*“Data is a precious thing and will last longer than the systems themselves.”*Tim Bernes-Lee.

|  |  |
| --- | --- |
| *Concepts to cover*   * How to search for datatypes.   + Add Sleep * What is a data type * How to work with datatypes   + Codable     - Add Exercise to existing application * Data Provenance   + Signing Weight readings * Extending Data types   + Adding a custom tracker / hypothesis | *To do*   * Add a diagram for the end result |

The Quantified Self [<http://quantifiedself.com/about/>] community enables self-knowledge through self-tracking. Self-tracking, when powered by appropriate data analysis, has been proven to trigger behavioral change. The act of self-tracking creates awareness and feedback. The hunger for, and success of, self-knowledge is evident from the growing number of 6000+ self-quantifiers in 41 cities around 14 countries.

Self-knowledge is possible only with a substantial collection of data about oneself. HealthVault provides more than 80 granular data types that enable tracking data regarding everything from daily exercise to genome sequences. In this chapter, we will build upon the understanding of the HealthVault application programming interface covered in Chapter 3 and extend it to develop a data intensive self-quantifying application. Through the Quantified Self application we will gain an understanding of HealthVault data types and application development.

# A Self-Experimentation Application

In Chapter 1 we analyzed weight data, while in Chapter 2 we worked with sleep information and correlated it with exercise. HealthVault offers a data type for tracking emotional state and daily dietary intake as well. Let’s consider building a simple Quantified Self utility that helps a user keep track of her emotional state, daily dietary intake, weight, sleep and exercise. Tracking these types of data and their relation to each other, would allow our user to form and prove interesting hypotheses such as: “I’m happier if I sleep well, and I sleep well if I drink less alcohol and exercise sufficiently.”

Self-tracking fosters awareness and a feedback loop; numerous participants in the Quantified Self movement, like Amy [link], have attributed improvement to insights generated by the data and the act of data collection. Our “Quantified Self” application will aim to emulate this pattern. Fig 4-1 summarizes the data pattern we wish to capture.

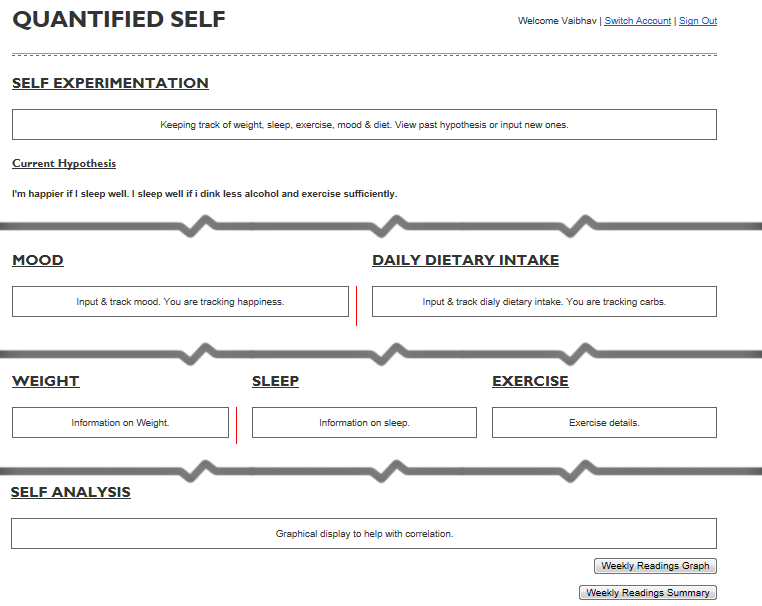
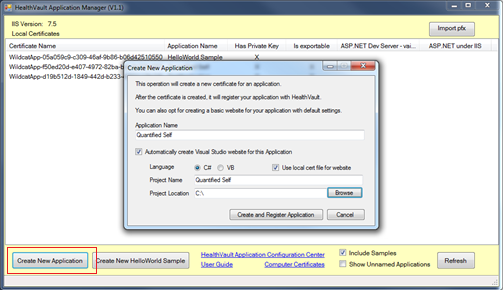


Fig 4-1 Data dashboard for Quantified Self application

## Setting Up New HealthVault Application

Let’s start by making a “Quantified Self” application with a unique application identifier In this chapter we will use the HealthVault .NET SDK in order to focus on understanding the HealthVault data types. However as [Chapter 4, Section 4] outlines, you can use other languages and HealthVault libraries as well.

The first step in creating the application is to download and install the HealthVault SDK from MSDN [http://msdn.microsoft.com/en-us/healthvault]. After installing the SDK, you will notice a utility called Application Manager. From the Windows Start buttonthis utility can be accessed through All Programs→Microsoft HealthVault→SDK→HealthVault Application Manager.

  
Fig 4-2 Using Application Manager to create a new HealthVault application

Once you open the Application Manager, you will notice the “Create New Application” button, which you should use now to create a new application. As Fig 4-2, shows, the new application creation process asks you for an application name and other details and creates a Visual Studio solution with the application starting point.

