Using ASP.NET MVC to Store Custom Data in HealthVault

Author: Mark Arteaga
Date: July 31, 2012

In previous articles, I have walked through on some of the key items to go through to get HealthVault SDK up and running with an ASP.NET MVC application. Here is the list in case you have not seen them

1. Quick Start Guide to HealthVault SDK
2. Integrating HealthVault SDK with ASP.NET MVC Part I
3. Integrating HealthVault SDK with ASP.NET MVC Part II
4. Integrating HealthVault SDK with ASP.NET MVC Part III

In this article, we'll look at submitting some custom data to HealthVault and be able to retrieve the data back from HealthVault using an API we will build. We will be submitting an "Asthma Journal" entry taken from the Asthma Journal app which can be downloaded from GitHub.

I'll assume you are familiar with HealthVault SDK and can get using the SDK in an MVC web application. We will be using the source code from Integrating HealthVault SDK with ASP.NET MVC Part III as a starting point.

Adding Our Model

First thing we need to do is create our model representing a journal entry. We will reuse the JournalEntry class from Asthma Journal with some modifications for the purpose of this article.

1. In the **Models** directory create a class called JournalEntry
2. Add the following code which will be our POCO

```
/// <summary>
/// Journal entry in the system
/// </summary>
public class JournalEntry
{
    public JournalEntry()
    {
        RecordId = Guid.Empty;
        UserId = Guid.Empty;
        Created = DateTime.Now.Date;
        EntryDate = DateTime.Now.Date;
        Latitude = -1;
        Longitude = -1;
    }

    /// <summary>
    /// The id of the journal entry
    /// </summary>
    public int Id { get; set; }
```

```
    /// <summary>
    /// the Guid user id assocociated with the record
    /// </summary>
    public Guid UserId { get; set; }

    /// <summary>
    /// The health vault record ID this entry is assocociated with
    /// </summary>
    [DisplayName("Create Record For: ")]
    [Description("The current selected HealthVault record the journal entry will be
assigned to")]
    public Guid RecordId { get; set; }

    /// <summary>
    /// Determins if person missed school or work
    /// </summary>
    [DisplayName("Did you miss school or work?")]
    public bool MissedWorkSchool { get; set; }

    /// <summary>
    /// Determins if user saw a doctor
    /// </summary>
    [DisplayName("Did you see a doctor?")]
    public bool SawDoctor { get; set; }

    /// <summary>
    /// Determines if user went to the emergency clinic because of symptons
    /// </summary>
    [DisplayName("Did you go to the emergency?")]
    public bool SawEmergency { get; set; }

    /// <summary>
    /// The peak flow reading for the entry
    /// <remarks>-1 is not entered should be a number from 0 to 500</remarks>
    /// </summary>
    [DisplayName("Enter your peak flow reading if there is one?")]
    public int PeakFlowReading { get; set; }

    /// <summary>
    /// Determines if the entry was done during the day or not
    /// </summary>
    [DisplayName("Was this during the day?")]
    public bool IsDay { get; set; }

    /// <summary>
    /// The location of the entry
    /// </summary>
    public double Latitude { get; set; }

    /// <summary>
    /// The location of the entry
    /// </summary>
    public double Longitude { get; set; }

    /// <summary>
    /// Date record was created
    /// </summary>
    public DateTime Created { get; set; }
```

```
    /// <summary>
    /// Date for the entry
    /// </summary>
    [DisplayName("What date is this entry for?")]
    [DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:MM/dd/yyyy}")]
    public DateTime EntryDate { get; set; }

}
```

Compile to make sure your project still compiles and continue to the next section.

Creating our DBContext

We'll need to create a DBContext for Visual Studio to create our View and Controller in the next step. Create a new file called HVDbContext.cs and implement the class as follows

