Integrating HealthVault SDK with APS.NET MVC Part II

Author: Mark Arteaga

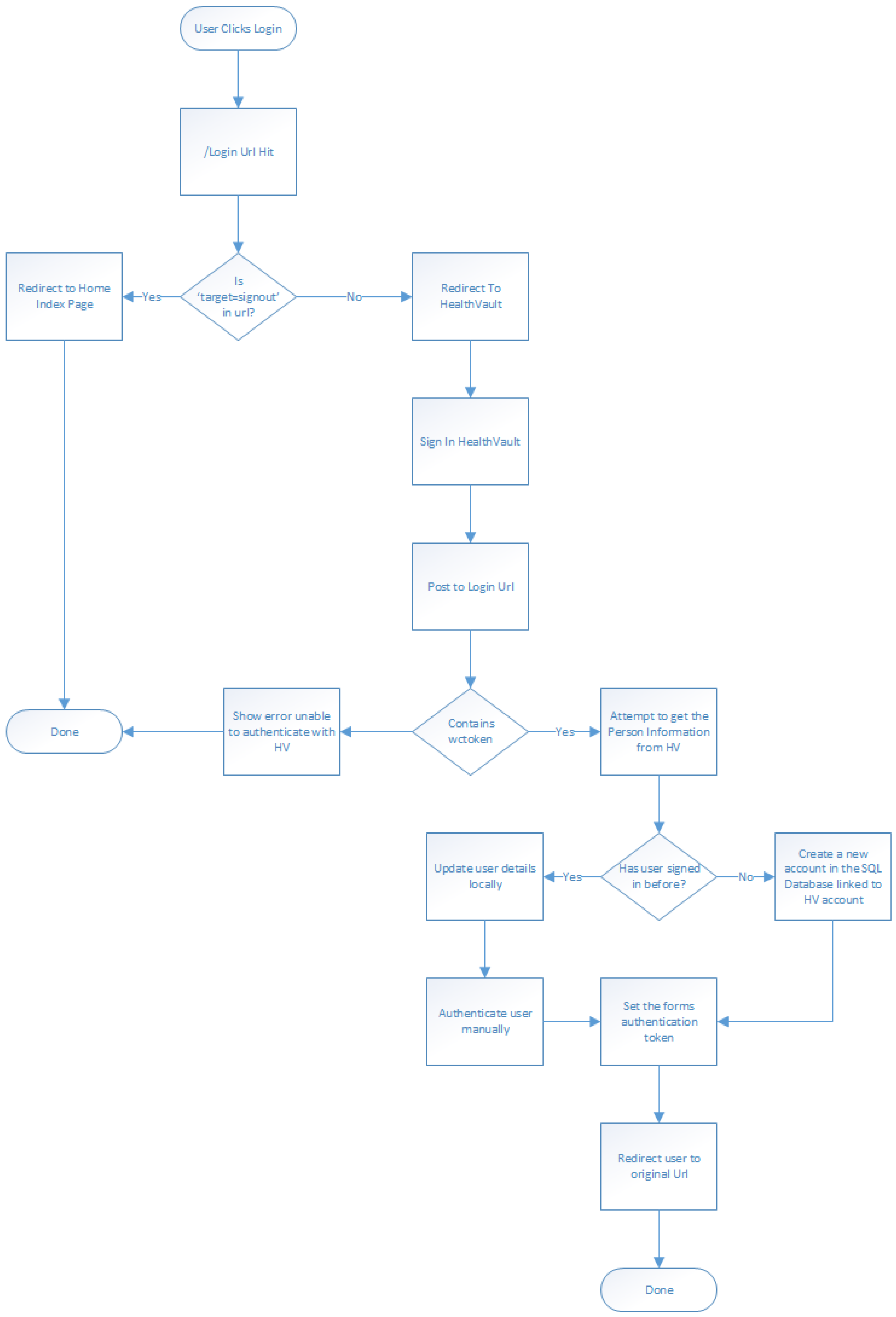
Date: July 31, 2012

In the first part of this article, we took the initial steps to have our ASP.NET MVC project work with the HealthVault SDK. In this part of the article, we'll add code so we can authenticate with HealthVault, get user information and integrate with default MembershipProvider.

Authenticating With HealthVault

To access a user's HealthVault data, we require them to authenticate with HealthVault. HealthVault's authentication is similar to OAuth but is not the OAuth implementation. All the details on Shell Redirect Interfaces can be found on MSDN and recommend you read through this to familiarize yourself with the code we are about to add.