The second step in the process is to register your application. Application Manager automatically opens a new browser window that signs you into the HealthVault Application configuration utility (https://config.healthvault.com) and creates the appropriate application in the HealthVault Development environment. The development environment is referred frequently also as PPE, which stands for pre-production environment. In the next chapter we will learn how the Application Configuration Center can be used to create a development application without using the Application Manager.

On the dashboard of HealthVault Application Configuration Center you will see the application you just created, as depicted in Figure 4-3.

Fig 4-3 HealthVault Application Configuration Center showing the application that was created

## Adding Data types

HealthVault offers more than 80 granular items to which a user can authorize access. They fall into categories such as fitness, condition, medications, health history, measurements, personal profile, files and custom data. A developer can obtain access for particular health data items by configuring an application’s authorization rule set. For our application, we need access to weight, sleep, and exercise data, which come directly from various devices. We also want the user to be able to track emotional state and daily dietary intake, information that she will enter manually.

To start the necessary configuration, click on the application ID in the HealthVault Application Configuration Center. Fig 4-4 illustrates the view of our “Quantified Self” application after clicking on the “Online rules” option. In this menu, select the appropriate data types for the application (weight measurement, sleep, exercise, etc.), select all permissions (read, write, update, delete), provide a reason why the application needs access to these types, and name the rule. A rule can also be configured to be optional and can have display settings. Why String, Is Optional and Display Flags items are currently not active for most HealthVault applications.

We are using HealthVault as the user authentication provider for our application, so we choose to operate in the online mode and create an authorization rule for such access. If we wanted our application to work through a back system provided by one of the other types of architecture discussed in Chapter 3 [link], we would configure the offline rules for access to appropriate data types.

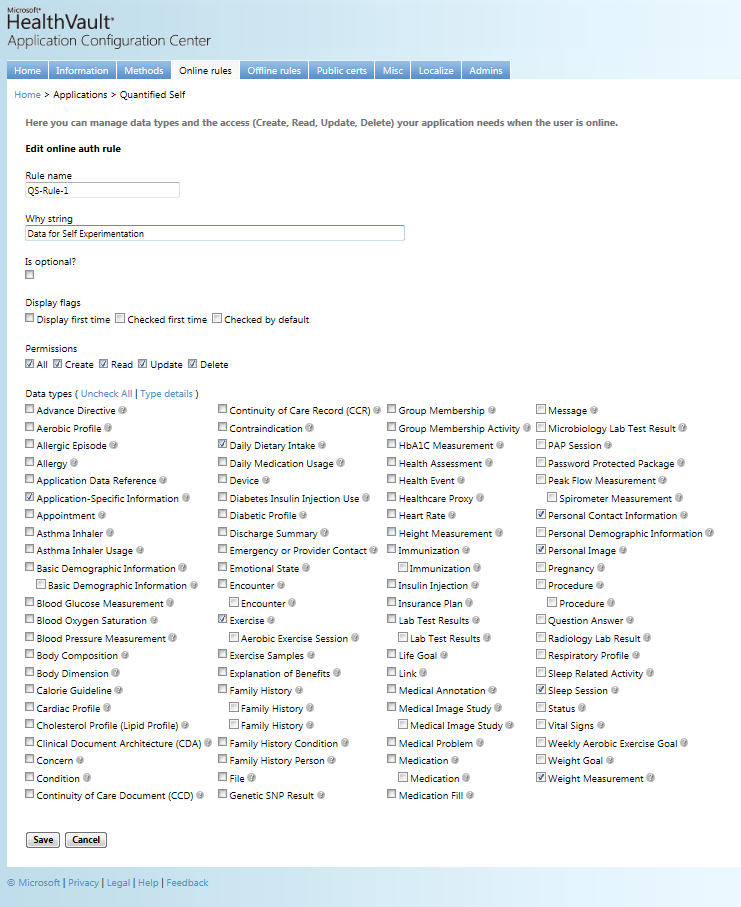


Fig 4-4 Configuring online rules for an application

We are finished selecting the appropriate data types for our application, and can now try accessing them through the application.

## Accessing the data types

The application manager utility creates a template application. Fig 4-5 shows the initial solution created by this utility.

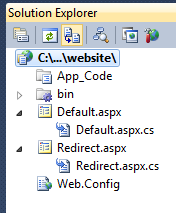
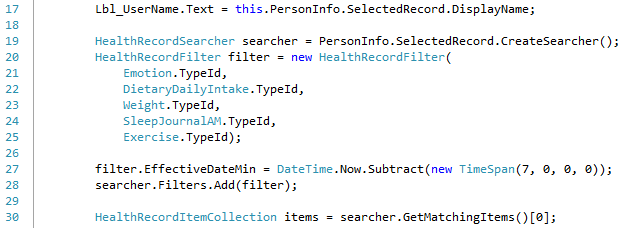


Fig 4-5 Solution created by Application Manager

The solution makes sure that your application is configured properly with an appropriate application ID, points it to appropriate HealthVault platform and shell development environments, and configures the application’s redirect URL; all of these configurations live in the Web.Config file. The Default.aspx page is derived from the HealthServicePage and handles authorization with HealthVault Platform, while the Redirect.aspx page is derived from the HealthServiceActionPage and handles authentication and interaction with HealthVault Shell. The bin folder contains HealthVault SDK libraries: Microsoft.Health.dll, which encapsulates the core Healthvault functionality; Microsoft.Health.Web.dll, which broadly encapsulates browser interaction, and Microsoft.Health.Itemtypes, which encapsulates an object model for all HealthVault data types.