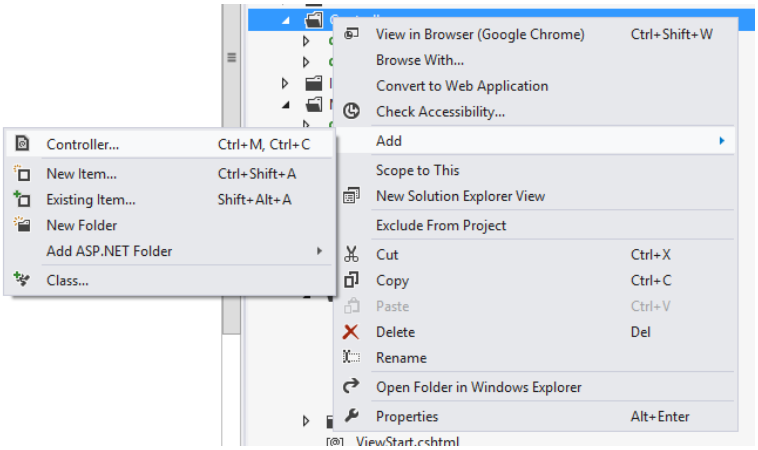
```
public class HVDbContext : DbContext
{
    public HVDbContext()
        : base("DefaultConnection")
    {
    }


    /// <summary>
    /// Journal Entries in the system
    /// </summary>
    public DbSet<JournalEntry> JournalEntries { get; set; }
}
```

Compile to make sure your project still compiles and continue to the next section.

Creating Our Views and Controller

Now that we have our POCO we can create a controller

1. Add a new Controller to your **Controllers** directory by right clicking Add -> Controller



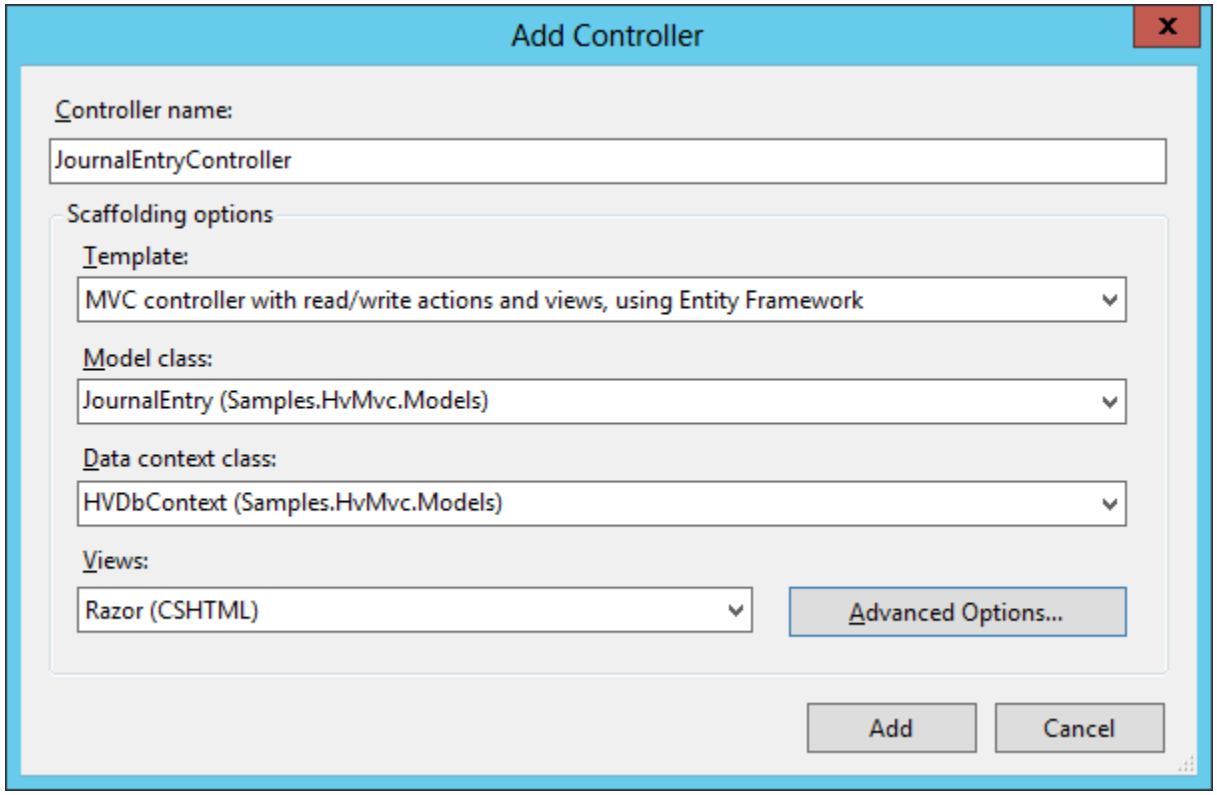
2. In the **Add Controller** dialog box set the Controller Name to **JournalEntryController**
3. For the Scaffolding options to the following
    a. Template: **MVC Controller with read/write actions and views, using Entity Framework**

