Integrating Windows 8 WinJS Metro App with HealthVault Part I

Author: Mark Arteaga
Date: July 31, 2012

In this first part of this article we will look at how to access HealthVault data from a Windows 8 application using WinJS (HTML/CSS/Javascript). Since there is no HealthVault SDK for WinJS to accomplish this task, we will be building a custom API using MVC to allow the application to access the appropriate data.

We will be continuing from the source code from the previous article **HealthVault User Image with Azure Storage** and adding updates to access the data and secure access to the data. We'll also be taking key learning's from Asthma Journal available for download on GitHub.

Accessing HealthVault

To access HealthVault data from an app, we require the use of a certificate generated to be used to encrypt data sent to the servers. Since distribution of this certificate within a Windows 8 application could potentially cause a security issue, we decided to use our MVC web application as a proxy to access HealthVault data.

Authorizing Doctor Roles

Since our Windows 8 application is only being accessed by 'doctors' we want to secure the access to the data to users with the Doctor role assigned to them. Usually we would just add the following attribute to our Controller `[Authorize(Roles="Doctor")]`and enable RoleManager in web.config.

Since we are using a custom IPrincipal class called HVPrincipal for our web application and setting the HttpApplication.User in the Global.asax.cs file, enabling the RoleManager will overwrite our HttpApplication.User with a RolePrincipal object.

To get around this we will create a custom IAuthorizationFilter and inherit from AuthorizeAttribute to override the AuthorizeCore method.

Add Database Model

Instead of using the System.Web.Security.Roles to check for roles, we will need to manually access the database and check to see if the user is in the role.

1. Create a class called DbModel.cs and implement as follows

```
/// <summary>
/// A user record in the database
/// </summary>
public class User
{
    [Key]
    public Guid UserId { get; set; }
```

```
    public string UserName { get; set; }
}

/// <summary>
/// A record in the roles table
/// </summary>
public class Roles
{
    [Key]
    public Guid RoleId { get; set; }
    public string RoleName { get; set; }
}

/// <summary>
/// Represents the roles for a user
/// </summary>
[Table("UsersInRoles")]
public class UserRole
{
    [Key]
    public Guid RoleId { get; set; }
    public Guid UserId { get; set; }
}
```

2. Open HVDbContext.cs and add the following properties

```
/// <summary>
/// The users in the system
/// </summary>
public DbSet<User> Users { get; set; }

/// <summary>
/// The roles in the system
/// </summary>
public DbSet<Roles> Roles { get; set; }

/// <summary>
/// The roles associated with users
/// </summary>
public DbSet<UserRole> UserRoles { get; set; }
```

Implement AuthorizeRole Class

Create a new class called AuthorizeRole.cs and implement it as follows

```
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false, Inherited = true)]
public class AuthorizeRole : AuthorizeAttribute
{
    protected override bool AuthorizeCore(HttpContextBase httpContext)
    {
        // first try and authorize the user
        if (httpContext == null)
        {
            throw new ArgumentNullException("httpContext");
        }

        IPrincipal user = httpContext.User;
        if (user.Identity.IsAuthenticated)
```

```
        {
            // now check to see if the user in the doctor role
            var context = new HVDbContext();

            // check to see if the role exists
            var roles = this.Roles.Split(',');
            if (roles.Length > 0)
            {
                // find the roles in the database
                foreach (var r in roles)
                {
                    var role = context.Roles.Where(t =>
t.RoleName.Equals(r)).FirstOrDefault();
                    if (role != null)
                    {
                        var userdb = context.Users.Where(t =>
t.UserName.Equals(user.Identity.Name)).FirstOrDefault();
                        if (userdb != null)
                        {
                            // now find the role and user association
                            var ru = context.UserRoles.Where(t =>
t.RoleId.Equals(role.RoleId) && t.UserId.Equals(userdb.UserId)).FirstOrDefault();
                            if (ru != null)
                            {
                                // we found the association so we are good
                                return true;
                            }
                        }
                    }
                }
            }

            // if we make it here user is not authenticated
            return false;
    }
}
```

The code is commented, but essentially overriding the AuthorizeCore method, we can implement our own check to see if the authorized user is part of the role.

New Controller

Next step is to add a new controller to allow the Windows 8 'Doctor App' to login and access the data. We will be creating an new MVC Area called v1 for the API.