The main solution doesn’t add a master page. In order to make it easy to extend functionality, we create a MasterPage named QuantifiedSelf.master and create a fresh Default.aspx page, after deleting the old one and ensure this page is derived from HealthServicePage.

As discussed in Chapter 3, Section 2], we can use the HealthVault GetThings API to access health items in a user’s health record. The code in listing [4-1] accesses Emotion, DietaryDailyIntake, Weight, Sleep, and Exercise from HealthVault. As in Line [27-28], we are making sure to fetch these elements for last 7 days only.



Listing 4-1 GetThings call to access multiple things

Before we display these types, let’s dig deeper to understand a HealthVault data type.

# Understanding HealthVault Data Types

A comprehensive list of all HealthVault data types is available from the HealthVault developer center at <http://developer.healthvault.com/types/types.aspx>. Each type has properties that determine to a great extent how items are created and used. To understand a type better, let’s take a deeper look at the example of the Weight Measurement type.

## Type Properties



Fig 4-6 Properties of Weight Measurement type

Fig [4-6] shows the properties of the Weight Measurement data type that are common to every data type from the HealthVault developer center (<http://developer.healthvault.com/types/type.aspx?id=3d34d87e-7fc1-4153-800f-f56592cb0d17>). Each HealthVault type has a unique identifier; this “id” is used by the HealthVault APIs to identify the type. In the case of Weight it is 3d34d87e-7fc1-4153-800f-f56592cb0d17. A type sets the “uncreateable” property to true if no application can create such a type in a user’s HealthVault record; a good example of this is Basic type. The “immutable” property is true if no application can modify or update an instance of that type in the user’s HealthVault record; a good example of this is CCR type. The property “singleton” is true if only one instance of that type can exist in a user’s HealthVault record; a good example of this is the Basic Demographic type.

### Type Transforms

Additionally the list of transforms is a property associated with the type. Transforms are inbuilt XSLT transformations available for a particular thing type. These transforms let you convert the XML associated with a particular type to various formats such as HTML, a representation compatible with various popular health care standards, or an older or newer version of the same type.

#### Form, MTT and STT Transforms

Common among all the types are the **form, stt** and **mtt** transforms. **form** provides an HTML table representation of an instance of the entire thing. **stt**, which stands for “single type transform,” provides a row-based representation of the type so that it can be viewed as a list of instances of the same type. **mtt**, or “multiple type transform,” provides a row-based representation of the type so that it can be combined and viewed with multiple HealthVault types. Each row in **mtt** has a summary attribute representing the details of the type. The main difference between stt and mtt is that, stt has an XML attribute for each meaningful data element of the type, while mtt summaries all the meaningful data elements in one string in the summary attribute.

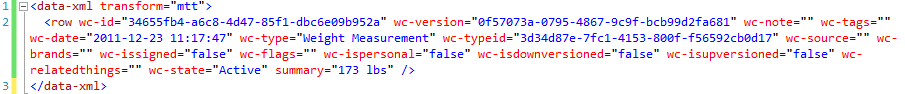
One can use the HealthVault Powershell plugin to view each source of the transforms. Listing [4.2] shows how to save the **form** transform for the Weight thing type.

**PS \> (Get-ThingType 3d34d87e-7fc1-4153-800f-f56592cb0d17).TransformSource["form"] | out-file Weight.xsl**

Listing4-2 Saving Form XSLT transformation for Weight thing-types to a file

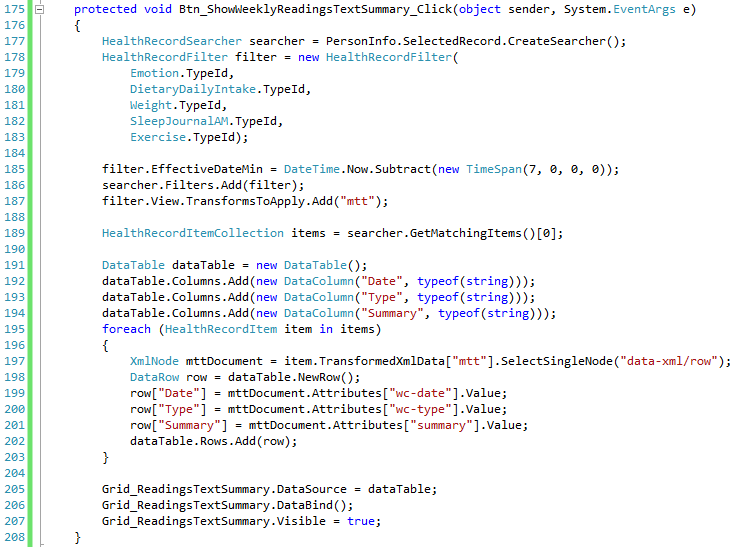
The columns on the type definition page in HealthVault Type Explorer define the column header, .NET data type, and width for each column. It’s handy to view this information about the type in a data grid.

