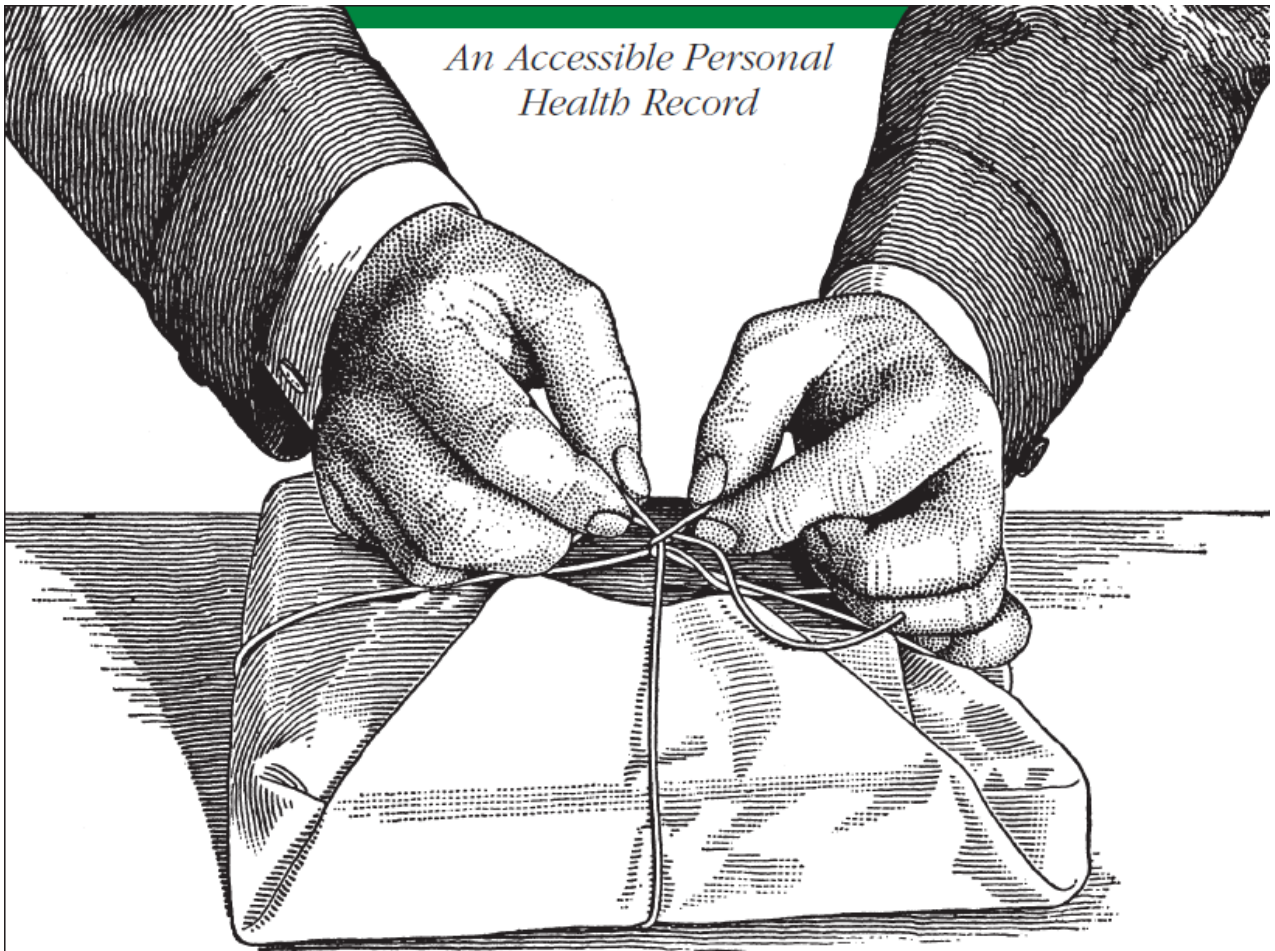


*An Accessible Personal
Health Record*



Enabling

Quantified Self with HealthVault

O'REILLY®

Vaibhav Bhandari

Outline of the work

Microsoft HealthVault is the most prominent example of a personally controlled health record (PCHR). With its open API, flexibility and connections with multiple health care providers, it gives people interested in monitoring their own health an unprecedented opportunity to do their own research on their own data. This concise book will explain what you can store in HealthVault, how to enable automatic updates from well-known fitness devices, and how to use programming libraries to create reports and investigate trends of interest to you.

Table of Contents

Outline of the work.....	2
Table of Contents.....	3
Foreword.....	5
Preface	6
Organization of the Book	6
Conventions Used in This Book.....	7
Acknowledgements.....	7
Chapter 1 Getting Started with HealthVault.....	8
What is HealthVault?	8
Getting started with HealthVault.....	8
Overview of HealthVault Features.....	10
Working with Health Data	13
Using partner applications.....	15
Chapter 2 Quantifying yourself.....	18
How Fitbit Tracks sleep	18
Exploring the HealthVault data.....	22
Analyzing the HealthVault Data	24
Chapter 3 Interfacing with HealthVault.....	28
Accounts and Records.....	28
HealthVault Application Programming Interface.....	30
HealthVault SDK and Open Source Libraries.....	44
Interfacing with HealthVault.....	47
Chapter 4 Using HealthVault Data Ecosystem for Self Tracking	52
A Self-Experimentation Application.....	52
Understanding HealthVault Data Types.....	57
Extending HealthVault Data Types	64
Creating Custom Types	66
Trusting Data in HealthVault Data Types.....	67
Relating HealthVault Data Types	68
Exploring HealthVault Data Types	69

Contributing to the Self-Experimentation Application	71
Chapter 5 Enabling mHealth for the Quantified Self	72
The Mood Tracker Mobile Application	72
Getting Started.....	73
Reading Data from HealthVault	79
Writing Data to HealthVault	82
Graphing Mood	83
Data Analysis – Mood Plant	85
What about Android and iOS?	86
Mobile Web Applications.....	86
Contributing to the Mood Tracker Application.....	87
Chapter 6 The Last Mile – Releasing Applications to Users.....	88
Testing your application.....	88
Releasing your application to end users	89
Monitoring and maintaining your application	90
Adding new features to your application	90
Taking your application international!.....	91
Further Resources	91

Foreword

Back in the Spring of 2006, I was getting headaches consistently around lunchtime every Saturday. It was really weird. At first I didn't recognize the pattern, I just knew that my head hurt a lot, and tried to make it go away by popping ibuprofen. The pills kind of worked, but not really. After way too long, I finally realized what must be going on.

One of the classic things everybody knows about Microsoft is that they give employees free soda. It's a pretty cool perk, but for those of us with no moderation switch; it can get a bit out of hand. When I came back to Microsoft in 2006 to start the HealthVault team, I quickly ran up a Diet Coke habit in the range of sixteen each day. All week – until Saturday, because the fridge in my house doesn't magically regenerate Diet Coke.

Suddenly it was just blindingly obvious: I was suffering from caffeine withdrawal. Now, a better man than I would have recognized that all that soda probably wasn't a good idea anyway. But instead, I just switched to caffeine-free Diet Coke and the headaches disappeared. I still spend a lot of time running to the restroom, but that's another issue altogether!

I love this story because it's so simple and obvious --- and yet it offers up a clear path to making improvements in all aspects of clinical care:

1. We have to **measure** our bodies over time and space.
2. We have to **correlate** the data we measure to identify patterns.

Doctors measure a lot of stuff to try to understand problems in the human body: labs, imagery, vital signs, and more. But these are all done as isolated snapshots, and all too often patterns that occur over time (weeks, months, years) and space (at home, at work, traveling, etc.) hide away undiscovered.

Historically this was understandable, because measuring the body has been hard and often inconvenient. In order to be useful, the amount and diversity of data required can be significant. But the world has changed, and now it's easy for anybody to create a holistic picture of their health with data.

This is one of the big reasons we created HealthVault. We recognized the importance of a comprehensive "hub" where people could collect all of this diverse information together, and where smart people could provide analysis tools to look for patterns and trends. For us, the "quantified self" has been a target from day one.

Vaibhav has been part of the HealthVault team for a long time, working with partners and our internal team to constantly improve the service. He's really done a great job in this book of showing what's possible when you take a platform like HealthVault, combine it with an ecosystem of innovative measurement devices, and make the data available for analysis in familiar tools like Microsoft Excel. And that's not all – he walks us through building HealthVault apps for the web and mobile phones, somehow cramming a ton of great information into a pretty manageable read. I hope he'll inspire an avalanche of new "body hackers" who can help show us what's possible.

It's pretty amazing stuff --- and frankly we've just gotten started. So have fun!

Sean Nolan, Distinguished Engineer
Microsoft Health Solutions

"The groundwork of all happiness is health."

Leigh Hunt

Preface

Organization of the Book

Although the chapters cover different topics, they have been arranged so that the concepts and techniques in earlier chapters form a foundation for the others.

Chapter 1. Getting Started with HealthVault

Health is critical to all of us. Healthcare and the infrastructure around it touches our lives and the lives of our loved ones. Many of us, in pursuit of long-term health, adopt goals ranging from controlling our weight to long-distance running. The Healthcare industry is in an early stage of realizing the power of the digital world, and the effectiveness of networks in helping drive a change.

This chapter introduces HealthVault as a powerful tool for interacting with health data. It also provides a walkthrough of functionality available to the end user through HealthVault.

Chapter 2. Quantifying yourself

Data is a powerful behavior change tool. The act of simply tracking something changes one's perception of that activity. Summarizing the data over time provides a yard stick by which to measure. The act of tracking activity overtime uncovers patterns in behavior. The structured data in HealthVault provides such an opportunity. Moreover the HealthVault ecosystem offers a variety of applications and devices to assist in this endeavor.

In this chapter we will explore how a consumer can use various devices to track critical health measures. We will also use common tools to explore the data stored by devices in to Microsoft HealthVault. We'll capture and view some data, then use a PowerShell plugin to extract selected data to a CSV format and manipulate the data in that format.

Chapter 3. Interfacing with HealthVault

As a platform HealthVault provides an innovative access management and programming interfaces for applications and devices to access a user's health information.

This chapter takes a closer look at the application programming interface offered by HealthVault to enable this interaction in a programmatic fashion. We will discuss various ways in which an application or device can interface with the HealthVault platform. The code samples will use .NET interfaces because they fit well with HealthVault, but the same interfaces are available in Java, PHP & other languages. The chapter will introduce the elements of programming that give the programmer access to data in HealthVault. Towards the end, we will discuss various architectural options available for interface an application or device with HealthVault.

Chapter 4. Using HealthVault Data Ecosystem for Self Tracking

The Quantified Self community is engaged in enabling self-knowledge through self-tracking. Self-tracking, 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 only possible with a substantial self-data. 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 2 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.

Chapter 5. Enabling mHealth for the Quantified Self

Having an accessible and programmable health record sets HealthVault apart. It enables a rich ecosystem of devices, mobile and web applications. Chapter 2 focused on introducing the HealthVault application programming interface; Chapter 3 gave a good overview of HealthVault data types using a data-intensive “Quantified-Self” application. This chapter takes a closer look at building mobile applications for HealthVault.

We will look at an end to end example of building a mood tracking application on top mobile platforms. The chapter will covers element of mobile client programming using the code samples for Windows Phone 7 (C-Sharp), similar interfaces are available for Android (Java) and iOS (Objective-C).

Chapter 6. The Last Mile – Releasing Applications to Users

HealthVault provides a secure and rapidly expanding platform with rich feature set for application developers. Developer can target a wide set of users targeting multiple languages to enable rich functionality enabling quantified self.

As part of an applications life-cycle the standard steps are testing the application, releasing it to the user and then monitoring it for anomalies. This chapter will highlight best practices for releasing, maintaining and marketing HealthVault application to end users.

Conventions Used in This Book

[O'Reilly Standard]

Acknowledgements

Special thanks to Fred Trotter for providing the weight data used in Chapter 1 of the book. Thanks to Eric Friedman and team for helping with sleep data and updated HealthVault integration for Fitbit.

Thanks to Heidi Klinck for touching reviewing initial drafts and Chris Tremonte for content layout ideas. Special thanks to Rob May for contributing content and sample for HealthVault Java library.

Many thanks to the technical reviewers for providing valuable comments on early drafts of the book. Especially Sean Nolan, Rob May, Umesh Madan, Ali Emami and other members of HealthVault team at Microsoft.

Last but not least thanks to Sean Nolan from HealthVault and [Gary Wolf] from Quantified self for the forward.

Chapter 1

Getting Started with HealthVault

Health is critical to all of us. Healthcare and the infrastructure around it touches our lives and the lives of our loved ones. Many of us, in pursuit of long-term health, adopt goals ranging from controlling our weight to long-distance running. The Healthcare industry is in an early stage of realizing the power of the digital world, and the effectiveness of personal health tools in helping drive a change.

This chapter introduces HealthVault as a powerful tool for interacting with health data. It also provides a walkthrough of functionality available to the end user through HealthVault.

What is HealthVault?

HealthVault is a personal data platform that allows a user to record, collect, and share all health information in a central location. A key benefit of using HealthVault is its application programming interface (API), which applications and devices can use to provide value for the end-user. As depicted in [Fig 1.1], HealthVault enables an ecosystem of devices and applications, with use cases ranging from tracking diet and nutrition to connecting to hospital or pharmacy systems. HealthVault currently supports more than 300 applications and 80 devices. Some devices connect to HealthVault via the HealthVault Connection Center; a complimentary client application that enables devices to upload information directly to HealthVault from a Windows PC.

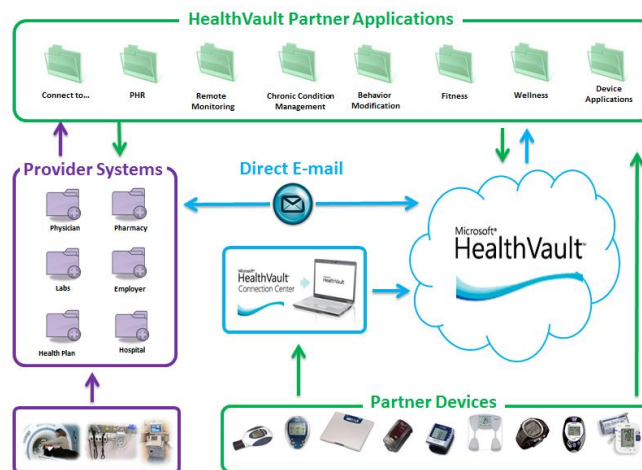


Fig 1.1 HealthVault Ecosystem with Devices and Applications

Getting started with HealthVault

On the HealthVault website, <http://www.healthvault.com>, a user can create an account using an existing Windows Live ID, Facebook, or OpenID account, or choose to create a new Windows Live ID. Fig 1.2 shows the sign-up screen for HealthVault.

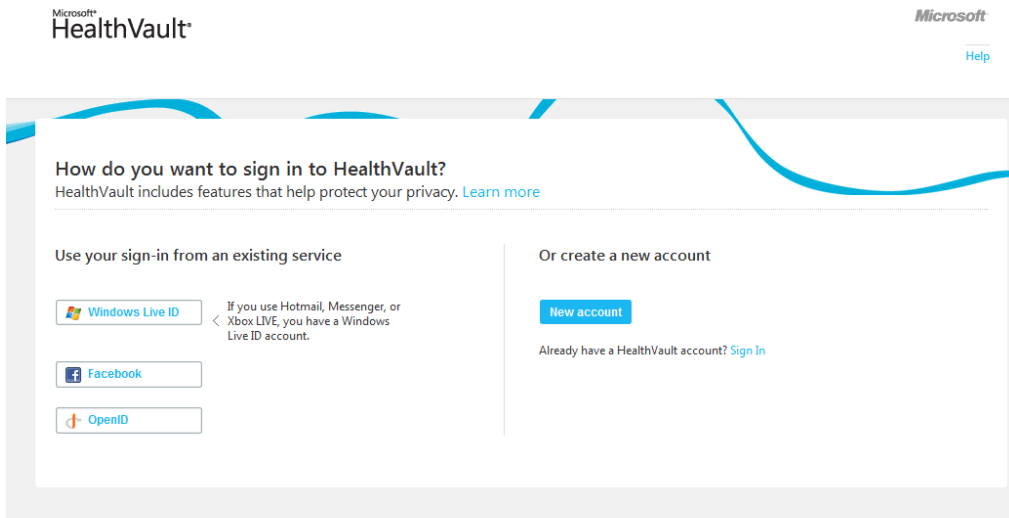


Fig 1.2. HealthVault sign-In page

HealthVault is currently publicly available in the United States and United Kingdom; you can create an account by entering basic demographic information and a proof of human computer interaction.

When a new user signs in to HealthVault, he is greeted with a new user wizard that enables him to select tasks and allows him to connect to various services [Fig 1.3].

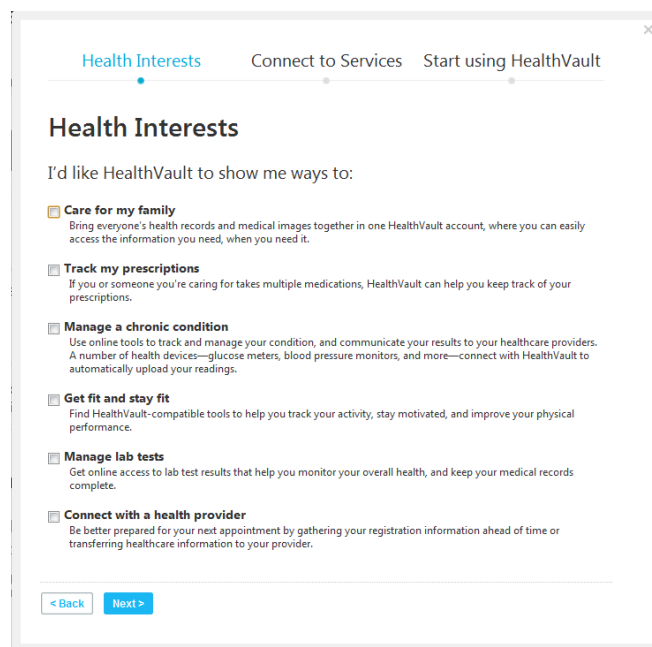


Fig 1.3. HealthVault new user wizard

Overview of HealthVault Features

In this section I'll go over a few of the most popular features in HealthVault, concentrating on ones that we'll use in this book to collect, manipulate and share information.

Health Information

The Health Information section of the health profile provides a view of all the information in the user's Health record. HealthVault supports more than 80 discrete kinds of data, from Advance Directive to Weight Goals. Through the user interface, you can edit and add health information. As Fig 1.4 indicates, you can add allergies, conditions, various measurements (blood glucose, blood pressure, peak flow, weight, height and lab test results), files (CCR, CCD, etc.), health history (family, immunizations, procedures) and emergency provider contact information.

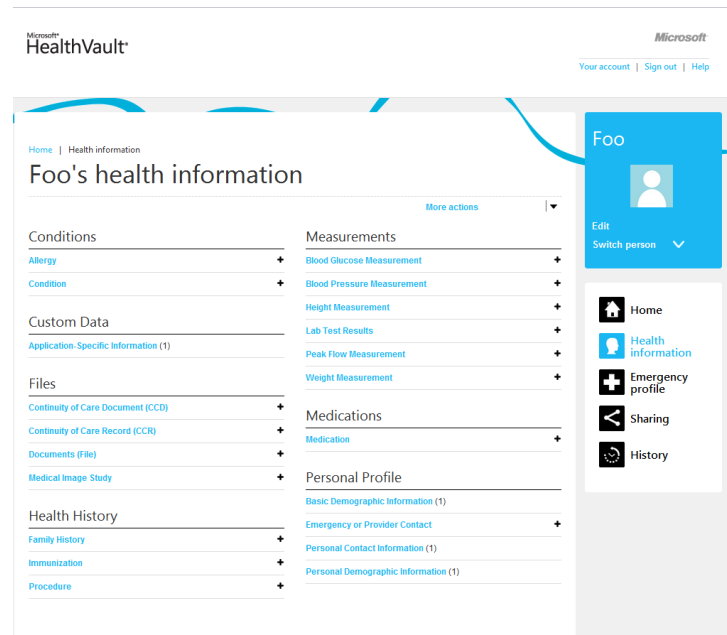


Fig 1.4. Health Information input support by HealthVault

You can also drill deeper and understand how the data entered your health profile, and see the audit trail to understand how the data evolved. [Fig 1.5] shows audit history of weight in HealthVault.

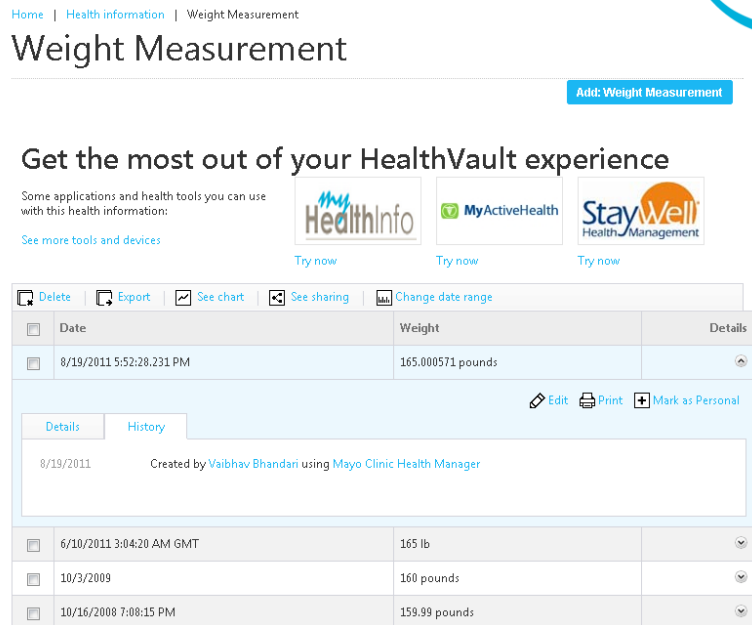


Fig 1.5. Viewing details of health data in HealthVault

Creating an emergency profile

Out of the box, HealthVault provides each user account with an emergency profile consisting of current allergies, conditions, medications, medical devices, and emergency contact information. A user can print, share, and update her emergency profile.

With an emergency profile, the user gets an emergency access code that could provide timely and up-to-date medical information to an emergency responder through HealthVault.com.

[Fig 1.6] shows the emergency access profile. Note that in addition to printing and sharing it, a user can also access a number of HealthVault tools that provide plethora of emergency services.