b. Model Class: **JournalEntry (Samples.HvMvc.Models)**

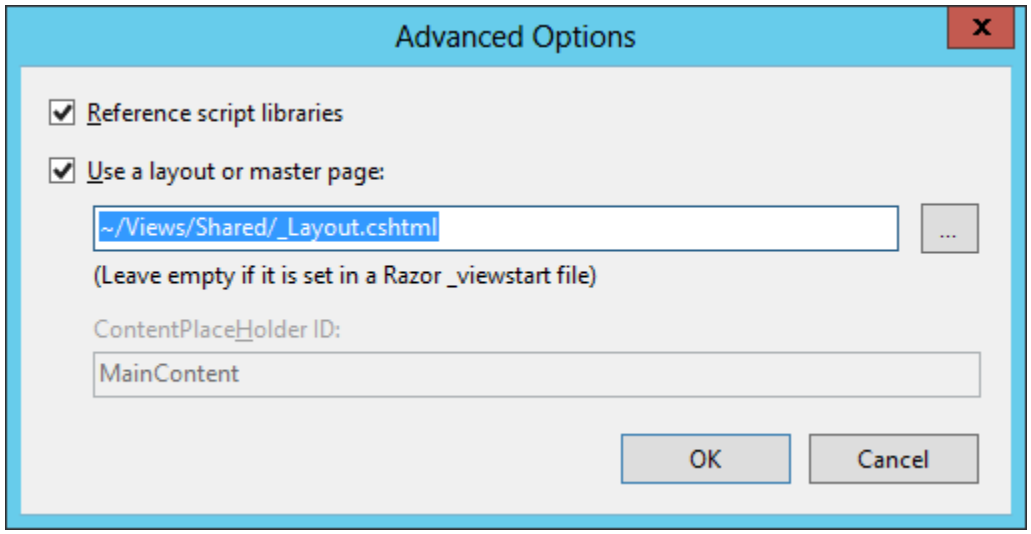
c. Data Context class: **HVDbContext (Samples.HvMvc.Models)**

d. Views: **Razor (CSHTML)**

4. The **Add Controller** dialog should look like this

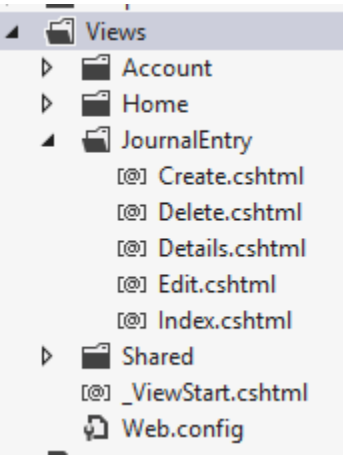


5. Click on **Advance Options** and check the **Use a layout or master page** and set it to **~/Views/Shared/_Layout.cshtml**

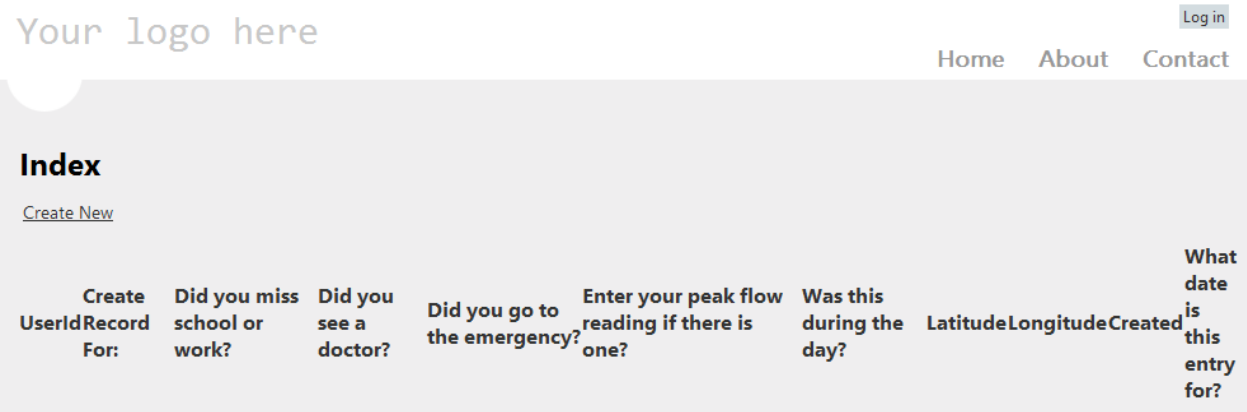


6. Click **OK** then click **Add**
7. In the **JournalEntryController.cs** file that was just created modify the **Index** method implementation to the following `return View(new List<JournalEntry>(0));`

