsite folder in an AP.Net WebSite project in VB.Net

Create a solution in Microsoft Visual Studio and add the corresponding project then proceed to edit the web.config file:

1. Add your license key
2. Edit the connection string to point to the newly created database

5.2 Default administrator's login to sample application

User name: **admin** Password: **admin**

5.3 Web User Controls

The Web User Controls delivered with the sample application include all the necessary functions to manage the application's Data Access Security and Module Access Security.

5.3.1 Data Access Security

- a. **DataHierarchy.ascx** : This web user control includes functionalities to add, delete and modify nodes in the data hierarchy. When adding the data hierarchy web user control, you have the choice to set the declarative parameter DataHierarchyTable to one of the 4 available data hierarchy tables:
 - Organizational Hierarchy
 - Client Hierarchy
 - Territory Hierarchy
 - Entity hierarchy

For example: `<uc1:DataHierarchy ID="DataHierarchy1" runat="server" DataHierarchyTable="Organization" />`

- b. **DataAccessSecurityPermissions.ascx** : This web user control enables the administrator to assign Data Access Security permissions to roles.
- c. **DataHierarchyImport.ascx**: This web user control includes the necessary functions to bulk import data in one of the data hierarchy tables (i.e. Organizations, Clients, Territories or Entities)

5.3.2 Module Access Security

- d. **ModuleAccessSecurityDevUI.ascx**: This control enables the Developer to define application functionalities in a hierarchy structure and generate the code corresponding to each of the function to be used during the development process. This module is intended to be used only by the developer in tandem with Visual Studio during the development phase of the project.
- e. **ModuleAccessSecurityPermissions.ascx**: This control enables the administrator to assign Module Access Security rights to roles.

6 Data Access Security

6.1 How does Data Access Security work

Assume you are building an application for a service company with offices all over the United States and you need to limit users to only view Service Requests pertaining to their location or region. You will need to include functionalities to dynamically add or remove offices and assign users to those offices. This can be achieved as follows:

1. Use the Organizational Hierarchy table provided with Visual Access Controls
2. Using the Data Access Security object library, include functionalities in your web application to add, delete and modify nodes (i.e. offices and regions) to the hierarchy.
3. Include a foreign Key in the target table that points to the primary key of your Hierarchy table. For example if you have a data table for “Service Requests”, you would add a column such as “Branch_id” to hold the node id from the Organizational hierarchy of the office where the service request belongs to.
4. Using the Data Access Security object library, include functionalities in your web application to assign one or more nodes from the hierarchy to an ASP.net Roles
5. Use the SQL Builder to create the select statement that populates the Grid view for Service Requests. As a result the end-user will only see Service Requests pertaining to his/her ASP.Net role

6.2 Functionalities

Data Access Security is driven by data Hierarchies. The Visual Access Controls comes with 4 Hierarchy tables (Organization, Clients, Territories and Entities). You can use one or more data hierarchies in your application. The object library includes function to:

1. Maintain data hierarchies
2. Assign data access permissions to ASP.net Roles
3. Add dynamic data access security constraint to Select statements

6.2.1 Maintain the data hierarchies

Using the Data Access Security library you can add, delete and modify nodes in the hierarchy. When adding a node, a function is called to create the security code. When deleting a node, the node is simply flagged as deleted and not removed from the table. Additional tools are provided to reset security codes and node path in the database tables

