Integrating HealthVault SDK with APS.NET MVC Part III

Author: Mark Arteaga
Date: July 31, 2012

In the first two parts of this series of articles we looked at preparing the HealthVault SDK to work with ASP.NET MVC since the original SDK is to be used with ASP.NET Web Forms. In the second part, we integrated to allow users to authenticate with HealthVault and our system to have access to authorized pages. In this third and final part, we will look at cleaning up the code and making it a little more of a complete system.

Custom IPrincipal Class

To start our cleanup process, one of the first things we want to do is create a custom IPrincipal class to hold some extra information about our signed in user.

1. Create a new class in your MVC Project called **HVPrincipal**
2. Add the following class implementation

```
/// <summary>
/// Class to hold the authenticated user from Health Vault for forms authentication
/// </summary>
public class HVPrincipal : IPrincipal
{
    public HVPrincipal()
    {
    }

    public HVPrincipal(MembershipUser aspUser, PersonInfo pi, string authtoken)
    {
         // null checking
         if (aspUser == null)
             throw new ArgumentNullException("aspUser cannot be null");

         //create the identity
         Identity = new GenericIdentity(aspUser.UserName);

         //save some values for easy retrieving from Database
         Name = pi.Name;
         PersonId = pi.PersonId;
         UserName = pi.PersonId.ToString();
         UserId = pi.PersonId;
         AuthToken = authtoken;
     }

     /// <summary>
     /// The auth token from HealthVault
     /// </summary>
    public string AuthToken { get; set; }

    /// <summary>
```

```
        /// The name of the healthvault user since the username in asp.net membership is a
    guid
        /// </summary>
        public string Name { get; set; }

        /// <summary>
        /// The HealthVault person id
        /// </summary>
        public Guid PersonId { get; set; }

        /// <summary>
        /// The username assigned to the user
        /// </summary>
        public string UserName { get; set; }

        /// <summary>
        /// The user id in the system
        /// </summary>
        public Guid UserId { get; set; }

        /// <summary>
        /// Generic identity object
        /// </summary>
        [ScriptIgnoreAttribute]
        public IIdentity Identity { get; set; }

       /// <summary>
       /// Determins if the user is in a role or not
       /// </summary>
       /// <param name="role"></param>
       /// <returns></returns>
       public bool IsInRole(string roleName)
       {
           return false;
       }

        /// <summary>
        /// Creates a generic idenity basically used when deserialized from json
        /// </summary>
        public void CreateGenericIdentity()
        {
            Identity = new GenericIdentity(UserName);
        }
    }
```

Compile to make sure your project still compiles. If your code does not compile the most likely reason is you are missing some using statements. Our next steps are to serialize this information to the AUTH cookie for future use and retrieval.

Serializing HVPrincipal Class

Now that we have our code setup to access the database, the following steps will allow us to serialize the data to the AUTH cookie.

1. Open AccountController.cs
2. Find the CreateAuthCookie method and change the implementation with the following

```
    private void CreateAuthCookie(PersonInfo personInfo, MembershipUser user, string
    authToken)
    {
        // Create a new principal and serialize it
        var hvPrincipal = new HVPrincipal(user, personInfo, authToken);
        var userData = new JavaScriptSerializer().Serialize(hvPrincipal);

        // create an auth ticket
        var authTicket = new FormsAuthenticationTicket(1,
            personInfo.PersonId.ToString(),
            DateTime.Now,
            DateTime.Now.AddHours(2),
            false,
            userData);

        // add the ticket to the cookies
        Response.Cookies.Add(new HttpCookie(FormsAuthentication.FormsCookieName,
    FormsAuthentication.Encrypt(authTicket)));
    }
```

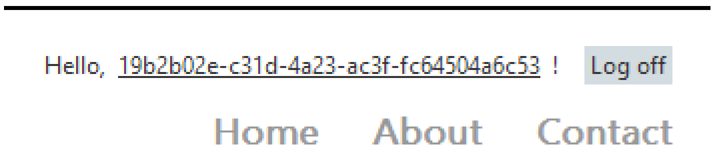
3. Open up **Global.asax.cs** and add the following method implementation which will set our HVPrincipal object to the current HttpContext.Current.User

```
protected void Application_AuthenticateRequest(Object sender, EventArgs e)
{
    HttpCookie authCookie = Request.Cookies[FormsAuthentication.FormsCookieName];

    if (authCookie != null)
    {
        FormsAuthenticationTicket authTicket =
FormsAuthentication.Decrypt(authCookie.Value);

        JavaScriptSerializer serializer = new JavaScriptSerializer();
        HVPrincipal hvPrin = serializer.Deserialize<HVPrincipal>(authTicket.UserData);
        hvPrin.CreateGenericIdentity();
        HttpContext.Current.User = hvPrin;
    }
}
```

Compile and run the code. You will notice, on the user interface level, there is not much change with the login details.


