TaskTracker Sample Application: ReadMe

This TaskTracker sample application demonstrates how to use Windows Azure Active Directory (AD) for authorization. TaskTracker is a Windows Azure Active Directory ASP.NET application that uses Role-Based Access Control (RBAC) and ACLs to ensure that only authorized users can access secured resources. You can use one or both of these techniques in your Windows Azure AD apps, or adapt your own methods to permit selective access to resources.

The TaskTracker sample application includes a Roles.xml file that stores a mapping of users and Active Directory security groups to special roles defined for this application. It uses features of the Microsoft .NET framework to enforce policies based on the user’s application roles.

This sample also demonstrates how to use an access control list (ACL) to manage access to application features. TaskTracker allows only users and Active Directory security groups listed in the ACL to view the Users page of the app. User with the Admin app role can change the ACL and role mappings, and the changes are saved in the corresponding XML files.

# Role Based Access Control

Role-based access control (RBAC) is commonly used to ensure that only authorized users can access secured resources. When using RBAC, you grant permissions to roles, not to individual users or groups. Then, when you assign roles to users and groups, the permissions that are granted to a role are granted to the users and groups in that role.

Most systems define permissions in the form of policies in which a particular action occurs only when a user or group has a particular role. The action might include granting access to resources, changing functionality, or even changing the user interface for that user.

This app defines four application roles: Admin, Observer, Writer, and Approver, and provides a different experience for users in each role. Take care not to confuse these application roles with the built-in directory roles that Windows Azure defines for its users, such as Company Administrator and Helpdesk Administrator. For a complete list of the built-in directory roles, use the [Get-MsolRole](http://technet.microsoft.com/en-us/library/dn194100.aspx) cmdlet.

# Determine a User’s Roles at Login

Let's begin with a walk through the code. If you want to get started right away, you can skip to the [Requirements](#BKMK_Requirements) section and then come back and read while you review the code.

The TaskTracker sample application uses [Windows Identity Foundation](http://msdn.microsoft.com/en-us/library/ee748475.aspx) (WIF) to authenticate users from Windows Azure AD. We'll extend WIF by adding role and group claims to the [ClaimsPrincipal](http://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx) object that represents the user.

As soon as a user logs in, the TaskTracker sample app gets the user's application role. To ensures that this is executed at login, in Web.config, we add the GraphClaimsAuthenticationManager class, a subclass of [ClaimsAuthenticationManager](http://msdn.microsoft.com/en-us/library/system.security.claims.claimsauthenticationmanager.aspx), to the WIF pipeline.

//From: web.config

<system.identityModel>

<identityConfiguration>

<claimsAuthenticationManager type="TaskTracker.GraphClaimsAuthenticationManager,TaskTracker" />

...

**Authenticate()**, the main function of the GraphClaimsAuthenticationManager class, begins by querying the Windows Azure AD Graph to determine the user’s built-in Windows Azure AD roles. If the user has the "Company Administrator" built-in role, which corresponds to the Global Administrator organizational role in the Windows Azure Management Portal, the app assigns the app-specific “Admin” role to that user.

//From: GraphClaimsAuthenticationManager.cs

public override ClaimsPrincipal Authenticate(string resourceName, ClaimsPrincipal incomingPrincipal

{

...

// get the user's built-in roles

new WebRetryHelper<object>(() => graphService.LoadProperty(currentUser, "memberOf"));

List<Role> currentRoles = currentUser.memberOf.OfType<Role>().ToList();

//if the user is a Company Administrator (Global Administrator),

//assign them the "Admin" role in the app.

foreach(Role role in currentRoles)

{

if (role.displayName.Equals("Company Administrator"))

{

((ClaimsIdentity)incomingPrincipal.Identity).AddClaim(new Claim(ClaimTypes.Role, "Admin", ClaimValueTypes.String, "TaskTrackerSampleApplication"));

}

}

...

Next, it gets the user's group memberships, stores them in a "Group" type claim, and adds the claim to the [ClaimsIdentity](http://msdn.microsoft.com/en-us/library/microsoft.identitymodel.claims.claimsidentity.aspx) class. To get the user's groups, including groups that the user belongs to because they are members of another group, it uses the **getCompleteGroupMembership** function (defined in DirectoryDataService\_partial.cs), which calls the [**getMemberGroups**](http://msdn.microsoft.com/en-us/library/dn424889.aspx)REST API function, which returns all groups to which the user belongs. For more information about the Windows Azure AD Graph REST API, see [Windows Azure Active Directory Graph](http://msdn.microsoft.com/en-us/library/hh974476.aspx).

From: GraphClaimsAuthenticationManager.cs

// Now, query transitive group membership of the user

List<string> completeGroupMembership = new WebRetryHelper<List<String>>(() => graphService.getCompleteGroupMembership(tenantId, currentUserObjectId, token)).Value;

//Store the user's groups as claims of type "Group"

foreach (string groupId in completeGroupMembership)

{

Debug.WriteLine("adding " + groupId);

((ClaimsIdentity)incomingPrincipal.Identity).AddClaim(new Claim("Group", groupId, ClaimValueTypes.String, "WindowsAzureADGraph"));

}

Then it determines the user's application roles by comparing the user’s group membership with the role mappings stored in the Roles.xml file and creates a "Role" claim.