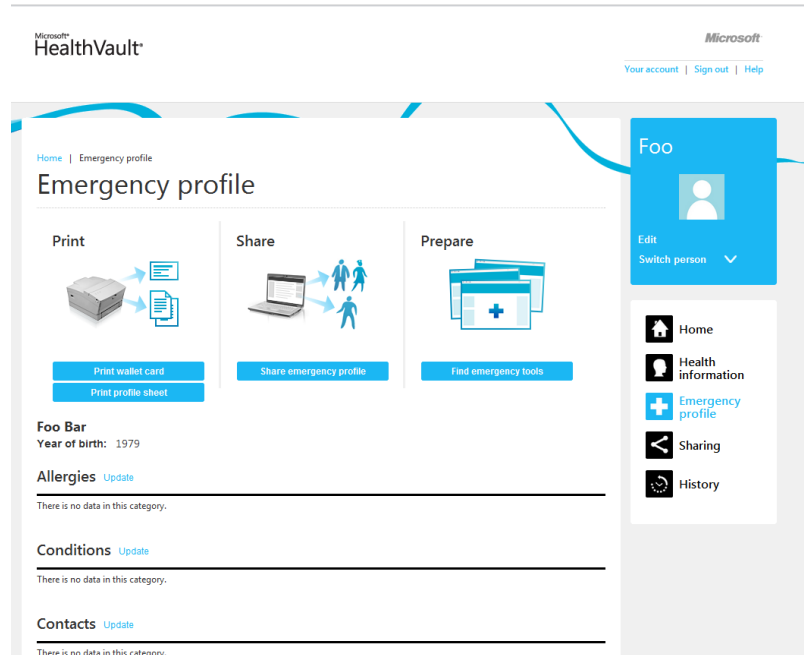


Fig 1.6. Emergency access profile

Sharing

Using the sharing section of Health profile the user can view with whom and how his information is being shared. A user can invite people to view granular information in their health profile. The data used in this book was shared by gracious contributors by using this sharing functionality for specific types of health data. As Fig 1.7 shows using the sharing pane, users can review and revoke access to online health tools.

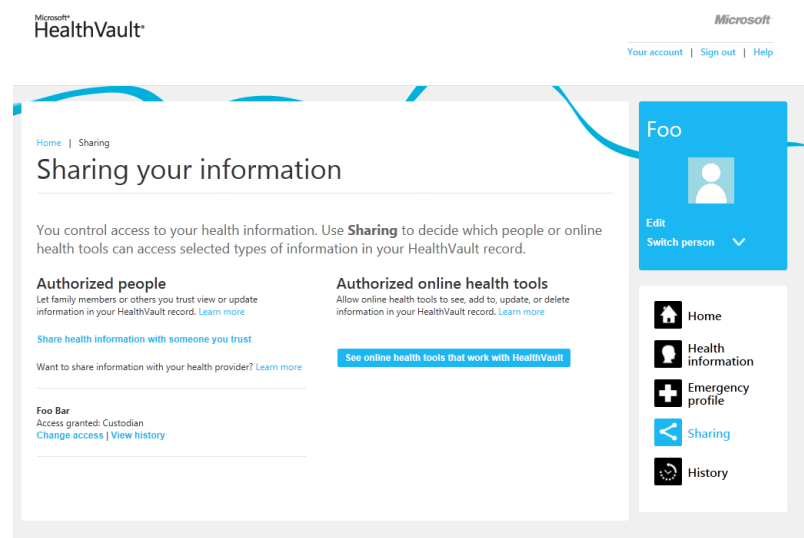


Fig 1.7. Sharing Health Information

History

Having a granularly shareable health profile enables a plethora of care co-ordination scenarios. However we do want to know how and when our sensitive health information is being accessed and updated. As Fig 1.8 shows using the “History” pane of the Health Profile, a user can view the ways their health information has been accessed. I frequently look at the “Changes made in last 30 days” and review who has accessed and updated my record.

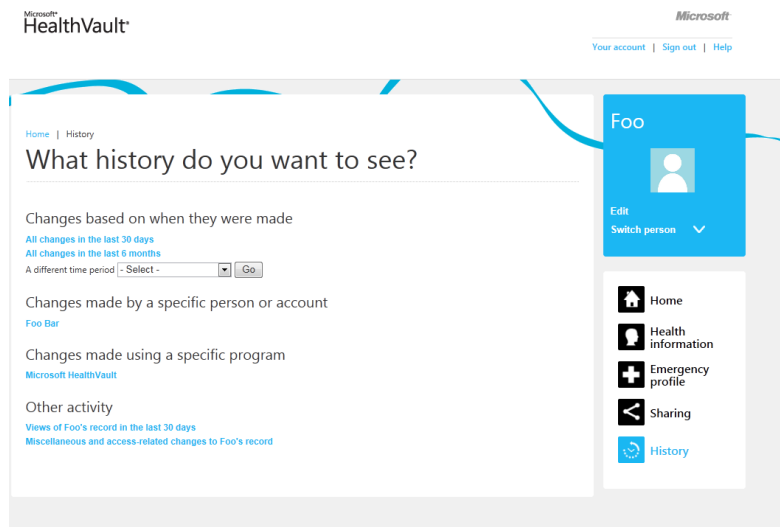


Fig 1.8. Reviewing History of Changes

Working with Health Data

Data is a powerful tool to understand behaviors and trigger appropriate measured change. Users can find out interesting trends by running calculations on their data stored in HealthVault, as I'll show throughout this book.

For instance, through the Health Information section, a user can chart his weight readings (Fig 1.9).

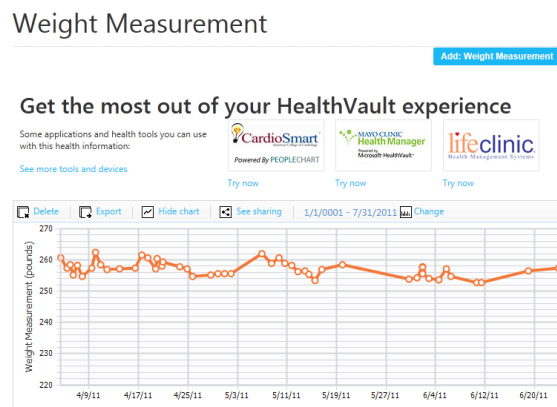
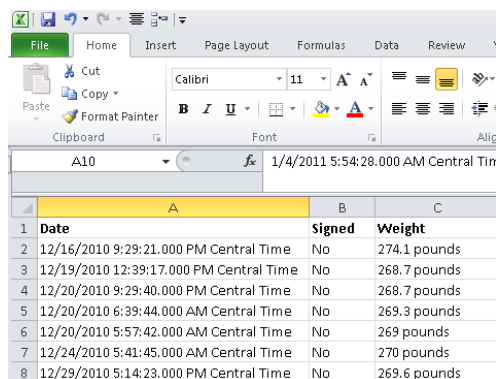


Fig 1.9. Tracking weight in HealthVault

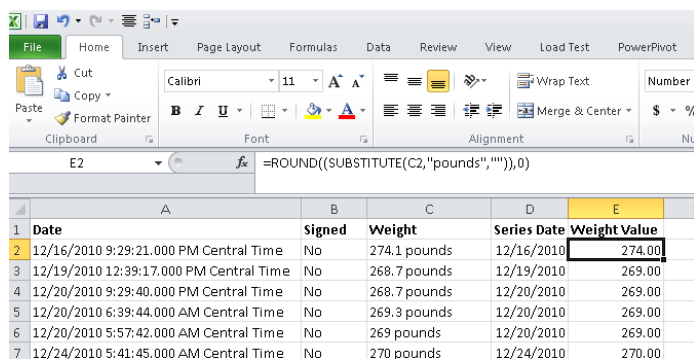
You will see that over last several readings, weight has been stable around 257 pounds. Nonetheless, I would like to take this a bit further and analyze these readings. To do this I click on the “export” button in the Health Information section. This gives me the readings in a comma separated values (CSV) format, which I can then open in Microsoft Excel or any spreadsheet program (Fig 1.10). If you don’t have weight data, I encourage you to download the sample spreadsheet with weight data included as part of this book’s examples, and follow along.



	A	B	C
1	Date	Signed	Weight
2	12/16/2010 9:29:21.000 PM Central Time	No	274.1 pounds
3	12/19/2010 12:39:17.000 PM Central Time	No	268.7 pounds
4	12/20/2010 9:29:40.000 PM Central Time	No	268.7 pounds
5	12/20/2010 6:39:44.000 AM Central Time	No	269.3 pounds
6	12/20/2010 5:57:42.000 AM Central Time	No	269 pounds
7	12/24/2010 5:41:45.000 AM Central Time	No	270 pounds
8	12/29/2010 5:14:23.000 PM Central Time	No	269.6 pounds

Fig 1.10. Weight readings in Microsoft Excel

Using Excel, I can clean the data so I can chart and analyze it further. I can add a series date attribute by just using the date from the first column (Fig 1.11).



	A	B	C	D	E
1	Date	Signed	Weight	Series Date	Weight Value
2	12/16/2010 9:29:21.000 PM Central Time	No	274.1 pounds	12/16/2010	274.00
3	12/19/2010 12:39:17.000 PM Central Time	No	268.7 pounds	12/19/2010	269.00
4	12/20/2010 9:29:40.000 PM Central Time	No	268.7 pounds	12/20/2010	269.00
5	12/20/2010 6:39:44.000 AM Central Time	No	269.3 pounds	12/20/2010	269.00
6	12/20/2010 5:57:42.000 AM Central Time	No	269 pounds	12/20/2010	269.00
7	12/24/2010 5:41:45.000 AM Central Time	No	270 pounds	12/24/2010	270.00

Fig 1.11. Using Excel to clean up the date

The formula `DATEVALUE(LEFT(A2, FIND(" "), A2)))` , converts the cell to a date value by picking the left side of the date format before the first space in column A2. The formula `ROUND(SUBSTITUTE(C2,"pounds"),0)`, removes the pound unit in column C and rounds the value to nearest integer.

Using Excel I can find the average weight over the last set of readings (Fig 1.12), and in fact plot my weight over months to uncover the monthly trend (Fig 1.12).

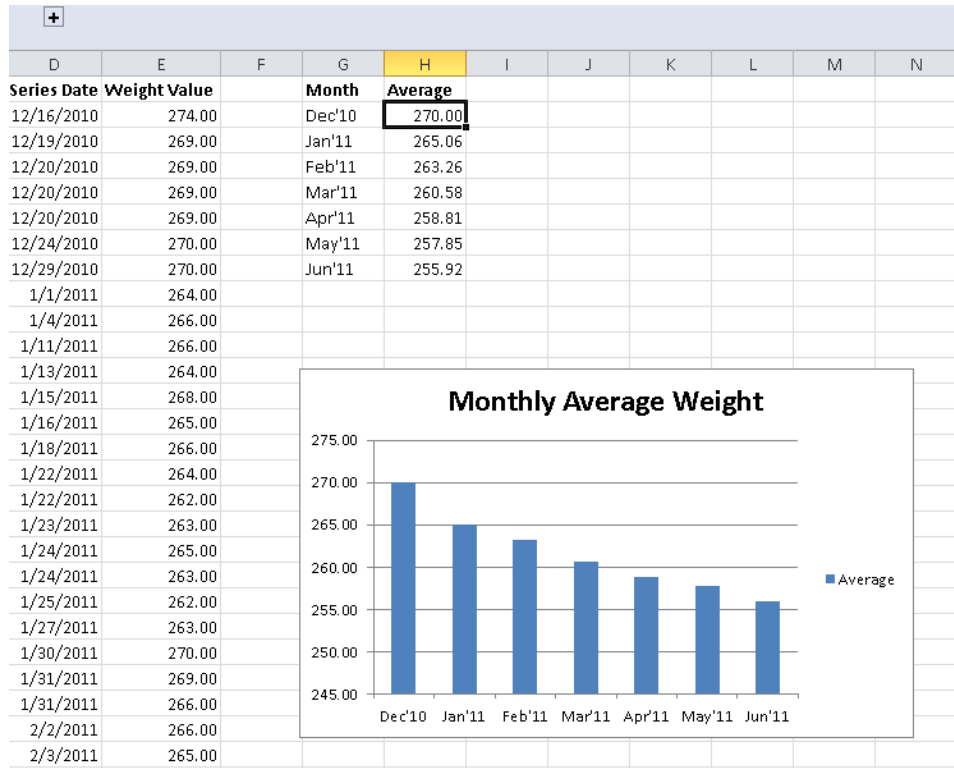


Fig 1.12. Monthly average weight as a bar chart

Managing weight is one scenario where you can use health tools to gain insights, the associated website with this book, enablingquantifiedself.com, has a repository of spreadsheets giving inspiration for additional care scenarios.

Using partner applications

So far, we have looked at the mechanisms provided within HealthVault to track, update and visualize health information. Outside applications, however, offer even more information. For instance, the Mayo Clinic Health Manager application, <https://healthmanager.mayoclinic.com>, can track your weight toward an intended goal (Fig 1.13).

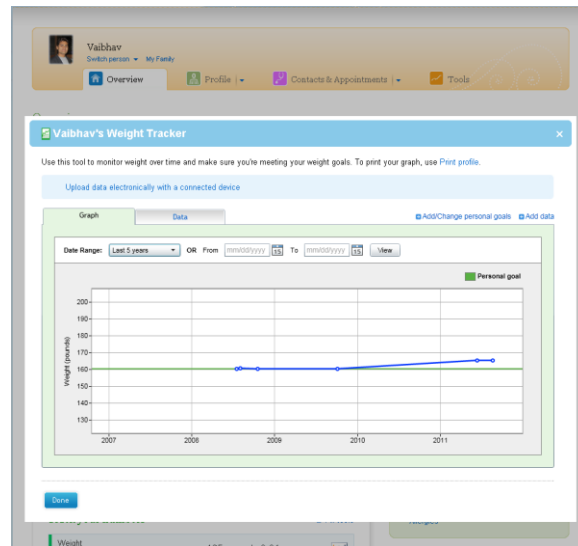


Fig 1.13. Tracking weight against a goal

The Mayo Clinic Health Manager is able to access all the weight information from a user's HealthVault account using the HealthVault API. If you're not a programmer you can benefit from many such applications that add value to HealthVault by allowing you to track and measure health data. If, however, you have modest programming skills in almost any modern language, this book will show you can create your own.

The HealthVault .NET Web SDK provides an abstraction on HealthVault APIs to simplify working with the platform. Listing 1.1 is a .Net program using the SDK to extract all the weights from a user's HealthVault record into a dictionary.

```

1 using System;
2 using System.Collections.Generic;
3
4 using System.Web;
5
6 using Microsoft.Health;
7 using Microsoft.Health.Web;
8 using Microsoft.Health.ItemTypes;
9 using Microsoft.Health;
10
11 public partial class HelloWorldPage : HealthServicePage
12 {
13     protected void Page_Load(object sender, EventArgs e)
14     {
15
16         HealthRecordSearcher searcher = PersonInfo.SelectedRecord.CreateSearcher();
17
18         HealthRecordFilter filter = new HealthRecordFilter(Weight.TypeId);
19         searcher.Filters.Add(filter);
20
21         HealthRecordItemCollection items = searcher.GetMatchingItems()[0];
22
23         Dictionary<string, string> weights = new Dictionary<string, string>();
24
25         foreach (Weight item in items)
26         {
27             weights[item.When.ToString()] = item.Value.ToString();
28         }
29
30         WeightView.DataSource = weights;
31         WeightView.DataBind();
32     }
33 }
34

```

Listing 1.1. Accessing HealthVault through the .NET Web SDK to read weight measurements

The steps in extracting data are; create a searcher, add a filter to restrict the output to the field or rows you want, and then run a search. The `searcher.GetMatchingItems()` in Line 21 of Listing 0.1 actually issues a HealthVault GetThings API request with a query configured to fetch all the Weight items from the users HealthVault record. We will learn more about the API & account management in chapter 3, and the data types in Chapter 4.

In the next chapter, we will dive deeper into the HealthVault device and application ecosystem.

"If you cannot measure it, you cannot improve it."
-- Lord Kelvin

Chapter 2

Quantifying yourself

Data is a powerful behavior change tool. The act of simply tracking changes one's perception of that activity. Summarizing the data over time provides a yard stick by which to measure. The act of tracking activity overtime uncovers patterns in behavior and provides definitive answers to self-experimentation questions. The structured data in HealthVault provides such an opportunity. Moreover the HealthVault ecosystem offers a variety of applications and devices to assist in this endeavor.

In this chapter we will explore how a consumer can use various devices to track critical health measures. We will also use common tools to explore the data stored by devices in to Microsoft HealthVault. We'll capture and view some data, then use a PowerShell plugin to extract selected data to a CSV format and manipulate the data in that format.

Fitbit is being used in this chapter just to illustrate the ways you can use data from all kinds of devices, so long as they provide a gateway to HealthVault. If you're not using Fitbit, I encourage you to download the sample Fitbit sleep data included as part of this book's examples, and follow along.

How Fitbit Tracks sleep

Fitbit is a pedometer on steroids that enables you to monitor a number of aspects of daily living. This chapter concentrates on sleep, Fitbit has been very popular with users trying to understand and change their sleep patterns. Fitbit provides an arm band (Figure 2-1) that tracks whether you're awake or asleep based on your activity level. Alternatively, the user can select an on/off mode to indicate whether she's asleep.

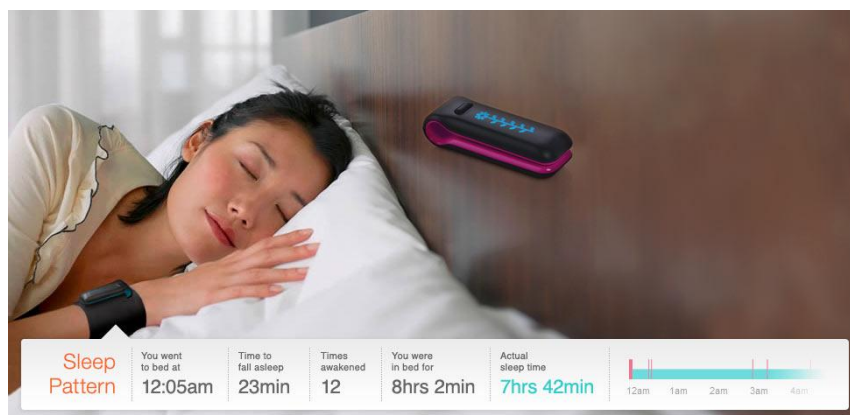


Fig 2-1. Fitbit being used to track sleep.

Fitbit also provides a base station that wirelessly uploads information from the device to the Fitbit web-service. Not having to worry about uploading information is a great value-add provided by this product.

Sending Data to HealthVault

Fitbit enables users to sync their data automatically with HealthVault. Once you have a Fitbit account, you can choose the Share stats page (Figure 2-2) which becomes available after clicking on the account settings.

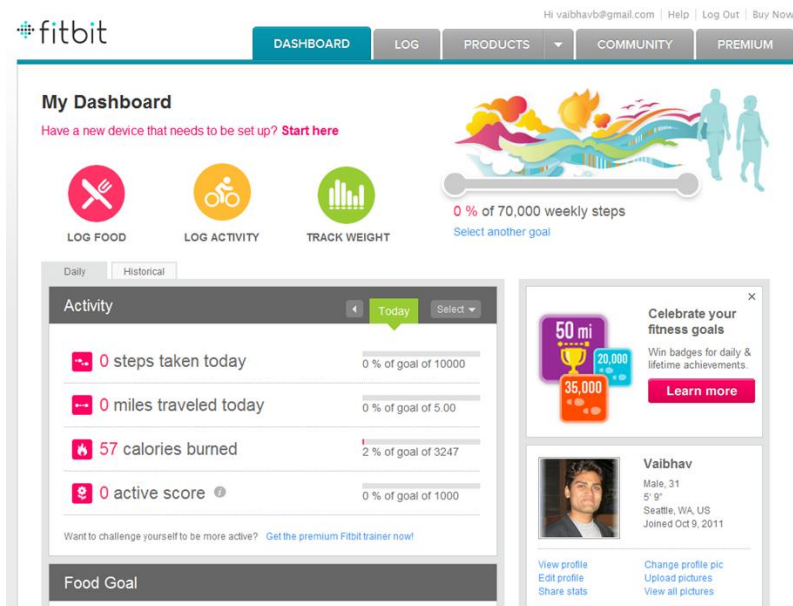
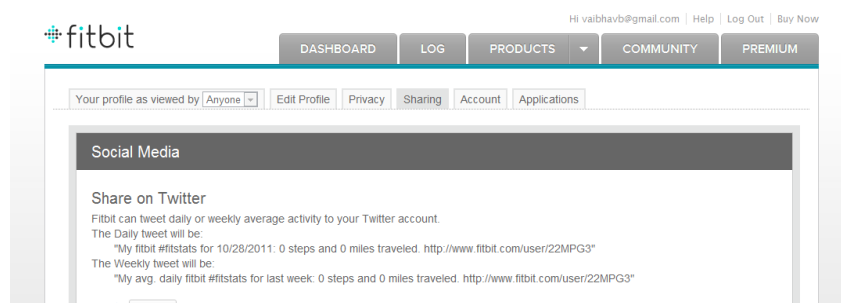


Fig 2-2. Fitbit Share stats feature

The Share Stats page, among other services, enables a link to HealthVault (Figure 2-3).



..

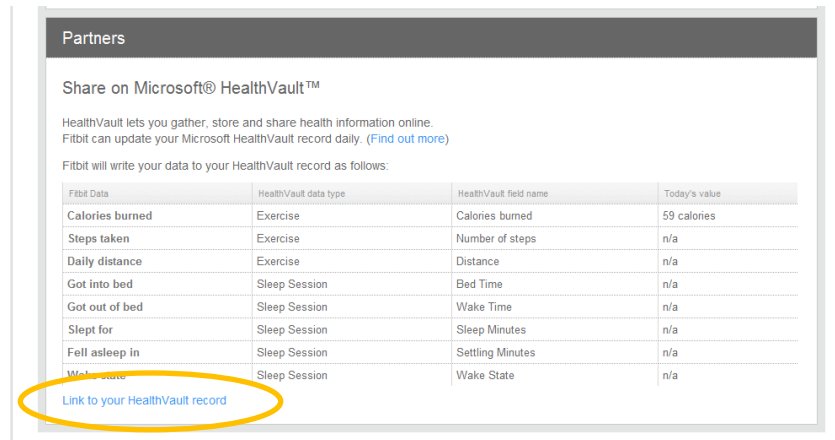


Fig 2-3. Connecting Fitbit with Microsoft HealthVault

Any application connecting to HealthVault has to get consent from the user for kinds of data it will be reading from or writing to Microsoft HealthVault. The user control is a two-step process. In the first step the user chooses the context of the record being authorized (Figure 2-4). As Figure 2-4 shows, in my case I have the option of using the application for my or my Mom's record. In the second step the user grants access to specific health data being shared with the application (Figure 2-5). As Figure 2-5 shows, the Fitbit application wants to access to my Exercise, Sleep Session and other health information. We will learn in more detail about the user authentication and authorization system in chapter 3.