At this point a bunch of files will be created in your **Views** directory and should look like the following



Run the project and navigate to http://localhost:[PORT]/JournalEntry and you will see the following which is not very pretty but will leave that to an exercise for the user



Securing the Controller

One of the requirements for this sample is to only have authorized users able to submit new Journal Entries. This is fairly easy and do the following

1. Open **JournalEntryControllers.cs** under the **Controllers** directory
2. Add the following line above the class definition
   `[Authorize]`

Compile and run the project and attempt to navigate to http://localhost/JournalEntry. This will re-direct you to the HealthVault login and once successfully logged in, will redirect back to the original URL.

Creating an Entry

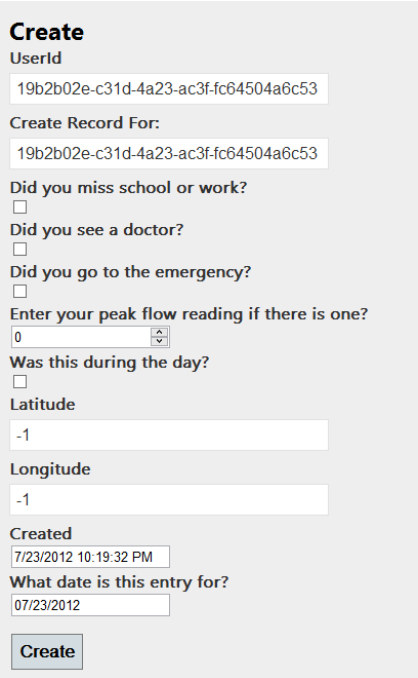
Once you are authorized, you can create an entry to submit to the database. There are a few fields on the form that are a bit cryptic so we can make some updates to make it a little easier to fill out the form.

1. Open **JournalEntryControllers.cs** under the **Controllers** directory

2. Find the Create() method and replace the existing code with the following

```
var e = new JournalEntry()
{
    EntryDate = DateTime.Now,
    Created = DateTime.Now,
    UserId = Guid.Parse(User.Identity.Name),
    RecordId = Guid.Parse(User.Identity.Name)
};
return View(e);
```

This code will pre-populate the fields on the form for the user.



UseId and RecordId we don't want to make visible so we can change the view implementation to make them hidden.

1. Open **Create.cshtml** in the **Views/JournalEntry** directory
2. Locate the following lines

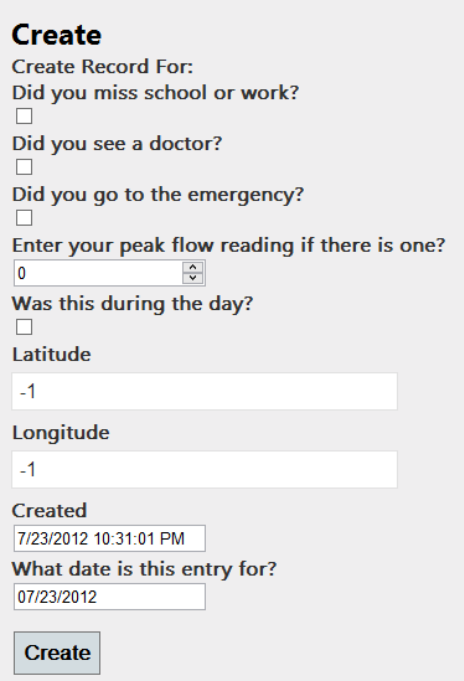
```
<div class="editor-label">
    @Html.LabelFor(model => model.UserId)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.UserId)
    @Html.ValidationMessageFor(model => model.UserId)
</div>

<div class="editor-label">
    @Html.LabelFor(model => model.RecordId)
</div>
<div class="editor-field">
```

```
    @Html.EditorFor(model => model.RecordId)
    @Html.ValidationMessageFor(model => model.RecordId)
</div>
```

3. Change the located lines with the following code

```
@Html.HiddenFor(model => model.UserId)
@Html.HiddenFor(model => model.RecordId)
```

Running the project, you will see the UserId and RecordId fields are now hidden



Clicking the Create button will cause an exception because it is trying to create the record in the database but our goal is to create in HealthVault. We will explore that in the next section

Submitting to HealthVault

In this section, we will submit the data to HealthVault and display the data in the list view. Since HealthVault has standard data types, we need to add a custom data type since our model does not fit into any of the standard types. For some more information on that, check [HealthVault Data Types: Custom Data Types](#).