From: GraphClaimsAuthenticationManager.cs

//Get role assignments

foreach(string role in getRoles(currentUserObjectId, completeGroupMembership))

{

//Store the user's application roles as claims of type Role

((ClaimsIdentity)incomingPrincipal.Identity).AddClaim(new Claim(ClaimTypes.Role, role, ClaimValueTypes.String, "TaskTrackerSampleApplication"));

}

# Use Application Roles to Enforce Policy

The TaskTracker application is a to-do list, but the user's roles determine which app features they can view and access. In RoleMapElem.cs (/Models), the application defines the four roles to which users can be assigned:

* **Admin**: Full control, including permission to assign role mappings, and change the ACL.
* **Writer**: Add tasks to the to-do list and set the task status.
* **Approver**: Change the status of tasks
* **Observer**: View, but not change, the to-do list

To enforce access to resources, the application uses the [ClaimsPrincipal.IsInRole](http://msdn.microsoft.com/en-us/library/microsoft.identitymodel.claims.claimsprincipal.isinrole.aspx)() method. This method returns True if the user has a role claim that matches the string parameter passed to the IsInRole() function, and returns False otherwise.

For example, in TaskTrackerController.cs, the IsInRole() method determines whether the user has the Admin or Writer role. If they do, they are permitted to add a task.

//From: TaskTrackerController.cs

if (User.IsInRole("Admin") || User.IsInRole("Writer"))

{

//add new task

if (formCollection["newTask"] != null && formCollection["newTask"].Length != 0)

{

XmlHelper.AppendTaskElemToXml(formCollection);

}

}

The TaskTracker application also uses the [Authorize attribute](http://msdn.microsoft.com/en-us/library/system.web.mvc.authorizeattribute(v=vs.108).aspx) to enforce access control. For example, in RoleController.cs, [Authorize(Roles = "Admin")] grants access to the RoleMappings controller/page only to users with the Admin role.

//From: RoleController.cs

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult ACLSubmit(FormCollection formCollection)

{

//remove ACL assignments marked by checkboxes

XmlHelper.RemoveACLElemFromXml(formCollection);

...

# Use an Access Control List to Enforce Access

In addition to RBAC, this app also demonstrates how to use an access control list (ACL) to restrict access to resources. In this app, only users and members of groups listed in the ACL can see the contents of the Users page. To change the ACL, use the **Users Resource Access Control List** feature of the RoleMappings page. Only users with the Admin application role can edit the ACL. All changes to the ACL are saved in the ACL.xml file in the App\_Data directory.

The ACL-based permission in the application works by storing the user’s group memberships as claims of type "Group" at sign on. Then, when the app needs to determine whether to allow you to see the Users page, it cross-references the user’s group membership claims with the stored ACL.

Because access to the Users page is determined entirely by the ACL, not by roles, even users with the Admin app role cannot view the Users page unless they (or a group to which they belong) appear in the ACL.

//From: HomeController.cs

[HttpGet]

public ActionResult Users()

{

...

//is user's objectId in the ACL?

if (elem.ObjectId.Equals(userObjectId))

{

onACL = true;

}

else

{

foreach (Claim groupClaim in ((ClaimsIdentity)User.Identity).FindAll("Group"))

{

//is a group the user belongs to in the ACL?

if (elem.ObjectId.Equals(groupClaim.Value))

{

onACL = true;

break;

}

}

}

...

//if user is not in ACL, do not grant permission

if (!onACL)

{

return RedirectToAction("Error", "Home", new { errorMessage = "Access Denied. To view this resource, have an admin add you or your group to the ACL." });

}

...

# Requirements

The following resources are required to run the TaskTracker sample application.

* An Internet connection
* Visual Studio 2012
* [Identity and Access Tool](http://go.microsoft.com/fwlink/?LinkID=245849) for Visual Studio 2012
* An active Windows Azure subscription. If you don't have one, can get a [90-day free trial](http://www.windowsazure.com/en-us/pricing/free-trial/).
* A Windows Azure AD tenant. If you don't have a tenant, follow the detailed instructions in the "How to get a Windows Azure AD tenant" section of [What is a Windows Azure AD tenant?](http://go.microsoft.com/fwlink/?LinkID=288845) To see these instructions used in the context of application development, see "Working with Your Windows Azure AD Directory Tenant" in [Adding Sign-On to You Web Application Using Windows Azure AD](http://msdn.microsoft.com/en-us/library/dn151790.aspx).

# Recommendations

* [Using the Graph API to Query Windows Azure AD Walkthrough](http://msdn.microsoft.com/en-us/library/windowsazure/dn151791.aspx). This sample builds directly upon the results of this walkthrough.