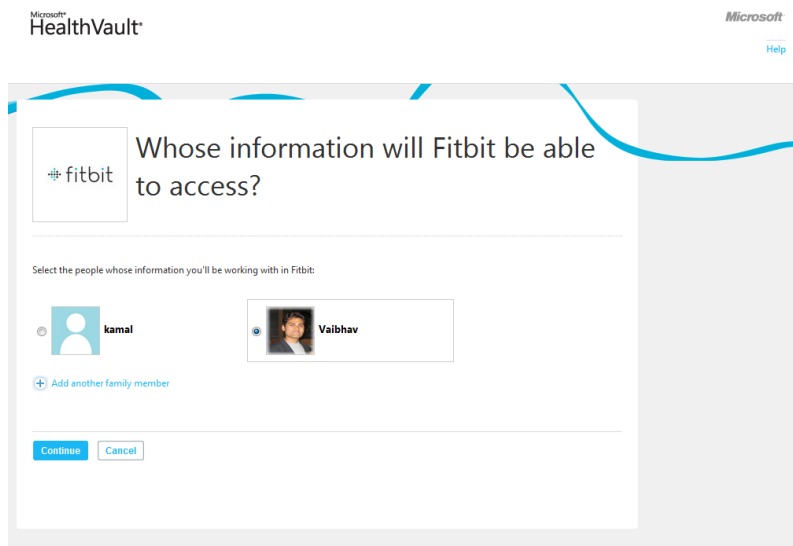


Fig 2-4. Choosing the context of a HealthVault record to work with an application

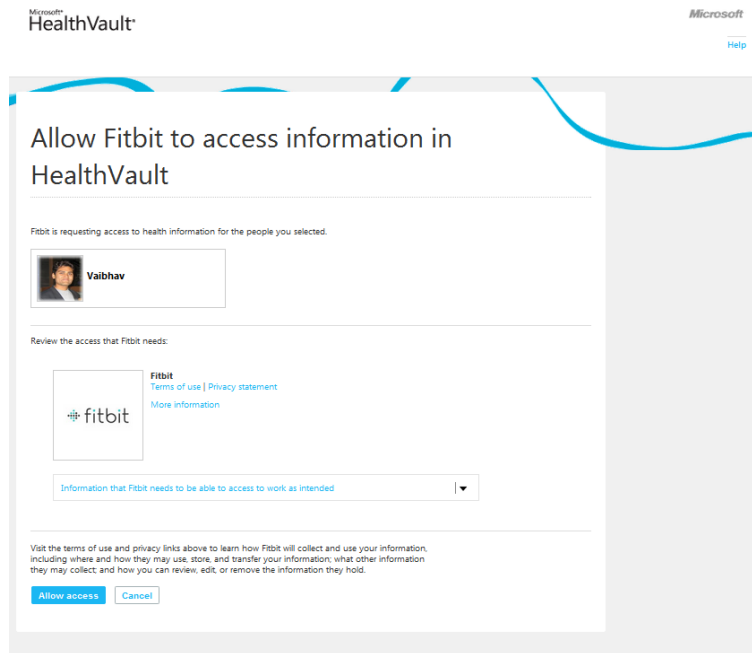


Fig 2-5. Authorizing an application to access a user's Health data.

Clicking on the “Information that Fitbit needs to be able to access to work as intended”, you will notice that Fitbit wants to access a user's Custom Data, Fitness, Measurements, and Personal Profile, as shown in Fig 2-6. In the line below the heading you will notice – Application Specific Information, Exercise, Sleep Session, Personal Contact Information and Personal Demographic Information, these are granular HealthVault data types. HealthVault has about 80+ granular data types which form the building blocks for various kinds of health information (Fitness, Measurement etc.). The data types are optimized to work with different devices and health systems. We will learn more about HealthVault data types and vocabularies in Chapter 4.

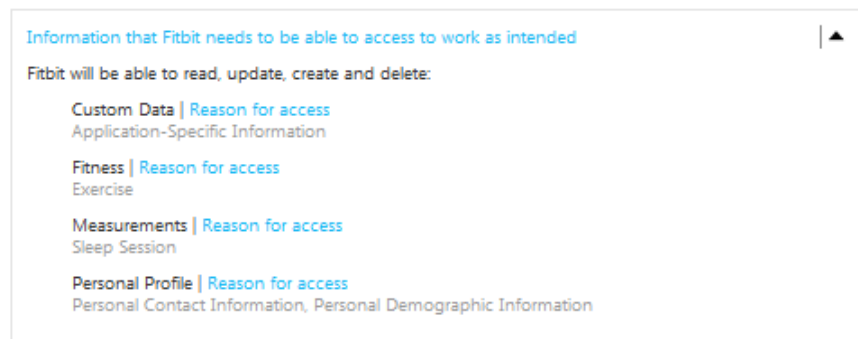


Fig 2-6. Granular details of the HealthVault data accessed by Fitbit

Understanding the Data Model

Fitbit collects pedometer and sleep data. When the device syncs its data to HealthVault's granular types it stores data as detailed in Table 2.1.

Fitbit Data	HealthVault Data type	HealthVault Field Name
Calories Burned	Exercise	Calories burned
Steps Taken	Exercise	Number of steps
Daily Distance	Exercise	Distance
Got in to bed	Sleep Session	Bed Time
Got out of bed	Sleep Session	Wake Time
Slept for	Sleep Session	Sleep Minutes
Fell asleep in	Sleep Session	Settling Minutes
Wake State	Sleep Session	Wake State

Table 2-1. Fitbit HealthVault data mapping

As a user we are interested in tracking all the information about sleep as collected by Fitbit. As you will note from Table 2-1, we should look at the HealthVault Sleep Session data type for tracking sleep, and HealthVault Exercise data type for tracking Fitbit pedometer data.

Exploring the HealthVault data

You can look at the data stored from Fitbit in the HealthVault user interface (sometimes referred as HealthVault Shell), as shown in Figure 2-7.

[Home](#) | [Health information](#) | [Sleep Session](#)


Sleep Session

Get the most out of your HealthVault experience

Some applications and health tools you can use with this health information:

[See more tools and devices](#)

[Try now](#)














	Delete		Export		See sharing		Change date range
<input type="checkbox"/>	Date	Bed Time	Wake Time	Sleep Time	Settling Time	Details	
<input type="checkbox"/>	6/29/2011 12:00:00 AM	10:51:00 PM	6:54:00 AM	463 min.	7 min.		
<input type="checkbox"/>	6/28/2011 12:00:00 AM	11:24:00 PM	4:28:00 AM	251 min.	5 min.		
<input type="checkbox"/>	6/27/2011 12:00:00 AM	11:13:00 PM	7:25:00 AM	437 min.	13 min.		
<input type="checkbox"/>	6/26/2011 12:00:00 AM	11:15:00 PM	8:00:00 AM	506 min.	7 min.		
<input type="checkbox"/>	6/25/2011 12:00:00 AM	11:27:00 PM	6:50:00 AM	428 min.	2 min.		
<input type="checkbox"/>	6/24/2011 12:00:00 AM	11:35:00 PM	7:00:00 AM	425 min.	7 min.		
<input type="checkbox"/>	6/23/2011 12:00:00 AM	11:30:00 PM	6:30:00 AM	404 min.	5 min.		

Fig 2-7. Sleep Session in HealthVault

Viewing information through the HealthVault user interface is convenient but a user cannot retrieve the entire information by exporting the data. As a power user and a quantifier I would like the data to be available to me to do some data-noodling. For that purpose, I would get this data in command-line format using the HealthVault PowerShell plugin (HvPosh). You can find the details of installing and extending this plug-in at <https://github.com/vaibhavb/HvPosh>. PowerShell can export data to a standard CSV format that can be consumed by a variety of other tools, simple or advanced, that let you do calculations and generate charts.

Once you have installed the PowerShell, load HealthVault's plugin in to Windows PowerShell using

Powershell> import-module HvPosh

Then grant access to HealthVault Powershell interface using the following command line. Note this command will walk you through the same record picking interface and authentication and authorization interface as we used earlier for the Fitbit application.

Powershell> Grant-HVPermission

Once you have access to HealthVault within PowerShell, you can start using the utility from the command line and extract information pertinent to Sleep Session:

Powershell>Get-Things -item Sleep-Session | format-table

The results for my sample data are shown in Figure 2-8.

When	Bedtime	WakeTime	SleepMinute s	SettlingMin utes	Awakenings	Medications	WakeState
5/29/201...	10:51:00 PM	6:54:00 AM	463	7	<Microso...		WideAwake
5/28/201...	11:24:00 PM	4:28:00 AM	251	5	<Microso...		WideAwake
5/27/201...	11:13:00 PM	7:25:00 AM	437	13	<Microso...		WideAwake
5/26/201...	11:15:00 PM	8:00:00 AM	506	7	<Microso...		WideAwake
5/25/201...	11:27:00 PM	6:50:00 AM	428	2	<Microso...		WideAwake
5/24/201...	11:35:00 PM	7:00:00 AM	425	7	<Microso...		WideAwake
5/23/201...	11:30:00 PM	6:30:00 AM	404	5	<Microso...		WideAwake
5/22/201...	10:52:00 PM	6:34:00 AM	440	4	<Microso...		WideAwake
5/21/201...	12:28:00 AM	6:21:00 AM	320	18	<Microso...		WideAwake
5/20/201...	11:17:00 PM	7:47:00 AM	481	3	<Microso...		WideAwake
12/19/20...	9:00:00 PM	7:00:00 AM	600	5	<Microso...		WideAwake

Figure 2-8. Structured data from Sleep Session as retrieved by PowerShell

--

If you don't have a Fitbit and want to follow along you can import the above data from the file Sleep-Data.xml available in the code associated with Chapter 3. Following is the command to import this data.

Powershell>Import-HvDataXml -File Sleep-Data.xml

--

You can understand the data further by using ways to look at this data by exploring the individual properties. Fig 2-9 shows how you can select particular properties of a HealthVault data type using PowerShell Select-object command.

Powershell> get-things sleep | select-object When, Bedtime, WakeTime | format-table

```
PS C:\> get-things sleep | select-object When, BedTime, WakeTime | format-table
```

When	Bedtime	WakeTime
6/29/2011 12:00:00 AM	10:51:00 PM	6:54:00 AM
6/28/2011 12:00:00 AM	11:24:00 PM	4:28:00 AM
6/27/2011 12:00:00 AM	11:13:00 PM	7:25:00 AM
6/26/2011 12:00:00 AM	11:15:00 PM	8:00:00 AM
6/25/2011 12:00:00 AM	11:27:00 PM	6:50:00 AM
6/24/2011 12:00:00 AM	11:35:00 PM	7:00:00 AM
6/23/2011 12:00:00 AM	11:30:00 PM	6:30:00 AM
6/22/2011 12:00:00 AM	10:52:00 PM	6:34:00 AM
6/21/2011 12:00:00 AM	12:28:00 AM	6:21:00 AM
6/20/2011 12:00:00 AM	11:17:00 PM	7:47:00 AM
12/19/2006 7:00:00 AM	9:00:00 PM	7:00:00 AM

Figure 2-9. Three columns of structured data from Sleep Session as retrieved by PowerShell

In fact, I want to be able to explore detailed of awakenings. This is particularly relevant in knowing patterns of sleep disturbances.

Powershell> get-things sleep | select-object -expandproperty Awakenings | format-table

```
PS C:\> get-things sleep | select-object -expandproperty Awakenings | format-table
```

When
3:00:00 AM
3:00:00 AM
3:00:00 AM
3:00:00 AM
3:00:00 AM
3:00:00 AM
3:00:00 AM
3:00:00 AM
3:00:00 AM
3:00:00 AM
3:00:00 AM
9:00:00 PM

Fig 2-10. Understanding the pattern of Awakenings

Now you can enable some data crunching by exporting this data to a CSV file and switching the data analysis to Microsoft Excel or Google Spread sheets.

Powershell> get-things sleep | select-object When, SleepMinutes, SettlingMinutes | export-csv SleepData.csv

This creates the file SleepData.csv with the selected data.

Analyzing the HealthVault Data

Once you have all the data in CSV file, you can open it in Excel (Figure 2-11) and analyze sleep patterns. You will notice that the spreadsheet has data for each sleep session specifying when that session occurred, the total sleep time in minutes and the time it took to get to bed termed as *settlingminutes*. I want to understand this data better so I create a sleep pattern X-Y scatter plot for this information [Fig 2-14].

	A	B	C	D	E
1	#TYPE Selected.Microsoft.Health.ItemTypes.SleepJournalAM				
2	When	SleepMinutes	SettlingMinutes		
3	6/29/2011 0:00	463	7		
4	6/28/2011 0:00	251	5		
5	6/27/2011 0:00	437	13		
6	6/26/2011 0:00	506	7		
7	6/25/2011 0:00	428	2		
8	6/24/2011 0:00	425	7		
9	6/23/2011 0:00	404	5		
10	6/22/2011 0:00	440	4		
11	6/21/2011 0:00	320	18		
12	6/20/2011 0:00	481	3		
13	12/19/2006 7:00	600	5		
14					

Fig 2-11. Sleep session data in an excel spread sheet

As Fig 2-12, reveals that for the duration of this week the median sleep has been around 400 minutes (i.e. around 6.5 hours), and as the data clearly shows the days when it took the longest to get to sleep the duration of sleep has been lower. So a good indicator of not been able to get to sleep in 10 minutes is a lower and poor quality of sleep.

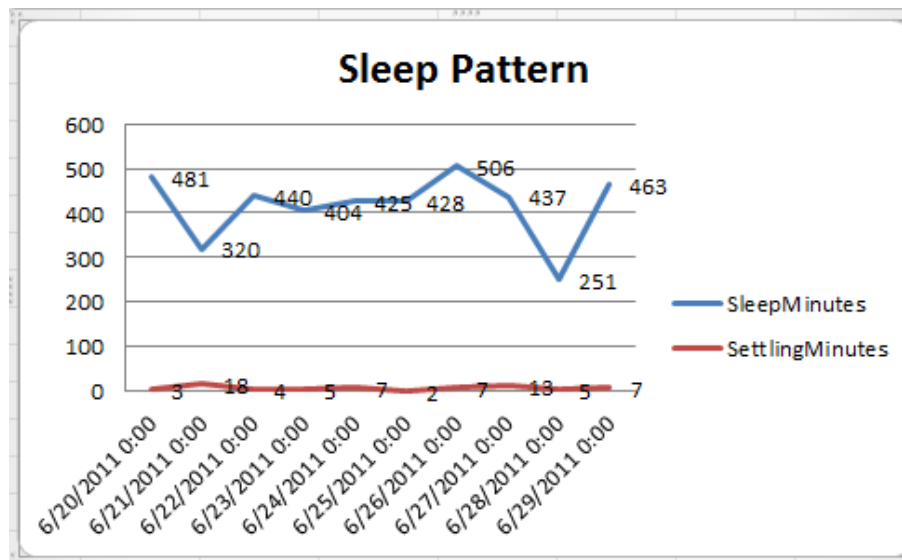


Fig 2-12. Sleep Pattern analysis

In fact for this duration I also want to understand the patterns around awakenings. So using powershell we generate another CSV file which focuses on awakenings.

Powershell> get-things sleep | select-object effectiveDate -expandproperty awakenings | Export-Csv d:\sleep-date-aw.csv

We can open the file in Excel and visualize how the awakenings are triggered. It's very obvious that the most awakenings are for duration of 10 minutes around 3am. I know that is because the workshop in neighborhood is doing an early project and the noise around that time wakes me up.

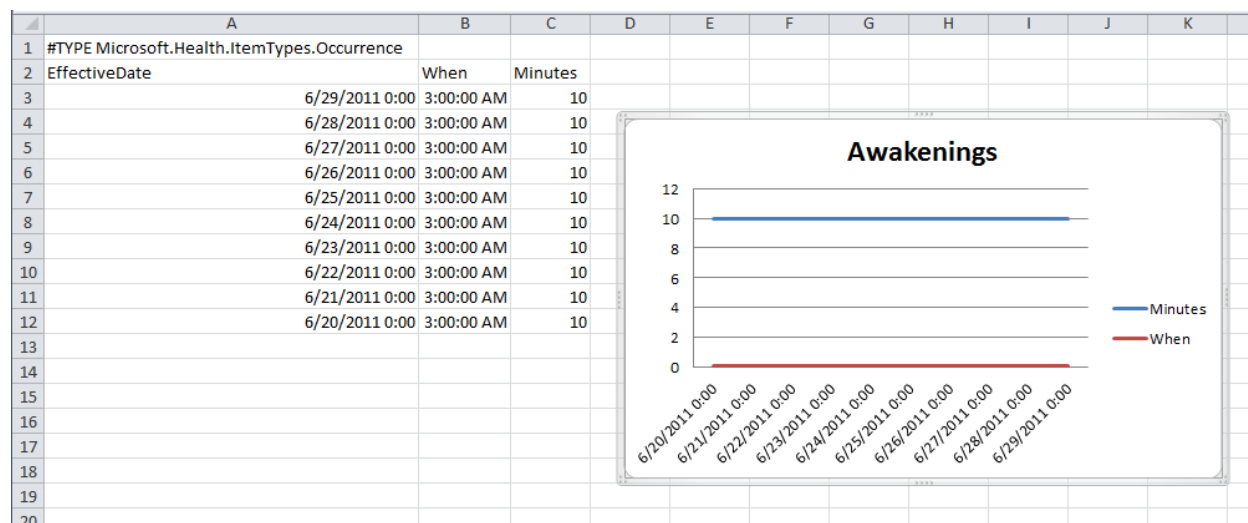


Fig 2-13. Awakenings Pattern Analysis

We can even take it a step further and correlate the sleep information with other types of data. With Fitbit we get the activity data and one can try and associated the steps information on the existing sleep data.

Using Powershell HealthVault plugin we can grab the appropriate fitness data from HealthVault.

Powershell> Get-Things exercise | where-object {\$_.Activity.Name -eq "walk"} | select-object -expandproperty Activity | export-csv pedometer.csv

Adding the pedometer data to what we obtained in Fig 2-12, gives us a way to correlate physical activity to sleep, as shown in Fig 2-14.

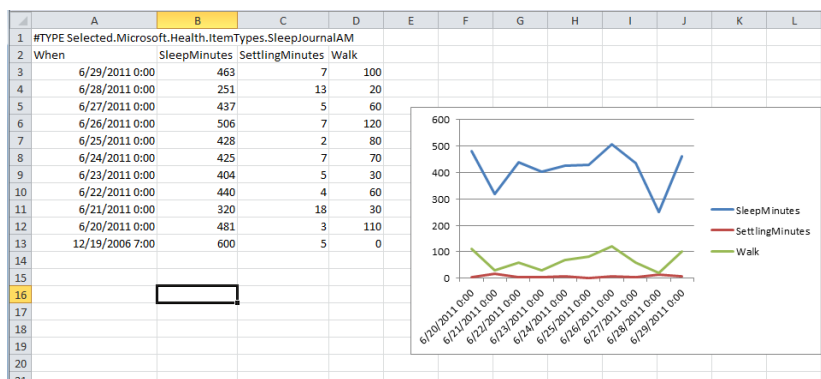


Fig 2.14 Correlating Sleep with Walking Exercise

Note that we scaled the steps information from pedometer to play nice on the graph by dividing it by 100. Using the above information one can say that possibly the days on which sleep was lower was possibly owing to lack of exercise on the other had the settling time was high on those days as well. So may be as a behavioral change one can mandate to get at least 5000 steps of walking to ascertain a good sleep.

This might change in long run, but that is the joy of learning from data and motivating a behavior shift! The associated website with this book, enablingquantifiedself.com, has a repository of spreadsheets giving inspiration for additional self-experimentation scenarios.

In upcoming chapters we will learn how we can automate some of the work we have done in this chapter with HealthVault application programming interface.

“Things would have changed if I had timely access to electronic medical records”

Regina Holliday

Chapter 3

Interfacing with HealthVault

As a platform HealthVault provides an innovative access management and programming interfaces for applications and devices to access a user’s health information.

In the previous chapter we discovered how to fetch and manipulate data stored in HealthVault. This chapter takes a closer look at the application programming interface offered by HealthVault to enable this interaction in a programmatic fashion. We will discuss various ways in which an application or device can interface with the HealthVault platform. The code samples will use .NET interfaces because they fit well with HealthVault, but the same interfaces are available in Java, PHP & other languages. The chapter will introduce the elements of programming that give the programmer access to data in HealthVault. Towards the end, we will discuss various architectural options available for interface an application or device with HealthVault. We’ll start by discussing accounts, because the first task is to get access to your own account.

Accounts and Records

HealthVault provides innovative access management to let a family health manager access and manage the records of various family members. Mom, serving as the family health manager, can create records for her husband and children. In Fig 3-1, Jane has created account for her husband, Tom and two kids, Chris and Sarah. She has full access to all information in her families HealthVault records.

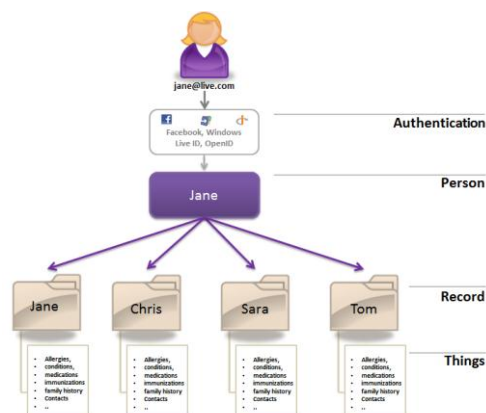


Fig 3-1. Multiple records under one HealthVault account

Additionally, HealthVault enables the same records to be accessed through multiple accounts. Full access can be thought of as custodial access to the record. In Fig 3-2, Jane has full access to health information of her family. Tom has also signed up to share the responsibility of managing health information of their kids Chris and Sarah, and also has full access to their health information.

