

TFS Practical Reporting Guide - Part 1

VS Online & OData

Visual Studio ALM Rangers

Part 1: Hosted Service & OData – Foreword

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Microsoft grants you a license to this document under the terms of the Creative Commons Attribution 3.0 License. All other rights are reserved.

© 2013 Microsoft Corporation.

Microsoft, Active Directory, Excel, Internet Explorer, SQL Server, Visual Studio, and Windows are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Table of Contents

Foreword	4
Introduction	5
Reporting on TFS Service	6
Reporting Challenges	6
Reporting Model Features	6
Open Data Protocol (OData)	10
What is it?	10
Report using PowerPivot	14
Limitations of the Reference Solution (at this stage)	14
Building your own solution	16
Final thoughts	19
Reporting using the Reference Solution	20
Customize OData for trending	24
Why extend the TFS OData project?	24
Customization walk-through	24
Customized OData Service deployment to IIS walk-through	28
Customize Reports for trending	30
Understanding the data produced by the extended OData feed	30
Creating Trend Reports Walkthrough	33
Sample Reports built using the trending walkthrough	36
In Conclusion	38
Appendix	39
Sample Code - OData Customization	39
Setting up Alternative Credentials for VS Online	52

Foreword

The reporting functionality in Team Foundation Server acts as the center of the application lifecycle management world. In almost every customer demo, managers are ready to sign on the dotted line as soon as they see how a SharePoint dashboard can be configured to give insight in to the project status of the multiple projects going on in an organization. This has been one of the key selling points when managers are choosing amongst different ALM solutions – TFS has all the data AND it's reportable. Unfortunately, our existing reporting stack is not portable to the cloud. As such, we are looking at making a significant investment in building a new reporting stack end-to-end that can work for on-premises and our cloud customers. We have heard a lot of feedback on where our existing reporting scenario is lacking and we're looking at how we can leverage the latest Microsoft Business Intelligence stack to not only meet existing functionality but surpass it with a simpler and faster solution. Tabular models, PowerPivot and PowerView are all amazing new technologies that we hope will help us achieve these long-term goals. Work by the product team is already underway with the goal of delivering an in-box and on-service offering that can deliver on the reporting promise TFS offers. Until then, we hope this paper acts as a good reference for a stop gap solution and we encourage feedback on your experiences using it so we can ensure the feedback is considered and incorporated as the product team paves the way forward.

Justin Marks – Senior Program Manager, Cloud Dev Services

Introduction

Welcome to TFS Reporting Guide where we, the ALM Rangers, will take you on a fascinating journey to discover the exciting and powerful reporting features provided by Team Foundation Server.

This guide focuses on providing practical guidance and a reference solution to enable Visual Studio Online users to generate reports based on WIT data.

Intended audience

We expect the majority of our audience to be developers; however, the guide is for everyone, from the developer to the administrator, to explain how to create and use reports that target Visual Studio Online to manage software development projects hosted in the cloud. The guide assumes a basic knowledge of the Visual Studio Online and a reporting mindset – in other words, intermediate to advanced Visual Studio Online users.

What you'll need

The following prerequisites needed and referenced in this guide as *supported editions*:

- Visual Studio 2012 or Visual Studio 2013
- Visual Studio Online ¹ or on-premises Team Foundation Server
- Microsoft Office Excel

Visual Studio ALM Rangers

The Visual Studio ALM Rangers are a special group with members from the Visual Studio Product group, Microsoft Services, Microsoft Most Valuable Professionals (MVP) and Visual Studio Community Leads. Their mission is to provide out-of-band solutions to missing features and guidance. A growing Rangers Index is available [online](#)².

Contributors

[Gordon Beeming](#)³, Jeff Levinson, [Jim Szubryt](#)⁴, [Mattias Sköld](#)⁵ and [Ted Malone](#)⁶.

Using the sample source code, Errata and support

All source code in this guide is available for download via the [TFS Practical Reporting](#)⁷ home page where we also provide the latest corrections and updates. Download the file **eBook1-Package.exe**.

Additional ALM Rangers Resources

Understanding the ALM Rangers – <http://aka.ms/vsarunderstand>

Visual Studio ALM Ranger Solutions – <http://aka.ms/vsarsolutions>

¹ <http://www.visualstudio.com>

² <http://aka.ms/vsarindex>

³ http://blogs.msdn.com/b/willy-peter_schaub/archive/2013/07/04/introducing-the-visual-studio-alm-rangers-gordon-beeming.aspx

⁴ http://blogs.msdn.com/b/willy-peter_schaub/archive/2011/07/31/introducing-the-visual-studio-alm-rangers-jim-szubryt.aspx

⁵ http://blogs.msdn.com/b/willy-peter_schaub/archive/2011/03/28/introducing-the-visual-studio-alm-rangers-mattias-sk-246-ld.aspx

⁶ http://blogs.msdn.com/b/willy-peter_schaub/archive/2010/07/23/introducing-the-visual-studio-alm-rangers-ted-malone.aspx

⁷ <http://aka.ms/treasure55>

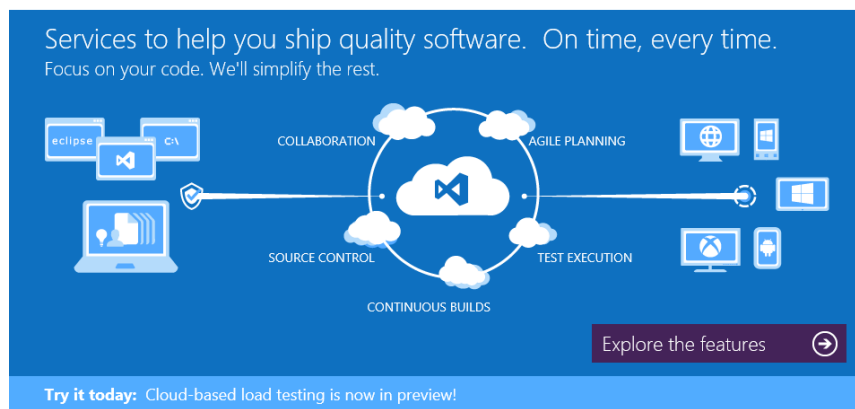
Reporting on TFS Service

Reporting Challenges

Team Foundation Server has always included a value proposition for reporting across work item tracking, source control, build, code quality and testing. The reporting architecture, which is based on the operational stores, the operational warehouse database and the analysis services cube, is part of the on-premises TFS infrastructure. It delivers insight and transparency—just what you need to react to the status and ever-changing conditions.

NOTE

Refer to the companion eBook **TFS Practical Reporting Guide - Part 2 Data Warehouse** for guidance and samples focused on the data warehouse.



When you commit to the [Visual Studio Online](http://tfs.visualstudio.com/)⁸ for your application lifecycle management (ALM) as [we did](http://aka.ms/vsarcloudhome)⁹ back in April 2011, you will not be able to enjoy the same wealth of reporting features (yet). Although Visual Studio Online will continue to evolve and add reporting features, we needed better transparency of our projects and an ability to spot the fires pro-actively.

The remainder of this guide will introduce you to the Tabular model, PowerPivot model, OData and walk you through a reference solution the ALM Rangers built for this guide and to use in their operational environment.