For the Asthma Journal application, we decided on using the standard SqlMemberShipProvider for forms authentication in combination with HealthVault authentication and create users locally to provide authorized access to certain pages. The basic flow is like this


User Clicks Login
/Login Url Hit
Is 'target=signout' in url?
Yes
Redirect to Home Index Page
No
Redirect To HealthVault
Sign In HealthVault
Post to Login Url
Contains wctoken
Show error unable to authenticate with HV
Done
Yes
Attempt to get the Person Information from HV
Has user signed in before?
Yes
Update user details locally
No
Create a new account in the SQL Database linked to HV account
Authenticate user manually
Set the forms authentication token
Redirect user to original Url
Done

Now that we have a basic flow, we will implement the code pieces to accomplish this.

1. In your VS2012 Project, add a reference to the HealthVault SDK projects.
2. Open **AccountController.cs** in the **Controllers** directory and find the **Login** method. Add the following code

```
[AllowAnonymous]
public ActionResult Login(string returnUrl)
{
    if (HttpContext.Request.RawUrl.Contains("target=SignOut"))
        //when we sign out it will come to this page, so we just want to redirect to the 
login page
        return RedirectToAction("Index", "Home");
    else
    {
        //Redirect to health vault login which will post to the login method
        var url = Request.Params["ReturnUrl"] == null ? (Request.UrlReferrer.PathAndQuery 
== null ? "/" : Request.UrlReferrer.PathAndQuery) : Request.Params["ReturnUrl"];
        WebApplicationUtilities.RedirectToLogOn(System.Web.HttpContext.Current, true, 
url);
    }
    return View();
}
```

This code will allow us to redirect to appropriate URLs depending on the request Url. When a user signs out from HealthVault, it will redirect to the Login page, so we need to handle that scenario. Add the appropriate using statements to eliminate any compile errors

Run the application, click the login link on the top right. This will re-direct you to HealthVault sign in and when you have successfully authenticated, it will redirect you back to the Login page but it will say it's unsuccessful because although we have authenticated with HealthVault, we have not authenticated with our local system. The next few steps will fix that.

Authenticating with SqlMembershipProvider

Once authenticated with HealthVault, we need to authenticate with our local system to allow the appropriate pages in our system. To do this we need to add appropriate code to our **Login()** method that handles the **HttpPost**.

Handling HTTPPOST for Login

First thing we need to make sure is check for the **wctoken** parameter in the queryString and if it's not there we just show an error that we cannot authenticate. Our HttpPost Login will be changed to the following

```
[AllowAnonymous]
[HttpPost]
public ActionResult Login(LoginModel model, string returnUrl)
{
    // here we are getting posted from HealthVault so extract the wctoken sent
    string authToken = Request.Params["wctoken"];
    if (authToken != null)
    {
    }
    else
    {
        // no wctoken so just redirect to home
        ModelState.AddModelError("", "Unable to authenticate with Microsoft
HealthVault.");
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}
```

Next we want to add the logic to the if statement to follow our flow chart. The code is commented, so should be able to read through it.

1. Using the HealthVault SDK, get the user information with the following code

```
// create a web app cred object
var appId = HealthApplicationConfiguration.Current.ApplicationId;
WebApplicationCredential cred =
new WebApplicationCredential(
    appId,
    authToken,
    HealthApplicationConfiguration.Current.ApplicationCertificate);

// setup the user
WebApplicationConnection connection = new WebApplicationConnection(appId, cred);
PersonInfo personInfo = HealthVaultPlatform.GetPersonInfo(connection);

// check to make sure there is access to records
if (personInfo.AuthorizedRecords.Count() == 0)
    throw new Exception("There are no authorized users for us to work with!");

// check to see if the user exists
var personId = personInfo.PersonId.ToString();
```

2. Once we have retrieved the person information from HealthVault, attempt to validate the user in our local system. The local system will be a user derived from the **PersonInfo** object and the username will be the **PersonInfo.PersonId** and the password will be the **PersonInfo.PersonId + HealthApplicationConfiguration.Current.ApplicationId**

```
// we found the user so authenticate them
var username = personId;
var password = personId + appId;
if (Membership.ValidateUser(username, password))
{
}
else
{
}
```

3. If the user has authenticated, we know they have used the system previously so we create an AUTH cookie to authenticate the user

```
// user has authenticated
```

```
var user = Membership.GetUser(personInfo.PersonId.ToString());

// save auth cookie
CreateAuthCookie(personInfo, user);
```