# XML Files

The Task Tracker application stores its persistent data in XML files in the TaskTracker/AppData directory. The app updates these files as you use it. We have also included a set of blank files in the TaskTracker/AppData/ App\_Data Files Backup directory for subsequent trials.

* **ACL.xml:** Contains entries that determine whether users have access to the Users page in the app. Because the ACL is independent of roles, users with the Admin app role can give users and groups access to the resource without changing their role membership.   
    
  When the application starts, the ACL has no user entries, so no one can view the content of the Users page. Users with the Admin app role can use the use **Users Resource Access Control List** section of the RoleMappings page to add users and groups to the ACL.
* **Roles.xml**: Contains elements that map users (by user ID) and groups with one of the four application roles: Admin, Writer, Approver, and Observer. When the application start, only the Admin role is assigned, but the Admin can use the features on the Users page to assign other users to roles. The role assignments are saved in the Roles.xml file.
* **Tasks.xml**: Stores the tasks that the Task Tracker app is tracking and their status. When you use the app features to add tasks to the Task Tracker, the tasks are saved in the Tasks.xml file.

# Prepare to Run the Sample

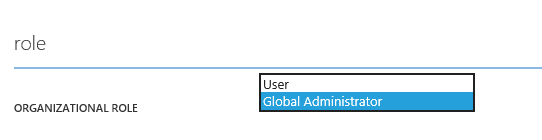
In this section, we'll get you ready to run the TaskTracker sample app. We'll select a Windows Azure AD tenant and directory, add a user with an organizational account (if you don't already have one), and add the TaskTracker app to your Windows Azure AD tenant.

## Select a Directory and Add Users

Before we run the sample, you need to select a Windows Azure AD directory. If you don't yet have a Windows Azure AD tenant that contains a directory, follow the detailed instructions in the "How to get a Windows Azure AD tenant" section of [What is a Windows Azure AD tenant?](http://go.microsoft.com/fwlink/?LinkID=288845) To sign into the TaskTracker app, you must have an organizational account, that is, a sign-in address in an enterprise domain, such as user@contoso.com or user@contoso.onmicrosoft.com. You cannot use Microsoft account address, such as user@outlook.com or user@live.com.

To create a user with an organizational account, use the "Add a User" instructions in the "Create a New Directory Tenant and Add a User" section of [Adding Sign-On to You Web Application Using Windows Azure AD](http://msdn.microsoft.com/en-us/library/dn151790.aspx).

To test the app, create a user with the Global Administrator organizational role, which corresponds to the Company Administrator built-in role and the Admin application role, and a user with the User organizational role. To set the user roles, use the Organizational Role field in the Windows Azure Management Portal.



During your test, the user with the Admin application role can use the RoleMappings page to change the application role of the user with the User role.

## Create a Security Group

The TaskTracker app lets you assign application roles to Active Directory security groups. Users in the security group have the privileges that are granted to the application role. You can also add a security group to the ACL. Members of group have access to the Users page that the ACL controls.

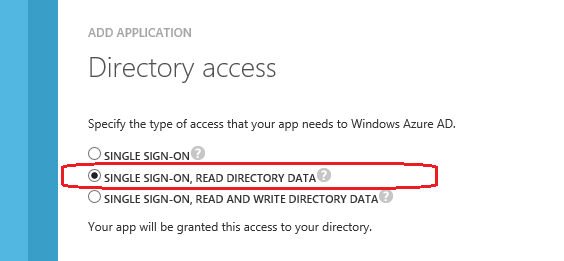
To test the ACL implementation, use the Group Management features of the Windows Azure Management Portal to add a security group to your Windows Azure AD tenant. For instructions, see [Add a group](http://go.microsoft.com/fwlink/?LinkID=331384).

## Add Task Tracker to Windows Azure AD

Next, you need to add the TaskTracker sample app to your Windows Azure AD tenant.

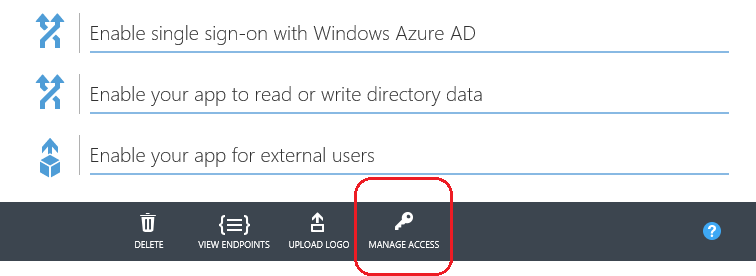
Follow the steps in “Register a New Application” section of [Adding Sign-On to You Web Application Using Windows Azure AD](http://msdn.microsoft.com/en-us/library/dn151790.aspx) to add a new web application.

However, on the Directory Access page, select the **Single Sign-on, Read Directory Data** option. This option allows the app to query the Windows Azure AD Graph. Name the application TaskTracker.