Reporting Model Features

This section provides comparison of the Tabular and the PowerPivot models to allow you to choose the correct model for your environment.

Scalability

Tabular models are server-based models and have the ability to scale well beyond anything that PowerPivot can offer. PowerPivot is limited to a 2GB size (for the Excel file) and only supports in-memory query mode versus *DirectQuery* mode.

In-memory query mode is exactly what it sounds like – the data must be loaded into memory before a query performed against it. This also means that when the underlying data changes, the in-memory data needs to refresh in order for the results to show up in a query.

DirectQuery mode bypasses these and other drawbacks by allowing the engine to process data directly from disk and supports better optimization and data security.

⁸ <http://tfs.visualstudio.com/>

⁹ <http://aka.ms/vsarcloudhome>

There are drawbacks to the *DirectQuery* mode. The biggest drawback is that only a single source of relational data can be used, once the model is set to use *DirectQuery*. Excel cannot be used as a client against a *DirectQuery* mode cube.

NOTE

For more information on **DirectMode** queries visit [here](#) ¹⁰.

Tabular models support data partitions, which allow the tabular model to be loaded and processed in chunks, unlike a PowerPivot model that processes data all at once with. Models can contain a large amount of data and yet process very quickly. An example of this might be storing sales data for twenty years. It is a safe bet that sales data from two years ago and longer does not change anymore, so the model can be set to only process recent sales data. By contrast, PowerPivot processes all historical sales data with each update of the underlying data.

Extensibility

PowerPivot supports extensibility that is provided through Excel but is limited. Tabular models allow developers to use SQL Server Management Studio, [Analysis Management Objects](#) ¹¹ (AMO), [ActiveX Data Objects Multidimensional Library](#) ¹² (ADOMD), [XML for Analysis](#) ¹³ (XMLA), [Analysis Services Deployment Wizard](#) ¹⁴, [AMO for PowerShell](#) ¹⁵ and [SQL Server Integration Services](#) ¹⁶ to import and manipulate data. This is a far greater toolbox than what is available with Excel PowerPivot models.

Security

PowerPivot models are secured by controlling access to the Excel file. That is the extent of the security model. Tabular models introduce the ability to do row level security and dynamic security through the use of Roles and role membership. Row level security is controlled by a filter, applied at query time, based on the user's role. Dynamic security allows security to be controlled based on the user who logged on to view the model data. Scenarios here involve aspects, which can change. For example, a manager being able to see information related to their direct reports who may change over time, or for accessing data based on the user's current geographic location.

NOTE

For more information, see this white paper: [Security the Tabular BI Semantic Model](#) ¹⁷.

Integrated Development Tools

PowerPivot lives in Excel and while you can put an Excel file in source control, managing versions or objects becomes somewhat difficult. In addition, the Excel file has to be managed outside of any professional development tools. Tabular models, on the other hand, are built with the SQL Server Data Tools within Visual Studio. That means that models can be controlled using Team Foundation Server and can participate in processes like automated builds using Team Build with automated deployment. Cathy Dumas has an excellent blog post on custom deployments with MSBuild tasks [here](#) ¹⁸.

¹⁰ [http://msdn.microsoft.com/en-us/library/hh230898\(v=SQL.110\).aspx](http://msdn.microsoft.com/en-us/library/hh230898(v=SQL.110).aspx)

¹¹ <http://msdn.microsoft.com/en-us/library/ms124924.aspx>

¹² <http://msdn.microsoft.com/en-us/library/ms123483.aspx>

¹³ <http://msdn.microsoft.com/en-us/library/ms186604.aspx>

¹⁴ <http://msdn.microsoft.com/en-us/library/ms176121.aspx>

¹⁵ <http://technet.microsoft.com/en-us/library/hh213141.aspx>

¹⁶ <http://blogs.msdn.com/b/cathyk/archive/2011/09/08/using-integration-services-with-tabular-models.aspx>

¹⁷ <http://msdn.microsoft.com/en-us/library/jj127437.aspx>

¹⁸ <http://blogs.msdn.com/b/cathyk/archive/2011/08/10/deploying-tabular-projects-using-a-custom-msbuild-task.aspx>

Choosing the right model

The following table summarizes the key considerations when you choose the right model for the right situation.

Tabular Model	PowerPivot
Scalability	
<ol style="list-style-type: none"> 1. No limit on maximum size 2. Supports DirectQuery mode 3. Single source of relational database 4. Supports Partitions 5. Usually suitable for large organizations. 6. Requires SQL Analysis Services running on a server with a lot of RAM (>8 GB) 	<ol style="list-style-type: none"> 1. Limited to 2GB (the size of Excel file) Supports only in-memory query mode 2. Multiple data sources 3. Does not support partitions 4. Usually suitable for small organizations
Extensibility	
Allow developers to use SQL Server Development Tools, Analysis Management Objects ¹⁹ (AMO), ActiveX Data Objects Multidimensional Library ²⁰ (ADOMD), XML for Analysis ²¹ (XMLA), Analysis Services Deployment Wizard ²² , AMO for PowerShell ²³ and SQL Server Integration Services ²⁴ to import and manipulate data	Only limited support for development – as much as can be done in Microsoft Excel
Security	
<ol style="list-style-type: none"> 1. Supports Row Level and dynamic security 2. Support for User roles, which define access to objects and data 	<ol style="list-style-type: none"> 1. Very limited security support. 2. Typical security is restricted access to file
Development Tools	
Usable with Source Control repository such as Team Foundation Server and built using Team Build	Versioning can be difficult
Conversion <->	
Cannot be converted to a PowerPivot model (though it can be downloaded into Excel)	Can convert to a Tabular model

Table 1 – Tabular Models versus PowerPivot

Scenarios that support a PowerPivot model

- The model maintained by personas who are not comfortable with Visual Studio.
- Not affected by the scalability, security, extensibility or integrated development tools feature restrictions.
- Models created for individual teams.
- Limited or no support for SQL Server Analysis Services server.
- Limited or no support for SharePoint environment configured for PowerPivot.

NOTE

To share PowerPivot models, they either have to be uploaded to SharePoint (and SharePoint has to be configured correctly) or the file has to be passed around – 2GB is probably a bit large to e-mail!

¹⁹ <http://msdn.microsoft.com/en-us/library/ms124924.aspx>

²⁰ <http://msdn.microsoft.com/en-us/library/ms123483.aspx>

²¹ <http://msdn.microsoft.com/en-us/library/ms186604.aspx>

²² <http://msdn.microsoft.com/en-us/library/ms176121.aspx>

²³ <http://technet.microsoft.com/en-us/library/hh213141.aspx>

²⁴ <http://blogs.msdn.com/b/cathyk/archive/2011/09/08/using-integration-services-with-tabular-models.aspx>

Scenarios that support a Tabular model

- Typical for large to enterprise scale environments.
- Need to access and share the data through different mediums.
- Support for SQL Server Analysis Services server.
- Support for SharePoint environment configured for PowerPivot.