The following steps will allow us to save to HealthVault.

1. Create a new HVJournalEntry.cs class
2. Change the implementation to the following

```
/// <summary>
/// Journal entry for health vault
/// </summary>
public class HVJournalEntry : HealthRecordItem
```

```
{
private static Guid m_typeId = new Guid("a5033c9d-08cf-4204-9bd3-cb412ce39fc0");
/// <summary>
/// Type id for the recird item.  NOTE any custom type must have this type id
/// http://msdn.microsoft.com/en-us/healthvault/bb968869
public static new Guid TypeId { get { return m_typeId; } }

private HealthServiceDateTime m_when;

public HVJournalEntry()
: base(new Guid("a5033c9d-08cf-4204-9bd3-cb412ce39fc0"))
{
}

public HVJournalEntry(JournalEntry entry)
: base(new Guid("a5033c9d-08cf-4204-9bd3-cb412ce39fc0"))
{
m_when = new HealthServiceDateTime(DateTime.Now);

// clone the entry coming in without reference to object
JournalEntry = new JournalEntry()
{
Created = entry.Created,
EntryDate = entry.EntryDate,
Id = entry.Id,
IsDay = entry.IsDay,
Latitude = entry.Latitude,
Longitude = entry.Longitude,
MissedWorkSchool = entry.MissedWorkSchool,
PeakFlowReading = entry.PeakFlowReading,
RecordId = entry.RecordId,
SawDoctor = entry.SawDoctor,
SawEmergency = entry.SawEmergency,
UserId = entry.UserId
};
}

/// <summary>
/// Journal Entry item
/// </summary>
public JournalEntry JournalEntry { get; set; }

/// <summary>
/// The app id or name used for the xml submitted to healthvault
/// </summary>
internal string AppId { get { return "test2"; } }

/// <summary>
/// Summary to include in the xml
/// </summary>
internal virtual string Summary { get { return "health record item from RB Asthma
Journal"; } }

/// <summary>
/// The format tag to use
/// </summary>
internal virtual string FormatTag { get { return "JournalEntryItem"; } }
```

```
/// <summary>
/// The element tag to use for the journal entry
/// </summary>
internal virtual string ElementTag { get { return "Data"; } }

/// <summary>
/// Writes the Xml to be save to health vault
/// </summary>
/// <param name="writer"></param>
public override void WriteXml(XmlWriter writer)
{
writer.WriteStartElement("app-specific");
{
writer.WriteStartElement("format-appid");
writer.WriteValue(this.AppId);
writer.WriteEndElement();
writer.WriteStartElement("format-tag");
writer.WriteValue("PeakZone");
writer.WriteEndElement();
m_when.WriteXml("when", writer);
writer.WriteStartElement("summary");
writer.WriteValue(this.Summary);
writer.WriteEndElement();

// start writing out the custom object
writer.WriteStartElement(this.ElementTag);
writer.WriteValue(this.SerializeEntry());
writer.WriteEndElement();
}
writer.WriteEndElement();
}

/// <summary>
/// Parses the xml from the healthvault system
/// </summary>
/// <param name="typeSpecificXml"></param>
protected override void ParseXml(IXPathNavigable typeSpecificXml)
{
XPathNavigator navigator = typeSpecificXml.CreateNavigator();
navigator = navigator.SelectSingleNode("app-specific");
XPathNavigator when = navigator.SelectSingleNode("when");
m_when = new HealthServiceDateTime();

m_when.ParseXml(when);

XPathNavigator formatAppid = navigator.SelectSingleNode("format-appid");
string appid = formatAppid.Value;

XPathNavigator peakZone = navigator.SelectSingleNode(ElementTag);
var data = peakZone.Value;
this.JournalEntry = DeserializeEntry(data);
}


/// <summary>
/// Serialized the journal entry
/// </summary>
/// <returns>json representation</returns>
```

```
private string SerializeEntry()
{
if (JournalEntry == null)
throw new ArgumentException("JournalEntry cannot be null");

return new JavaScriptSerializer().Serialize(JournalEntry);
}

private JournalEntry DeserializeEntry(string data)
{
if (string.IsNullOrEmpty(data))
throw new ArgumentException("data cannot be null");

return new JavaScriptSerializer().Deserialize<JournalEntry>(data);
}
}
```

This is a lot of code that was just added so we'll touch on a few points.