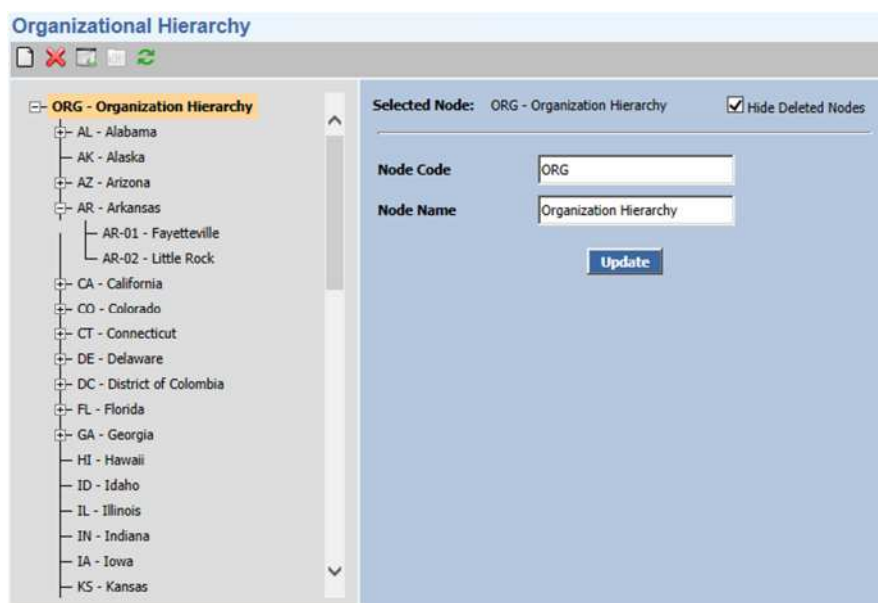


Figure-1 Organizational Hierarchy maintenance screen (sample application)

6.2.2 Assign data access permissions to ASP.net Roles

Using the Data Access Security object library you can display a data hierarchy and assign node permissions to ASP.Net roles. Assigning a higher level in the hierarchy implies that all child nodes are accessible to the role.



Figure-2 Data Access Security permissions (sample application)

6.2.3 Add dynamic data access security constraint to “Select” statements

The Data Access Security constraint is set on the select statement to limit the result to a subset of data allowed by the user’s role. This is applicable when populating Grid View, Drop Down List and any other objects.

6.2.4 Import Data Hierarchy

The Sample application includes a web user control to import data hierarchy from a pipe delimited flat file. The structure of the flat file consists of:

- **Code** – String – max length 25 Characters
- **Code Name** – String – max length 256 Characters
- **Parent Code** – String – max length 25 Characters



Figure-3 Import Data Hierarchy

Here is an example pipe delimited file content where T000 is the code for the root node in the Territory hierarchy:

```
AL|Alabama|T000
AK|Alaska|T000
AL-01|Anniston|AL
AL-02|Birmingham|AL
AL-03|Dothan|AL
```


6.3 Data Access Security -Object library

6.3.1 Display and maintain data hierarchy tree

1) *GetRootNode*

Used to populate the root level of a hierarchy tree

Parameters:

- String *tableName*: the data hierarchy table name
- String *order*: takes one of 2 values "Name" or "Code"

Return SQLDataAdapter

2) *GetChildNodes*

Used to populate the child nodes in a hierarchy tree

Parameters:

- String *parentID*: Parent Node Id
- String *tableName*: the data hierarchy table name
- Boolean *excludeDeletedNodes*: Hides deleted nodes from hierarchy (True or False)
- String *order*: takes one of 2 values "Name" or "Code"

Return SQLDataAdapter

3) *GetNodeCode*

Retrieve the short code for the tree node

Parameters:

- String *parentID*: Parent Node Id
- String *ItemID*: Node Id

Return String

4) *InsertNode*

Add a node to the hierarchy

Parameters:

- String *tableName*: the data hierarchy table name
- String *itemName*: Node Title
- String *itemCode*: Node short code
- Integer *itemParentID*: Parent Node Id

Return Integer

5) *UpdateNode*

Update tree node name and short code

Parameters:

- String *tableName*: the data hierarchy table name
- String *itemName*: Node Title
- String *itemCode*: Node short code
- Integer *itemID*: Tree Node Id

6) *DeleteNodeRecursive*

Flag nodes as deleted including all children nodes

Parameters:

- String *tableName*: the data hierarchy table name
- Integer *itemID*: Tree Node Id

7) *UndeleteNode*

Remove the Deleted Flag from the node and its parent nodes if applicable

Parameters:

- String *tableName*: the data hierarchy table name
- Integer *itemID*: Tree Node Id

8) *UndeleteNodeRecursive*

Remove the DeletedFlag from the node and child nodes

Parameters:

- String *tableName*: the data hierarchy table name
- Integer *itemID*: Tree Node Id

9) *InitNodeSecurityCode*

Should be called after a new node is inserted in the data hierarchy to initialize the node security settings

Parameters:

- Integer *itemID*: Tree Node Id
- String *tableName*: the data hierarchy table name

10) *RebuildSecurityCode*

Resets the Security Codes for all nodes and member roles

Parameters:

- String *tableName*: the data hierarchy table name

6.3.2 Maintain Role permissions

1) *InsertRolePermission*

Add node to role permissions

Parameters:

- String *tableName*: the data hierarchy table name
- String *itemID*: Tree Node Id
- String *roleID*: ASP.Net RoleID

2) *DeleteRolePermission*

Remove node from role permissions

Parameters:

- String *tableName*: the data hierarchy table name
- String *itemID*: Tree Node Id
- String *roleID*: ASP.Net RoleID

3) *DeleteAllRolePermission*

Remove all nodes from role permissions

Parameters:

- String *roleID*: ASP.Net RoleID

4) *DeleteAllTablePermissions*

Deletes all role permissions related to the specified data hierarchy table

Parameters:

- String *tableName*: the data hierarchy table name

5) *isNodeInRoleDasPermissions*

Find out if a node is checked while building the data hierarchy tree

Parameters:

- Integer *itemID*: Tree Node Id
- String *tableName*: the data hierarchy table name
- String *roleID*: ASP.Net RoleID

Return Boolean

6.3.3 SQL Builder

Use the SQLBuilder class to add Data Access Security to your select statement (for example when populating a grid). The class exposes a number of properties to join your select statement to the data hierarchy and a function that returns the joint SQL statement with embedded Data Access Security controls restricting the user to only view data pertaining to his/her associated role(s).

1) *Properties to add the Select query*

a) *SelectStatement (required)*

Add the SELECT statement as a string. Do not include [WHERE],[HAVING],[ORDERBY]

Property SelectStatement() *As String*

b) *WhereClause*

(Optional) Add the WHERE clause of the SQL query as string

Property WhereClause () *As String*

c) *HavingClause*

(Optional) Add the HAVING clause of the SQL query as string

Property HavingClause() *As String*

d) *OrderClause*

(Optional) Add the ORDER BY clause of the SQL query as string

Property OrderClause () *As String*

2) *Properties to join a data hierarchy*

One of the following properties is required for the BuildSqlQuery function.

a) *JoinToOrganizationalHierarchyOn*

Property JoinToOrganizationalHierarchyOn() *As String*

Assign the value of your [Table Alias].[ColumnName] equivalent to vac_DH_Organization.ItemID

b) *JoinToClientHierarchyOn*

Assign the value of your [Table Alias].[ColumnName] equivalent to vac_DH_Client.ItemID

Property JoinToClientHierarchyOn() *As String*

c) *JoinToTerritoryHierarchyOn*

Assign the value of your [Table Alias].[ColumnName] equivalent to vac_DH_Territory.ItemID

Property JoinToTerritoryHierarchyOn () *As String*

d) *JoinToEntityHierarchyOn*

Assign the value of your [Table Alias].[ColumnName] equivalent to vac_DH_Entity.ItemID

Property JoinToEntityHierarchyOn () *As String*

3) *Properties to add the node name*

The following properties are used to append the column “NodeName” to the list of column names in the select statement.