Prerequisites once you have made a choice

Prerequisites to create a Tabular model

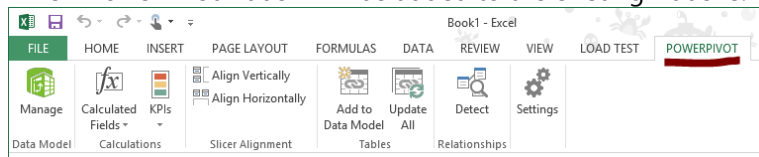
Setting up to build a tabular model is relatively straightforward and this section does not add to the components of various front ends you can use to report on the data. For the purposes of this document, reporting on the model can be accomplished using the data analysis features of Microsoft Excel.

- There must be a source of data for the data model. Because this document focuses on Team Foundation Server, the source of the data should be one of the following: the TFS Service OData feed, an on-premises TFS Warehouse database or a collection relational database (i.e. TFS_<collection name>). Most of the focus will be on the OData feed and the relational database.
- SQL Server 2012 Analysis Services in the Tabular mode must be installed. Instructions for this installation can be found [here](#)²⁵.
- For creating, publishing and reporting on tabular models, these are all of the components that a developer requires.

Prerequisites to create a PowerPivot model

In terms of the tools required, creating a PowerPivot model is somewhat simpler than a tabular model.

- As with a tabular model, a data source is required.
- Beyond that, Microsoft Excel 2010 or above is required. If the version is Excel 2010, you have to install an add-in. This add-in can be downloaded from [here](#)²⁶. If you use Excel 2013, the PowerPivot add-in is installed automatically. However, it is not enabled by default.
 - To enable the PowerPivot add-in:
 - Open a blank workbook in Excel 2013.
 - Select **File, Options**.
 - Select the **Add-Ins** tab.
 - Select **Com Add-ins** and click **Go**.
 - Select Microsoft Office PowerPivot for Excel 2013 and click **OK**.
 - A new PowerPivot ribbon will be added to the existing ribbons.



REVIEW

We have explored the Tabular model the PowerPivot model. In the next section, we'll introduce OData and walk you through building a PowerPivot model against the Visual Studio Online environment.

²⁵ <http://msdn.microsoft.com/en-us/library/hh231722.aspx>

²⁶ <http://www.microsoft.com/en-us/download/details.aspx?id=29074>

Open Data Protocol (OData)

What is it?

The **Open Data Protocol (OData)** is a Web protocol for querying and updating data. It provides a way to unlock your data and free it from silos that exist in applications today. OData does this by applying and building upon Web technologies such as [HTTP](#)²⁷, [Atom Publishing Protocol](#)²⁸ (AtomPub) and [JSON](#)²⁹ to provide access to information from a variety of applications, services, and stores.

The protocol emerged from experiences implementing AtomPub clients and servers in a variety of products over the past several years. OData helps us expose and access information from a variety of sources including, but not limited to, relational databases, file systems, content management systems and traditional websites.

Why use it?

Currently, common ways of designing web Application Programming Interfaces (APIs) have emerged on the web. Many of them apply REST or CQRS principles and use JSON or XML to expose data.

Frequently asked questions:

- What does it mean to do a POST to a particular resource?
- What is required to identify related resources?
- Is it possible to apply filters to retrieve specific resources?

These and many more questions must be covered in developer documentation for each API.

OData solves this problem by **providing a uniform way to expose, structure, query and manipulate data** using REST practices and JSON or ATOM syntax to describe the payload.

OData also **provides a uniform way to represent metadata** about the data, allowing computers to know more about the type system, relationships and structure of the data.

By using OData, you tap into a large ecosystem that can target your dataset without having to learn a lot to get going with your dataset.

²⁷ <http://www.w3.org/Protocols/>

²⁸ <http://www.ietf.org/rfc/rfc4287.txt>

²⁹ <http://json.org/>

Setting up TfsOData

We are using the OData solution shared by [Brian Keller](#), which can be downloaded from [here](#) ³⁰.

NOTE

When targeting <https://tfs.visualstudio.com>, OData is enabled by default.
If you have no plans to customize the OData feed this section is for your information only.
 See [Bringing OData to Team Foundation Service](#) ³¹ for more details.

NOTE

When using Brian Keller's ALM VM TFS 2012 (<http://aka.ms/almvms>) you need to ensure that it is a fresh VM and that you have turned on time synchronization. Otherwise, you will not be able to connect to the hosted service OData feed because of a certificate issue. You will also need to enable internet access.

Prerequisite(s)	
Build the OData Service	<ul style="list-style-type: none"> Visual Studio 2012 with Update 1 ³² Windows Azure SDK for .NET (optional) ³³ Necessary only if building for Windows Azure environment. See the sections specific to IIS Express and IIS Server for details on how to build without installing the SDK.
Deploy the OData Service	<ul style="list-style-type: none"> Microsoft .NET Framework 4.5 ³⁴ Team Foundation Server 2012 Update 1 Object Model Installer (included as part of ODataTFS.Web project) IIS 7 or IIS 8 with ASP.NET installed – this means Windows 7 and up, Windows Server 2008 and up SSL certificate to secure HTTPS endpoint (if deploying publically) Windows Azure account (optional)
Third-party	<ul style="list-style-type: none"> The OData Service uses the WCF Data Services Toolkit ³⁵ to help with common tasks associated with building an OData endpoint with WCF Data Services. This dependency is located in the code\References folder. The ODataTFS.Tests project, which houses the included unit tests, depends on a mocking library called Moq ³⁶. Version 4.0 of this dependency is included as part of the OData Service download.

Table 2 – Prerequisites

NOTE

If you want to run or debug using the Azure Emulator, you will also need to install the IIS feature with ASP.NET on your development machine.

Deploying the OData Service to IIS Server

NOTE

This walkthrough was verified with Windows Server 2008 R2 Standard SP1, Windows Server 2012 Standard and Windows Server 2012 R2. You might need to adapt the steps to match up with your environment if there are any differences.

Step	Instructions
1 Install Web Server (IIS) role <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Install the Web Server (IIS) role and include the ASP.NET role service. Add in the dependent role services when prompted. In Server 2012, select Application Development, ASP.NET 4.5. Include the Dynamic Content Compression option.

³⁰ <http://blogs.msdn.com/b/briankel/archive/2013/01/07/odata-service-for-team-foundation-server-v2.aspx>

³¹ <http://blogs.msdn.com/b/briankel/archive/2013/01/24/bringing-odata-to-team-foundation-service.aspx>

³² <http://www.microsoft.com/visualstudio/eng/products/visual-studio-overview>



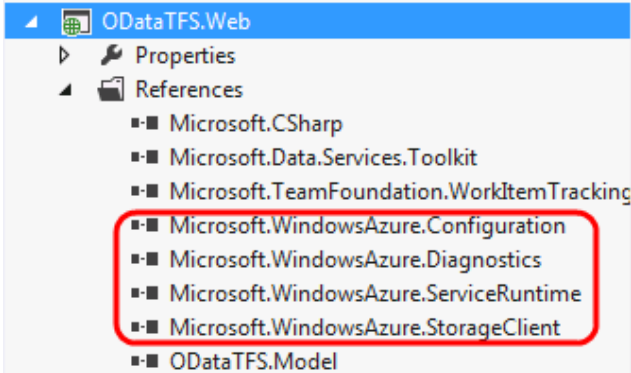

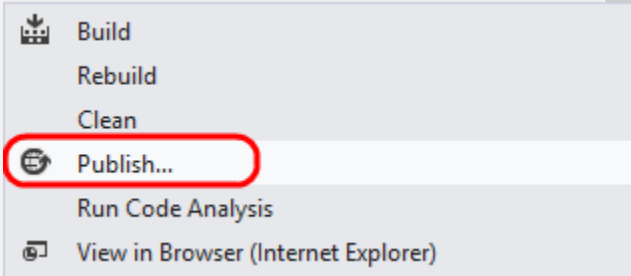


³³ <http://www.windowsazure.com/en-us/develop/downloads/>

³⁴ <http://www.microsoft.com/en-us/download/details.aspx?id=30653>

³⁵ <http://wcfdatakit.codeplex.com/>

³⁶ <http://code.google.com/p/moq/downloads/list>

Part 1: Hosted Service & OData – Open Data Protocol (OData)