Listing 43 shows the multi type table transformation XML returned by the HealthVault platform for the Weight type. We can see the columns ranging from wc-id (type identification) to summary (summary information of the type).



Listing 4-3 Weight MTT Xml for weight type.

In our Quantified Self application we can use the **mtt** transform to easily display multiple types in the same table for self-analysis. Lines 177-189 construct and fetch our query from HealthVault; note that in Line 187 we ask the HealthVault platform to apply the **mtt** transform on the returned items. In Line 197, we select the row for each data-xml **mtt** transform. We then display the wc-date, wc-type, and summary columns (Line 199-202). Different applications can choose to show different columns. Individual type columns, like **weight** for Weight, are available in single type transform (**mtt**) whereas a summary column summarizes this information in **mtt**. The HealthDataItemGrid control is also available from HealthVault .NET SDK to show this information automatically.



Listing 4-4 Viewing Multiple HealthVault types in a data grid.

Once we have the Data grid configured, we can view the summary of all types in the same column structure. Figure 4-7 shows how this information is displayed in our Quantified Self application.

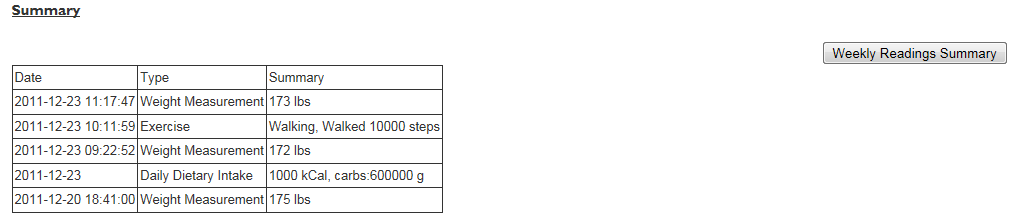


Figure 4-7 Quantified Self application showing multiple types in a data grid

<aside>  
The CCR HealthVault type (1e1ccbfc-a55d-4d91-8940-fa2fbf73c195) has a **tohv** transform that converts the data in that type to individual HealthVault elements.

In addition to the use of transforms to convert types to different representations, the HealthVault method schema provides a <final-xsl> element in each method header. final-xsl converts the data returned by the method call to in-built transforms, such as converting to CCR (toccr), CCD (toccd), CSV (tocsv), or RSS (torss). Final-xsl also allows the caller to specify a custom-built XSLT transform that the HealthVault platform runs on the output before sending it to the requestor.  
  
The final-xsl element is specified between the <country> and <msg-time> elements in header of a method. In the HealthVault .NET SDK, one can call this functionality by using the GetTransformedItems [link] method. In the Java .NET Open Source library, this functionality can be used through a call to request.setFinalXsl(“transform name or transform source”).

</aside>

#### Versioning Transforms

HealthVault data types can have multiple versions. As the HealthVault ecosystem matures, existing types need to be updated or modified to match new use cases. Medications, Basic Demographic Information, are Family History are good examples of types that have multiple versions. You will notice that the older Medication, available at <http://developer.healthvault.com/pages/types/type.aspx?id=5c5f1223-f63c-4464-870c-3e36ba471def>, has an up version transform and the newer Medication , <http://developer.healthvault.com/pages/types/type.aspx?id=30cafccc-047d-4288-94ef-643571f7919d>, has a downversion transform. Through these transforms, HealthVault provides an easy way to move data between an older and newer version of a data-type.

<aside>

Versioning of data types is unique to HealthVault among personal health data platforms. Personal health records are meant to exist over a lifetime, and this feature makes moves seamless from older health items to newer health items.

</aside>

#### Other Transforms

Transform names containing wpd\*and \*hvcc\* enable the HealthVault Connection Center to convert Windows Portable Device Data to and from HealthVault XML.

## Type Schemas

Now that we have understood the high-level properties associated with a type, and have used the MTT display transform to show the summary all data types in our application, let’s take a closer look at what is entailed in a type’s schema, with the specific goal of displaying appropriate values for the Weight type.

Weight is a simple type that scales and applications can write to or read from. The XML schema and associated sample representation for this type are shown in Fig 4-8

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Fig 4-8 XML and schema representation of the HealthVault Weight type

The Weight type consists of a sequence of date/time and weight values. The use of date/time in HealthVault is defined in the *dates.xsd* schema file (<https://platform.healthvault-ppe.com/platform/XSD/dates.xsd>) and the weight values are defined in the *types.xsd* schema file ( <https://platform.healthvault-ppe.com/platform/XSD/types.xsd>).