- a) AddOrganizationalHierarchyNodeName
(Optional) Add the Organizational Hierarchy node name to the SELECT statement as "NodeName"
`Property AddOrganizationalHierarchyNodeName () As Boolean`
- b) AddClientHierarchyNodeName
(Optional) Add the Client Hierarchy node name to the SELECT statement as "ClientName"
`Property AddClientHierarchyNodeName () As Boolean`
- c) AddTerritoryHierarchyNodeName
(Optional) Add the Territory Hierarchy node name to the SELECT statement as "TerritoryName"
`Property AddTerritoryHierarchyNodeName () As Boolean`
- d) AddEntityHierarchyNodeName
(Optional) Add the Entity Hierarchy node name to the SELECT statement as "EntityName"
`Property AddEntityHierarchyNodeName () As Boolean`

4) *Function to build the query*

- a) BuildSqlQuery
Requires the [SELECTStatement] and [JoinTo...Hierarchy] properties
`Function BuildSqlQuery() As String`
Return String

7 Module Access Security

The Module Access Security consists of two sets of functionalities:

1. Functionalities for the developer to use while developing the web app. It is recommended to simply re-use the web user control **ModuleAccessSecurityDevUI.ascx** available in the sample application rather than recreate the tools.
2. Functionalities for the Administrator to use to assign permissions to roles.

7.1 Developer's UI

7.1.1 Setup

The Developer's UI is setup within the web application then removed or hidden from use before publishing the application to a production environment. You can create your own Developer's UI or reuse the web user control available in the sample web application:

`\App_VisualAccessControl\UserControls\ModuleAccessSecurity\Developer\ModuleAccessSecurityDevUI.ascx`