Step	Instructions
2 Install .NET Framework 4.5  - Done	<ul style="list-style-type: none"> This is a pre-requisite for the Team Foundation Server client object model. On Server 2012, this prerequisite is part of ASP.NET 4.5. <p>NOTE If you install the .NET Framework before IIS, you'll need to use the ASP.NET IIS Registration tool that comes with the .NET Framework.</p> <ul style="list-style-type: none"> It is recommended to check for Windows Updates after you install the .NET Framework.
3 Build OData to Server  - Done	<ul style="list-style-type: none"> In Visual Studio, open the odatatfs.sln solution file. If you already have the Windows Azure SDK installed, go ahead and skip to the next section. If you want to avoid installing the Windows Azure SDK and will ultimately be deploying to IIS, you can simply remove the Microsoft.WindowsAzure.* references from the Web project.  <ul style="list-style-type: none"> Select Build Rebuild Solution to ensure everything is up-to-date.
4 Publish OData  - Done	<ul style="list-style-type: none"> Right-click on the ODataTFS.Web project in Solution Explorer and select the Publish option. You can use any publish method that you want, but the remaining instructions in the section are based on the File System method. 
5 Copy published website to server  - Done	<ul style="list-style-type: none"> Run the InstallTFSObjectModel.cmd script from the \bin\SetupFiles folder of the published website. This will install the Team Foundation Server client object model assemblies to the GAC. Run the SetupIIS.cmd script from the \bin\SetupFiles folder of the published website to create the Team Foundation Server client object model cache folder and set permissions for the IIS_IUSRS group to the cache folder.
6 Setup Security  - Done	<ul style="list-style-type: none"> Ensure that the IIS_IUSRS group has full permissions to that location also, via the command line or by going to the folder properties and security tab in a Windows Explorer window. Create and configure a new Web Site in IIS. Most of the site details such as name, port, and host name are specific to your environment. Make sure that you use HTTPS and your SSL certificate for the binding. Preferably, use the physical path c:\inetpub\wwwroot\TfsOdataAlmRangers, in place of c:\odata.

Part 1: Hosted Service & OData – Open Data Protocol (OData)

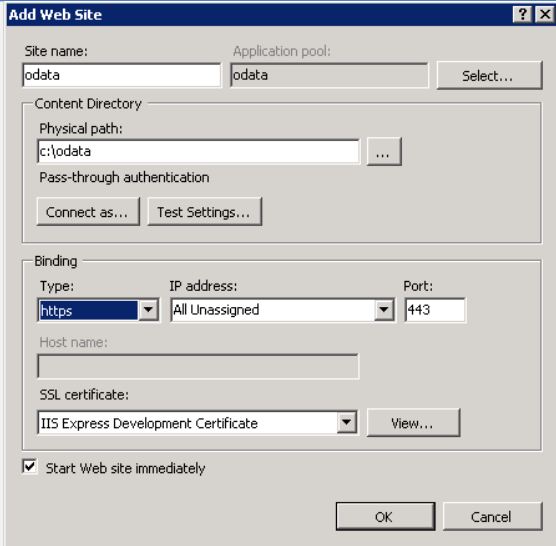
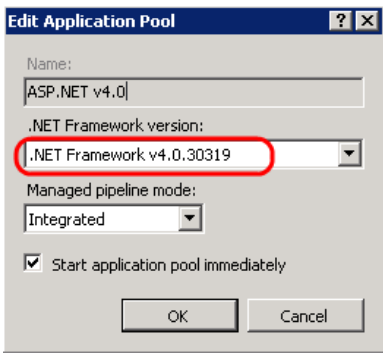
Step	Instructions
	<div data-bbox="354 197 906 739">  </div> <div data-bbox="354 739 1474 877"> <p>NOTE For testing purposes, you can use a self-signed certificate. One way to do this using IIS Manager is to navigate to the machine node, open Server Certificates, and select Create Self-Signed Certificate. To learn more about SSL and IIS, please see http://learn.iis.net/page.aspx/144/how-to-set-up-ssl-on-iis-7/.</p> </div>
<p>7</p> <p>Configure .NET Framework</p> <p>☐ - Done</p>	<p>Once the website has been created, we also need to make sure that the associated application pool is configured to use .NET Framework 4. It should be okay if the minor version is greater than that showed in the screenshot.</p> <div data-bbox="354 982 734 1331">  </div>
<p>8</p> <p>Ready!</p> <p>☐ - Done</p>	<p>At this point, the OData Service is ready to go.</p>

Table 3 – Walkthrough: Deploying the OData Service to IIS Server

Report using PowerPivot

The Hosted TFS Service does not provide a mechanism out of the box to provide reports on data. Because of this, the ALM Rangers wanted to provide at least a base level of capability for reporting on various pieces of information. To do this, the reference solution relies on the OData feed built on top of the Hosted TFS Solution (for detailed information on the OData feed, see this link [here](#)³⁷).

The goal of this solution is to allow teams to extract data from the Hosted TFS Service so that they can create their own reports using the familiar interface of Microsoft Excel. The current reference solution uses Microsoft PowerPivot to enable this capability, which then allows users to create Pivot Charts or Power View reports either on the desktop or by using Microsoft Excel 2013.

Important Information to Prepare for Walkthroughs

NOTE

Refer to **ALMRangersReferenceSolution.xlsx** which is a completed reference sample workbook done by completing this walkthrough.

We are using @TOKENS@ in place of real-world data, both to protect IP and to make this walkthrough flexible. Before you proceed, please invest a few minutes to gather and record the necessary information to replace the @TOKENS@ with.

Information	ALM Rangers Environment Example	TOKEN used during the walkthroughs	YOUR VALUE
Hosted Service URL	almrangers.visualstudio.com	@SERVICE@	
Team Project Collection	DefaultCollection	@TPC@	
User ID with access to @SERVICE@		@USER@	
User password with access to @SERVICE@	-	@PASSWORD@	
An area path you wish to report against	VisualStudio.ALM	@AREAPATH@	
A team project you wish to report against	VisualStudio.ALM	@TP@	
A team you wish to report against, based on an area path	VisualStudio.ALM\vsarUnitTestFx	@TEAM@	
Excel file name	AlmRangersReferenceSolution.xlsx	@FILE@	

Table 4 – Reference Solution Information

Limitations of the Reference Solution (at this stage)

The OData feed currently has the following limitations:

- Does not expose a majority of the data contained within TFS.
- The following is **not** available for reporting:
 - Trend data on the work items table
 - Team names, making it necessary to correlate area and iteration paths with teams.