1. Right Click on your project
2. Click on Add -> New Item -> Area'
3. In the **Add Area** dialog box, enter **api.** At this point you will have a new folder called **Area\api** with other folders inside there
4. Right click on **Controllers** then click **Add -> Controller**
5. In the **Add Controller** dialog box
    a. Name the controller **DoctorAccountController**
    b. Set the Template to **Empty MVC Controller**

6. Click **Add**

Now that we have our new class, we can implement our methods. First we'll create our **Login** HttpPost method as follows

```
[HttpPost]
public ActionResult Login(string name, string password)
{
    var status = "The user name or password provided is incorrect.";
    var key = "";
    if (Membership.ValidateUser(name, password))
    {
        FormsAuthentication.SetAuthCookie(name, true);
        status = "ok";
    }

    // If we got this far, something failed, redisplay form
    return Json(new { status = status }, JsonRequestBehavior.AllowGet);
}
```

We will also create a method called **Ping** to allow the WinJS app to check if they are authorized to access the system. Using WinJS.xhr, it will automatically save the Auth-Cookie for us when we call login. Our WinJS app will call ping, to see if the user has ever logged in or if the cookie has expired.

```
[AuthorizeRole(Roles = "Doctor")]
[HttpPost]
public ActionResult Ping()
{
    return Json(new { status = "ok" }, JsonRequestBehavior.AllowGet);
}
```

Compile and run the program and leave running. We will be using the new API once we build out the Windows 8 app.

Creating our Windows 8 App

Now that we have the start of our API for our MVC web app, we can start creating our Windows 8 WinJS app. You will have to use Visual Studio 2012 to create a Windows 8 app and the steps below will be using VS 2012 RC.

1. Open Visual Studio 2012
2. Click on **File -> New Project**
3. In the **New Project Dialog** select the **JavaScript** templates and select **Navigation App**
4. Name your app, set the location and click OK.

Login Functionality

First thing we will build is for the WinJS application to be able to access the MVC API built in the previous sections. We will build a WinJS.UI.SettingsFlyout page for the login

1. Right click on the **pages** folder and click **Add -> New Folder**
2. Name the new folder **login**

3. Open up the login.html and replace the <body> tag with the following

```
<!-- BEGINSETTINGSFLYOUT -->
<div id="login" data-win-control="WinJS.UI.SettingsFlyout" aria-label="App Settings
Flyout" data-win-options="{settingsCommandId:'login'}">
    <!-- Use either 'win-ui-light' or 'win-ui-dark' depending on the contrast between
the header title and background color -->
    <div class="win-ui-dark win-header" style="background-color: #00b2f0">
        <!-- Background color reflects app's personality -->
        <button type="button" onclick="WinJS.UI.SettingsFlyout.show()" class="win-
backbutton"></button>
        <div class="win-label">Sign In</div>
    </div>
    <div class="win-content">
        <div class="win-settings-section">
            <p>Enter your username and password to Asthma Journal to sign in</p>
            <div>
                <label>Username</label>
                <input type="text" aria-label="Enter email account" id="user" />
            </div>
            <div>
                <label>Password</label>
                <input type="text" aria-label="Enter email account" id="pass" />
            </div>
            <button id="btnlogin" type="button">Login</button>
        </div>

    </div>
</div>
<!-- ENDSETTINGSFLYOUT -->
```

4. Open **login.js** and add the following code to the **ready** function

```
// wire up the login click
btnlogin.addEventListener('click', function () {
    // hide the flyout
    var divSecurity = document.getElementById("login").winControl;
    divSecurity.hide();
    //thish does not work!!
    //WinJS.UI.SettingsFlyout.hide();

    // attempt to login in
    Application.loginManager.login(
        document.getElementById('user').value,
        document.getElementById('pass').value);
});
```