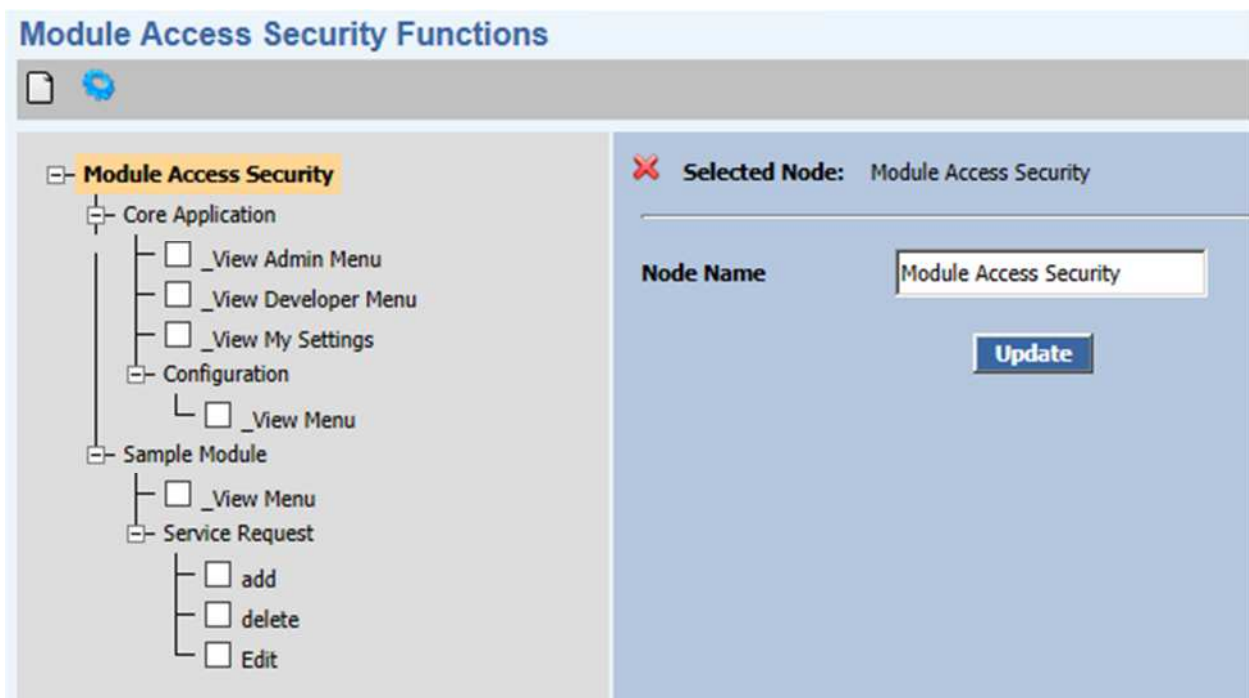



Figure-1 Developer's UI

7.1.2 Workflow and Functionalities

Once the developer's UI is added to the web application, the developer would use it in tandem with Visual Studio. For example, assume that you need to create a form to manage Service Requests with module access security controls to limit who can do what on this form. Here is what you would need to do:

1. Create a Web form in visual studio to manage Service Requests then add a grid with a series of action buttons to Add, Delete and Edit a Service Request record.
2. Launch the web application in a browser, navigate to the Developer's UI for Module Access Security then add nodes in a tree view display for the function to Add, Delete and Edit a Service request.
3. Check the box for one or more function nodes then select  to generate snippet code as depicted in figure-4.

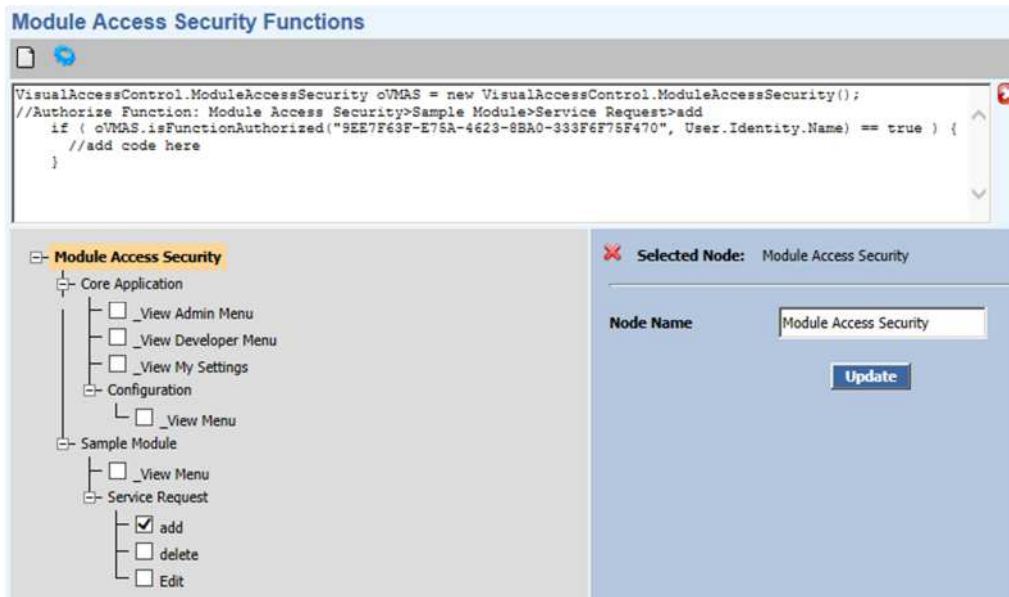


Figure-4 Developer's UI with snippet code

- Copy and paste the code in Visual Studio then proceed to add the business logic within the IF statement where it indicates "add code here" implying that your code will execute if the logged in User is authorized to use this function.