³⁷ <https://tfsodata.visualstudio.com/>

Exploring the reference solution

The entirety of the solution includes the Work Items, Projects and Builds tables as shown in Figure 1.

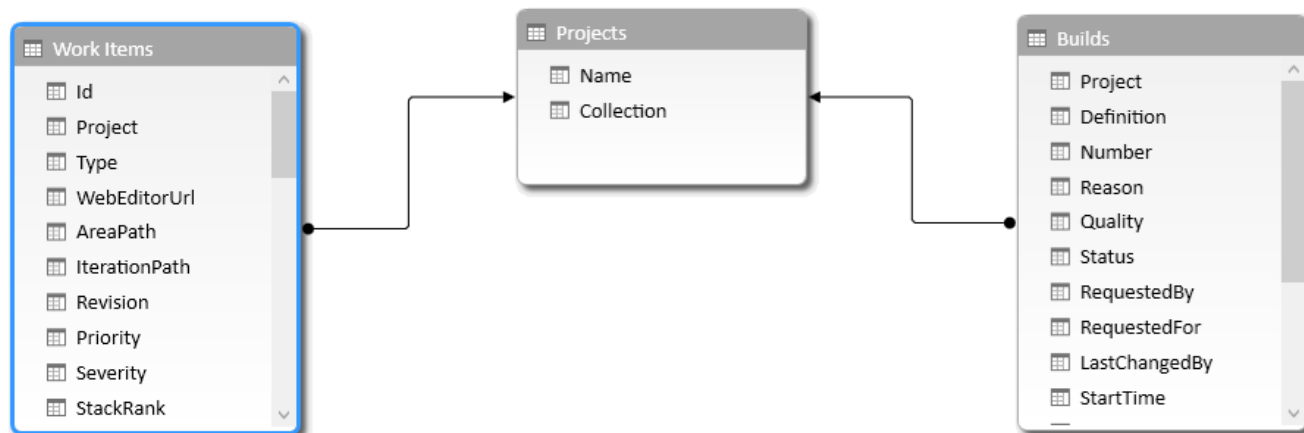


Figure 1 - Reference Model

Each dimension is joined by the project name. Both the Work Items dimension and the Builds dimension contain measures to provide information on the current state of information.

Work Items dimension

This dimension includes all information about the current state of the work items and allows you to do aggregations of work. Common uses of this might be reporting how much work is assigned to a user, how many of each work items are assigned, how much work is left to do in the iteration, etc. The measures available against this dimension are the following:

- Work Item Count
- Revision Count
- Completed Work
- Remaining Work

There are any number of measures a team could add, based on the available data, but most likely these will be the four base measures that teams will need.

Builds dimension

The Builds dimension includes all of the information about completed builds. This is the standard information from the OData feed with one exception – there is an added column, which calculates the build duration in order to demonstrate how to add additional columns. The measures available against this dimension are the following:

- Build Count
- Build Duration Total

Projects dimension

Because this solution is designed to work with the off the hosted service, everything available to a team is limited to a single collection. Each account (such as accountname.visualstudio.com) consists of a collection called "DefaultCollection". Because of that, this dimension contains the project names within the default collection, which is a list of all team projects.

Building your own solution

The provided reference solution offers a starting point for developing your own reports but other information is available from the hosted service. The OData feed page located [here](#) contains other information available to a consumer. PowerPivot and Tabular Model solutions will become a larger part of reporting against Team Foundation Server in the future because they allow flexibility to add measures and additional fields at will without having to build data warehouse adapters. In addition, the reference solution allows users to pull out data which contain hours and minutes rather than just days which means that users can report in as granular a fashion as is called for.

For this reason, this section walks through how to create a PowerPivot model from an OData feed. This will give users the ability to build their own solutions on top of the available feeds, but these capabilities translate into being able to pull and use data in PowerPivot with any data source.

Connecting to the TFS OData Feed

To begin with, data needs to be pulled into PowerPivot in order to build the data model. Before beginning this walkthrough, be sure to read the instructions [here](#)³⁸. Pay particular attention to the section titled “Team Foundation Service Authentication” and set up the authentication as described. Once this has been set up, you’re ready to begin.

Step	Instructions
1 Get Started ☐ - Done	<ul style="list-style-type: none"> Open Microsoft Excel and select a blank workbook.
2 Configure OData ☐ - Done	<ul style="list-style-type: none"> Select the PowerPivot tab on the ribbon and click Manage. Click Click Get External Data and then From Data Service. In the Table Import Wizard dialog, enter the Friendly name as Builds. Refer to the TFS OData feeds page and notice the first URL next to Builds For reference, this URL is https://tfsodata.visualstudio.com/DefaultCollection/Builds. Copy this URL and paste it into the Data Feed Url box in Excel.
3 Setup Security ☐ - Done	<ul style="list-style-type: none"> Click the Advanced Settings button and make the following changes: <ul style="list-style-type: none"> Integrated Security should be set to Basic. User ID (@USER@) should be set in the form of [account]\[username] where account is the first part of the Hosted TFS @URL@ (for example in the URL test.visualstudio.com, “test” would be the account) and the username is the username set when configuring the alternate authentication settings as a pre-requisite to this walkthrough. Password (@PASSWORD@) is the password configured for the alternate authentication settings. <div> NOTE The procedure of setting alternative credentials is described at http://tfodata.visualstudio.com and on page 52. </div>
4 Test Connection ☐ - Done	<ul style="list-style-type: none"> Click Test Connection. After the connection succeeds, click OK to close the Advanced Settings dialog box.
5 Pull Data	<ul style="list-style-type: none"> In the Table Import Wizard dialog box, click Next and then click Finish. At this point the data will be pulled from the OData feed into PowerPivot. Depending on the amount of data this may take from 30 seconds to several minutes.

³⁸ <https://tfsodata.visualstudio.com/>

Part 1: Hosted Service & OData – Report using PowerPivot

Step	Instructions
<input type="checkbox"/> - Done	
6 Finish up <input type="checkbox"/> - Done	<ul style="list-style-type: none"> After you see the green Success check, click Close. A single table now resides in PowerPivot. To add other OData feeds, simply repeat this process but enter the URL of the OData feed to be imported.

Table 5 – Connect to TFS OData Feed

Adding additional tables and connecting them

In this part of this walkthrough, you'll add two additional tables (Projects and Work Items) to the model and connect them to each other).

Step	Instructions
1 Add Projects Table <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Repeat steps 1 – 10 in Connecting to the TFS OData feed section, except this time use the following Url: https://tfsodata.visualstudio.com/DefaultCollection/Projects.
2 Add WorkItems Table <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Repeat steps 1 – 10 in Connecting to the TFS OData feed section, except this time use the following Url: https://tfsodata.visualstudio.com/DefaultCollection/WorkItems.
3 Cleanup Workbook <input type="checkbox"/> - Done	<p>At this point there are three tables in the model.</p> <ul style="list-style-type: none"> Rename the WorkItems table to Work Items. <ul style="list-style-type: none"> Right-click the worksheet that says "WorkItems" and select Rename. Add a space between "Work" and "Items" and press ENTER. Switch to the Diagram view by clicking the icon in the far lower right corner of PowerPivot window (right next to the drag handle in the lower right corner). From the Projects table, drag the Name field to the Project field in the Builds table. From the Projects table, drag the Name field to the Project field in the Work Items table.
4 Save the workbook <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Save the Excel workbook as file @FILE@.