The HealthVault .NET Web SDK encapsulates a nice object model on top of this XML and gives a user access to Value and When fields as shown in Fig 4-9

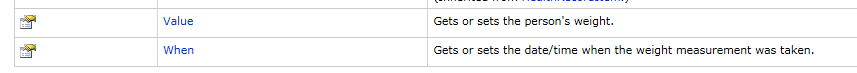
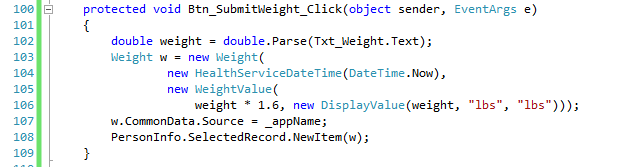


Fig 4-9 Properties in HealthVault .NET SDK for the Weight Class

### Units and measurements

Note that the Value field of this type contains display and units data. HealthVault stores the underlying measurement in kilograms, but the application can show it to the user in the same form in which it was entered. In our example Quantified Self application, we ask the user to input values in pounds. Listing 4-5 shows how we convert this value to kilograms for storage while displaying it to the user as pounds (lbs).



Listing 4-5 Creating a new Weight Value.

### Dates

The When field or date is a special type called HealthServiceDateTime. As Line 104 shows, an instance of this date can be created by using the System DateTime. HealthVault enables a user to enter varying degrees of date precisions; hence it has a custom date time.

In fact, the HealthVault approximate datetime construct allows you to create a date as flexible as “when I was a kid” or “Jan 2011” or “Dec”. All the kinds of HealthVault dates are defined in dates.xsd available at <https://platform.healthvault-ppe.com/platform/XSD/dates.xsd>.

<aside>One of the core HealthVault design tenets is to ingest all kinds of data. Flexible dates enable a user to enter unstructured data. Furthermore, constructs like approx.-date-time allow HealthVault to receive data from standards such as CCR or CCD. </aside>

### Common Data

All types share some common data elements. In Listing 3.4, line 107 we are writing to the common data element that shows the source of the application.

Other commonly used data elements are notes, tags, and related-items.

### Terminologies

HealthVault provides an extensible mechanism to specify strings coded for use across various systems, through codable-value. The codable-value consists of a text associated with the code, and the code represented in a structured format called coded-value.

Terminologies are used in a HealthVault data element called codable. Codable provides a structured way to represent semantically meaningful text.

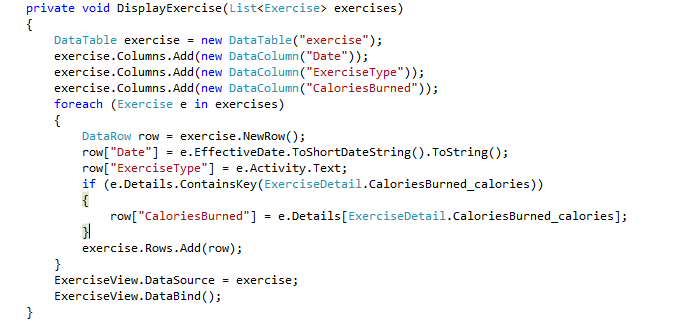


Fig 4-10 codable-value Schema

Figure 3.10 shows a codable-value schema. The family data field of the coded-value represents if the code belongs to particularly code system; wc refers to HealthVault code system, HL7 refers to system adopted by Health Language 7.

HealthVault has more than 150 terminologies. The wiki [link] describes how these terminologies are used by the HealthVault user interface, and this article [link] describes how meaningful use as proposed by federal regulations dictates the use of these terminologies.

Listing 4-6 shows the how one can read the Exercise data type for showing calories burned. The Exercise data type stores various kinds of attributes in key value pairs. These attributes are listed in the ExerciseDetails terminology. As line [] show one can use the code value of CaloriedBurned from the ExerciseDetail terminology to lookup the appropriate value, and display it in the user interface.



Listing 4-6 Listing Calories Burned in DisplayExercise Function.

# Extending HealthVault Data Types

Applications frequently have to represent something that is not encapsulated by the data-structure of the HealthVault data-types. Out-of-box HealthVault provides a mechanism by which a data type can be extended.

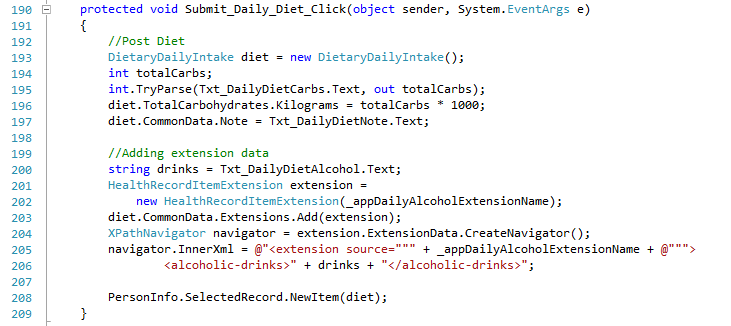
Every application can choose to write XML information in the extension tag with the common data section of a data type. It is recommended that applications distinguish their extension elements by using a unique source attribute on the extension element.