You can view and change the directory access of the TaskTracker application at any time.

1. Go to the [Windows Azure Management Portal](https://manage.WindowsAzure.com) and sign in.
2. Click **Active Directory**, double-click a directory, click **Applications**, and then click TaskTracker.
3. Click **Manage Access** and then click **Change the directory access for this app**.



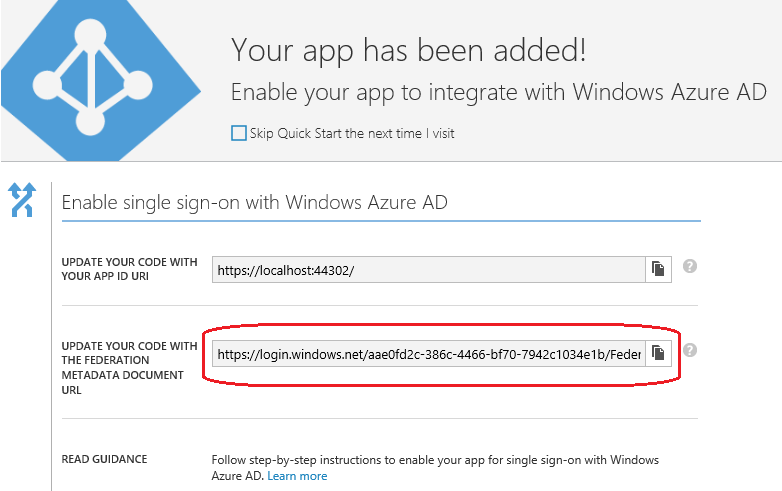
1. On the **Directory access** page, select a new access type and click the check mark, or , click the X to close the page without changing the access type.

## Customize the App for your Tenant

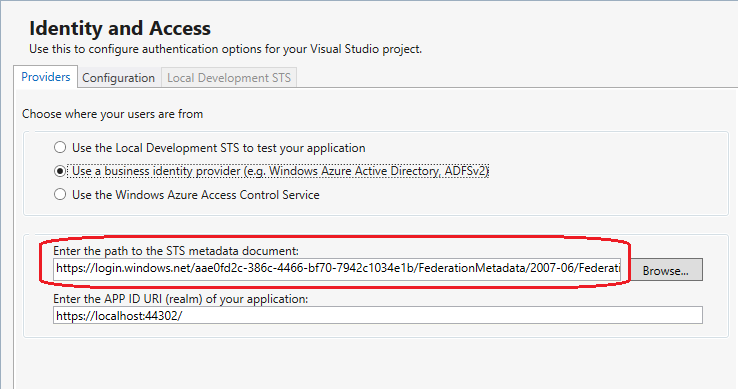
Before you run the Task Tracker app, you need to update the web.config file with information about your Windows Azure AD tenant. We'll use the Identity and Access tool in Visual Studio 2012 to make most of the changes for us.

We'll begin by copying some information from the Windows Azure AD tenant.

1. Go to the [Windows Azure Management Portal](https://manage.WindowsAzure.com) and sign in.
2. Click **Active Directory**, double-click a directory, click **Applications**, and then click TaskTracker.
3. Copy the Federation Metadata URL of the app. You'll find it in the value of the **Update Your Code with the Federation Metadata Document URL** field.



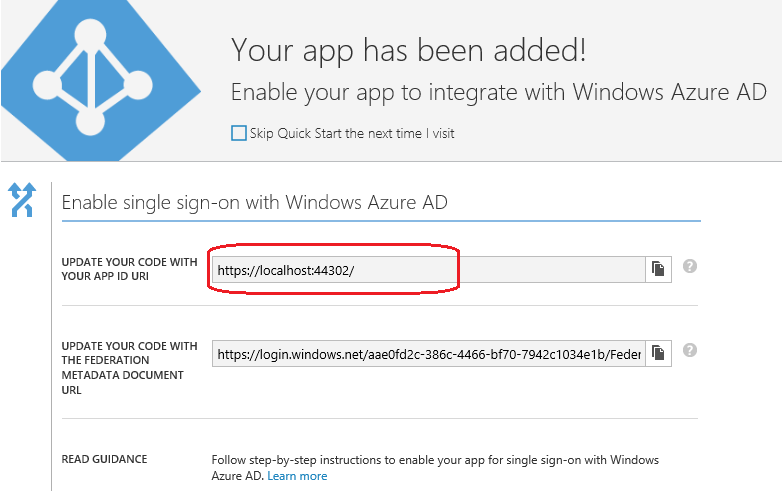
1. In Visual Studio 2012, start the TaskTracker application.
2. In Solution Explorer, right-click TaskTracker, and then click **Identity and Access**. (If this item is not in the menu, it might not be installed. To install, see [Identity and Access Tool](http://go.microsoft.com/fwlink/?LinkID=245849).
3. In the Identity and Access window, on the Providers tab, select **Use a business identity provider**.
4. In the **Enter the path to the STS metadata document** field, paste the Federation Metadata URL.