HVJournalEntry Constructor

```
public HVJournalEntry(JournalEntry entry)
: base(new Guid("a5033c9d-08cf-4204-9bd3-cb412ce39fc0"))
```

Every HealthVault data type has a GUID id assigned to it and when creating a custom data type you have to assign your own id. The constructor for HVJournalEntry does exactly this.

HVJournalEntry Parsing

```
public override void WriteXml(XmlWriter writer)
protected override void ParseXml(IXPathNavigable typeSpecificXml)
```

Since we have custom data that we are adding and we need to adhere to data type specifications, our WriteXML and ParseXml methods serialize and deserialize to the appropriate formats. HealthVault documentation recommends you serialize your custom data to XML, but in our sample code, we opted for JSON serialization instead.

Submitting Data

Now that our HVJournalEntry class is ready, we can add code to submit data to HealthVault.

1. Open **JournalEntryController.cs** in **Controllers** directory
2. Change the **HttpPost Create** implementation to the following

```
if (ModelState.IsValid)
{
    //add the custom type for health vault
    ItemTypeManager.RegisterTypeHandler(HVJournalEntry.TypeId, typeof(HVJournalEntry),
true);

    // get the authed user
    var authorizedUser = (User as HVPrincipal);
    if (authorizedUser != null)
    {
        //get the auth token
```

```
        var authToken = authorizedUser.AuthToken;

        // create the appropriate objects for health vault
        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);

        // before we add make sure we still have permission to add
        var result = personInfo.SelectedRecord.QueryPermissionsByTypes(new List<Guid>() {
HVJournalEntry.TypeId }).FirstOrDefault();
        if
(!result.Value.OnlineAccessPermissions.HasFlag(HealthRecordItemPermissions.Create))
            throw new ArgumentNullException("unable to create record as no permission is
given from health vault");

        //Now add to the HV system
        personInfo.SelectedRecord.NewItem(new HVJournalEntry(journalentry));

        // redirect
        return RedirectToAction("Index");
    }
}

return View(journalentry);
```

This code is very similar to the code in AccountController.Login method where we need to connect to the HealthVault system and retrieve the personInfo object that we will be saving the data to. There are two key items of interest

Registering Custom Data Type

TO use a custom data type, you need to register it with the HealthVault SDK. You need to do this every time you retrieve or add data to/from HealthVault. The following line accomplishes this

```
    //add the custom type for health vault
    ItemTypeManager.RegisterTypeHandler(HVJournalEntry.TypeId, typeof(HVJournalEntry),
true);
```

Check for Access

The second bit of code is the piece where we first check for 'Create' access to HealthVault before we attempt to submit the data. The following few lines accomplish this

```
var result = personInfo.SelectedRecord.QueryPermissionsByTypes(new List<Guid>() {
HVJournalEntry.TypeId }).FirstOrDefault();
        if
(!result.Value.OnlineAccessPermissions.HasFlag(HealthRecordItemPermissions.Create))
            throw new ArgumentNullException("unable to create record as no permission is
given from health vault");
```

Compile and run the program and attempt to create a new entry in HealthVault. The result should be you creating an entry in HealthVault and then being redirected back to the JournalEntry Index page. You will see no items listed, but we will fix that in the next section.

Retrieving Data

Now that we are able to submit data, we need to retrieve the appropriate data items from HealthVault.