In our example, let’s assume we are extending the daily dietary intake type to add information on alcohol consumption.

## Creating a type extension

We would like to track the amount of alcohol consumed in a simple element called “alcoholic-drinks”. The simply things further we assume this element represents the number of alcoholic drinks including wine, beer, cocktails etc., and is normalized to mean average alcohol per unit.

The first step is to write an alcoholic-drinks XML element within the extension tag using a unique source (\_appDailyAlcholoExtensionName) in the extension element. Listing 4-7, Line [204-206] in the following example shows how one can do it in the .NET SDK.



Listing 4-7 Creating a type extension

## Consuming a type extension

The second step is to read information from the extension. In our application, the user enters alcoholic drink through a text box associated with the Daily Dietary intake section, as shown in Fig 4-11

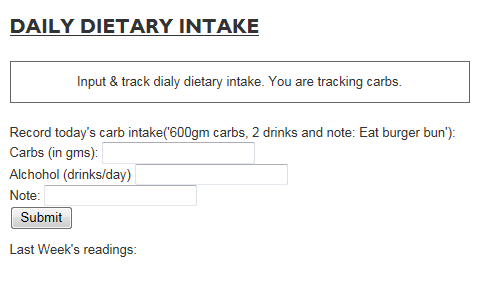


Fig 4-11Input section for Daily Dietary Intake

Line [228-230] in listing [4-8] shows how one can read <alcoholic-drinks> XML. To parse this information we use XPath, in the data type document the element of interest resides at extension/alcoholic-drinks. Using the .NET XPathNavigator class we select a single note signifying this value (Line 228-229]. Line 236 fetches the note associated with this instance of daily–dietary intake. Potentially the user can input clarify information like drank 3 tequila shots in this element.



Listing 4-8 consuming a type extension

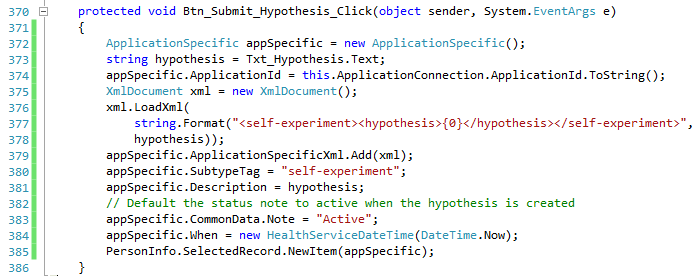
Applications may choose to combine and formalize the two steps just shown and create an extension class, which could be then registered with HealthVault SDK so that every time the extend type is accessed by the application the appropriate extension properties are available.

# Creating Custom Types

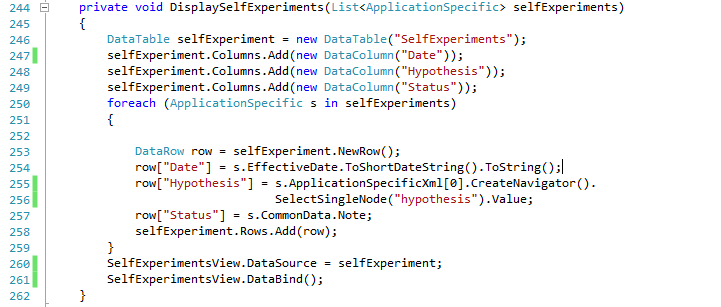
Extending a HealthVault data type might not always solve your data needs. Many times there are legitimate use cases for which the application needs a unique data repository. For example, in our Quantified-Self application, we need repository to store all of the users self-experiments.

HealthVault provides a mechanism called an *application specific type* for this purpose. This type is not shareable with other applications. Once application developers find a broader use for their data, they can work with Microsoft to create a first class data type for their need.

Listing 4-9 shows how one can use an application-specific type to store self-experiment hypothesis for the Quantified-Self application. In our application we are asking a user to create a hypothesis use a simple text box. The value of this text box is read as hypothesis string in Line 373. In Line 375-378, we create an xml document with the data for this specific type and then add it to the using the ApplicationSpecificXml in Line 379. Each application specific type requires a subtypetag and description (Line 380-381). We also specify the application creating this type in Line 374. Additionally we use the common note element to capture the status of the type in Line 383 and the When element captures the date.



Listing 4-9 Writing Application Specific Custom Type

On the other hand, we can show the list of Self experiments by reading the ApplicationSpecificXml using an XPath navigator. In Listing 4-10, note in Line 255-256, we assume that the document for this type contains only one element and the first node is hypothesis. 

Listing 4-10 Reading Application Specific Type

# Trusting Data in HealthVault Data Types

Knowing the origin of data is often critical for an application which is using it for sensitive purposes. Some use cases warrant working with only trusted data, some warrant knowing whether the data is from a device or self-entered by the users and in some cases the application might just want to work with known data providers.