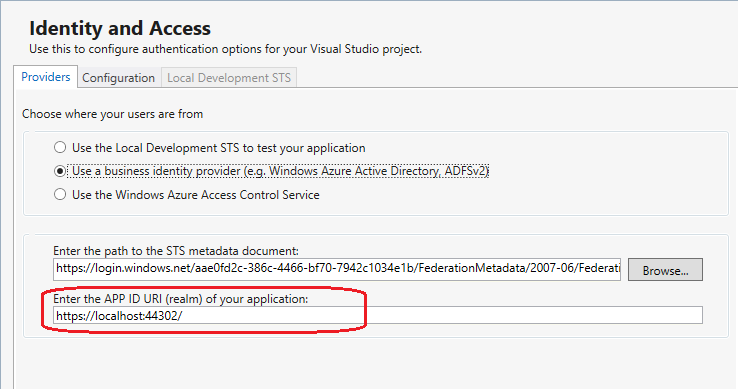
Note: This page might warn you that the test certificate that we used for the sample is not valid.

1. Go back to the TaskTracker application page in the Windows Azure Management portal.   
   Copy the App ID URI of the application from the **Update Your Code with your App ID Uri** field.

The App ID URI and Federation Metadata URL are added when you add your application to Windows Azure AD. In this example, the app is using port 44302, but the port number will vary.



1. Now, back in Visual Studio, paste the App ID Uri in the **Enter the APP ID URI (realm) of your application** field.



1. Click OK.

The Identity and Access Tool has entered information from your Windows Azure AD tenant into the appSettings element of the web.config file. The values in the key=value pairs should now match the tenant value.

The following is a sample appSettings element. The values of the keys vary with each tenant.

<appSettings>

<add key="ClientId" value="**d1499339-8059-4558-9837-d1eb05eef085**" />

<add key="Password" value="4t4hnaDZ8y7FgI50CMTmu03B3Ss2sbXvBSj1axn18Mc=" />

<add key="TenantDomainName" value="Contoso.onmicrosoft.com" />

<add key="webpages:Version" value="2.0.0.0" />

<add key="webpages:Enabled" value="false" />

<add key="PreserveLoginUrl" value="true" />

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

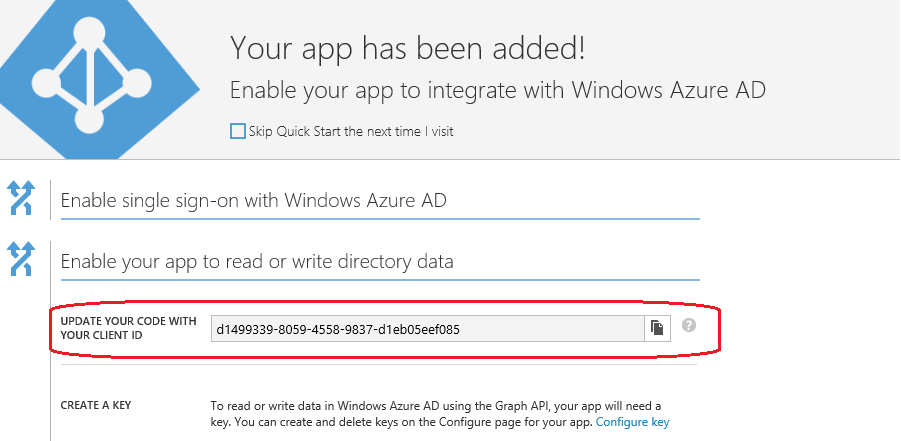
<add key="ida:FederationMetadataLocation" value="https://login.windows.net/aae0fd2c-375a-4466-bf60-7942a1034e1b/FederationMetadata/2007-06/FederationMetadata.xml" />

<add key="ida:Issuer" value="https://login.windows.net/aae0fd2c-375a-4466-bf60-7942a1034e1b/wsfed" />

<add key="ida:ProviderSelection" value="productionSTS" />

</appSettings>

1. Replace the ClientId value in appSettings with the client ID of your Windows Azure AD tenant. To find your client ID, on the TaskTracker page of the Windows Azure Management Portal, Expand the **Enable your app to read or write directory data** section. Copy the value of the **Update your code with your client ID** field.