Table 6 – Adding tables

Modifying Columns and Measures

Now that data resides within the PowerPivot model you can manipulate it: . rename, delete, or add columns or add measures. This walkthrough will demonstrate how the reference model was created.

As a note, it usually helps to make the column names user-friendly, which means spelling out words, putting spaces between words, etc. These changes should be made before doing anything else. Otherwise, you'll have to correct formulas because renaming a column does not automatically fix all formulas associated with the column.

This walkthrough will require entering a few formulas. These are not typical Excel formulas. They are Data Analysis Expression (DAX for short) formulas. For more information on DAX, please see the MSDN reference located [here](#) ³⁹.

For this walkthrough, you are only going to rename two columns.

Step	Instructions
1 Rename StartTime Column ☐ - Done	<ul style="list-style-type: none"> Go back to the grid view and switch to the builds data. Select the StartTime column by clicking the header for this column. Right-click the column and select Rename. Put a space in between Start and Time and press ENTER.
2 Rename FinishTime Column ☐ - Done	<ul style="list-style-type: none"> Repeat the first step for the FinishTime column, adding a space between Finish and Time.
3 Add Calculated Columns ☐ - Done	<p>A calculated column is any column that does not exist in the source data.</p> <ul style="list-style-type: none"> Scroll all the way to the right of the grid, to the last column, which says Add Column. Click in the first empty cell below the Add Column header, enter the following formula and press ENTER: = 24 * 60 * ([Finish Time] – [Start Time]) <div> <div>NOTE</div> <div>Do not copy and paste this formula because Word formatting invalidates the formula.</div> </div> <ul style="list-style-type: none"> This formula converts the difference between the finish time and start time to minutes and formats it as a number. On the Home tab of the ribbon in the Formatting section, change the Format from General to Decimal Number. Right-click the CalculatedColumn1 (the default name given to the new column) and rename it to Build Duration.
4 Value becomes a measure ☐ - Done	<p>Once this is done, the value needs to become a measure so that it can be aggregated. Data in a column alone is simply treated as a field (although some technologies like Power View offer an automatic aggregation function because it is a number, not all tools allow this). By turning this into a measure, when a user adds a team project to the pivot table and then adds the Build Duration Total measure (which will be added next), PowerPivot will automatically sum the build duration for all builds associated with the selected team project.</p> <ul style="list-style-type: none"> Below the grid of data in PowerPivot is a blank grid. This is called the Measures grid. Select the top cell of the left most column Please note that measures can be placed in any cell in the measures grid, but as a convention, keep it at this location because it is easily visible In the formula bar at the top, enter the following formula: Build Duration Total:=CALCULATE(SUM([Build Duration])) <p>The text before the “:=” is the name of the measure. When you create a measure, it’s usually critical that you use the CALCULATE function. The reason for this is that CALCULATE ensures that when PowerPivot is calculating the measure, it takes into account all filters currently in place on the table at the time. If CALCULATE is not used, the value will be the same regardless of how the user has filtered the data and this is not the expected behavior.</p>
5 Add Measures ☐ - Done	<ul style="list-style-type: none"> The last step is to add the measures to the Work Items table. Add the following measures to the measures grid in the Work Items table: Work Item Count:=CALCULATE(COUNTROWS("Work Items")) Revision Count:=CALCULATE(sum([Revision])) Completed Work:=CALCULATE(SUM([CompletedWork])) Remaining Work:=CALCULATE(SUM([RemainingWork]))

Table 7 – Modifying Columns and Measures

Final thoughts

This completes the creation of the reference model. Everything will be identical when doing this against another project except for the account\username and password that is entered. In addition, the model does not have to be re-created to be hooked up to other projects – the reference model can be used as-is.

To hook the model up to another hosted collection, do this:

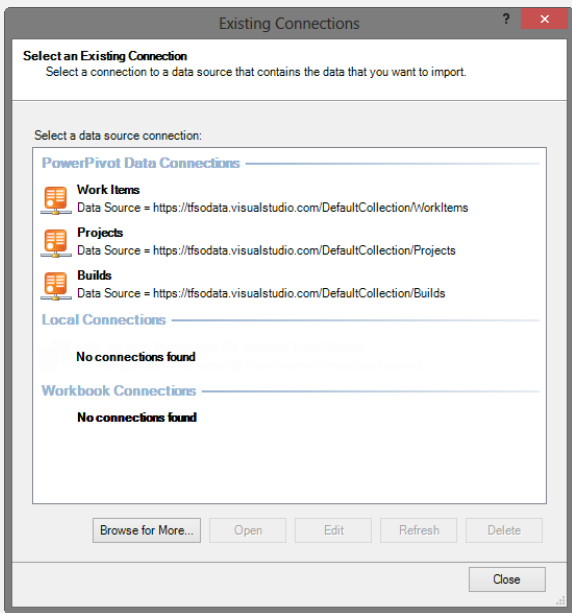
Step	Instructions
1 Open Excel ☐ - Done	<ul style="list-style-type: none"> Open the @EXCEL@ file.
2 Open Connections ☐ - Done	<ul style="list-style-type: none"> Select the PowerPivot tab from the ribbon and click Manage. On the Home tab click Existing Connections. 
3 Update Connections ☐ - Done	<ul style="list-style-type: none"> For each of the existing connections, do this: <ul style="list-style-type: none"> Select the connection. Click Edit. This will display the Table Import Wizard dialog <ul style="list-style-type: none"> Click Advanced. Change the username and password fields, click OK and then click Save. Once this is done for all three connections, the model will be hooked up to the new data source. At this point, simply refresh the data Click Refresh All from the Home tab.
4 Save ☐ - Done	<ul style="list-style-type: none"> Save the @EXCEL@ file.

Table 8 – Hookup to different host

Reporting using the Reference Solution

There are three ways to create reports based on the PowerPivot data model: using SharePoint to host the PowerPivot model (which allows for Excel Services or Power View), Pivot tables within Excel and Power View within Excel (Office 2013 only at this point).

This section describes using Pivot tables within **Excel** and **Power View** and will demonstrate the basics of creating reports from the hosted service.

Pivot Tables

Pivot tables are a standard Excel mechanism for creating reports based on tabular data and are the basis of Excel Services reports so this information is applicable to many scenarios.

This walkthrough assumes that you are using Excel 2013 with the PowerPivot add-on enabled or Excel 2010 with the PowerPivot add-on installed (instructions on enabling PowerPivot in Excel 2013 and downloading the Excel 2010 add-on can be found [here](#)).