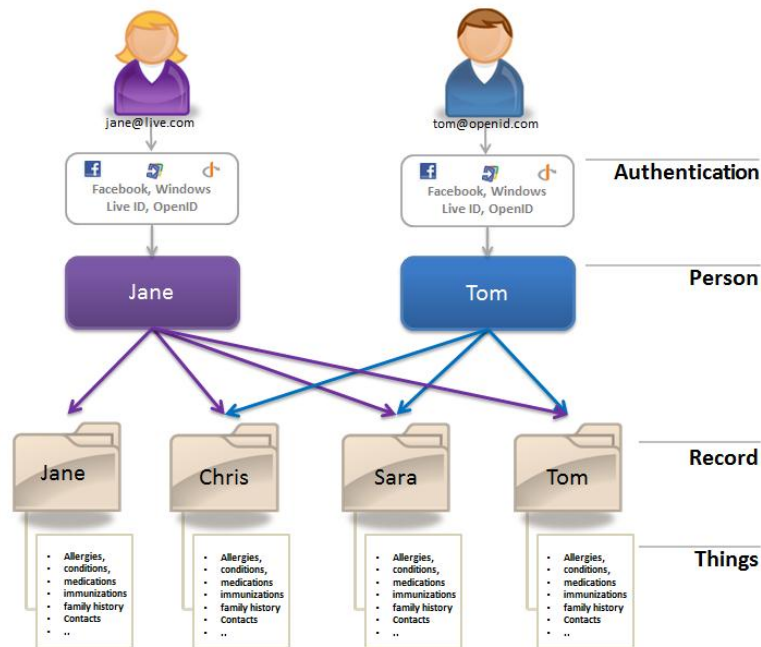


Fig 3-2 Multiple accounts pointing to same HealthVault records

Account Information

An application gets access to HealthVault account information through an API called Get-PersonInfo. This API returns a structure called PersonInfo, which in turn consists of the records associated with a HealthVault account.

```

Select Windows PowerShell
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> import-module hvposh
PS C:\Windows\system32> Get-PersonInfo

PersonId       : 4c15c687-110d-4585-b091-d3fceb93de
Name           : Vaibhav Bhandari
ApplicationSettings :
SelectedRecord  : Vaibhav Bhandari
AuthorizedRecords : <14606ac9e-138a-4f8a-82cf-c771ff6f9249, Vaibhav Bhandari>
PreferredCulture : en-US
PreferredUICulture : en-US
Connection      :
ApplicationConnection : Microsoft.Health.HealthClientAuthorizedConnection
  
```

Fig 3-3 PersonInfo Data Structure

Using the HealthVault Powershell plug-in, you can try out the Get-PersonInfo API using the command *get-personinfo*. The structure returned consists of a unique PersonId for the account and a set of records identifier authorized to be used with the application for this particular account. Any pertinent

information for the application is stored in ApplicationSettings. The user's preferred language and display settings are stored in the PreferredCulture and PreferredUICulture fields respectively.

An application can decide to work with only one record at a time termed as *single record application* (SRA), or provide an interface to work with multiple records associated with the signed-in user termed as *multiple record application* (MRA). Section 2.4 describes how to enable each of these record management capabilities in detail.

HealthVault Application Programming Interface

A HealthVault application interacts with two distinct resources:

<https://platform.healthvault.com/platform/wildcat.ashx>: the HealthVault Platform–

<https://account.healthvault.com>: the HealthVault Shell.

The HealthVault Platform provides XML over HTTP requests to manipulate data hosted by the service, and the HealthVault Shell provides account management, user authentication and other services.

HealthVault provides a development environment for partners to develop their applications. The development environment is hosted at <https://platform.healthvault-ppe.com/latform>, for HealthVault platform and <https://account.healthvault-ppe.com> for HealthVault Shell.

Each HealthVault application gets a unique identifier called AppID. Developers can get a free application identifier using the HealthVault Configuration Center (<https://config.healthvault-ppe.com>).

<aside> For our example application we are using an application ID that was already created for a HelloWorld sample application. In Chapter 4, we will create our own application. </aside>

HealthVault Shell Interface

The HealthVault Shell provides its functionality primarily by redirecting the end-user's browser. HealthVault Shell presents a secure user interface dialog in the browser. These dialogs help with user authentication, authorization, record selection, and managing the user's experience around health data.

An application communicates its intention to HealthVault Shell using a URI construct like <https://account.healthvault.com/redirect.aspx?target={ShellTarget}&targetqs={ShellTargetParams}>

The **ShellTarget** parameter specifies the intent of the application, which could range from prompting the user to authorize the application to letting the user view their health items. Table 3-1 summarizes some of these targets; a detailed list of the Shell Targets is available on HealthVault MSDN at <http://msdn.microsoft.com/en-us/library/ff803620.aspx>.

HealthVault Shell Target	Purpose
AUTH	Prompts the user to authorize himself and select record(s) to be used with an application
APPREDIRECT	Redirects a user to another HealthVault application.
CREATEACCOUNT	Allows an application to create a new HealthVault account and redirects

	to the application after account authorization
CREATEAPPLICATION	Enables a client or desktop application to create an instance of its application on the device.
RECONCILE	Enables an application to redirect to the HealthVault Shell for reconciling a CCR/CCD with a user record.
VIEWITEMS	Allows an application to redirect a user to view or create health record items using HealthVault Shell.

Table 3-1 HealthVault Shell redirect Interface (partial list)

In certain circumstances, the HealthVault Shell needs to communicate with the application. For example, if a user wants to know the privacy statement of the application or if the user decides to not authorize permission for the application to access their health items, the HealthVault shell would then need to communicate with the application. HealthVault requires applications to register a “Redirect URL” for functionality they provide. The Redirect URL should be a secure (https) URL that can respond to request of this nature:

`https://{application redirect url}?target={ApplicationTarget}&targetqs={ApplicationTargetParameters}`

The **ApplicationTarget** specifies the desired action to get serviced; it could range from the user asking for a Privacy statement to the user rejecting the application’s authorization request. Table 3-2 summarizes some of these targets; a detailed list of the Application Targets is available on a HealthVault MSDN [page](#).

HealthVault Application Target	Purpose
APPAUTHSUCCESS	Notifies the application that the user successfully logged in and/or granted authorization to the application.
SIGNOUT	Notifies the application that the user logged out of their HealthVault session. The application can then do cleanup and show a sign-out page.
SELECTEDRECORDCHANGED	Notifies the application that the user successfully change the selected record. Section 2.2 shows example of handling this.
PRIVACY	Notifies the application that the user wants to view its privacy statement.
SERVICEAGREEMENT	Notifies the application that the user wants to view its terms of use or service agreement.

Table 3-2 HealthVault application targets (partial list)

HealthVault Platform APIs

The HealthVault Platform provides a number of APIs to enable access to application and user data; these APIs are well documented at <http://developer.healthvault.com/pages/methods/methods.aspx>. The following discussion will focus on kinds of functionality provided by these APIs. Table 3-3 summarizes the APIs available from the HealthVault platform

HealthVault API Category	API Names	Purpose
Authentication	CreateAuthenticatedSessionToken, RemoveApplicationRecordAuthorization,	Authenticate an application and a user.

	NewApplicationCreationInfo, NewSignupCode, GetPersonInfo, GetAuthorizedRecords	
Reading Health Items	GetThings	A rich interface to retrieve Health items along with an associated digital signature or streamed BLOBs.
Adding & Updating Health Items	PutThings, OverwriteThings BeginPutBlob	Enable an application to add or update Health item data.
Delete Health Items	RemoveThings	Enables an application to delete data.
Patient Connect	AssociatePackageId, BeginPutConnectPackageBlob, CreateConnectPackage, CreateConnectRequest, GetAuthorizedConnectRequests, DeletePendingConnectPackage, DeletePendingConnectRequest	Enable clinical applications to create a temporary drop-off or permanent connection for consumers without having a web interface.
Asynchronous Processing	GetAuthorizedPeople, GetUpdatedRecordsForApplication, GetEventSubscriptions, UpdateEventSubscription, SubscribeToEvent, UnsubscribeToEvent	Enable application to work asynchronously with HealthVault and create a publish/subscribe model.
Messaging	SendInsecureMessage, SendInsecureMessageFromApplication	Enable applications to send messages to consumers using these APIs.
Terminology	GetVocabulary, SearchVocabulary	Enable applications to retrieve or search terminologies hosted by HealthVault.
Application Management	SetApplicationSettings GetApplicationSettings AddApplication, UpdateApplication	Enable an application to store a record specific setting, and manage derivative applications.
Service Discovery	GetServiceDefinition, GetThingType	Help with service discovery.
OpenQuery	SaveOpenQuery, GetOpenQueryInfo, DeleteOpenQuery	These are hardly used, but they give the ability to run pre-canned queries for a health record.

Table 3-3 HealthVault API Summary

Authentication & Authorization APIs

Applications authenticate themselves to the HealthVault platform using the CreateAuthenticatedSessionToken API. Users are authenticated through HealthVault.com and applications can get tokens for authorization using the HealthVault Shell redirect interface at <http://msdn.microsoft.com/en-us/library/ff803620.aspx>.

<aside>CreateAuthenticatedSessionToken or CAST is the most commonly used HealthVault API. The API provides authentication token for clients as well as web applications. Most HealthVault wrappers provide an API for this purpose.</aside>

Individual methods are available for an application to fetch record and person authorization details. Most notably, NewApplicationCreationInfo is used by mobile clients to receive security keys from the HealthVault platform.

Reading Health Items

The core function that lets an application read items from a user's HealthVault record is GetThings. We will discuss this API in detail in section [link], but to summarize, this API provides the ability to query HealthVault, fetch a health item with granular details, and fetch large BLOB items such as images.

<aside>After CreateAuthenticatedSessionToken, GetThings is the most commonly used HealthVault API.</aside>

Creating and Updating Health Items

The counterpart of GetThings is PutThings. It is used by most applications to update and create health items. We will discuss this function in detail in section 2.3.1.

OverwriteThings allows applications to force overwrites on existing health items. This API is not generally used.

HealthVault, unlike most personal health platforms, provides a mechanism to store large files such as medical images. Application can upload large chunks of information by using the BeginPutBlob API. It is fairly tricky to use, but there is good documentation on how to use it via raw XML interfaces at <http://msdn.microsoft.com/en-us/library/ff803584.aspx>, as well as use the HealthVault .NET SDK at <http://msdn.microsoft.com/en-us/library/ff803576.aspx>.

Delete Health Items

DeleteThings is the one of the simplest HealthVault functions, allowing applications to delete individual health items from a user's record. HealthVault keeps an audit trail of all operations, including the delete operation.

<aside> Only users can view the audit trail for Health items by using HealthVault Shell's history functionality. Applications do not have access to the audit trail of a health item. </aside>

Patient Connect

As discussed later in section [link], several clinical applications use HealthVault to send information to consumers either a single time or continually through back-end systems. CreateConnectPackage allows applications to create a one-time package for the user to receive in their HealthVault account, and DeleteConnectPackage allows applications to perform cleanup as necessary. On the other hand, CreateConnectRequest allows application to establish a continual link with a patient's HealthVault record. Application can get the details needed to make the link by using GetAuthorizedConnectRequests.

<aside>HealthVault will delete the validated connect requests after a period of time. It is advised that applications calls GetAuthorizedConnectRequests daily or weekly to ensure that all validated connect requests are retrieved. </aside>

Asynchronous Processing

HealthVault provides several mechanisms for an application to perform asynchronous processing. GetAuthorizedPeople gets information about the people that are authorized to use the application. This function paginates results using a PersonID-cursor and provides a way to query authorizations created after a given point in time. Applications have found this function useful to send email updates and reminders to their subscribers. Similarly, GetUpdatedRecordsForApplication retrieves a list of records for an application with things that have been updated since a specified date.

HealthVault provides a powerful publish/subscribe mechanism. Applications can subscribe to events around create, read, update, and delete operations on HealthVault thing-types. These events are registered with the platform using the SubscribeToEvent method. In addition to defining the subscribe event, the application registers a secure URI to which the HealthVault platform publishes events. The HealthVault Eventing mechanism is documented in detail with appropriate examples at | <http://msdn.microsoft.com/en-us/library/gg681193.aspx>.

The InstantPHR, <http://www.getrealconsulting.com/instantphr/>, application from GetReal Consulting is a good example of an application that uses the HealthVault asynchronous processing and eventing in particular to notify users of changes in their Health records.

Messaging

Applications using HealthVault can send e-mail messages to HealthVault users. SendInsecureMessageFromApplication allows the application to choose the sender address and specify its domain.

<aside>The HealthVault Messaging APIs are insecure. It would be better for an institution to set up the Direct email protocol and send secure email to the HealthVault user. HealthVault users get free Direct email addresses.</aside>

Terminology

Terminologies, also known as vocabularies, are a list of codes associated with well-known terms in a particular domain.] HealthVault hosts numerous terminologies. Most as tagged as wc, and are created

by Microsoft. However several third party terminologies from the National Library of Medicine, USDA, HL7, and other institutions are also hosted.

<aside>

You can use the PowerShell HealthVault plugin (Listing 2.4) to verify that approximately 150 terminologies are hosted by HealthVault.

PS C:\> (Get-Vocabulary).Count

150

Listing 2.4 Using HvPosh to use Get-Vocabulary

</aside>

The terminologies can be accessed using the GetVocabulary API. Accessing the terminologies does not require user authentication; these are application only APIs. Some terminologies hosted by HealthVault, like RxNorm, are huge. RxNorm is a terminology that attempts to normalize all medication names and contains more than 200,000 entries. The SearchVocabulary API provides an XML interface as well as a JSON interface to search vocabularies. In fact, one can get an auto completion text box for entering a medication by using SearchVocabulary on RxNorm. The HealthVault user interface provides auto completion for medications, conditions, and other health item types using the SearchVocabulary API.

Application Management

The SetApplicationSettings and GetApplicationSettings functions provide a way for applications to store and retrieve their record-specific settings in HealthVault. Information like theme selection by a particular user or order of authorized records can be stored in application settings. The multiple record management (MRA) functionality detailed in Section 2.4.2 can be implemented using these APIs.

HealthVault also allows certain kind of applications, Master Applications, to create and manage other applications. Master Applications use the AddApplication and UpdateApplication functions to manage the “child” applications they create.

<aside> There are very few Master Applications in the HealthVault ecosystem. The GoLive or publication bar for these applications is high. Once an application is created, it cannot be deleted. </aside>

Service Discovery

The GetServiceDefinition function provides access to all the details of HealthVault applications, including Methods, Schemas, and Configurations for the service. Using GetServiceDefinition, an application can programmatically discover HealthVault service information and keep it up to date.

<aside>

You can use the PowerShell HealthVault plugin (Fig 3-4) to explore GetServiceDefinition.

```

PS C:\> Get-ServiceDefinition

HealthServiceUrl      : https://platform.healthvault-ppe.com/platform/wildcat.ashx
Version               : 1.8.1034.7107
HealthServiceShellInfo : Microsoft.Health.HealthServiceShellInfo
Assemblies             : {}
Methods               : {AddApplication, AllocatePackageId, AssociateAlternateId, BeginPutBlob...}
IncludedSchemaUrls    : {https://platform.healthvault-ppe.com/platform/XSD/types.xsd, https://platform.healthvault-ppe.com/platform/XSD/auth.xsd, https://platform.healthvault-ppe.com/platform/XSD/application.xsd, https://platform.healthvault-ppe.com/platform/XSD/vocab.xsd...}
ConfigurationValues   : {allowedDocumentExtensions, .avi, .bluebutton, .bmp, .ccd, .ccr, .cda, .doc, .docm, .docx, .eml, .gif, .jpg, .mp3, .one, .pdf, .png, .ppsm, .ppsx, .ppt, .pptm, .pptx, .pub, .rpsmsg, .rtf, .scp, .tif, .tiff, .txt, .vsd, .wav, .wma, .wmv, .xls, .xlsb, .xism, .xlsx, .xml, .xps}, {autoReconcilableTypes, 1e1ccbfc-a55d-4d91-8940-fa2fbf73c195, 9c48a2b8-952c-4f5a-935d-f3292326bf541, {blobHashBlockSizeBytes, 2097152}, {blobHashDefaultAlgorithm, SHA256Block1...}

```

Figure 3-4 Using GetServiceDefinition in HvPosh

</aside>

Open Query

Open Query is an insecure mechanism for running preconfigured queries invoked with an identifier on HealthVault data. For example, the following query:

<https://platform.healthvault.com/platform/openquery.ashx?id=9C4C77CF-1DF0-4c41-BD3D-EC9232B5BC8A>

invokes a saved request that corresponds to the specified identifier. Only queries associated with GetThings can be saved with SaveOpenQuery. The invocation of the open query doesn't require authentication and authorization and the HealthVault team discourages its use and it might be removed in future updates.

Read and Write API - Diving Deep

The most important HealthVault functions an application should become familiar with are GetThings and PutThings. In the following sections we will dive deeper in to each of these functions, discuss how their treatment in the HealthVault .NET SDK, and see some examples. The code in this chapter can be a starting point for the more complex applications introduced in the rest of the book.

GetThings

The best way to understand GetThings API is to look at the XML that an application would send to HealthVault platform to request a set of things.

```

1 <wc-request:request xmlns:wc-request="urn:com.microsoft.wc.request">
2   <auth>...</auth>
5   <header>...</header>
22  <info>
23    <group name="GetWeights" max="10">
24      <id>d8460ea8-50d4-4c30-ad92-49d1a1020b52</id>
25      <filter>
26        <type-id>d8460ea8-50d4-4c30-ad92-49d1a1020b52</type-id>
27        <thing-state>Active</thing-state>
28        <created-app-id>1F82D899-22E0-43F2-A645-59EDB6927645</created-app-id>
29        <xpath>/thing/data-xml/weight/value/kg[.>=60]</xpath>
30      </filter>
31      <format>
32        <section>core</section>
33        <xml />
34      </format>
35      <current-version-only>true</current-version-only>
36    </group>
37  </info>
38 </wc-request:request>

```

Listing 3-1 GetThings XML request

Each GetThings request can have multiple queries called a **group** (Line 23). Each group is identified by its **name**. The response from HealthVault combines the items returned by the group; group-name is used to index items returned for a particular query group. The group element can take one or more attributes to control the results. The attribute used in the previous example is **max**, which tells the HealthVault platform to return the top 10 items for this query.

Each query group can contain `<filter>` sub elements to return particular items with specific identification. Listing 3-1 requests items with a specific type-id (Line 24). The **d84..52** identifier is associated with a particular instance of the Weight type. Multiple IDs can be specified in each request. Other elements, such as client identifiers and thing keys, can also be used instead of an ID. The **filter** also restricts results to those that have an active thing-state (Line 27) and were created using the Withings Application (Line 28). Items can also be filtered by using XPath; on Line 29 we are looking for Weight item, whose values are greater than 60kg, maybe because we know that the scale was misconfigured during this time. To sum up, the query in Fig 2.11 returns the core xml sections of the top ten weight elements that were created by the Withings application and have values greater than 60kg.

The format and quantity of information returned by the GetThings query can be controlled by **format** specifiers. Using the **section** tag, we can specify that we just want the **core** elements of the requested thing-types (Line 32). Other section tags could specify digital signatures, audit information, or *effective* permissions for the request. Using the **xml** tag (Line 33) one can run an XSL transform on the thing-types or choose from existing transforms available for the type. Our xml tag is empty, so no transform is run in the sample query. We will discuss MTT, STT, and other transforms in Chapter 4.

An application can request only the current-versions of HealthVault thing types (Line 35), although the platform does store older versions of the data.

To complete the discussion on the format of the GetThings request, note that I have collapsed the auth (Line2) and header (Line 5) tags. These elements specify the authorization information for the method and various other header elements, like method name, final-xsl, version, user, and application tokens.

Table 3-4 summarizes the querying ability of GetThings method. To learn more, please refer to the methods schema documentation at <http://developer.healthvault.com/pages/methods/methods.aspx> and associated HealthVault SDK reference at <http://msdn.microsoft.com/en-us/library/hh672196.aspx>.

Search Criteria			
	.NET SDK Name	XML Element Name	Description
Group Attribute	Name	<code>name</code>	identifies the group
	Max	<code>max</code>	Maximum number of things to be returned
	Max-Full	<code>max-full</code>	Maximum number of full things
Identifiers	ClientItemIds	<code>client-thing-id</code>	Client Id of the thing
	ItemIds,	<code>Id</code>	Thing Instance Id
	ItemKeys	<code>key</code>	Thing Key
Filters	EffectiveDateMax, EffectiveDateMin, UpdatedDateMax, UpdatedDateMin	<code>eff-date-min</code> <code>eff-date-max</code> <code>updated-date-min</code> <code>updated-date-max</code>	Various thing filters
	CreatedDateMax, CreatedDateMin	<code>created-date-min</code> <code>created-date-max</code>	
	CreatedPerson, UpdatedPerson,	<code>created-person-id</code> <code>updated-person-id</code>	
	CreatedApplication, UpdatedApplication	<code>created-app-id</code> <code>updated-app-id</code>	
	XPath		
	States	<code>xpath</code>	
		<code>thing-state</code>	
Formats	View	Section	Section to be retrieved (core, audits, effectivepermissions, digitalsignatures)
		Xml	Name of transform to apply
		<code>type-version-format</code>	Version ID of type format
		<code>blob-payload-request</code>	Sequence of blob-filters (blob names) and blob-format-spec (information, inline or streamed)
Versions	CurrentVersionOnly	<code>current-version-only</code>	

Table 3-4 GetThings query parameters

Having understood the paradigm through which one can access things from HealthVault using the GetThings methods, let's look at how we can utilize querying to display Weight values in our application.

Our application currently fetches all the weight readings. However we want to be able to explore the readings on a graph one week at a time. Listing 3-2 shows how we can do it. As Line 52 shows, we begin by creating a searcher object for the record we are working with, and then create a new filter for Weight type (Line 53). Once we have the filter, we add properties to filter the value so that we get only weight items that are dated for previous week using the EffectiveDateMin (Line 54).

Once the query is constructed, we issue a GetThings request to HealthVault (Line 57). Since we have only one group, we index for the results in the first set of matching results (**GetMatchingItems()[0]**).

```
50 protected void Btn_ShowWeeklyWeightReadings_Click(object sender, EventArgs e)
51 {
52     HealthRecordSearcher searcher = PersonInfo.SelectedRecord.CreateSearcher();
53     HealthRecordFilter filter = new HealthRecordFilter(Weight.TypeId);
54     filter.EffectiveDateMin = DateTime.Now.Subtract(new TimeSpan(7, 0, 0, 0));
55     searcher.Filters.Add(filter);
56
57     HealthRecordItemCollection items = searcher.GetMatchingItems()[0];
58
59     TimeSeries t = new TimeSeries("Weight Graph");
60
61     foreach (Weight item in items)
62     {
63         //Assuming all data is in one unit
64         t.SeriesValue.Add(new TimeSeries.TimeSeriesValues(
65             item.EffectiveDate, item.Value.DisplayValue.Value));
66     }
67     TimeplotView.Plots.Add(t);
68     TimeplotView.DataBind();
69     TimeplotView.Visible = true;
70 }
```

Listing 3-2 Using the EffectiveDateMin filter to get weekly data

After getting the matching items, we create a new TimeSeries (Line 59) and add each item individually to the series. The TimeSeries.ascx.cs file contains an implementation of this class. You can choose to use your own implementation with any other graphing library. In this example, I'm using the Flot javascript graphing library (<http://code.google.com/p/flot/>). TimeplotView (Line 67) is an instance of a user control that has a Flot graphing object.