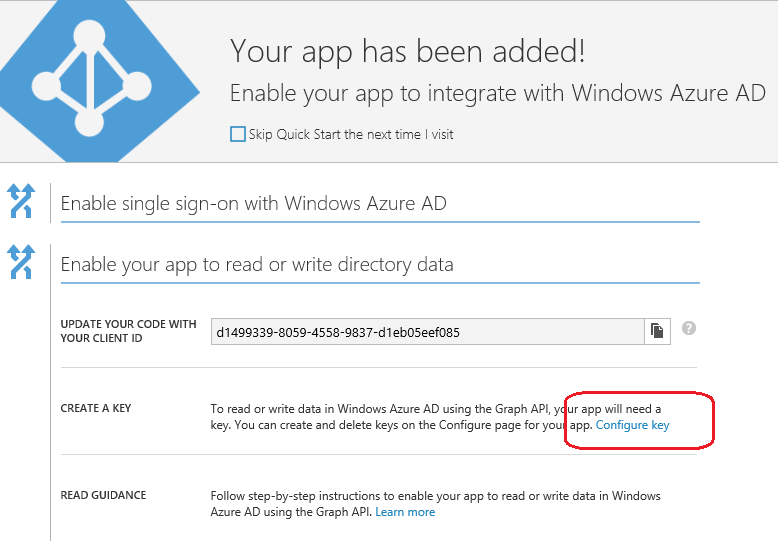


Paste the client ID in the value of the ClientId key in the appSettings element of your web.config file.

<appSettings>

<add key="ClientId" value="d1499339-8059-4558-9837-d1eb05eef085" />

1. Replace the Password value with the value of the key for your instance of the app. On the TaskTracker page of the Windows Azure Management Portal, Expand the **Enable your app to read or write directory data** section. Click **Configure key**, scroll to the Keys section, and copy the key value.



Paste the key in the value of the Password key in appSettings.

<appSettings>

<add key="ClientId" value="d1499339-8059-4558-9837-d1eb05eef085" />

<add key="Password" value="4t4hnaDZ8y7FgI50CMTmu03B3Ss2sbXvBSj1axn18Mc=" />

1. Replace the value of the TenantDomainName key with the name of your tenant. Enter the full name, including the "onmicrosoft.com" if applicable.

<appSettings>

<add key="ClientId" value="d1499339-8059-4558-9837-d1eb05eef085" />

<add key="Password" value="4t4hnaDZ8y7FgI50CMTmu03B3Ss2sbXvBSj1axn18Mc=" />

<add key="TenantDomainName" value="Contoso.onmicrosoft.com" />

1. Verify that the GUID in the **authority name** and **validIssuers** name values of the **issuerNameRegistry** element is the same as the GUID of your Federation Metadata Document URL. You can find the Federation Metadata Document URL on the TaskTracker application page of the Windows Azure Management portal.

For example, if the Federation Metadata XML URL is

https://login.windows.net/aae0fd2c-386c-4466-bf70-7942c1034e1b/FederationMetadata/2007-06/FederationMetadata.xml

the GUID is:

aae0fd2c-386c-4466-bf70-7942c1034e1b

If the GUID in the **authority name** or **validIssuers** values does not match, replace it with the GUID of your Federation Metadata XML URL.

<issuerNameRegistry type="System.IdentityModel.Tokens.ValidatingIssuerNameRegistry,...">

<authority name="https://sts.windows.net/aae0fd2c-386c-4466-bf70-7942c1034e1b/">

<keys></keys>

<validIssuers>

<add name="https://sts.windows.net/aae0fd2c-386c-4466-bf70-7942c1034e1b/" />

</validIssuers>

</authority>

</issuerNameRegistry>

...

1. Verify that the GUID in the value of the **issuer** attribute of the **wsFederation** element is the same as the GUID of your Federation Metadata XML URL. If it is not the same, replace the GUID in the **issuer** attribute with the GUID of your Federation Metadata XML URL.

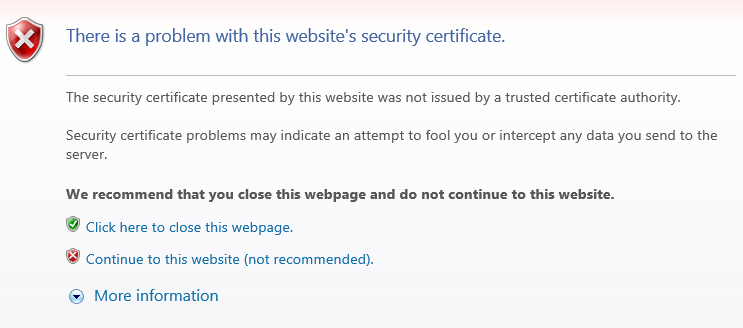
<wsFederation passiveRedirectEnabled="true" issuer="https://login.windows.net/aae0fd2c-386c-4466-bf70-7942c1034e1b/wsfed" realm="https://localhost:44302/" requireHttps="false" />

# Running the TaskTracker App

Now you can run the TaskTracker app.

1. In Visual Studio, click **Run**.

Because the sample app does not include a security certificate, your Internet browser might display a security error. Select the option to continue to the application.