HealthVault provides several ways to look at data provenance. Applications can look at the created\_by and updated\_by fields of a data-type and see whether they were updated by devices or known applications. Additionally, HealthVault provides digital signing of data, which can create a very secure ecosystem of trust

In our example, we look at the Source attribute of Weight items to see whether they were uploaded by a Withings scale or manually added by the user.

<aside>

### Digitally signing data

HealthVault provides a way to digitally sign all data types. **For instance,** using the .NET SDK, an application can Sign HealthVault types very easily, Listing 4-11shows a snippet of code which can be used to sign weight data type. The certificate in cert.Import can be obtained from a trust provider like VeriSign, Comodo etc.

protected void Btn\_SubmitAndSignWeight\_Click(object sender, EventArgs e)

{

double weight = double.Parse(Txt\_Weight.Text);

Weight w = new Weight(

new HealthServiceDateTime(DateTime.Now),

new WeightValue(

weight \* 1.6, new DisplayValue(weight, "lbs", "lbs")));

X509Certificate2 cert = new X509Certificate2();

cert.Import("..\\cert\valid\_cert.pfx");

w.Sign(cert);

PersonInfo.SelectedRecord.NewItem(w);

}

Listing 4-11. Signing Weight data

The verification of a digitally signed certificate is available through IsSignatureValid() and ValidateCertificate() methods in HealthVault .NET SDK.

In the samples associated with this Chapter (ThingSignatureSample.java) you can review code for doing digital signing of HealthVault data type using the HealthVault Java library.

</aside>

# Relating HealthVault Data Types

HealthVault data types are intended to be self-contained units of health information. The data types have distinct health items like medications, immunizations, and weight readings. This approach is characteristically different from a relational data modeling; in which the data is normalized and stored in distinct tables which have explicit relationship with each other. For example in a relational model medications may be expressed as separate medication-name and medication-dosage tables.

Many times there is a need to represent relationships between individual health items. For example, a medication is inherently related to medication-fill. Medications are associated with a person’s profile as prescribed by a physician, and medical-fill is used by a pharmacy to prescribe units of medications to a consumer as they consume the prescribed medications.

The relationship between medication and medication-fill is expressed by ***related-items***. HealthVault offers related-items as an inherent mechanism to link and associate data-types. Related-item is a special chunk of XML that resides in the common data of a health-item. Relationships are usually described in the dependent item and link to the more independent one. For instance, to express the relationship between medication-fill and medication, one places related-items in the medication-fill and point to the medication.

Another interesting use of related-items is to link together a set of readings being uploaded from a single device. For example, a device calculating body fat-percentage and cholesterol can associate them through related-items while uploading them. Since this association is done before uploading to HealthVault, a special unique identifier called client-id can be used. Client-Ids are usually unique identifiers associated to instances of HealthVault data types and are created by the client uploading the data.

One can take relationships even further and associate a set of medical-images, medications, and conditions as a result of a particular health incident, maybe an accident. The Mayo Clinic Health Manager application provides a way to create a web of related HealthVault items.

Related-items lie beyond the scope of this book, but the reader is encouraged to explore them and contribute interesting uses and examples at enablingquantifiedself.com.

# Exploring HealthVault Data Types

In our example, we picked some HealthVault types to be used in the application based on our device, data availability, and purpose. Every application programmer needs to go through this data exploration based on your needs and goals. This section gives an overview of all HealthVault types so that the reader can have a good understanding of what is available in the system.

**Categorizing HealthVault Data Types**

HealthVault stores personal health information ranging from fitness data to medical images. Table 4-1 shows the categorization of the data as displayed to end user.

|  |  |
| --- | --- |
| **Category** | **HealthVault Types** |
| Fitness | Aerobic Exercise Session, Aerobic Profile, Aerobic Weekly Goal, Calorie Guideline, Daily Dietary Intake, Exercise, Exercise Samples, Weight Goal |
| Conditions | Allergy, Concern, Condition, Contraindication, Emotional State, Pregnancy, Problem, Respiratory Profile |
| Medications | Asthma Inhaler, Asthma Inhaler Use, Daily Medication Usage, Insulin Injection  Insulin Injection Use, Medication, Medication Fill |
| Health History | Allergic Episode, Annotation, Cardiac Profile, Diabetic Profile, Discharge Summary, Encounter, Explanation of Benefits, Family History, Family History Condition, Family History Person, Health Assessment, Immunization, Procedure,  Question Answer |
| Measurements | Blood Glucose, Blood Oxygen Saturation, Blood Pressure, Body Composition, Body Dimension, Cholesterol Profile, Device, Genetic SNP Results, HbA1C  Heart Rate, Height, Lab Test Results, Microbiology Lab Results, PAP Session, Peak Flow, Radiology Lab Results, Sleep Journal AM, Sleep Journal PM, Spirometer, Vital Signs, Weight |
| Personal Profile | Advance Directive, Appointment, Basic, Contact, Healthcare Proxy, Life Goal,  Payer, Person (emergency or provider contact), Personal Demographic Information, Personal Image |
| Files | Clinical Document Architecture (CDA), Continuity of Care Document (CCD), Continuity of Care Record (CCR), File, Medical Image Study, Password-Protected Package |
| Custom Data | Application Data Reference, Application Specific, Group Membership, Group Membership Activity, Link, Status |

Table 4-1 End User Categorization of HealthVault data

### Fitness

HealthVault offers a range of fitness types. The most commonly used fitness data type is Exercise. Exercise provides a terminology-based categorization of kinds of exercise: e.g., walking or running. Each activity can also be associated with terminology driven units: Count, Mile, etc.