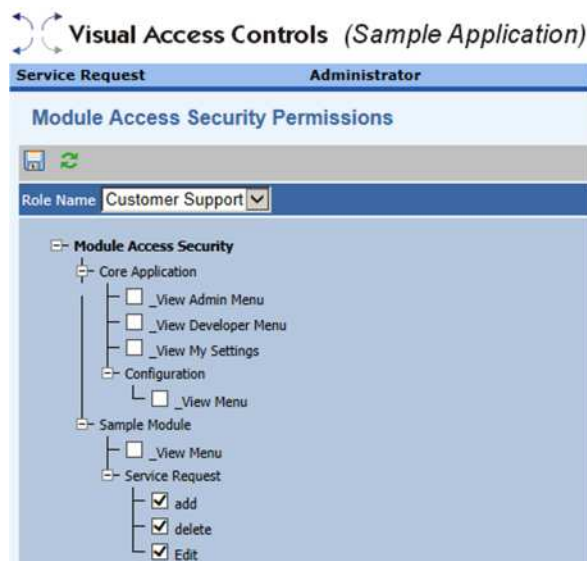
```
VisualAccessControl.ModuleAccessSecurity oVMAS = new VisualAccessControl.ModuleAccessSecurity();
//Authorize Function: Module Access Security>Sample Module>Service Request>add
if (oVMAS.isFunctionAuthorized("9EE7F63F-E75A-4623-8BA0-333F6F75F470", User.Identity.Name) == true) {
    //add code here
}
```

7.2 Administrator's UI

Module Access Security permissions are set at the Role level. Roles are created and managed using the standard ASP.Net Role Manager provider. Examples of the functions available to the administrator are in the web user control

`\App_VisualAccessControl\UserControls\ModuleAccessSecurity\Administrator\ModuleAccessSecurityPermissions.ascx`

In essence, the Administrator will be able to select a Role and assign permissions to the role as depicted hereunder:



7.3 Module Access Security Objects Library

The developer's UI is a web form added to the web application and used by the developer to define the hierarchy tree of the various pages and functions embedded in the target web application.

1) *AddFunction (Used in Developer's UI)*

Add a new function or function group to the functions hierarchy tree.

Parameters:

- String *masName*: the name of the function or page you want to include in the hierarchy
- String *masParentId*: the uniqueidentifier of the parent function node

2) *AddFunctionToRole*

Associate a function with a role

Parameters:

- String *roleId*: the uniqueidentifier of the role
- String *masID*: the uniqueidentifier of the function

3) *DeleteFunction (Used in Developer's UI)*

Deletes selected function node and all child nodes.

Parameters:

- String *masID*: The unique identifier of the selected function node

4) *GetChildNodes*

Used to populate and display the child nodes in a hierarchy tree.

Parameters:

- String *parentId*: The uniqueidentifier of the parent node

5) *GetFunctionPath*

Used when generating the code snippet to display the hierarchy of the function.

Parameters:

- String *masID*: The uniqueidentifier of the function

6) *GetRootNodes*

Used to populate and display the root level of the functions hierarchy tree.

Parameters:

- none

7) *isFunctionAuthorized*

Check if function is associated to one of the user's roles.

Parameters:

- String *masID*: The uniqueidentifier of the function
- String *username*: The login id of the logged in user

Returns "True" if the function is authorized for the user to access. Otherwise returns "False".

8) *isFunctionAssignedToRole*

To check if a function is associated with a role

Parameters:

- String *masID*: The unique id of the function
- String *rolename*: The role name

Returns “True” or “False”

9) *ModifyFunctionName (Used in Developer’s UI)*

Update the name of the tree node.

Parameters:

- String *masName*: the new name or title of the module access security node
- String *masParentId*: the uniqueidentifier of the parent function node

10) *RemoveAllFunctionsFromRoleByRoleID*

Remove All Role Permissions by role id. Usually used before deleting a role to avoid orphan records

Parameters:

- String *roleid*: The uniqueidentifier of the role.

11) *RemoveAllFunctionsFromRoleByRoleName*

Remove All Role Permissions by role name. Usually used before deleting a role to avoid orphan records

Parameters:

- String *roleName*: The role name.

12) *RemoveFunctionFromRole*

Remove permission from a role

Parameters:

- String *roleID*: The uniqueidentifier of the role.
- String *masID*: The uniqueidentifier of the function.