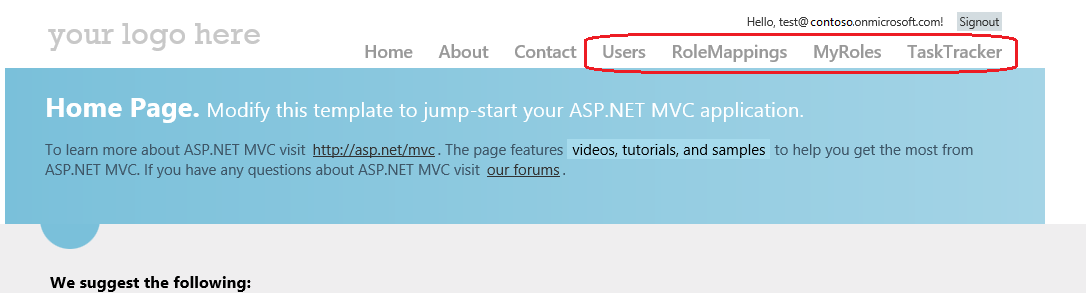
1. On the sign-in page, enter the address of a user with an organizational account. If that account isn't displayed on the sign-in page, click **Use another account**.

When you begin, by default, no users or security groups are mapped to roles or appear the ACL. Initially, only the Global Administrator of the Windows Azure account has the Admin application role. The Admin can use the features of the application to assign roles to other users.

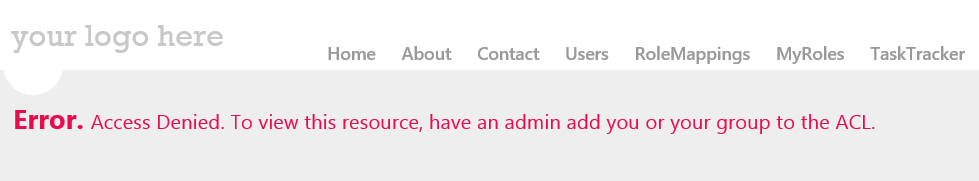
Feel free to experiment with different role combinations to see how it affects the view and functionality of the user.

## Features Managed by Roles

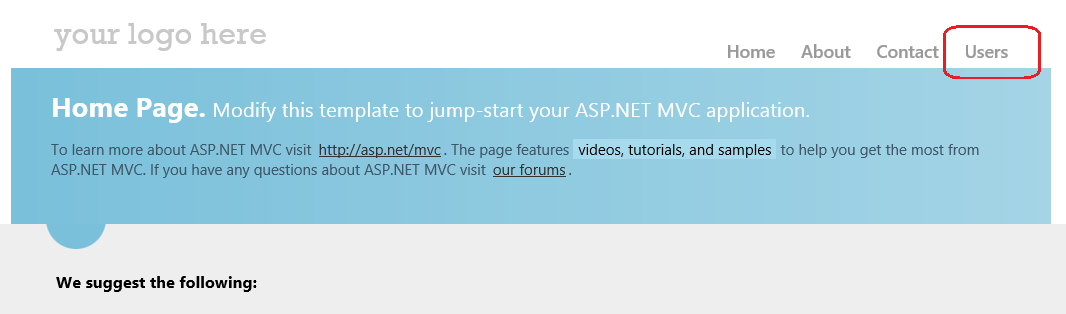
When you sign in with an account that has the Admin application role, you'll see the TaskTracker home page. It's the default MVC page, but the app has added Users, RoleMappings, MyRoles, and TaskTracker tabs. (See the cshtml files in the Views directory.) You can use the features of the application on those pages.



However, if you click the Users page, you see an "access denied" error message. Only users and members of groups that appear in the ACL see the content of the Users page.



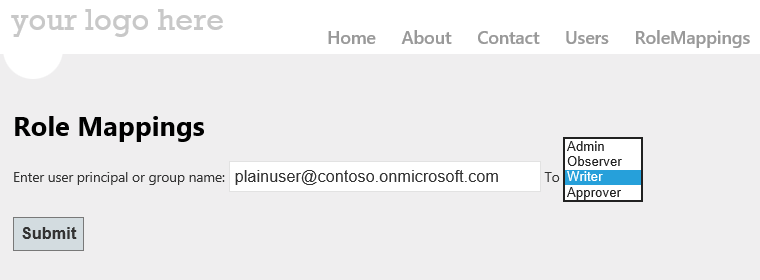
If you sign in with an account that has the User organizational role, only the Users tab is displayed. You cannot use any features of the application.



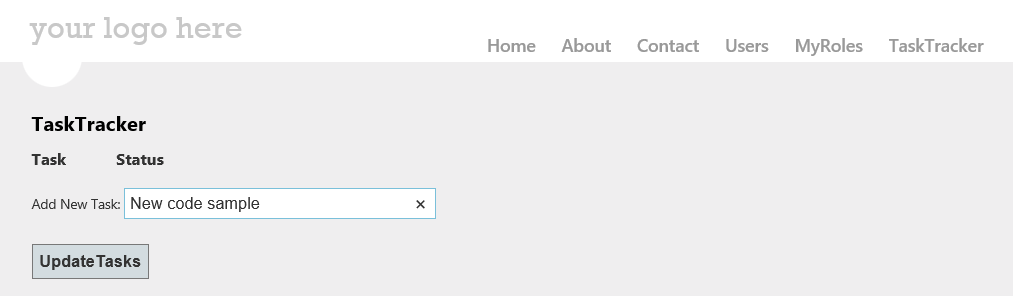
To change a user's role, sign in with an account that has the Admin application role and click **RoleMappings**. Enter the User Principal Name (UPN) of a user or the display name of a group.