In line 67-69 we add the constructed TimeSeries to the Graphing object and make it visible on the screen. Fig 3-5 shows results of running this information on the selected record.

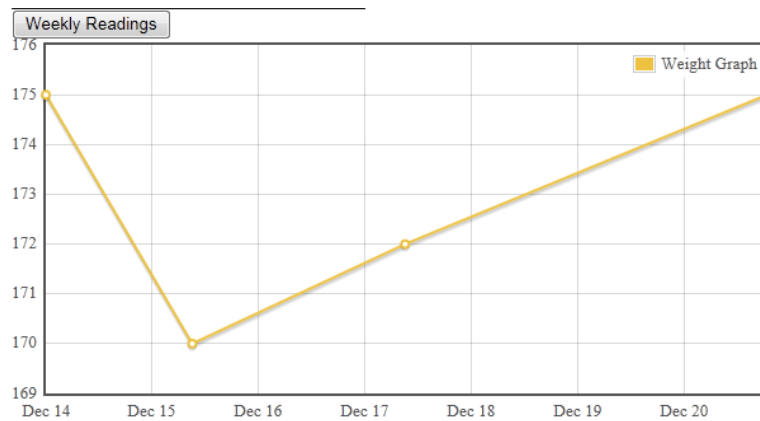


Fig 3-5 Filtering weight data for the week

PutThings

As with GetThings, we'll begin our coverage of the PutThings API by looking at the XML an application would send to HealthVault to create or update a set of things. Compared to GetThings, this is a simple request.

```

1 <wc-request:request xmlns:wc-request="urn:com.microsoft.wc.request">
2   <auth>
3     <hmac-data algName="HMACSHA1">IB7cdWxFNKE+xhrvE5poT5uTue=</hmac-data>
4   </auth>
5   <header>...</header>
22  <info>
23    <thing>
24      <type-id>0a5f9a43-dc88-4e9f-890f-1f9159b76e7b</type-id>
25      <thing-state>Active</thing-state>
26      <data-xml>...</data-xml>
718   </thing>
719   </info>
720 </wc-request:request>

```

Listing 3-3. Put-Things XML Sample

Each PutThings request can add instances of a **thing** (Line 23). Each thing has a type-id (Line 24) that identifies what kind of data item it is. In Listing 3-3, the thing is of type **Weight**. Because we are adding a weight thing, the **data-xml** (Line 23) part of this request needs to adhere to the schema for this particular type. In Chapter 4, we will discuss the thing-type schema. The important aspect of using this schema is to make sure to use a unique thing-id. In case of an update, the thing-id is the instance of weight you want to update. Additionally the thing-version-id element should be the version id of the element that is currently in the HealthVault. HealthVault offers optimistic concurrency: if an application tries to update an old thing that has already been updated by some other application, its version ID will have changed and the new put won't succeed. The thing-version-id is critical to make sure one update does not override another.

Let's modify our application to add a new Weight element to HealthVault. On the Default.aspx page we construct a text box to enter weight in pounds and associate a "save" button with it, with the action as "Btn_SubmitWeight_Click".

Listing 3-4 illustrates how we will go about saving a new element to HealthVault. Notice the HealthVault-specific DateTime field (HealthServiceDateTime) on Line 37 and a way to differentiate the actual WeightValue from DisplayValue on Lines 38-39. HealthVault enables health items to have a flexible date time. It also stores v measurements in a canonical format and allows the user to see them in the format in which they entered the data. In the case of Weight, it is stored canonically in kilograms, but we assume the user prefers to enter and display the weight in pounds. So on Line 39 we multiply the value entered by the user by 1.6. The DisplayValue of the weight is in pounds (lbs), which the applications working with this type can use to show the value to the user

```
33  protected void Btn_SubmitWeight_Click(object sender, EventArgs e)
34  {
35      double weight = double.Parse(Txt_Weight.Text);
36      Weight w = new Weight(
37          new HealthServiceDateTime(DateTime.Now),
38          new WeightValue(
39              weight * 1.6, new DisplayValue(weight, "lbs", "lbs")));
40
41      PersonInfo.SelectedRecord.NewItem(w);
42  }
```

Listing 3-4 Put-Things Example

Having a well-formed weight health item, we can send it to HealthVault by calling NewItem method in HealthVault .NET SDK. Under the hood NewItem call issues a PutThings request to HealthVault platform. The HealthVault SDK also has an UpdateItem method which saves changes to an existing item using the PutThings method.

To update an existing item, one essentially does the same thing as shown, except you use the UpdateItem SDK method instead of NewItem.

<aside>

The following code in Java is equivalent of the .NET Listing in 3-4. The OnlineRequestTemplate is generated using the libraries SimpleRequestTemplate, and it sets appropriate UserAuthToken and PersonId on the request.

```

public void PutThing() throws Exception
{
    long weightValueInKg = 80;

    DisplayValue dv = new DisplayValue();
    dv.setUnits("lb");
    dv.setUnitsCode("lb");
    dv.setValue(weightValueInKg/2.2);

    WeightValue wv = new WeightValue();
    wv.setKg(weightValueInKg);
    wv.setDisplay(dv);

    Weight weight = new Weight();
    weight.setValue(wv);
    weight.setWhen(DateTime.FromCalendar(Calendar.GetInstance()));

    Thing thing = new Thing();
    thing.setData(weight);

    SimpleRequestTemplate requestTemplate = new SimpleRequestTemplate(
        ConnectionFactory.getConnection());
    requestTemplate.setPersonId("75ac2c6c-c90e-4f7e-b74d-bb7e81787beb");
    requestTemplate.setRecordId("8c390004-3d41-4f5c-8f24-4841651579d6");

    PutThingsRequest request = new PutThingsRequest();
    request.getThing().add(thing);

    PutThingsResponse response = (PutThingsResponse)requestTemplate.makeRequest(request);
}

```

Listing 3-5 Reading Weight Using Java SDK
</aside>

Record Management – Diving deep

Single Record Application (SRA)

For an application working with single record, HealthVault provides a simple mechanism to switch to another record using a switch record hyperlink. In order for an application to switch to a different record, the application needs to tell the HealthVault shell to give the user the ability to switch the record and have a pre-configured receiving end-point for HealthVault shell to send the user back to a notification to change the selected record, termed a SELECTEDRECORD. The application redirects the user with the hyperlink URL as in the following example, the AUTH target implies an authentication request, the *appid* implies the identifier associated with the calling application and the *forceappauth* string makes sure that the user gets an ability to change the record.

```

https://account.healthvault-
ppe.com/redirect.aspx?target=AUTH&targetqs=appid%3D82d47a5a-d435-4246-895a-
746c475090d3%26forceappauth%3Dtrue.

```

Using the HealthVault .NET Web SDK this can be done with following line of code. The last variable enables applications to pass any optional parameters which need to pass-through the url redirection.

```

this.RedirectToShellUrl("AUTH", "appid=" + this.ApplicationId.ToString() +
"&forceappauth=true", "passthroughParam=optional");

```

As a simple example, Fig 3.6 shows how one can associate “switch account” functionality to an existing HealthVault application. Enabling this functionality involves two steps. First is to create an appropriate URL to send the user to HealthVault, Listing 3-6 shows the associated code. Second step in this process is to create a receiving URL so that HealthVault can send the user back to your application, Listing 3-7

shows the associated code required to configure a SelectedRecordChanged end-point page in the HealthVault SDK's web.config file. These two steps are relatively simple and can be accomplished with any programming language, section 2.1 explains the HealthVault shell interface and section 3 lists the available libraries.

Welcome Vaibhav | [Switch Account](#)

Weight Values

Key	Value
Weight	150

Fig 3.6. Adding ability to switch accounts.

In Listing 3-6 notice the Line 51, we are using the RedirectToShellUrl HealthVault SDK functionality to enable creating the appropriate redirection URL for HealthVault.

```

49 protected void Lnk_SwitchAccount_Click(object sender, EventArgs e)
50 {
51     this.RedirectToShellUrl("AUTH", "appid=" + this.ApplicationId.ToString() +
52         "&forceappauth=true", "passthroughParam=optional");
53 }

```

Listing 3-6 Adding Redirect Url in Default.aspx.cs

In Listing 3-7, we are creating a key in web.config file to associate a receiving end-point for selected record changed action. In the next section, we explain the HealthVault shell interface in detail.

```

26 <!-- Handling selected record changed -->
27 <add key="WCPage_ActionSelectedRecordChanged" value="default.aspx"/>
28

```

Listing 3-7 Handling SelectRecordChanged Target in Web.Config

Multiple Record Application (MRA)

The Mayo Clinic health manager works with all the records associated with the account and it termed as a multiple record application (MRA). As illustrated in Fig 3-7, I can switch from my record to my Mom's record seamlessly within the Mayo Clinic application. This flexibility can be achieved in an application by including IsMRA=true while communicating with HealthVault Shell's Authentication (AUTH) mechanism, as detailed in Section 2.1. Using this capability the application can then authenticate multiple records at the same time, and make requests to access health information for each one of them. For the application to remember the previous active record before the switch any associated record, it should store the current record identifier in the application settings associated with the person before making the switch.



Fig 3-7. An application working with multiple records

HealthVault SDK and Open Source Libraries

The HealthVault team offers a .NET SDK [link]. Additionally, a number of open-source libraries offer higher-level abstractions for interacting with the HealthVault platform. This section outlines the level of abstractions available in each of these.

HealthVault .NET SDK

The HealthVault .NET SDK is the official SDK available from Microsoft for working with the HealthVault platform. The HealthVault team maintains this SDK and provides interfaces for all HealthVault interfaces.

This SDK does not support the HealthVault Client APIs for mobile phones, but it does support the HealthVault Client APIs for Windows Applications. The Shell Redirect Interface is supported, but not all capabilities are supported. Notably, this is the only SDK that supports signing health items and streaming large files to HealthVault. HealthVault uses Azure, Microsoft's cloud storage service, to store these large files.

Throughout this book, we will be looking at code that uses this SDK, and refer to it as the HealthVault .NET SDK. Officially each major release of this SDK is supported for two years, and the SDK is currently compatible with .NET framework version 2.0.

The source code of this SDK is available for reference, but the license terms don't allow modifications to the SDK.

HealthVault Open Source Java SDK

This is the second most popular SDK for the HealthVault platform. The HealthVault open source Java SDK is available under a very permissive open source license at <http://healthvaultjavaib.codeplex.com>. This SDK was developed by members of HealthVault team and provides interfaces for most HealthVault interfaces. The source code of the SDK is available under the Microsoft Public license, and modifications and redistribution of this code are permitted for commercial and non-commercial purposes.

Notably, this SDK supports the HealthVault Client APIs for Android mobile phones, and provides a complete abstraction layer for Shell redirect interfaces. But it does not support Patient Connect,

asynchronous processes, signing of health items, or streaming large files to HealthVault. However, there are samples or documentation available for signing and streaming.

Additionally the SDK provides an object wrapper for thing-types using code generation tools. If these classes don't meet your needs, you can use the method schemas and create suitable wrappers.

This SDK is fully available for JDK 1.6, however raw authentication is supported for JDK 1.4. The SDK is community supported, and patches for bug fixes or missing functionality are welcomed.

<aside>

Certificate Management

In case of the .NET SDK and Windows platform the HealthVault SDK offers the Application Manager tool to make it easy to work with public and private key for your application. In case of Java, the best way to handle certificates for the application is to create one using the keytool for the Java Development kit.

The following keytool command creates a public and private key pair for your application in the java keystore:

```
keytool -genkeypair -keyalg RSA -keysize 2048 -keystore keystore -  
alias java-wildcat -validity 9999
```

Note the algorithm used is RSA and keysize is 2 kilobytes, its recommend to have the keysize as large as your installation supports. The generated key pair is valid for 9999 days, and you can choose to configure it. The name of the key-pair is java-wildcat this needs to be added to hv-application.properties file in the java sdk.

The HealthVault platform needs to have the certificate associated with the public key of your application. The keytool can also be used to export this certificate. Following is a sample command to do the same:

```
keytool -export -alias java-wildcat -keystore keystore > my-pub.cer
```

java-wildcat is the name of the application's key-pair and its exported as my-pub.cer.

</aside>

HealthVault Open Source iOS Mobile Library

The HealthVault team provides an open source and community supported library for the iOS platform available at <https://github.com/microsoft-hsg/HealthVault-Mobile-iOS-Library> . This library provides basic functionality to authenticate mobile clients. It doesn't provide support for any additional HealthVault features.

Applications such as iTriage, <http://www.itriagehealth.com/>, have used this library to create HealthVault iOS applications.

This library is available under the Apache 2.0 open source license, and modifications and redistribution of this code are permitted for commercial and non-commercial purposes.

HealthVault Open Source Windows Phone Library

Like the iOS library, the Windows Phone library at <http://healthvaultwp7.codeplex.com/> provides an authentication abstraction for Windows Phone mobile clients. Applications such as LiveScape (<http://livescape.mobi/>) have used this library to create HealthVault-enabled Windows Phone applications.

This library is available under the Apache 2.0 open source license, and modifications and redistribution of this code are permitted for commercial and non-commercial purposes.

In Chapter 5, we will walk through a detailed application showing how to work with HealthVault mobile interfaces.

HealthVault Open Source Python, PHP and Ruby Library

The HealthVault team has helped create Python, PHP, and Ruby libraries. These libraries are primarily driven by partners and provide the basic authentication layer for working with the HealthVault service. Applications such as TrailX (Python), Teladoc (PHP) and podfitness (Ruby) have used these libraries to create successful HealthVault applications.

These libraries are available under the Apache 2.0 open source license, and modifications and redistribution of this code are permitted for commercial and non-commercial purposes.

Table 2.2 summarizes the functionality available in various HealthVault libraries.

SDK library	Distribution	Supported Platform	Features Available	License & Support
HealthVault .NET	MSDN	Windows XP, Vista, 7 (.NET 2.0)	All HealthVault features	MS-RL Microsoft Supported
Java	Codeplex	JDK 1.6 JDK 1.4 (limited)	Authentication, Method wrappers, Thing-Type Wrappers.	MS-PL Community Support
Java	Codeplex	Android (1.6+)	Authentication, Thing-type wrappers	MS-PL Community Support
iOS	GitHub	iOS 4.0+	Mobile Authentication	Apache 2.0 Community Support
Windows Phone	Codeplex	Windows Phone 7+	Mobile Authentication	Apache 2.0 Community Support
Python	Google Code	Python 2.7	Authentication (Basic)	Apache 2.0 Community Support
PHP	SourceForge	PHP	Authentication (Basic)	Apache 2.0 Community Support
Ruby	RubyForge	Ruby	Authentication (Basic)	Apache 2.0 Community Support

Table 3-5 HealthVault SDK and Open Source Libraries

Interfacing with HealthVault

We touched on the HealthVault APIs and interface; these interfaces are usually combined in multiple ways to create integration architectures with HealthVault. This section discusses high-level options for integrating applications and devices with HealthVault. This discussion should be useful for understanding different architectural patterns available for interfacing devices and applications with HealthVault.

Device Connectivity

As of this writing, more than 80 types of devices connect with HealthVault. These devices vary from pedometers and weighing scales to blood pressure meters and pulse oximeters. Fig 3-8 shows the various interfaces available for a device to connect with HealthVault.

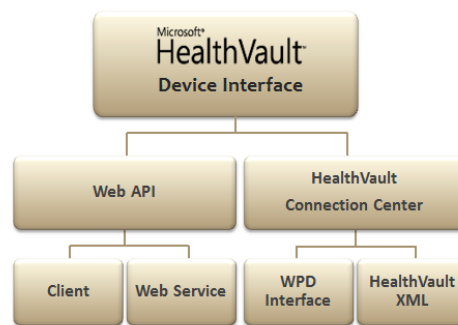


Fig 3-8. Interfaces for Device Integration with HealthVault.

Currently, a large number of devices interface with HealthVault through HealthVault's Windows client utility, called HealthVault Connection Center. HealthVault Connection Center enables device integration using the Windows Portable Devices (WPD) standard.

If a device already has a Windows device driver, the appropriate data can be communicated to HealthVault using WPD standard. The HealthVault team has a Device Development Kit (DDK) that can be used for this integration. It lies outside the scope of this book.

When there are no WPD supported elements for a device, it can still integrate with HealthVault through HealthVault Connection Center by sending and receiving HealthVault XML directly, Chapter 4 describes the HealthVault data type XML. This approach referred to as HealthVault XML in Fig 3-7. The HealthVault DDK has an example of how to go about configuring such an interface.

In addition to interfacing devices through HealthVault Connection Center, device manufacturers can write their own client application to enable data to upload to HealthVault using the HealthVault Client SDKs. ECG Glove which is available at <http://ineedmd.com/>, is a good example of a device that sends information to HealthVault using this interface.

Devices such as Fitbit and Withings actually take integration a step further and interface with HealthVault directly through the cloud using HealthVault APIs.

Continua

Continua Health Alliance is a non-profit, open industry organization of healthcare and technology companies joining together in collaboration to improve the quality of personal connected healthcare. With more than 230 member companies around the world, it is the leading consortium for personal healthcare devices. HealthVault has announced support for Continua drivers will be available in the future. In the future, devices will be able to play well in HealthVault and Continua ecosystem either by using the HealthVault Web API or converting data into IEEE 11073 formats.

Continua is not a standards body, but has identified a set of standards that together enable a personal connected healthcare ecosystem. At heart of it is the IEEE 11073 Personal Health Data Standard, which dictates various data standard profiles for devices ranging from blood pressure cuffs to weighing scales. IEEE 11073 is a data standard and is independent of transport.

On the transport layer, Continua supports USB personal Health Class devices, Bluetooth Healthcare device profiles, and other transports as they become compliant in the future.

Figure 3-9 shows the interfaces supported by Continua.



Fig 3-9. Continua complaint devices

Application Connectivity

As of this writing, there are more than 300 HealthVault applications are live in the United States. HealthVault Applications work with the HealthVault personal health data store by using various APIs over the HTTP protocol, as we have seen with PutThings and GetThings. Fig 3-10 depicts various ways in which applications have interfaced with HealthVault, depending on their use case.

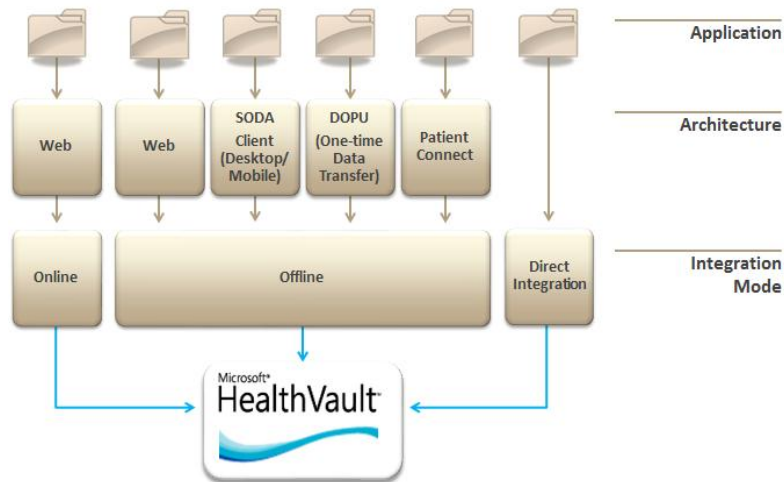


Fig 3-10. Connectivity Types with HealthVault

The following discussion goes into detail on various modes of connecting with HealthVault depending on the application needs as they pertain to the underlying platform, user consent, authentication, and user interface.

Online HealthVault Application

In addition to data storage, native HealthVault applications can use HealthVault for user authentication and authorization. Any data access using this mode requires the user's explicit permission each time the application interacts with HealthVault.

Mayo Clinic Health Manager, <https://healthmanager.mayoclinic.com/>, is a Native HealthVault application.

Offline HealthVault Application

Applications can choose to simply store and access data from HealthVault, without using HealthVault as a primary authentication and authorization entity.

FitBit, <http://www.fitbit.com/>, is a good example of an offline application. It links a FitBit account with a HealthVault user record and account, and then interacts with health items therein.

Drop Off Pick Up

Drop Off Pick Up (DOPU) is analogous to sending a secured fax to HealthVault user. The data flow in this architecture is one-way. The application drops the data into HealthVault and the user picks it up.

If a consumer happens to visit a health care institution and does not intend to have an on-going relationship with the care entity, DOPU provides an effective mechanism for the institution to provide documents.

Patient Connect

Some entities don't intend to maintain a public facing web site, but would like to have an ongoing relationship with their users through HealthVault. Patient connect provides an ideal mechanism for such

institutions. Via this mechanism, the user authorizes an application to read or write data to their HealthVault record through a user interface on HealthVault.com.

Clinical systems like Electronic Medical Records commonly use this model to connect to HealthVault, which is why this model is called Patient Connect. It should be noted, however, that this model is not limited to clinical systems and can be used by any back-end system.

Client Connectivity

Client Connectivity referred as Software on Device Authentication (SODA) enables applications to run on client platforms like desktop or mobile device, outside of the web browser. Every time a user installs a SODA application, the user must authorize that installation of the application to access their HealthVault record. For instance, if the user is running the same application on both their laptop and desktop, they will need to authorize both installations to access their HealthVault record.

A number of mobile health applications like iTriage, LiveScape, and Weight4Me use this architecture. In Chapter 5 we will develop a mobile application and do a detailed walk through of the APIs available to use this type of interface.

Direct Integration

The Direct project, formerly known as NHIN Direct, is collaboration between the public and private sector to develop a simple, secure, and standards-based method to send encrypted health information directly to known, trusted recipients over the Internet. This project aims at replacing the fax machine in Healthcare. Providers are able to send documents to each other securely.

Direct integration is the easiest kind of integration with HealthVault; trusting applications can actually send and receive documents to and from HealthVault using a Direct-enabled e-mail address. HealthVault users get an e-mail address in the format [handle]@direct.healthvault.com. As part of its Direct implementation, HealthVault automatically adds any recognized attachments like Clinical Care Documents or Clinical Care Records to the user's record.

Google Health was able to interface with HealthVault using the Direct integration. For HealthVault to accept e-mails from a new direct domain, the application needs to register the public key with HealthVault. If the application e-mails to newuser@direct.healthvault.com with the user's email address in the subject line, HealthVault stores the e-mail in a password encrypted package and send an e-mail to the user to associate the dropped off information to their record. The user can also sign in to their HealthVault record and read the message in HealthVault Message Center.