Step	Instructions
1 Prepare Workbook ☐ - Done	<ul style="list-style-type: none"> Open the @EXCEL@ file.
2 Manage PowerPivot ☐ - Done	<ul style="list-style-type: none"> Click the PowerPivot tab on the ribbon and click the Manage icon (first icon on the left). Once the model is displayed, click the PivotTable icon on the Home tab. Select Existing Worksheet and click OK on the Insert Pivot dialog box. Expand the Work Items dimension in the Pivot Table list. All Measures are at displayed at the bottom of the dimension in which the measure exists. Select the check box next to Work Item Count (or drag the Work Item Count measure to the Values area). This will display a count of work items in Pivot table area in Sheet 1. Expand the Projects dimension. Drag the Name field to the Filters box. In cell B1, click the drop-down list and select the relevent @AREAPATH@. In the Work Items dimension, drag the Type field to the Rows area.
3 Create Pivot Chart ☐ - Done	<ul style="list-style-type: none"> Click the Analyze tab on the ribbon. Note that this tab will only show up if a cell within the Pivot table area is selected. Select the Pivot Chart icon. Change the Chart Type to Pie and click OK. The results should look similar to that in Figure 2, which has been cleaned up and formatted for readability.

Part 1: Hosted Service & OData – Report using PowerPivot

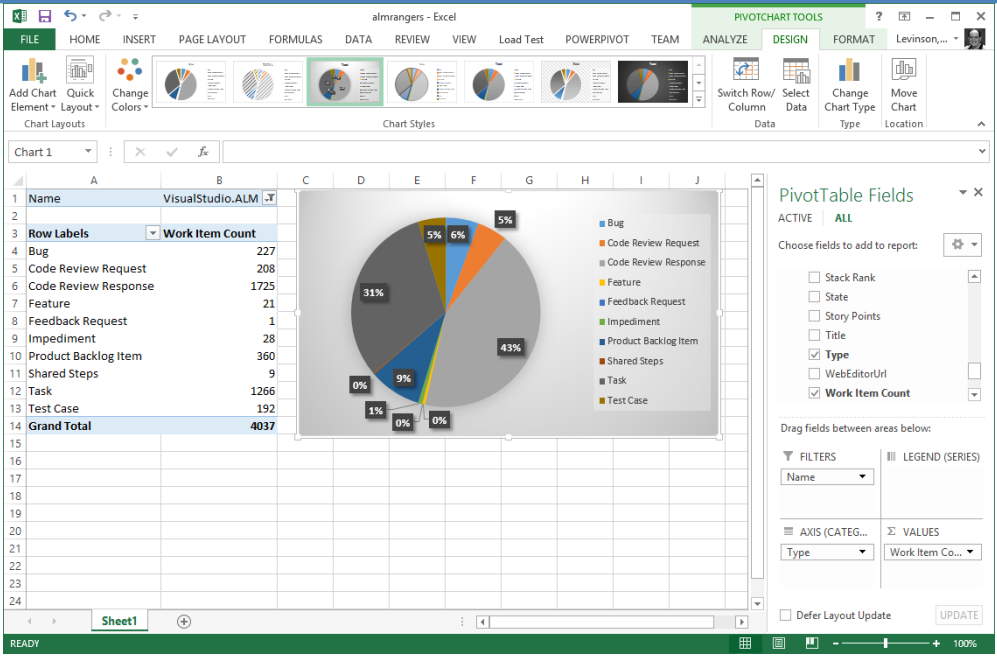
Step	Instructions
	 <p>The screenshot shows an Excel workbook with a PivotTable and a PivotChart. The PivotTable is located in the range A1:B14, with 'Name' in column A and 'Work Item Count' in column B. The PivotChart is a pie chart located in the range C1:E14, showing the distribution of work item types. The PivotTable Fields task pane is visible on the right, showing the 'Type' field in the 'VALUES' area and 'Work Item Count' in the 'FILTERS' area.</p>
4 DONE ☐ - Done	Once this is completed it can be placed on a SharePoint site and used in Excel Services. This process is the same as reporting off of the normal on premises TFS cube.

Table 9 – Walkthrough: PivotTables

Power View

Power View is an interactive reporting mechanism that Microsoft created and released with SQL Server 2012. A front-end version of this tool is built into Excel 2013. It is also available in SharePoint 2010 and SharePoint 2013. This walkthrough assumes that you use Excel 2013 and have enabled the Power View add-on.

This walkthrough [here](http://www.mssqlinsider.com/2013/02/how-to-enable-powerpivot-and-power-view-in-excel-2013/)⁴⁰ provides the steps to enable the Power View add-in and demonstrates the interactivity of the Power View tool.

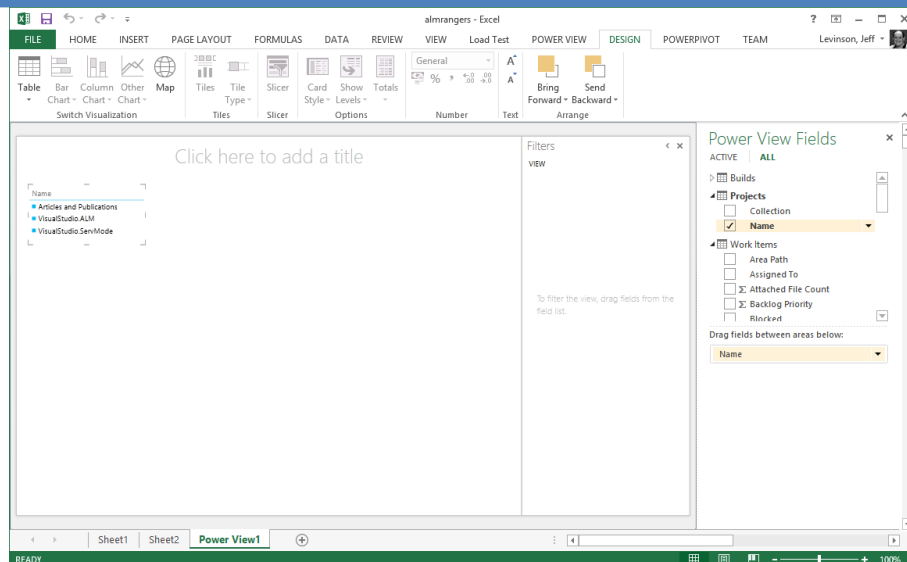
Step	Instructions
1 Prepare Workbook ☐ - Done	<ul style="list-style-type: none"> Open the @EXCEL@ file.
2 Add Power View Tab ☐ - Done	<ul style="list-style-type: none"> Select the Insert tab from the ribbon and click Power View. This will add a new tab to the workbook called "Power View1."
3 Create Table ☐ - Done	<ul style="list-style-type: none"> From the Power View field list on the right, expand the Projects dimension and select the check box next to Name. This will create a table on the chart with the list of projects. From the Design tab click the Slicer icon. This will add a dot next to each team project and make it click-able (as shown below);

⁴⁰ <http://www.mssqlinsider.com/2013/02/how-to-enable-powerpivot-and-power-view-in-excel-2013/>