Hello, 19b2b02e-c31d-4a23-ac3f-fc64504a6c53 !

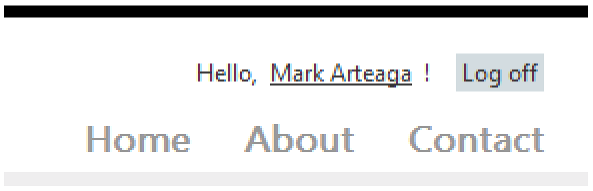
The next step will accomplish showing the user name on the screen.

Showing the User Their Name

One of the first things we want to do is get rid of the GUID that is shown as the username.   With the code created above, this is easy to accomplish.

1. Open up **Views/Shared/_LoginPartial.cshtml**
2. Find **User.Identity.Name** and replace with `(User as Samples.HvMvc.HVPrincipal).Name`

Run the project, sign in to HealthVault and you will notice the name is now displayed to the user.



Redirecting User to HealthVault

Now that the user can see their name, they also have the ability to change passwords. Since we are automatically creating users we do not want to have that functionality and we have no user management that we need to do on our system. We still want to have a way for the users to manage their HealthVault account and we'll do that now.

1. Open up **AccountController.cs**
2. Add the following method to redirect the user to HealthVault

```
[Authorize]
public ActionResult Details()
{
    // redirects the user to the healthvault account management page
    WebApplicationUtilities.RedirectToShellUrl(System.Web.HttpContext.Current,
"MANAGEACCOUNT");
    return RedirectToAction("Index", "Home");
}
```

3. Open up **Views/Shared/_LoginPartial.cshtml**
4. Replace the 'true' section of the if statement with the following

```
<ul>
    <li>
        Hi, @Html.ActionLink((User as Samples.HvMvc.HVPrincipal).Name, "Details",
"Account" , routeValues: null, htmlAttributes: new { @class = "username" }) |
    </li>
    <li>
        @Html.ActionLink("Log off", "LogOff", "Account")
    </li>
</ul>
```

Run the program and click the name. This will result in being redirected to HealthVault management page. Note, that there is no link in HealthVault page to return back to your web application.

For a list of all the Shell Urls available see on HealthVault Shell Redirect Interfaces on MSDN.

Final Tidying Up

In this final section we will deal with a few housekeeping items that need to be done.

Logging Out

We need to allow the user to logout of our system and the HealthVault system.

1. Open up **AccountController.cs**
2. Add the following line to the **LogOff** method

```
WebApplicationUtilities.SignOut(System.Web.HttpContext.Current, null);
```

This will make sure that the HealthVault/LiveId will always be requested when the user logs off of our local system.

Removing AccountController Actions

We need to remove some Actions in our AccountController class.

1. Open up **AccountController.cs**
2. Change the implementation for **Register(), Register(RegisterModel), ChangePassword, ChangePassword(ChangePasswordModel), ChangePassowrdSuccess()** with the following

```
return RedirectToAction("Index", "Home");
```

This will make sure none of the functionality is available in the system and just redirect the user to the home page. The other option is to delete the methods, but you may decide to use them in the future at some point.

Removing the Register Link

In the _loginPartial.cshtml there is a link to register. Since HealthVault system handles the registration, we need to remove this functionality. You can also re-direct the user to the **CREATEACCOUNT** HealthVault shell url but I will leave that up to the reader to explore.

1. Open up **Views/Shared/_LoginPartial.cshtml**
2. Find the following line and delete it `<li>@Html.ActionLink("Register", "Register", "Account", routeValues: null, htmlAttributes: new { id = "registerLink" })</li>`

Conclusion

In this article we looked at integrating with the HealthVault SDK some more with our system. We created a custom IPrincipal class, cleaned up the user interface and cleaned up some code that was not required anymore.

This concludes our three part articles on Integrating HealthVault SDK with ASP.NET MVC. There are a few articles I will follow up on such as storing data in HealthVault, Retrieving Avatar Images, deploying to Azure and integrating with a Windows 8 WinJS application, but these are much shorter articles.