Application Provisioning and Master Applications

Application Provisioning refers to providing an application in HealthVault's production environment, which the HealthVault team does for all of connectivity models discussed so far. However, in special cases it provides the ability for applications known as "master" applications to provision individual HealthVault "child" applications.

Frequently, solution providers develop HealthVault integration for common scenarios such as uploading lab information or sending clinical care record information from a facility's electronic medical record

system. These solutions are deployed separately for each institution. The HealthVault team delegates the responsibility for creating these individual application instances to the solution provider through the Master Application mechanism.

Thus, for instance, if we wanted to deploy an individual instance of a Weight Tracker application per institution, we would use the AddApplication API available from the HealthVault Platform. The Appendix has links to examples and resources about how to create a child application.

“Data is a precious thing and will last longer than the systems themselves.”

Tim Bernes-Lee.

Chapter 4

Using HealthVault Data Ecosystem for Self Tracking

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.

QUANTIFIED SELF

Welcome Vaibhav | [Switch Account](#) | [Sign Out](#)

SELF EXPERIMENTATION

Keeping track of weight, sleep, exercise, mood & diet. View past hypothesis or input new ones.

Current Hypothesis

I'm happier if I sleep well. I sleep well if I drink less alcohol and exercise sufficiently.

MOOD

Input & track mood. You are tracking happiness.

DAILY DIETARY INTAKE

Input & track daily dietary intake. You are tracking carbs.

WEIGHT

Information on Weight.

SLEEP

Information on sleep.

EXERCISE

Exercise details.

SELF ANALYSIS

Graphical display to help with correlation.

[Weekly Readings Graph](#)

[Weekly Readings Summary](#)

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 button this utility can be accessed through All Programs → Microsoft HealthVault → SDK → HealthVault Application Manager.

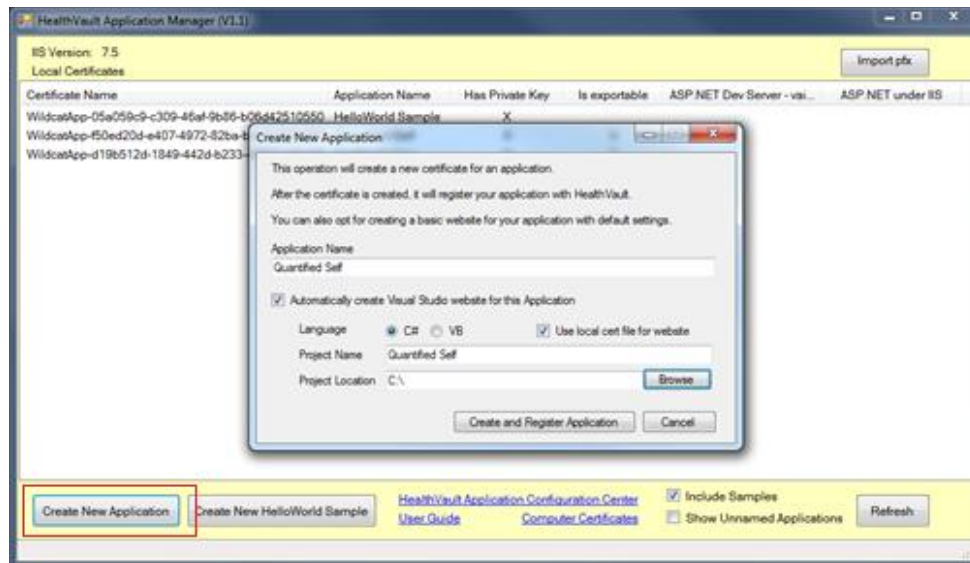


Fig 4-2 Using Application Manager to create a new HealthVault application

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.

f50ed20d-e407-4972-82ba-bdc2b726d363	Quantified Self	Default	Start Go-Live process	View app config in production Disable
--------------------------------------	-----------------	---------	-----------------------	---

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.

Microsoft HealthVault Application Configuration Center

Home > Applications > Quantified Self

Here you can manage data types and the access (Create, Read, Update, Delete) your application needs when the user is online.

Edit online auth rule

Rule name
QS-Rule-1

Why string
Data for Self Experimentation

Is optional?
☐

Display flags
☐ Display first time ☐ Checked first time ☐ Checked by default

Permissions
☒ All ☒ Create ☒ Read ☒ Update ☒ Delete

Data types ([Uncheck All](#) | [Type details](#))

<input type="checkbox"/> Advance Directive	<input type="checkbox"/> Continuity of Care Record (CCR)	<input type="checkbox"/> Group Membership	<input type="checkbox"/> Message
<input type="checkbox"/> Aerobic Profile	<input type="checkbox"/> Contraindication	<input type="checkbox"/> Group Membership Activity	<input type="checkbox"/> Microbiology Lab Test Result
<input checked="" type="checkbox"/> Allergic Episode	<input checked="" type="checkbox"/> Daily Dietary Intake	<input type="checkbox"/> HbA1C Measurement	<input type="checkbox"/> PAP Session
<input type="checkbox"/> Allergy	<input type="checkbox"/> Daily Medication Usage	<input type="checkbox"/> Health Assessment	<input type="checkbox"/> Password Protected Package
<input type="checkbox"/> Application Data Reference	<input type="checkbox"/> Device	<input type="checkbox"/> Health Event	<input type="checkbox"/> Peak Flow Measurement
<input checked="" type="checkbox"/> Application-Specific Information	<input type="checkbox"/> Diabetes Insulin Injection Use	<input type="checkbox"/> Healthcare Proxy	<input type="checkbox"/> Spirometer Measurement
<input type="checkbox"/> Appointment	<input type="checkbox"/> Diabetic Profile	<input type="checkbox"/> Heart Rate	<input checked="" type="checkbox"/> Personal Contact Information
<input type="checkbox"/> Asthma Inhaler	<input type="checkbox"/> Discharge Summary	<input type="checkbox"/> Height Measurement	<input type="checkbox"/> Personal Demographic Information
<input type="checkbox"/> Asthma Inhaler Usage	<input type="checkbox"/> Emergency or Provider Contact	<input type="checkbox"/> Immunization	<input checked="" type="checkbox"/> Personal Image
<input type="checkbox"/> Basic Demographic Information	<input type="checkbox"/> Emotional State	<input type="checkbox"/> Immunization	<input type="checkbox"/> Pregnancy
<input type="checkbox"/> Basic Demographic Information	<input type="checkbox"/> Encounter	<input type="checkbox"/> Insulin Injection	<input type="checkbox"/> Procedure
<input type="checkbox"/> Blood Glucose Measurement	<input type="checkbox"/> Encounter	<input type="checkbox"/> Insurance Plan	<input type="checkbox"/> Procedure
<input type="checkbox"/> Blood Oxygen Saturation	<input checked="" type="checkbox"/> Exercise	<input type="checkbox"/> Lab Test Results	<input type="checkbox"/> Question Answer
<input type="checkbox"/> Blood Pressure Measurement	<input type="checkbox"/> Aerobic Exercise Session	<input type="checkbox"/> Lab Test Results	<input type="checkbox"/> Radiology Lab Result
<input type="checkbox"/> Body Composition	<input type="checkbox"/> Exercise Samples	<input type="checkbox"/> Life Goal	<input type="checkbox"/> Respiratory Profile
<input type="checkbox"/> Body Dimension	<input type="checkbox"/> Explanation of Benefits	<input type="checkbox"/> Link	<input type="checkbox"/> Sleep Related Activity
<input type="checkbox"/> Calorie Guideline	<input type="checkbox"/> Family History	<input type="checkbox"/> Medical Annotation	<input checked="" type="checkbox"/> Sleep Session
<input type="checkbox"/> Cardiac Profile	<input type="checkbox"/> Family History	<input type="checkbox"/> Medical Image Study	<input type="checkbox"/> Status
<input type="checkbox"/> Cholesterol Profile (Lipid Profile)	<input type="checkbox"/> Family History Condition	<input type="checkbox"/> Medical Image Study	<input type="checkbox"/> Vital Signs
<input type="checkbox"/> Clinical Document Architecture (CDA)	<input type="checkbox"/> Family History Person	<input type="checkbox"/> Medical Problem	<input type="checkbox"/> Weekly Aerobic Exercise Goal
<input type="checkbox"/> Concern	<input type="checkbox"/> File	<input type="checkbox"/> Medication	<input type="checkbox"/> Weight Goal
<input type="checkbox"/> Condition	<input type="checkbox"/> Genetic SNP Result	<input type="checkbox"/> Medication Fill	<input checked="" type="checkbox"/> Weight Measurement
<input type="checkbox"/> Continuity of Care Document (CCD)			

© Microsoft | [Privacy](#) | [Legal](#) | [Help](#) | [Feedback](#)

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.

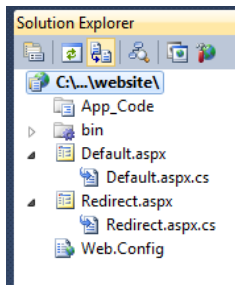


Fig 4-5 Solution created by Application Manager

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.

```
17         Lbl_UserName.Text = this.PersonInfo.SelectedRecord.DisplayName;
18
19         HealthRecordSearcher searcher = PersonInfo.SelectedRecord.CreateSearcher();
20         HealthRecordFilter filter = new HealthRecordFilter(
21             Emotion.TypeId,
22             DietaryDailyIntake.TypeId,
23             Weight.TypeId,
24             SleepJournalAM.TypeId,
25             Exercise.TypeId);
26
27         filter.EffectiveDateMin = DateTime.Now.Subtract(new TimeSpan(7, 0, 0, 0));
28         searcher.Filters.Add(filter);
29
30         HealthRecordItemCollection items = searcher.GetMatchingItems()[0];
```


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

Weight Measurement

[Return to types list](#)

Type Properties

Properties	
Property	Value
id	3d34d87e-7fc1-4153-800f-f56592cb0d17
name	Weight Measurement
uncreateable	false
immutable	false
singleton	false
transforms	form, mtt, stt, wpd-0A4A7F3D-877D-4541-BE27-3F08D0709B27, include-hvcc-mapping, hvcc-display, include-hvcc-props

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** summarizes 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).

```
1 <data-xml transform="mtt">
2   <row wc-id="34655fb4-a6c8-4d47-85f1-dbc6e09b952a" wc-version="0f57073a-0795-4867-9c9f-bcb99d2fa681" wc-note="" wc-tags=""
   wc-date="2011-12-23 11:17:47" wc-type="Weight Measurement" wc-typeid="3d34d87e-7fc1-4153-800f-f56592cb0d17" wc-source="" wc-
   brands="" wc-issigned="false" wc-flags="" wc-ispersonal="false" wc-isdownversioned="false" wc-isupversioned="false" wc-
   relatedthings="" wc-state="Active" summary="173 lbs" />
3 </data-xml>
```

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.

```

175 protected void Btn_ShowWeeklyReadingsTextSummary_Click(object sender, System.EventArgs e)
176 {
177     HealthRecordSearcher searcher = PersonInfo.SelectedRecord.CreateSearcher();
178     HealthRecordFilter filter = new HealthRecordFilter(
179         Emotion.TypeId,
180         DietaryDailyIntake.TypeId,
181         Weight.TypeId,
182         SleepJournalAM.TypeId,
183         Exercise.TypeId);
184
185     filter.EffectiveDateMin = DateTime.Now.Subtract(new TimeSpan(7, 0, 0, 0));
186     searcher.Filters.Add(filter);
187     filter.View.TransformToApply.Add("mtt");
188
189     HealthRecordItemCollection items = searcher.GetMatchingItems()[0];
190
191     DataTable dataTable = new DataTable();
192     dataTable.Columns.Add(new DataColumn("Date", typeof(string)));
193     dataTable.Columns.Add(new DataColumn("Type", typeof(string)));
194     dataTable.Columns.Add(new DataColumn("Summary", typeof(string)));
195     foreach (HealthRecordItem item in items)
196     {
197         XmlNode mttDocument = item.TransformedXmlData["mtt"].SelectSingleNode("data-xml/row");
198         DataRow row = dataTable.NewRow();
199         row["Date"] = mttDocument.Attributes["wc-date"].Value;
200         row["Type"] = mttDocument.Attributes["wc-type"].Value;
201         row["Summary"] = mttDocument.Attributes["summary"].Value;
202         dataTable.Rows.Add(row);
203     }
204
205     Grid_ReadingsTextSummary.DataSource = dataTable;
206     Grid_ReadingsTextSummary.DataBind();
207     Grid_ReadingsTextSummary.Visible = true;
208 }

```

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.

Summary

Date	Type	Summary
2011-12-23 11:17:47	Weight Measurement	173 lbs
2011-12-23 10:11:59	Exercise	Walking, Walked 10000 steps
2011-12-23 09:22:52	Weight Measurement	172 lbs
2011-12-23	Daily Dietary Intake	1000 kCal, carbs:600000 g
2011-12-20 18:41:00	Weight Measurement	175 lbs

Weekly Readings Summary

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

<aside>

The CCR HealthVault type (1e1ccbf7c-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

```

1 <schema xmlns:weight="urn:com.microsoft.wc.thing.weight"
2         xmlns:t="urn:com.microsoft.wc.thing.types"
3         xmlns:d="urn:com.microsoft.wc.dates"
4         xmlns="http://www.w3.org/2001/XMLSchema"
5         targetNamespace="urn:com.microsoft.wc.thing.weight">
6   <import namespace="urn:com.microsoft.wc.thing.types" schemaLocation="base.xsd" />
7   <import namespace="urn:com.microsoft.wc.dates" schemaLocation="dates.xsd" />
8
9   <element name="weight">
10     <complexType>
11       <sequence>
12         <element name="when" minOccurs="1" maxOccurs="1" type="d:date-time"/>
13         <element name="value" minOccurs="1" maxOccurs="1" type="t:weight-value"/>
14       </sequence>
15     </complexType>
16   </element>
17 </schema>

```

```

1 <data-xml>
2   <weight>
3     <when>
4       <date>
5         <y>1990</y>
6         <m>1</m>
7         <d>1</d>
8       </date>
9       <time>
10        <h>1</h>
11        <m>0</m>
12        <s>0</s>
13        <f>0</f>
14      </time>
15    </when>
16    <value>
17      <kg>60</kg>
18      <display units="lb">132</display>
19    </value>
20  </weight>
21 </common/>
22 </data-xml>

```

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



	Value	Gets or sets the person's weight.
	When	Gets or sets the date/time when the weight measurement was taken.

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).

```

100 protected void Btn_SubmitWeight_Click(object sender, EventArgs e)
101 {
102     double weight = double.Parse(Txt_Weight.Text);
103     Weight w = new Weight(
104         new HealthServiceDateTime(DateTime.Now),
105         new WeightValue(
106             weight * 1.6, new DisplayValue(weight, "lbs", "lbs")));
107     w.CommonData.Source = _appName;
108     PersonInfo.SelectedRecord.NewItem(w);
109 }

```

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.

```

1 <complexType name="codable-value">
2   <sequence>
3     <element name="text" type="string">
4     </element>
5     <element name="code" type="this:coded-value" minOccurs="0" maxOccurs="unbounded">
6     </element>
7   </sequence>
8 </complexType>
9 <complexType name="coded-value">
10  <sequence>
11    <element name="value" type="string">
12    </element>
13    <element name="family" type="string" minOccurs="0">
14    </element>
15    <element name="type" type="string">
16    </element>
17    <element name="version" type="string" minOccurs="0">
18    </element>
19  </sequence>
20 </complexType>
21

```

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.

```

private void DisplayExercise(List<Exercise> exercises)
{
    DataTable exercise = new DataTable("exercise");
    exercise.Columns.Add(new DataColumn("Date"));
    exercise.Columns.Add(new DataColumn("ExerciseType"));
    exercise.Columns.Add(new DataColumn("CaloriesBurned"));
    foreach (Exercise e in exercises)
    {
        DataRow row = exercise.NewRow();
        row["Date"] = e.EffectiveDate.ToShortDateString().ToString();
        row["ExerciseType"] = e.Activity.Text;
        if (e.Details.ContainsKey(ExerciseDetail.CaloriesBurned_calories))
        {
            row["CaloriesBurned"] = e.Details[ExerciseDetail.CaloriesBurned_calories];
        }
        exercise.Rows.Add(row);
    }
    ExerciseView.DataSource = exercise;
    ExerciseView.DataBind();
}

```

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 (_appDailyAlcoholExtensionName) in the extension element. Listing 4-7, Line [204-206] in the following example shows how one can do it in the .NET SDK.

```
190 protected void Submit_Daily_Diet_Click(object sender, System.EventArgs e)
191 {
192     //Post Diet
193     DietaryDailyIntake diet = new DietaryDailyIntake();
194     int totalCarbs;
195     int.TryParse(Txt_DailyDietCarbs.Text, out totalCarbs);
196     diet.TotalCarbohydrates.Kilograms = totalCarbs * 1000;
197     diet.CommonData.Note = Txt_DailyDietNote.Text;
198
199     //Adding extension data
200     string drinks = Txt_DailyDietAlcohol.Text;
201     HealthRecordItemExtension extension =
202         new HealthRecordItemExtension(_appDailyAlcoholExtensionName);
203     diet.CommonData.Extensions.Add(extension);
204     XPathNavigator navigator = extension.ExtensionData.CreateNavigator();
205     navigator.InnerXml = @"<extension source="" + _appDailyAlcoholExtensionName + @"">
206         <alcoholic-drinks>" + drinks + "</alcoholic-drinks>";
207
208     PersonInfo.SelectedRecord.NewItem(diet);
209 }
```

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

DAILY DIETARY INTAKE

Input & track dialy dietary intake. You are tracking carbs.

Record today's carb intake('600gm carbs, 2 drinks and note: Eat burger bun'):

Carbs (in gms):

Alcohol (drinks/day)

Note:

Last Week's readings:

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.

```
211 private void DisplayDailyDiet(List<DietaryDailyIntake> dailydiets)
212 {
213     DataTable dailydiet = new DataTable("DailyDiets");
214     dailydiet.Columns.Add(new DataColumn("Date"));
215     dailydiet.Columns.Add(new DataColumn("Carbs (in gm)"));
216     dailydiet.Columns.Add(new DataColumn("Alcohol (#drinks)"));
217     dailydiet.Columns.Add(new DataColumn("Note"));
218     foreach (DietaryDailyIntake e in dailydiets)
219     {
220         DataRow row = dailydiet.NewRow();
221         row["Date"] = e.EffectiveDate.ToShortDateString().ToString();
222         row["Carbs (in gm)"] = e.ToString();
223         foreach (HealthRecordItemExtension extension in e.CommonData.Extensions)
224         {
225             if (extension.Source == _appDailyAlcoholExtensionName)
226             {
227                 XPathNavigator navigator = extension.ExtensionData.CreateNavigator();
228                 XPathNavigator alcoholicDrinksNavigator =
229                     navigator.SelectSingleNode("extension/alcoholic-drinks");
230                 if (alcoholicDrinksNavigator != null)
231                 {
232                     row["Alcohol (#drinks)"] = alcoholicDrinksNavigator.Value;
233                 }
234             }
235         }
236         row["Note"] = e.CommonData.Note;
237         dailydiet.Rows.Add(row);
238     }
239     DailyDietView.DataSource = dailydiet;
240     DailyDietView.DataBind();
241 }
```

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 subtype tag 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.

```
370 protected void Btn_Submit_Hypothesis_Click(object sender, System.EventArgs e)
371 {
372     ApplicationSpecific appSpecific = new ApplicationSpecific();
373     string hypothesis = Txt_Hypothesis.Text;
374     appSpecific.ApplicationId = this.ApplicationConnection.ApplicationId.ToString();
375     XmlDocument xml = new XmlDocument();
376     xml.LoadXml(
377         string.Format("<self-experiment><hypothesis>{0}</hypothesis></self-experiment>",
378             hypothesis));
379     appSpecific.ApplicationSpecificXml.Add(xml);
380     appSpecific.SubtypeTag = "self-experiment";
381     appSpecific.Description = hypothesis;
382     // Default the status note to active when the hypothesis is created
383     appSpecific.CommonData.Note = "Active";
384     appSpecific.When = new HealthServiceDateTime(DateTime.Now);
385     PersonInfo.SelectedRecord.NewItem(appSpecific);
386 }
```

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.

```
244 private void DisplaySelfExperiments(List<ApplicationSpecific> selfExperiments)
245 {
246     DataTable selfExperiment = new DataTable("SelfExperiments");
247     selfExperiment.Columns.Add(new DataColumn("Date"));
248     selfExperiment.Columns.Add(new DataColumn("Hypothesis"));
249     selfExperiment.Columns.Add(new DataColumn("Status"));
250     foreach (ApplicationSpecific s in selfExperiments)
251     {
252
253         DataRow row = selfExperiment.NewRow();
254         row["Date"] = s.EffectiveDate.ToShortDateString().ToString();
255         row["Hypothesis"] = s.ApplicationSpecificXml[0].CreateNavigator().
256             SelectSingleNode("hypothesis").Value;
257         row["Status"] = s.CommonData.Note;
258         selfExperiment.Rows.Add(row);
259     }
260     SelfExperimentsView.DataSource = selfExperiment;
261     SelfExperimentsView.DataBind();
262 }
```

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 4-11 shows 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.

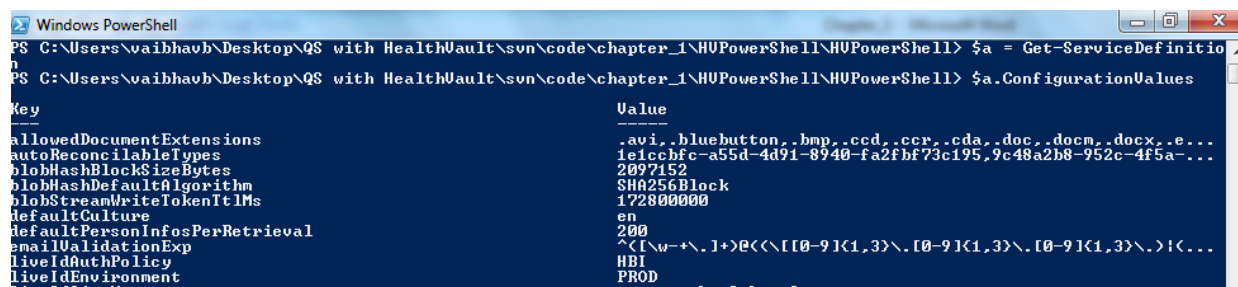
Personal Profile

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 GetServiceDefinition 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>]



```
Windows PowerShell
PS C:\Users\vaibhav\Desktop\QS with HealthVault\svn\code\chapter_1\HUPowerShell\HUPowerShell> $a = Get-ServiceDefinition
PS C:\Users\vaibhav\Desktop\QS with HealthVault\svn\code\chapter_1\HUPowerShell\HUPowerShell> $a.ConfigurationValues

Key                                     Value
----
allowedDocumentExtensions              .avi,.bluesbutton,.bmp,.ccd,.ccr,.cda,.doc,.docm,.docx,.e...
autoReconcilableTypes                  1e1ccbfc-a55d-4d91-8940-fa2fbf73c195,9c48a2b8-952c-4f5a-...
blobHashBlockSizeBytes                 2097152
blobHashDefaultAlgorithm                SHA256Block
blobStreamWriteTokenItMs               172800000
defaultCulture                         en
defaultPersonInfosPerRetrieval         200
emailValidationExp                     ^(<[\w+.\ ]+>@<([\0-9]{1,3}\. [\0-9]{1,3}\. [\0-9]{1,3}\. )!<...
liveIdAuthPolicy                       HBI
liveIdEnvironment                      PROD
```

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!