1. Open **JournalEntryController.cs** in **Controllers** directory
2. Change the **Index** implementation to the following

```
// register the custom type
ItemTypeManager.RegisterTypeHandler(HVJournalEntry.TypeId, typeof(HVJournalEntry), true);

// get the user
var hvUser = (User as HVPrincipal);
if (hvUser != null)
{
    // get the auth token
    var authToken = hvUser.AuthToken;

    // create the appropriate objects for health vault
    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 = null;
    personInfo = HealthVaultPlatform.GetPersonInfo(connection);

    // before we add make sure we still have permission to add
    var result = personInfo.SelectedRecord.QueryPermissionsByTypes(new List<Guid>() {
HVJournalEntry.TypeId }).FirstOrDefault();
    if (!result.Value.OnlineAccessPermissions.HasFlag(HealthRecordItemPermissions.Read))
        throw new ArgumentNullException("unable to create record as no permission is
given from health vault");

    // search hv for the records
    HealthRecordSearcher searcher = personInfo.SelectedRecord.CreateSearcher();
    HealthRecordFilter filter = new HealthRecordFilter(HVJournalEntry.TypeId);
    searcher.Filters.Add(filter);

    // get the matching items
    HealthRecordItemCollection entries = searcher.GetMatchingItems()[0];

    // compile a list of journalEntryItems only
    var ret = entries.Cast<HVJournalEntry>().ToList().Select(t => t.JournalEntry);

    // return the list to the view
    return View(ret);
}
```

```
else
{
    // if we make it here there is nothing to display
    return View(new List<JournalEntry>(0));
}
```

The code is commented and similar to the create code with the exception of the last few lines.

```
// search hv for the records
HealthRecordSearcher searcher = personInfo.SelectedRecord.CreateSearcher();
HealthRecordFilter filter = new HealthRecordFilter(HVJournalEntry.TypeId);
searcher.Filters.Add(filter);

// get the matching items
HealthRecordItemCollection entries = searcher.GetMatchingItems()[0];

// compile a list of journalEntryItems only
var ret = entries.Cast<HVJournalEntry>().ToList().Select(t => t.JournalEntry);
```

Here we create a HealthRecordSearcher to allow us to search for records, get the entries, and cast the entries to an HVJournalEntry and then do a lambda expression to only return the JournalEntry objects as a List<>.

Compile and run the code and when you navigate to http://localhost/JournalEntry you will see the entry you created in the previous section.

Deleting Data

To delete items from HealthVault from our list, we need to change our Index implementation so when the Delete link is clicked, we know what record to delete from HealthVault.

1. Open **JournalEntryController.cs** in **Controllers** directory
2. In the **Index** method, delete the following line `var ret = entries.Cast<HVJournalEntry>().ToList().Select(t => t.JournalEntry);`
3. Add the following code

```
var items = entries.Cast<HVJournalEntry>().ToList();
var ret = new List<JournalEntry>(items.Count());
foreach (var t in items)
{
    var je = t.JournalEntry;
    je.HvId = t.Key.ToString();
    ret.Add(je);
}
```

4. Open **JournalEntry.cs** and add the following new property

```
/// <summary>
/// The key and record id of the linked record (or thing as HV Likes to call it) in
HealthVault
/// </summary>
public string HvId { get; set; }
```

5. Open **Index.cshtml** in **Views/JournalEntry**

6. Find the following lines

```
@Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
@Html.ActionLink("Details", "Details", new { id=item.Id }) |
@Html.ActionLink("Delete", "Delete", new { id=item.Id })
```

7. Replace the lines with the following

```
@Html.ActionLink("Edit", "Edit", new { id=item.HvId }) |
@Html.ActionLink("Details", "Details", new { id=item.HvId }) |
@Html.ActionLink("Delete", "Delete", new { id=item.HvId })
```

8. Find the **Delete, Edit, Details** methods and change the **id** param to a string. For example the Delete method should look like this `Delete(string id)`
9. Change the **Delete** method implementation to the following

```
// create the item key
var t = id.Split(',');
var key = new HealthRecordItemKey (Guid.Parse(t[0]), Guid.Parse(t[1]));

// get the user
var hvUser = (User as HVPrincipal);
if (hvUser != null)
{
    // get the auth token
    var authToken = hvUser.AuthToken;

    // create the appropriate objects for health vault
    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 = null;
    personInfo = HealthVaultPlatform.GetPersonInfo(connection);

    // delete the record
    personInfo.SelectedRecord.RemoveItem(key);
}

// redirect
return RedirectToAction("Index");
```

Compile and run the code and attempt to delete the records already created. NOTE: we do not give confirmation in this sample code, but it is good practice to confirm with the user if they want to delete.

Viewing and Editing Data

The code for viewing and editing data will be very similar to retrieving and deleting data so I'll leave it as an exercise for the reader to investigate those options.

Conclusion

In this article we covered how to store custom data in HealthVault from an ASP.NET MVC application. We created a new model and associated Views and Controller, we setup our view to submit data, and finally we submitted data to HealthVault and also wrote some code to delete records from HealthVault.

In the next article, we'll go through how to get this application running on Windows Azure WebRole.