When you assign an application role to the user or group, their view of the app and their privileges change.

For example, you can assign the Writer application role to a user.

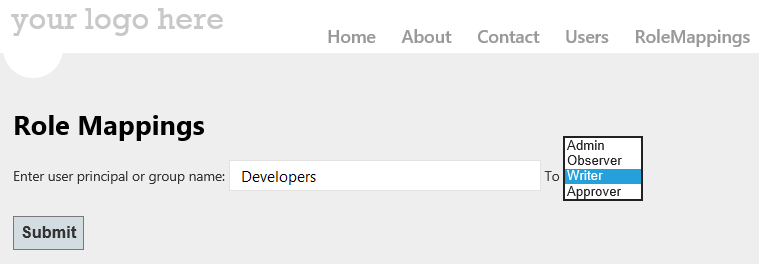


When the user with the Writer application role logs in, they see all tabs and can use the TaskTracker tab to add a task.



You also can grant privileges to the members of a security group. Because getRoles takes the user's complete group membership, this action has the same effect as assigning the user to the application role.

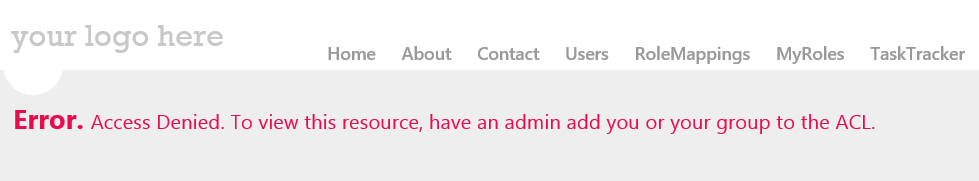
For example, if you create a Developers security group in your Windows Azure AD tenant, enter the group name in the RoleMappings field, and assign the application role of Writer, the members of the Developers group have the Writer application role and its privileges.



## Features Managed by ACL

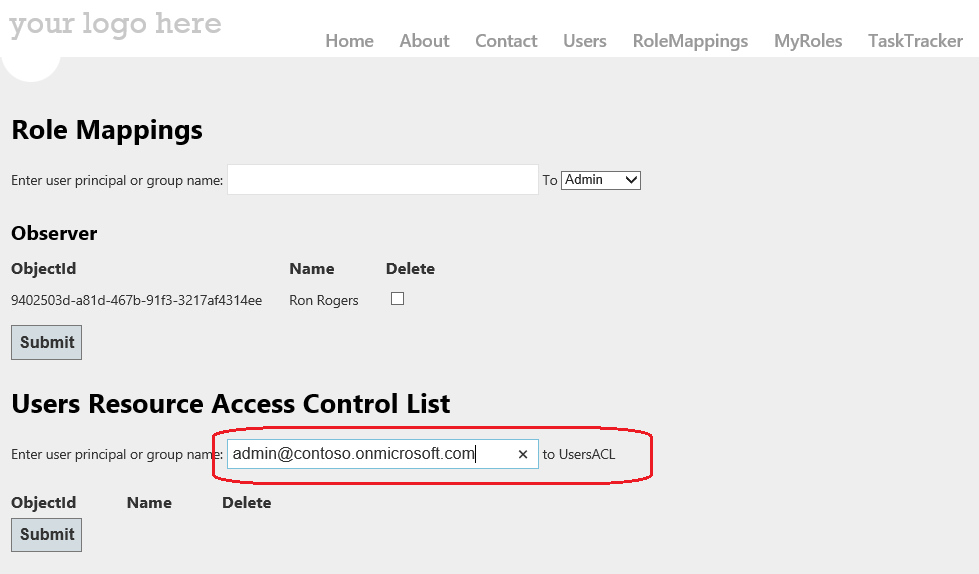
Access to the Users page of TaskTracker is determined by the ACL, which is independent of all application roles. Even users with the Admin application role cannot see the content on the Users page unless the user or a group to which the user belongs appears in the ACL. Only users with the Admin application role can add users and groups to the ACL.

If your ID, or a group to which you belong, is not specified in the ACL, the Users page displays an error that explains that you don't have access.



To add a user or group to the ACL, use **Users Resource Access Control List** section of the RoleMappings page. You must have the Admin app role to add a user or group to the ACL.

Enter the UPN of a user or the display name of a group that is registered in your Windows Azure AD tenant and click Submit. You can enter your own username or the name of a group to which you belong. Because the RoleController updates the ACL immediately and the Users action in the HomeController checks the ACL when access is requested, you have access to the Users page immediately.



Then, when you click the Users tab in app, the users in your tenant appear on the Users page.