“Think: mHealth as personal health reform”

Jane Sarasohn-Kahn

Chapter 5

Enabling mHealth for the Quantified Self

Having an accessible and programmable health record sets HealthVault apart. It enables a rich ecosystem of devices, mobile, and web applications. Chapter 3 focused on introducing the HealthVault application programming interface; Chapter 4 gave a good overview of HealthVault data types using a data-intensive “Quantified-Self” application. This chapter takes a closer look at building mobile applications for HealthVault.

We will look at an end-to-end example of building a mood tracking application on top mobile platforms. The chapter will cover elements of mobile client programming using the code samples for Windows Phone 7 (C#). Similar interfaces are available for Android (Java) and iOS (Objective-C).

The Mood Tracker Mobile Application

In Chapter 3, we built an end-to-end web-application that enables a user to track several kinds of data and use that data to help with self-experimentation. Many elements of self-tracking data, such as sleep, weight, and exercise have the capability to be measured through devices; however, it’s very hard to measure elements of happiness such as mood and stress automatically.

In recent years, we have seen a surge in mobile smartphone devices. Mobile devices offer a very effective tool for efficient data entry. Mobile devices are an ideal platform to build data collection tools. So our manual “mood tracking” need could be served by an application that makes it easy and engaging for a user to track mood using a smart phone. For purposes of our example, let’s build the application on Windows Phone 7 platform.

So, what should we build?

The application will allow the user to input their mood, well-being, and stress level; present a way to look at the history of the data; and add a bit of zest using an avatar, a mood plant. The mood plant summarizes the user’s emotional state over time. When the user is happy, stress-free, and fit for a long time, the plant thrives, showing a happy face (☺) and in case of depression and stress it shows the effects of bad health (☹).

Figure 5-1 is a sketch of what the app might look like.

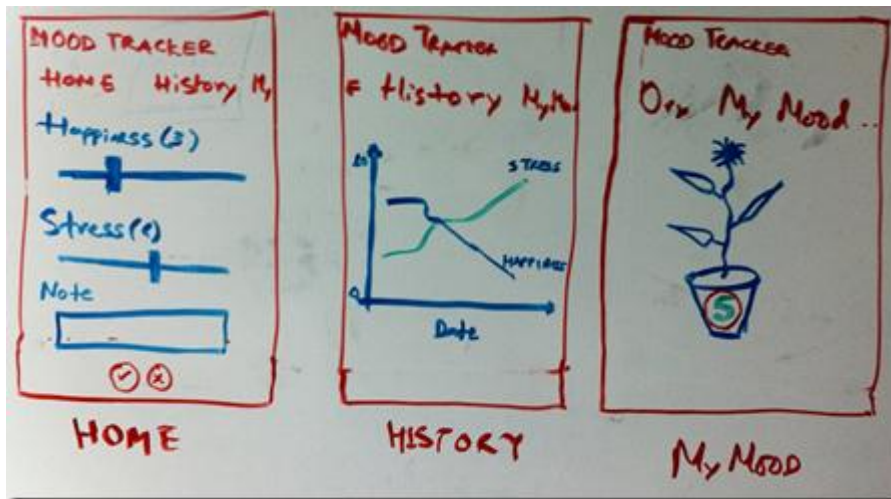


Fig 5-1. White board wire-frame of our mood tracking application.

Choosing HealthVault Integration

The first question we need to answer is **what kind of HealthVault connectivity does this application require?** We discussed several models of connecting with HealthVault in chapter 3. As this application is only for a client device, the model we will be using a client application, and will utilize the HealthVault Windows Phone 7 client library. Having a client application allows you to provide a rich interface and the potential capability to store the readings locally.

Selecting Appropriate HealthVault data types

The next question we should solve is **what HealthVault data types to use?** We discussed HealthVault data types in detail in chapter 4. Various data types could apply in this context, but browsing the [HealthVault data types](#) reveals one relevant data type in particular: [Emotional State](#).

On further analysis, it turns out that this type is almost perfect for our use. Mood, stress, and well-being are rated on a scale of 1-5. We do a further reading of associations for each of these values, and add appropriate textual elements for each of the values (mood, stress, and well-being).

Getting Started

I assume you have Visual Studio installed with Window Phone 7 (WP7) tools. If not, you can get them from [here](#).

Next, go over to [codeplex](#) and download the HealthVault library with sample applications.

I extracted the library to my desktop and the folder structure looks like Fig 5-2. HvMobileRegular has the relevant C# code to abstract for working with the HealthVault web service, and HvMobilePhone uses the code in HvMobileRegular to build a library that works with Windows Phone 7 platform. TestRegular directory has a unit-test for HealthVault mobile Windows Phone 7 library. WegithTrackerDemo is a sample application that shows case uses of the library for a Weight Tracking application.

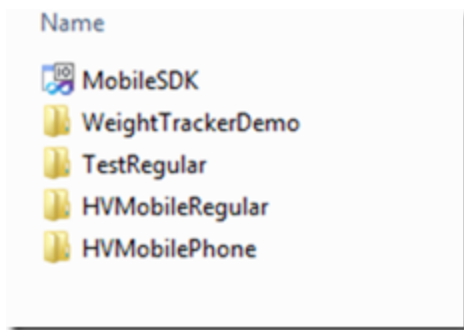


Fig 5-2. HealthVault WP7 library extracted.

If you open the MobileSDK solution in visual studio and press F5, the library compiles and the WeightTracker demo starts. Fig 5-3 shows this application in action; we will use this application as a template for building ours.

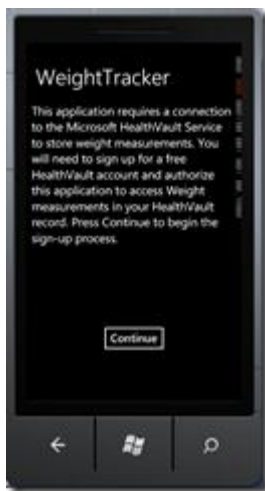


Fig 5-3. Compiling and running HealthVault WP7 Sample Application

Without further ado, let's create our new Silverlight for Windows phone project. We can create a solution for MoodTracker, and reference the HVMobilePhone library in that project. You can also use the existing project, MobileSDK, and associate a new application in it; in the source code associated with this chapter I created a new Project called MoodTracker (Figure 5-4).

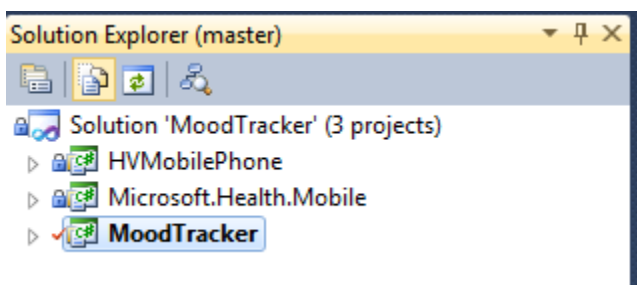


Fig 5-4. Beginnings of Mood Tracker

First things first, let's set up the application to talk to HealthVault. In the App.xaml.cs class, add a reference to the HealthVault Service and HealthVault Shell. We also need to make sure we get a unique application ID in the developer environment of HealthVault. To do that, we head over to the [HealthVault Application Configuration Center](#), and create a new application by clicking on the "Create a new application" button (Figure 5-5). Note that in Chapter 3 we used the Application Manager utility to create a web application, in this chapter we use an alternative method which enables creating client as well as web applications.

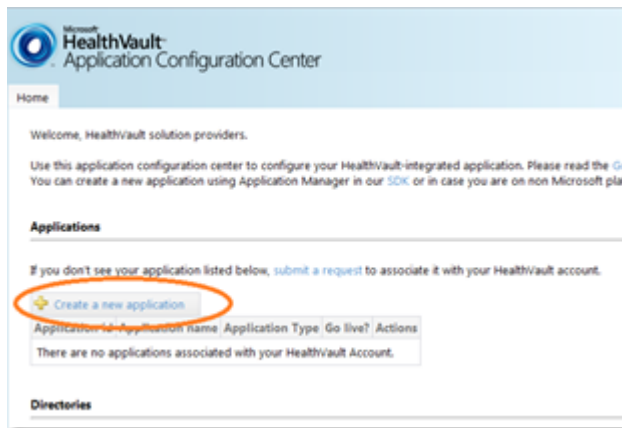


Fig5-5. Creating a new application in Application Configuration Center

We create an application of type Software on Device Auth (SODA), which is an authentication mechanism for client applications, and pick the name Mood Tracker for it, as shown in figure 5-6.

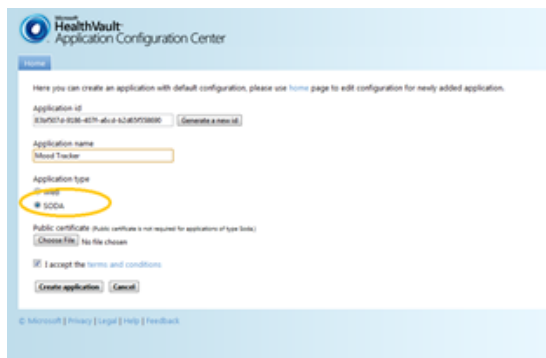


Fig 5-6. Creating Mood tracker as a SODA application

Once the application is created, we need to assign appropriate authorization rules for the data types that the application will access. To do that, click on the app's link and assign appropriate data types for the application, as shown in Figure 5-7.

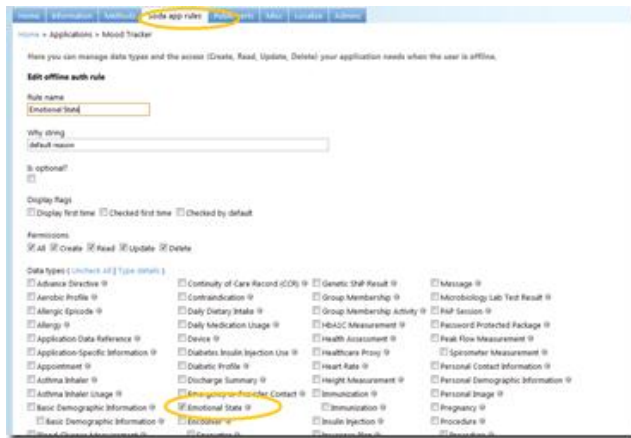


Fig 5-7. Adding Emotional State to our Mood tracker application.

Having created the client application and assigned data type authorization rules, we are all set! Now let's configure the base page to work with the HealthVault pre-production environment (PPE). The PPE is the development environment publicly available for all HealthVault developers. The HealthVault platform in this environment is available at <https://platform.healthvault-ppe.com/platform/wildcat.ashx> and the HealthVault shell in this environment is available at <https://account.healthvault.com>. Chapter 6 will show how to deploy your app to the general public after you have developed and tested it.

Listing 5-1 shows the initial code for configuring the application. In Line 27, we assign the `platformUrl`, In Line 28, we assign the `shellUrl`, and Line 29 is the application identifier that we created using the application configuration center. The `HealthVaultService` object initialized the HealthVault Windows Phone 7 library with appropriate configuration variables. Using this object, we can make all the relevant HealthVault web service requests.

```

18 namespace MoodTracker
19 {
20     public partial class App : Application
21     {
22         public static string SettingsFilename = "Settings";
23         public static HealthVaultService HealthVaultService { get; set; }
24         public static string HealthVaultShellUrl { get; set; }
25
26
27         static string platformUrl = @"https://platform.healthvault-ppe.com/platform/wildcat.ashx";
28         static string shellUrl = @"https://account.healthvault-ppe.com";
29         static string masterAppId = "83bf507d-9186-407f-a6cd-b2d65f558690";
30
31         // Code to execute when the application is launching (eg, from Start)
32         // This code will not execute when the application is reactivated
33         private void Application_Launching(object sender, LaunchingEventArgs e)
34         {
35             HealthVaultService = new HealthVaultService(platformUrl, shellUrl, new Guid(masterAppId));
36         }
37         // Code to execute when the application is activated (brought to foreground)
38         // This code will not execute when the application is first launched
39         private void Application_Activated(object sender, ActivatedEventArgs e)
40         {
41             HealthVaultService = new HealthVaultService(platformUrl, shellUrl, new Guid(masterAppId));
42         }
43     }

```

Listing 5-1 configuring the client application

We can make this project a startup project, press F5, and get to the first page of our application. We're in business!

Authenticating the Application and User with HealthVault

In order for the Mood Tracker application to work with HealthVault, we will get appropriate application creation credentials from the HealthVault Service. We must also set up a method by which the user can authorize the application using the HealthVault Shell.

1. To get the credentials from the HealthVault Service, the application contacts the HealthVault service to get application creation URL. The code for that is outlined in [MyMood.xaml.cs \(Listing 5-2\)](#).

```

47 void MainPage_Loaded(object sender, RoutedEventArgs e)
48 {
49     App.HealthVaultService.LoadSettings(App.SettingsFilename);
50     App.HealthVaultService.BeginAuthenticationCheck(AuthenticationCompleted,
51         DoShellAuthentication);
52     SetProgressBarVisibility(true);
53 }
54
55 void DoShellAuthentication(object sender, HealthVaultResponseEventArgs e)
56 {
57     SetProgressBarVisibility(false);
58
59     App.HealthVaultService.SaveSettings(App.SettingsFilename);
60
61     string url;
62
63     if (!_addingRecord)
64     {
65         url = App.HealthVaultService.GetUserAuthorizationUrl();
66     }
67     else
68     {
69         url = App.HealthVaultService.GetApplicationCreationUrl();
70     }
71 }

```

Listing 5-2 Authenticating the application with HealthVault Service

2. The application creation needs to be validated on behalf of the user.

The best mechanism to achieve this is by having a page with a hosted browser that redirects appropriately to HealthVault and then closes the browser and navigates back to the application page after a successful authorization.

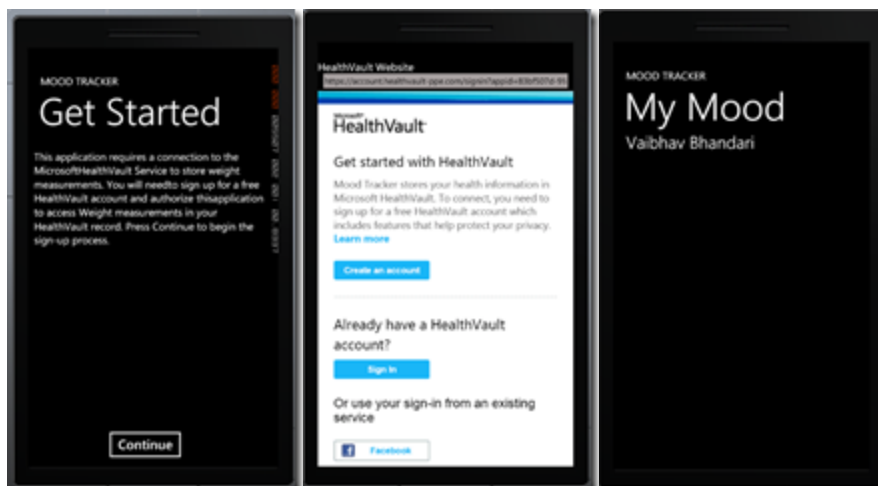
Listing 5-3 is the relevant code in HostedBrowserPage.xaml.

```
33 void c_webBrowser_Navigated(object sender, System.Windows.Navigation.NavigationEventArgs e)
34 {
35     if (e.Uri.OriginalString.Contains("target=AppAuthSuccess"))
36     {
37         Uri pageUri = new Uri("/MyMood.xaml", UriKind.RelativeOrAbsolute);
38
39         Deployment.Current.Dispatcher.BeginInvoke(() =>
40         {
41             NavigationService.Navigate(pageUri);
42         });
43     }
44 }
45
46 void HealthVaultWebPage_Loaded(object sender, RoutedEventArgs e)
47 {
48     string url = App.HealthVaultShellUrl;
49
50     c_webBrowser.Navigate(new Uri(url));
51 }
```

Listing 5-3 using a hosted browser to show HealthVault user authentication

Note that on success the Application is being redirected to MyMood.xaml, which is our application's landing page.

Figure 5-8 shows the flow of how the authentication shown here works.



Step1. Home page Step2. Sign-in & Auth Step3. Mood tracker in action.

(Note to artist: please include the previous 3 steps as a legend in the figure.)

Fig 5-8 Authentication Model with HealthVault

Reading Data from HealthVault

The data type we settled on for our application was EmotionalState. Our first goal is to be able to read data for this type and display it in our application. To do this, we need test data for emotional state. Add test information in to the test or developer account for this application from [type samples](#), associated with Emotional type in the HealthVault Developer Center, as shown in Fig 5-9. An important thing to note is that you need to be signed in to developer.healthvault.com to add the sample; otherwise this application gives an error.

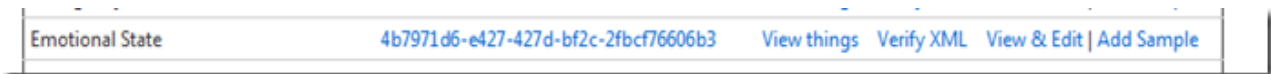


Fig 5-9. Adding Emotional State sample to HealthVault record.

We can verify that this sample is added to our record by viewing the information in the HealthVault PPE shell interface (<https://account.healthvault-ppe.com>), as shown in Fig 5-10.

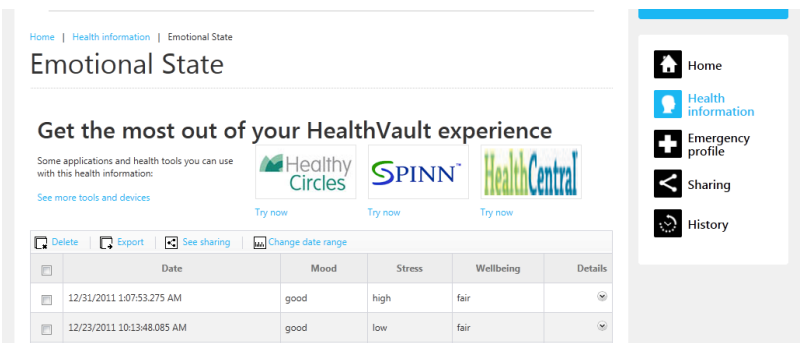


Fig 5-10. Emotional State Samples in developer account.

Chapter 2 explained the HealthVault GetThings method. GetThings enables an application to read data from the user's health record. A read request for health data can be performed using various querying mechanisms. For the purposes of this application, we will retrieve the last item active for the user's Emotional State data type.

```
103 // We are only interested in the last item
104 HealthVaultMethods.GetThings(EmotionalStateModel.TypeId, 1, null, null, GetThingsCompleted);
```

Listing 5-4 Using GetThings

To make it easier to work with GetThings, I implemented a simple abstraction on the method in HealthVaultMethods class. Listing 5-5 shows code for this abstraction. It allows the construction of a GetThings query for one typeid, with maximum items returned and with appropriate minimum and maximum effective dates for these health items. Chapter 4 explains the XML query sent by the GetThings method in detail.

```

26     public static void GetThings(string typeId,
27         int? maxItems,
28         DateTime? effDateMin,
29         DateTime? effDateMax,
30         EventHandler<HealthVaultResponseEventArgs> responseCallback)
31     {
32         string thingXml = @"
33         <info>
34             <group {0}>
35                 <filter>
36                     <type-id>{1}</type-id>
37                     <thing-state>Active</thing-state>
38                     {2}
39                     {3}
40                 </filter>
41                 <format>
42                     <section>core</section>
43                     <xml/>
44                     <type-version-format>{1}</type-version-format>
45                 </format>
46             </group>
47         </info>";
48
49         XElement info = XElement.Parse(string.Format
50             (thingXml,
51             MaxItemsXml(maxItems),
52             typeId,
53             EffDateMinXml(effDateMin),
54             EffDateMaxXml(effDateMax)));
55         HealthVaultRequest request = new HealthVaultRequest("GetThings", "3", info, responseCallback);
56         App.HealthVaultService.BeginSendRequest(request);
57     }

```

Listing 5-5 GetThings abstraction

Now, once we can get emotional state things, we need to perform two actions things on the client side.

First, pick the *thing* item we are interested in from the *GetThings* response. To choose the appropriate item, LINQ to XML comes in very handy, offering a SQL-like select clause for XML data as shown in Listing 5-6. LINQ stands for Language Integrated Querying, and it allows for making queries natively from C#.

```

148     // using LINQ to get the latest reading of emotional state
149     XElement latestEmotion = (from thingNode in responseNode.Descendants("thing")
150         orderby Convert.ToDateTime(thingNode.Element("eff-date").Value) descending
151         select thingNode).FirstOrDefault<XElement>();

```

Listing 5-6 Choosing things from a Get-Thing response

Second, parse the items returned for mood, stress, and well-being data. We can achieve this by creating a model for Emotional State. This model is available to review in the *EmotionalStateModel.cs* file. The parse method on this model parses the appropriate elements in the *thing* XML. Chapter 4 details the format of this XML. Notice that in Line 68, we parse the common element to fetch the *note* data for emotional state type. In 66, we are setting the When date of the instance to the *eff-date* element. We have created enumerations for Mood, Stress and Wellbeing values, and we can parse the integers for those values using the *Enum.Parse* method.


```

58 public void Parse(XElement thingXml)
59 {
60     this.Mood = Mood.None;
61     this.Stress = Stress.None;
62     this.Wellbeing = Wellbeing.None;
63
64     XElement emotionalState = thingXml.Descendants("data-xml").Descendants("emotion").First();
65
66     this.When = Convert.ToDateTime(thingXml.Element("eff-date").Value);
67
68     if (thingXml.Descendants("common") != null &&
69         (thingXml.Descendants("common").Descendants("note").Count() != 0))
70     {
71         this.Note = thingXml.Descendants("common").Descendants("note").First().Value;
72     }
73
74     if (emotionalState.Element("mood") != null)
75     {
76         try
77         {
78             this.Mood = (Mood)System.Enum.Parse(typeof(Mood),
79                 ((XElement)emotionalState.Element("mood")).Value, true);
80         }
81         catch (Exception) { }
82     }
83     if (emotionalState.Element("stress") != null)
84     {
85         try
86         {
87             this.Stress = (Stress)System.Enum.Parse(typeof(Stress),
88                 ((XElement)emotionalState.Element("stress")).Value, true);
89         }
90         catch (Exception) { }
91     }
92     if (emotionalState.Element("wellbeing") != null)
93     {
94         try
95         {
96             this.Wellbeing = (Wellbeing)System.Enum.Parse(typeof(Wellbeing),
97                 ((XElement)emotionalState.Element("wellbeing")).Value, true);
98         }
99         catch (Exception) { }
100     }
101 }

```

Listing 5-7 parsing a thing for Emotion State data type

After retrieving the data in our emotional state model, we can use XAML to view it in our application. XAML is the user interface markup technology for Windows Phone 7. For the purposes of this book, we won't go in detail of XAML. Fig 5-11 shows the display of the latest emotional state reading from HealthVault.