Devices like FitBit, Withings work with this type. The exercise activities terminology lists a range of exercise values including running, walking, swimming etc. Devices that fetch detailed information on exercise can write individual samples to the Exercise Sample type. For instance, Polar watch [link] writes to exercise samples in addition to summarizing the workout in Exercise type.

This category of types is implemented in a fairly generic way so that various industry formats, such as the one used by Garmin’s connect web site (<http://connect.garmin.com/> ), can translate easily to these types. Units from ISO can also be translated easily to HealthVault units.

### Conditions

Health problems, allergies, contra-indications, and mental health (emotional state) are categorized in the Condition set of types. Conditions are sensitive health problems that usually have an onset date and a status associated with them.

The HealthVault Shell uses the Condition type to record conditions. Conditions entered through the user interface are mapped to SNOMED-CT terminology.

### Medication

Medications are the center of modern medicine. HealthVault offers a number of granular types to capture the essence of medications.

A number of pharmacies, including CVS and Walgreens, offer applications for importing prescription data in to HealthVault. But the user interface and integration for these applications is a bit challenging.

The most frequently used data types in this category are Medication and Medication Fill. Each prescription could be broken in to Medication and Medication Fill. Medication Fill is the recurring part of one’s prescription. As you may recall from the Data Relationship section (Section 5), Medication Fill type is usually related to Medication using the related-item semantics when entered through HealthVault Shell, medications are mapped or coded to the RxNorm Active Medicines terminology.

### Health History

Immunizations, Procedures, Family history, Health events, etc. form the basis of Health history.

The most notable application using types in this category is the Surgeon General‘s Family History application. This powerful application, <https://familyhistory.hhs.gov/fhh-web>, enables individual to easily create family Health history tree.

### Measurements

Measurements are the most extensive category of HealthVault data types. Measurements range from the output of various fitness devices to lab results. For instance, the Withings weighing scale writes to the weight measures, and FitBit writes to sleep. The measures are granular records of daily activity and make it traceable.

On the other hand, the Lab test result, one of the most complicated HealthVault types, represents results from labs. It can be used in conjunction with industry standard terminologies.

### Personal Profile

The Personal profile category of HealthVault types contains data pertaining to Healthcare proxy, personal image, and demographics. Almost every HealthVault application showing a user’s picture or looking at their age or other demographic information is using types in this category.

### Files

HealthVault, unlike most personal health records, allows you to upload a number of types of files, and therefore supports data types for these files. Fig 4-12 shows the file extensions supported, displayed through a GetServiceDefintion call in Powershell. The GetServiceDefinition information can also be viewed online at healthvault developer center in service definition section [<http://developer.healthvault.com/pages/methods/methods.aspx>]

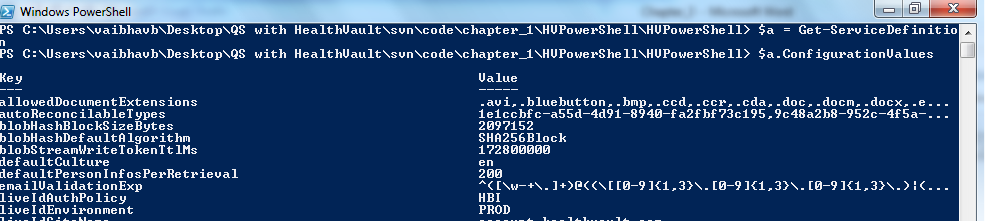


Fig 4-12 List of file extensions supported by HealthVault

The Medical Image study type used by the HealthVault connection center uploads DICOM medical images in this type. The CCD/CCR types are industry standard ways by which various hospital information systems send care records to HealthVault. Google health users, for instance, migrated to HealthVault using the CCR type. The Message file type is the backbone of HealthVault Direct integration. Any email message received by the user is stored in the Message type.

### Custom Data

The Application Specific Type, already covered in the Creating Custom Types (Section 4) with regard to adding a repository in which to store self-experiments, is the most important custom data type. The Application Specific type is used by various application to store information in HealthVault for which no other type or extension to a type is appropriate. For instance, the spending scout application used to store explanation of benefit information in this type, until the HealthVault team created an Explanation of Benefits (EOB) type to support it more directly.

# Contributing to the Self-Experimentation Application

In next Chapter we will see how we can augment the self-experimentation web application by creating mobile applications. The source for the application is available at enablingquantifiedself.com, and we are inviting you, dear reader to extend this application and make it your own. Perhaps forks the git repository and contribute your code back or create java, ruby, python versions of it!