5. Right click on the **js** folder and a new **Javascript** file called **loginManager.js** and add the following code

```
(function () {
    "use strict";

    var self;
    WinJS.Namespace.define("Application", {
        LoginManager: WinJS.Class.define(
```

```
            function LoginManager() {
                // save a ref
                self = this;

                // store this object globally
                Application.loginManager = this;
            },
            {
                _baseUrl: 'http://localhost:10190/api/v1/DoctorAccount/',

                // method to call when login is complete
                onLoginComplete: null,

                // login failed
                onLoginFailed: null,

                // check access and let the caller know the success
                ping: function (success, fail) {
                    WinJS.xhr({
                        type: 'POST',
                        headers: { "Content-type": "application/x-www-form-urlencoded"
},
                        url: this._baseUrl + 'Ping'
                    }).then(
                        function (result) {
                            try {
                                var res = JSON.parse(result.responseText);
                                if (res.status === 'ok') {
                                    // we are ok to make request
                                    if (success)
                                        success();
                                }
                                else {
                                    // we are not ok
                                    if (fail)
                                        fail(result);
                                }
                            }
                            catch (e) {
                                // just assume there is no cookie saved
                                if(fail)
                                    fail()
                            }

                        },
                        function (result) {
                            // there was an error so let the caller know
                            if (fail)
                                fail(result);
                        });
                },

                // login method
                login: function (username, password) {
                    // make a request
                    WinJS.xhr({
                        type: 'POST',
                        url: this._baseUrl + 'Login',
```

```
                        headers: { "Content-type": "application/x-www-form-urlencoded"
},
                        data: 'name=' + username + '&password=' + password
                    }).then(
                    function (result) {
                        if (result.status === 200) {
                            // we are good see if the status code is ok
                            var res = JSON.parse(result.responseText);
                            if (res.status === 'ok') {
                                // we are ok so callback
                                if (self.onLoginComplete)
                                    self.onLoginComplete(result);
                            }
                            else {
                                // tell the user we can't login
                                if (self.onLoginFailed)
                                    self.onLoginFailed(result);
                            }
                        }
                        else {
                            // there was a wrong reposne from server
                            if (self.onLoginFailed)
                                self.onLoginFailed(result);
                        }
                    },
                    function (result) {
                        // there was an error so report back to user
                        if (self.onLoginFailed)
                            self.onLoginFailed(result);
                    });
                },
            }
        )
    });
})();
```

6. Open **default.js** and add the following line to the end of the **activated** function

```
//Just create a new login manager to store globally
var t = new Application.LoginManager();
```

7. Open **home.js** and add the following two functions

```
showSettings: function () {
    // wire up the completed events
    Application.loginManager.onLoginComplete = function (result) {
        // we are good so attempt to get data
        console.log('sign in successful');
    };

    // wire up the failed event
    Application.loginManager.onLoginFailed = function (result) {
        // show a message to user
        console.log('login failed!!');
    };
```

```
    // since we can't sign in show the login
    WinJS.UI.SettingsFlyout.showSettings("login", "/pages/login/login.html");
}
```

8. In the **ready** method add the following

```
var self = this;
// attempt to make the request
Application.loginManager.ping(function () {
    // we are good
    console.log("already logged in and cookie set");
},
function (result) {
    console.log("need to sign in");
    self.showSettings();
});
```

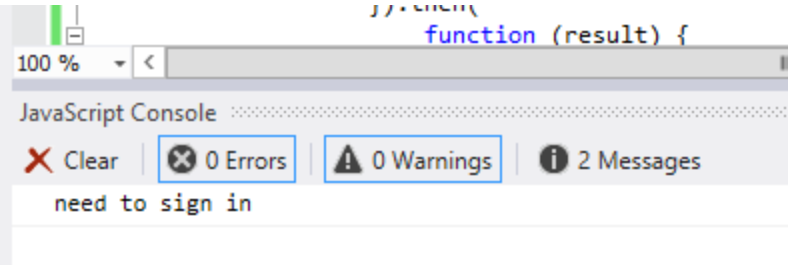
9. Open **default.html** and add the following reference

```
<script src="/js/loginManager.js"></script>
```

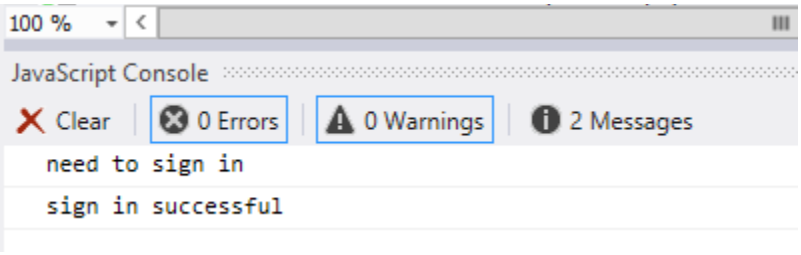
After all that, here is a quick summary of what was done. We first created a login flyout to allow a user to sign into the system. Then we created a loginManager.js file that would manage logging into the system. We then updated home.js to start the process of logging into the system.

Compile and run the program, you will be presented with the sign in screen as follows

And your Javascript console should have the following



Upon logging in, the Javascript console should have the following



Conclusion

In this first part of this article, we looked at setting up our MVC application to be accessed from a Windows 8 WinJS application. We created the functionality in our MVC application including a custom attribute to handle the doctor's role and in our Win8 WinJS application we started creating the functionality to access the services. In the next part we'll finish up adding new functionality to our MVC web application to access HealthVault data from a WinJS application.