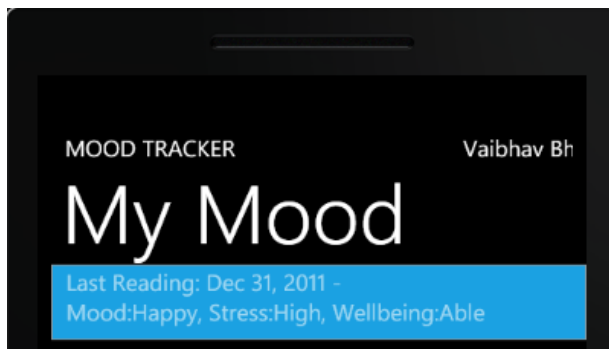


Fig 5-11. Latest Emotional State Reading in Mood Tracker!

Writing Data to HealthVault

In the previous [section](#) we discussed how one can display the data retrieved from the HealthVault Emotional State data-type. Before we get to the topic of this section and discuss how we can put new items into HealthVault, Fig 5-12 shows a screen shot of how the application looks once we have enabled the put.

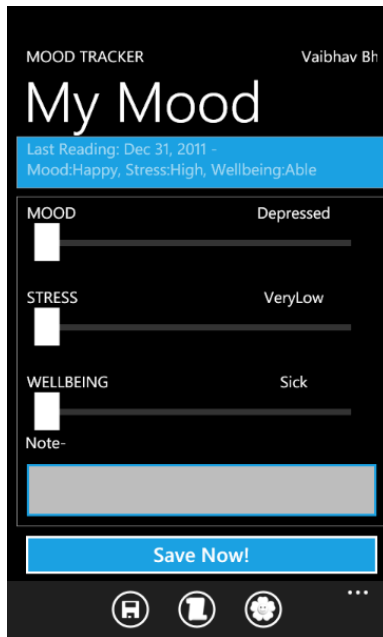


Fig 5-12. MoodTracker with put enabled

For each of the emotional states--mood, stress and well-being--we have a slider that lets the user capture their emotional state. They can also add a note pertaining to their mood using a text box. We want this information to be uploaded with the current time stamp once the user hits Save Now!

Listing 5-6 shows how the save button submits information to HealthVault. Note that in Line 190 we are calling an abstraction for PutThings method.

```
181 // Save the reading to HealthVault
182 private void Btn_SaveReadingToHealthVault_Click(object sender, RoutedEventArgs e)
183 {
184     EmotionalStateModel model = new EmotionalStateModel();
185     model.Mood = (Mood)c_MoodSlider.Value;
186     model.Stress = (Stress)c_StressSlider.Value;
187     model.Wellbeing = (Wellbeing)c_WellbeingSlider.Value;
188     model.When = DateTime.Now;
189     model.Note = GetNote();
190     HealthVaultMethods.PutThings(model, PutThingsCompleted);
191     SetProgressBarVisibility(true);
192 }
```

Listing 5-6 Saving new data to HealthVault

In Chapter 3, we looked at the PutThings method in detail. PutThings enables an application to add or update health items in a user's record. As Listing 5-7 line 97 shows, our abstraction fetches the relevant information from the base health record item object and submits that to HealthVault using the PutThings version 2 API. The response for this request is handled by the responseCallback function, which in turn can check for various return codes from the service.

```

97         public static void PutThings(HealthRecordItemModel item,
98                                     EventHandler<HealthVaultResponseEventArgs> responseCallback)
99     {
100         XElement info = XElement.Parse(item.GetXml());
101         HealthVaultRequest request = new HealthVaultRequest("PutThings", "2", info, responseCallback);
102         App.HealthVaultService.BeginSendRequest(request);
103     }

```

Listing 5-7 PutThings abstraction

Now that we are able to write data to HealthVault, we have a mobile application which can read and update information to HealthVault!

Graphing Mood

In the last section we enabled Mood Tracker to enter new data in to HealthVault. We want to be able to discover patterns in mood, stress and well-being, and graphing them over time is a great mechanism by which to achieve this goal. Let's start with a simplistic approach, showing the emotional state readings for mood, stress and well-being over a week. As Fig 5-13 shows, a user can browse mood reading based on a weekly margin and move forward or backward a week at time.

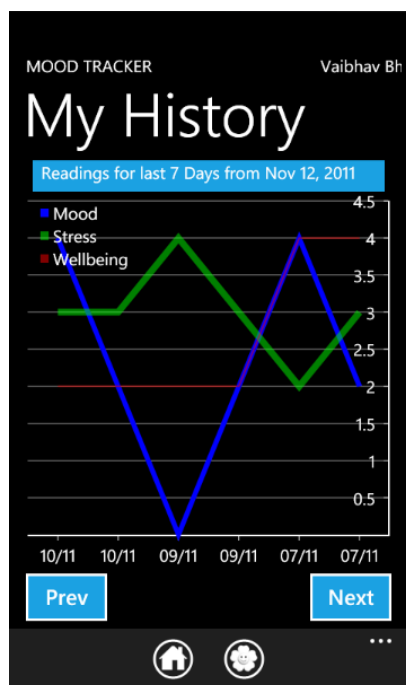


Fig 5-13 Graphing Emotional State over 7 days

In order to be able to get data from HealthVault for a specific time period, the GetThings[link] method needs to have the effective date filter enabled to look for appropriate readings. Line 66 in Listing 5-8 shows how the GetThings abstraction is configured to return elements for the last seven days only.

```
64 // Get the last emotional state info and try to plot a graph
65 HealthVaultMethods.GetThings(EmotionalStateModel.TypeId, null,
66     BaseTimeForGraph.Subtract(new TimeSpan(7, 0, 0, 0)),
67     BaseTimeForGraph,
68     GetThingsCompleted);
```

Listing 5-8 Fetching readings for last seven days

Note that the eff-date-min element, as implemented in GetThings class in HealthVaultMethods.cs, must be formatted in the ISO 8601 format. Line 73 in Listing 5-9 shows how we do the formatting.

```
69 private static string EffDateMinXml(DateTime? effDateMin)
70 {
71     if (effDateMin != null)
72         return
73             string.Format(@"<eff-date-min>{0}</eff-date-min>",
74                 effDateMin.Value.ToString("yyyy-MM-ddTHH:mm:ss.FFFZ",
75                     CultureInfo.InvariantCulture));
76     else return "";
77 }
78 }
```

Listing 5-9 Formatting eff-date-min for Get-Things request

Once we can selectively get information from HealthVault, we can use a graphing library to show the readings. In our case, I chose the open-source graphing library amCharts for its ease of use. In fact, I added it to the project with one click using the [NuGet Package manager](#). Listing 5-10 shows a snippet of the configuration code showing how the graph is set up for mood, stress and well-being using a serial chart. Note that the series values are bound in Line 65 using a DataSource called EmotionList; it is a list of *observable* emotional states.

```

63         <amq:SerialChart x:Name="EmotionsChart"
64             BorderThickness="1"
65             DataSource="{Binding EmotionList}"
66             CategoryValueMemberPath="FormattedWhen"
67             AxisForeground="White"
68             PlotAreaBackground="Black"
69             GridStroke="DarkGray" Height="463" Width="450">
70         <amq:SerialChart.Graphs>
71             <amq:LineGraph ValueMemberPath="Mood"
72                 Title="Mood" Brush="Blue"
73                 StrokeThickness="6"
74                 BorderBrush="Cornsilk"/>
75             <amq:LineGraph ValueMemberPath="Stress"
76                 Title="Stress" Brush="#8000FF00"
77                 StrokeThickness="8" />
78             <amq:LineGraph ValueMemberPath="Wellbeing"
79                 Title="Wellbeing"
80                 StrokeThickness="2"
81                 Brush="#80FF0000"/>
82         </amq:SerialChart.Graphs>
83     </amq:SerialChart>

```

Listing 5-10 Graphing Emotional State

Data Analysis – Mood Plant

We want a user to engage with the emotional state readings, and a good way to achieve this goal is by providing a zestful visualization for their emotional state. We use a mood plant as a mechanism to gauge a user's emotional state over recent past.

The flower of the plant represents the average mood, the leaves represent average stress and the roots represent the average wellbeing. Individual flower, leaf and root ligatures map the values 0 through 5 to mood, stress and wellbeing. The final mood plant is a result of super-imposing these values. Fig 5-14 shows an instance of mood plant with mood 3, stress 3, and wellbeing 3.

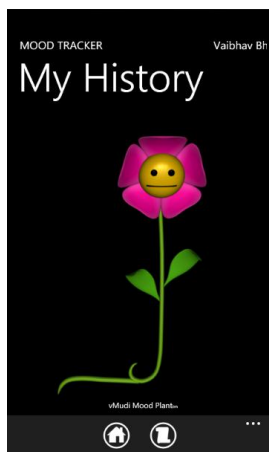


Fig 5-14 Mood Plant

So how do we find average mood, stress, or well-being? Various correlations and algorithms can be used to express the average emotional state over time. We will start with a simple function that creates an average mood, stress, or well-being score based on a weighted average of the values. The function takes the readings for mood, stress, or well-being for the past month. It assigns 50% weight to the more recent week, 30% to the week 3, and 20% to weeks 1 and 2 of the month's readings. It is left as an exercise to the reader to evaluate and try different functions. Listing 5-11, shows the code for the weighting algorithm

```

83      /*
84      * Algorithm
85      * 1. Read last 1 month's readings
86      * 2. Weight 50% to 4th week
87      * 3. Weight 25% to 1-3 week
88      * 4. Weight 25% to how many readings (Good is 4/wk)
89      */
90      DateTime time50p = baseTime.Subtract(new TimeSpan(7,0,0,0));
91      DateTime time30p = baseTime.Subtract(new TimeSpan(14, 0, 0, 0));
92
93      int m = 0; int s = 0; int w = 0;
94      int c50 = 0; int c30 = 0; int c20 = 0;
95      foreach (EmotionalStateModel emotion in emotionList)
96      {
97          if (emotion.When >= time50p)
98          {
99              m += (int)emotion.Mood * 50 ;
100             s += (int)emotion.Stress * 50 ;
101             w += (int)emotion.Wellbeing * 50;
102             c50++;
103         }
104         else if (emotion.When >= time30p)
105         {
106             m += (int)emotion.Mood * 30;
107             s += (int)emotion.Stress * 30 ;
108             w += (int)emotion.Wellbeing * 30;
109             c30++;
110         }
111         else
112         {
113             m += (int)emotion.Mood * 20;
114             s += (int)emotion.Stress * 20;
115             w += (int)emotion.Wellbeing * 20;
116             c20++;
117         }
118     }
119
120     // Final numbers
121     int c = 50 * c50 + 30 * c30 + 20 * c20;
122     m = m / c;
123     s = s / c;
124     w = w / c;

```

Listing 5-11 Algorithm for calculating mood, stress, and well-being over the past month

What about Android and iOS?

In Chapter 3, we discussed various libraries available for HealthVault. In particular, there are libraries available for Android (Java) and iOS (Objective-C) that allow a developer to implement a mobile application on these platforms. These libraries are open source and commercial friendly.

We don't show the code for a sample application these platforms, but the functionality available is very similar to the Windows Phone 7 library and the material covered in this chapter will be equally useful.

The Android library is available with the Java SDK on Codeplex [link], and the iOS library is available on GitHub [link]. It is left as an exercise to the reader to create solutions for these platforms.

Mobile Web Applications

Earlier in the chapter, we discussed the architectural choice to develop a client application for our Mood Tracker. However, web applications do have a choice to use the Web for delivery. In fact, we might want

to link the Mood tracker application to our Quantified Self web application so that a user can see other relevant data in a web browser.

It's important to remember that a web application is a separate application entity from a native client application and that the user has to authorize it separately. The Quantified Self application, when launched from the Mood tracker, will ask for the user to sign in and authorize it.

Web applications can use standard browser detection techniques to present a mobile view of the content. HealthVault Shell does configure its view for mobile applications, and the calling application can always force a mobile view by adding ***mobile=true*** to the URL parameters. It would be a good exercise for the reader to implement a mobile view of the quantified self-application.

Contributing to the Mood Tracker Application

The source for MoodTracker windows phone application 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 iOS, Android versions of it!

“Be careful about reading health books. You may die of a misprint.”

Mark Twain

Chapter 6

The Last Mile – Releasing Applications to Users

Over the last few chapters we have gained an understanding of the HealthVault application programming interface (Chapter 3), learned about building a HealthVault web application with a focus on the HealthVault data types (Chapter 4), and built an engaging mobile application (Chapter 5). An application’s life cycle typically involves testing the application, releasing it to the user, and then monitoring it for anomalies, tasks that entail special requirements in a HealthVault context. This chapter will highlight best practices for releasing, maintaining, and marketing HealthVault application to end users.

Testing your application

Well written software goes through multiple test cycles, including both automated and manual tests. This section outlines some valuable test scenarios around HealthVault account management, API interfaces, and data types, which you should consider in addition to other tests.

HealthVault enables people to share and manage multiple health records. It’s important to make sure that your application will work with multiple records associated with the same HealthVault account. In Chapter 3, we covered account management and ways to configure record switching. You need to ensure your application works with an account with multiple records. The best way to achieve this is to create several test accounts with multiple health records and try your application with them.

Another important aspect of account management is sharing. You can test this by sharing a healthvault record with another person and then making sure they can authenticate your application in association with that record. In case of insufficient permissions, your application should show an error message.

HealthVault provides an XML-based Web API. This API is accessible through programming libraries available through various HealthVault libraries and SDKs, as discussed in Chapter 3. While developing an application, you should pay special attention to any failure codes returned from HealthVault. In fact, make sure you log non-success return codes from API calls to HealthVault so that you can investigate reasons for failure. HealthVault provides a comprehensive list of status codes returned by the service at <http://msdn.microsoft.com/en-us/library/hh567902.aspx>. Particularly interesting is CREDENTIAL_TOKEN_EXPIRED, which your application should handle by requesting a new credentials token from HealthVault. The HealthVault .NET SDK and Mobile SDKs handle this status appropriately for you. In case you see the INVALID_XML status code, you should look closer at your request to make sure the XML is valid for various HealthVault methods and data type schemas, which are available at <http://developer.healthvault.com>.

HealthVault enables a coherent and well adopted data ecosystem. It is very important to make sure that your application works well in this data ecosystem and uses the HealthVault data types appropriately. Chapter 4 explains HealthVault data types in detail. The best way to that make sure you are reading a data type appropriately is to create a few instances of the type you are consuming with various possible permutations and combinations and then access them in your application. Get Real Consulting offers a great tool, HealthVault X-ray, <http://xray.getrealconsulting.com/>, which enables you to create myriad instances of the data type you are consuming in an appropriate test account.

The second aspect of working with the data types is to make sure that the information you are writing is consumable by other applications. The HealthVault team offers a tool called HealthVault Data type checkup at <http://datacheckup.healthvault-ppe.com/hvappcheckup>, it works against data written by your application in a test record and finds any compatibility issues. Currently, the HealthVault data checkup tool supports only a limited number of types. Another mechanism to ensure that you play well is to copy your test records through HealthVault X-Ray in a production test account, authorize other HealthVault applications to access this account, and then confirm that the information is consumable by these applications.

Frequently, applications code properties of HealthVault data types improperly, or are not able to parse a flexible date format as used in HealthVault. Review Chapter 4 to make sure you handle these cases.

Releasing your application to end users

After testing your application thoroughly, including the conditions listed in the previous section, you are ready to release it to end users. The HealthVault team has documented the release process, termed the Go-Live Process, on the HealthVault developer network at <http://msdn.microsoft.com/en-us/healthvault/bb962148>.

The first step in this process it to ensure that you have a signed a business agreement with HealthVault, and have an identifier associated with your partner account. This is a non-technical step and could be done way before your application is ready to be released.

Having established a partner account with Microsoft, you can submit a request to the technical team to review your application in the pre-production environment and push it to the HealthVault production environment. The review typically tests that your application plays well in the HealthVault ecosystem and uses the brand appropriately.

An important step, once your application is available for the world to use, is to network with fellow applications! This can be done using the Wiki provided by the HealthVault team at http://partners.mshealthcommunity.com/hv_eco/w/wiki/partner-directory.aspx. In addition to business networking, this wiki is used by applications to notify the development community of any extensions to data types that they have implemented. Having well documented extensions available makes it easy for other applications to work with data created by your application, giving you the benefit of network effects.

The last step in your Go-Live Process is to make sure that your application is discoverable to end users. This typically means working with the HealthVault team to become part of their application directory at <http://www.healthvault.com>.

Monitoring and maintaining your application

Congratulations on getting that application out there! Whether you have created a client or a web application, it's very important to monitor its health.

You should log all the failed calls to the HealthVault web service. For additional debugging, the HealthVault SDK provides a tracing mechanism which you can use to log all the request responses. The mechanics of it are detailed at <http://msdn.microsoft.com/en-us/library/ff803626.aspx>. The HealthVault team monitors their development forums available at <http://www.msdn.com/healthvault>, and you can use them to report any anomalies or failures in the service.

Each release of the HealthVault .NET SDK is supported for two years, and the team frequently adds enhancements and bug fixes to the newer releases. SDKs and libraries available in other language--Java, Python, etc. -- are also updated by the community. As part of maintaining your application, you should make sure you monitor the underlying libraries so that you can upgrade your service to use the most robust offerings.

Another important aspect of maintaining a HealthVault application is to maintain its security artifacts, such as the application certificate and user tokens. HealthVault uses X509 certificates to authenticate web applications; you should make sure that the certificates you use for your application have the appropriate validity to function for a long time. HealthVault uses long-lived user tokens for client applications and you should make sure that these applications frequently refresh the tokens.

Adding new features to your application

Having a well-tested and well-maintained usable application will frequently result in a number of feature requests from users, not a bad problem to have! Many of these feature requests will necessitate support for additional HealthVault data types.

When you are updating your application to have access to new HealthVault data types, you must request the user to re-authorize your application so that you can get access to these additional data types. Another feature available from HealthVault is optional rules; these rules are data type authorization rules that ask permission for only additional data types. In addition to providing a smooth upgrade curve, optional rules also enable you to run an older version of your application side by side with the new version in case your users prefer not to upgrade. You can read more about optional rules at <http://msdn.microsoft.com/en-us/library/ff803609.aspx>.

Updating the data type rules of an application is not automatic, and an application typically needs to go through the HealthVault Go-Live process at <http://msdn.microsoft.com/en-us/healthvault/bb962148> to release this update in HealthVault production environment.

Taking your application international!

Throughout the book, we have worked with HealthVault as it is available in United States. However HealthVault has a growing list of implementation partners and the platform is available in Canada, the UK, and Germany as of this writing. You can work with the HealthVault team to explore releasing your application in each of these countries.

The important aspect to keep in mind is that HealthVault as a service is completely globalized and internationalized. The request and response string HealthVault displays to a user can be changed to appropriate locales. In fact, you can also make a Spanish version of your application available in the United States. The HealthVault application configuration center through its *Localize* tab, allows developers to configure their applications with Spanish strings so that when a user with Spanish browser settings accesses HealthVault, the application's authorization screen is shown with the appropriate language. The HealthVault Shell redirect interface, which is discussed in detail in Chapter 3, also respects an *lcid*= {*Locale ID*} parameter in its query string to show the appropriate display language in the user interface.

Additionally, HealthVault offers an internationalized set of units and time formats to allow applications to work with appropriate standards in the target country.

With this information, you are all set to stride into the exciting world of enabling quantified self with HealthVault!

Further Resources

This section highlights some important resource available for HealthVault development.

Need reference information?

The HealthVault MSDN Site, <http://www.msdn.com/healthvault>, is a great resource on all things HealthVault. HealthVault features and SDK information is available in the reference section at <http://msdn.microsoft.com/en-us/library/aa155110.aspx>. The HealthVault team has an active blog at <http://blogs.msdn.com/b/healthvault/> and a list of frequently asked questions at <http://blogs.msdn.com/b/healthvaultfaq/>.

Have a Question?

The HealthVault Forums at <http://social.msdn.microsoft.com/forums/en-US/healthvault/> are a great place to ask technical questions.

Development tools

Fiddler is a great tool to enable request – response tracing for web applications. This tool will help you look at and analyses XML information being exchanged with HealthVault platform.

Get Real Consulting's X-Ray for HealthVault at <https://xray.getrealconsulting.com/> is an indispensable tool for HealthVault development. It offers the ability to create data in a HealthVault pre-production and production environment and export & import information.

The HealthVault team hosts the developer center tool at <http://developer.healthvault.com/>; it is very handy to look at HealthVault method and data type schemas.

HealthVault Data Type Checkup at <http://datacheckup.healthvault-ppe.com/hvappcheckup>, offers lint functionality for data written by your application. This tool highlights best practices for writing data to HealthVault for a select set of data types.

Mapping your data to HealthVault

Chapter 4 summarizes the intent and use of various data types in HealthVault. Didier Thizy has a good post at http://www.macadamian.com/insight/healthcare_detail/mapping_hl7_phm_to_healthvault/ on mapping the HL7 PHM standard to HealthVault.

The HealthVault team maintains a detailed mapping of the ASTM Continuity of Care Record standard to HealthVault data types on MSDN at <http://msdn.microsoft.com/en-us/healthvault/ee663895>. The reference article at <http://msdn.microsoft.com/en-us/library/ff803579.aspx> is a great resource for using CCR data in HealthVault.