4. If the user has not authenticated with the system previously, we want to create them a new account and then create them an AUTH cookie to authenticate them with the system. If there is an error creating the new user, we want to report that back.

```
// the user has not registered with us so create one
// Attempt to register the user
MembershipCreateStatus createStatus;
var newUser = Membership.CreateUser(model.UserName, model.Password, "", passwordQuestion:
null, passwordAnswer: null, isApproved: true, providerUserKey: null, status: out
createStatus);

if (createStatus == MembershipCreateStatus.Success)
{
    //save auth cookie
    CreateAuthCookie(personInfo, newUser);
}
else
{
    ModelState.AddModelError("", ErrorCodeToString(createStatus));
    return View(model);
}
```

5. Finally if everything goes well, we want to re-direct the user back to the URL that they attempted to login in from

```
// redirect to the actionqs
NameValueCollection query = HttpUtility.ParseQueryString(Request.Url.Query);

var r = HttpUtility.UrlDecode(query["actionqs"]);
return Redirect(new Uri(string.Format("http://{0}{1}{2}",
    Request.Url.Host,
    (Request.Url.IsDefaultPort ? "" : ":" + Request.Url.Port), r)).ToString());
```

After all that you're HttpPost Login method should look like the following

```
[AllowAnonymous]
[HttpPost]
public ActionResult Login(LoginModel model, string returnUrl)
{
    // here we are getting posted from HealthVault so extract the wctoken sent
    string authToken = Request.Params["wctoken"];
    if (authToken != null)
    {
        // create a web app cred object
        var appId = HealthApplicationConfiguration.Current.ApplicationId;
        WebApplicationCredential cred =
        new WebApplicationCredential(
            appId,
            authToken,
            HealthApplicationConfiguration.Current.ApplicationCertificate);

        // setup the user
        WebApplicationConnection connection = new WebApplicationConnection(appId, cred);
        PersonInfo personInfo = HealthVaultPlatform.GetPersonInfo(connection);

        // check to make sure there is access to records
        if (personInfo.AuthorizedRecords.Count() == 0)
            throw new Exception("There are no authorized users for us to work with!");

        // check to see if the user exists
        var personId = personInfo.PersonId.ToString();

        // we found the user so authenticate them
        var username = personId;
        var password = personId + appId;
        if (Membership.ValidateUser(username, password))
        {
            // user has authenticated
            var user = Membership.GetUser(personInfo.PersonId.ToString());

            // save auth cookie
            CreateAuthCookie(personInfo, user);
        }
        else
        {
            // the user has not registered with us so create one
            // Attempt to register the user
            MembershipCreateStatus createStatus;
            var newUser = Membership.CreateUser(model.UserName, model.Password, "",
passwordQuestion: null, passwordAnswer: null, isApproved: true, providerUserKey: null,
status: out createStatus);

            if (createStatus == MembershipCreateStatus.Success)
            {
                //save auth cookie
                CreateAuthCookie(personInfo, newUser);
            }
            else
            {
                ModelState.AddModelError("", ErrorCodeToString(createStatus));
                return View(model);
            }
        }

        // redirect to the actionqs
        NameValueCollection query = HttpUtility.ParseQueryString(Request.Url.Query);

        var r = HttpUtility.UrlDecode(query["actionqs"]);
        return Redirect(new Uri(string.Format("http://{0}{1}{2}",
            Request.Url.Host,
            (Request.Url.IsDefaultPort ? "" : ":" + Request.Url.Port), r)).ToString());
    }
    else
    {
        // no wctoken so just redirect to home
        ModelState.AddModelError("", "Unable to authenticate with Microsoft
HealthVault.");
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}
```

Manually Creating Authentication Cookie

If you have read through the code, you would have noticed a method called **CreateAuthCookie().** This is a helper method created to manually create the authentication Cookie.

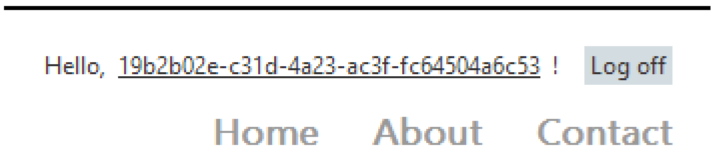
Now that our models and database contexts are setup, we can go ahead and find users in the system after they have authenticated with HealthVault. The following method accomplishes this.

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

    // 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)));
}
```

If you now run the project, attempt to sign into HealthVault, you will notice the 'login section' of the page is now displaying user information and a 'log off' link' as follows.



Ideally, showing a GUID is not what we want but we will do clean up and tidying up in Part III of our article.

Conclusion

In this part of the series, we looked at authenticating with HealthVault. We also looked at authenticating with our local system without the need of having the user enter two usernames and passwords, one for HealthVault and one for our local system. In Part III, we will look at tidying up the code to allow a better experience.