Part 1: Hosted Service & OData – Report using PowerPivot

Step

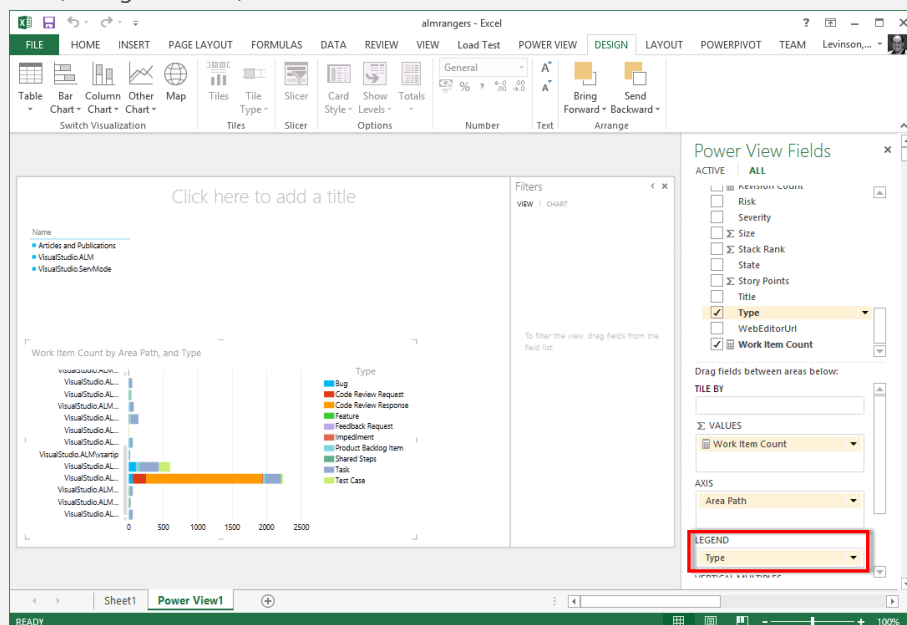
Instructions



4
Manage Power
View Surface

□ - Done

- Click any whitespace area on the Power View surface. This clears the current selection so the next selection starts a new table..
- In the **Power View** field list, check the **Area Path** in the Work Items dimension.
- This will create a new table with the list of area paths.
- Next, select the check box next to the **Work Item Count** measure under **Work Items**.
- Click the **Column Chart** button on the **Design** tab.
- This will convert the list to a bar chart.
- Drag the Type field from the Work Items dimension to the legend box in the Power View Fields list (see figure below):



5
Refine Power
View Surface

□ - Done

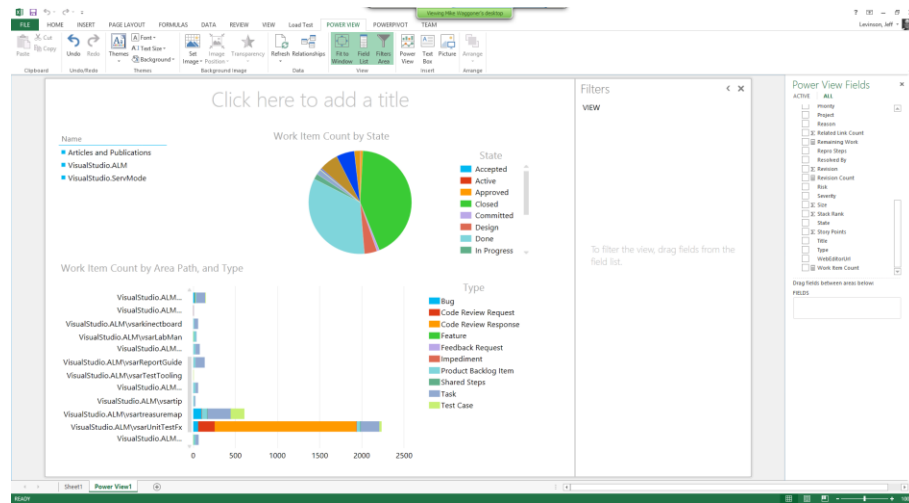
- Click in the whitespace anywhere on the design surface. This clears the current selection so the next selection starts a new table..
- Check the **State** field under **Work Items**.
- This will create a new table again with the list of states.
- Check the **Work Item Count** measure under **Work Items**.
- Click the **Other Chart** from the **Design** menu and select **Pie**.
- (Optional) Add a chart title by clicking **Click here to add a title**.

Part 1: Hosted Service & OData – Report using PowerPivot

Step

Instructions

- The final view should look like this:



6
Explore more
"goodness"
□ - Done

- Click the one of the team projects (@TP@) on the graph and note that the other two charts are updated to reflect the fact that the report is now filtered by the area path.
- Next, scroll down in the list of areas to the team (@TEAM@) area and on the bar chart, click the part of the chart that shows the Code Review Requested work item type. Notice that the report is further filtered again.
- Next, hover you mouse over the Closed area of the pie chart.
- The results of all of this should be similar to this:

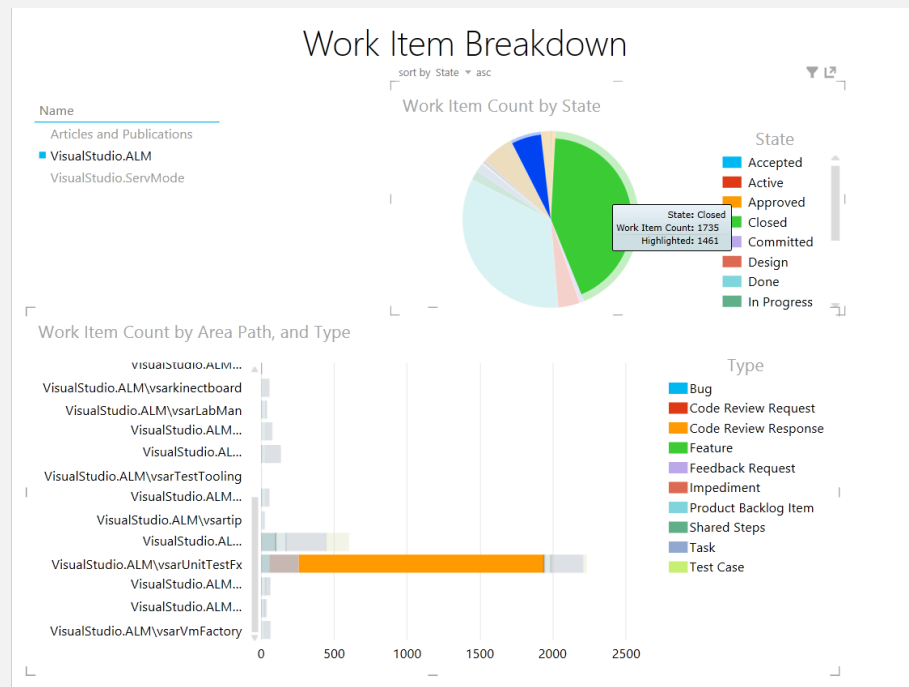


Table 10 – Walkthrough: PowerView

Customize OData for trending

NOTE

You can download the **latest** version of the OData solution shared by [Brian Keller](#) from [here](#) ⁴¹. The companion **eBook1-Package.zip** contains the original v2.3 version of OData and the customized version of OData as documented herein.

Why extend the TFS OData project?

There are several reasons why you might want to extend the TFS OData feed, for example:

- **Custom fields in your work items**
The TFS OData project only returns a core set of fields. To access to your custom fields, add them by extending the OData project.
- **Missing data**
The TFS OData project provides access to lot, but not all of the data handled in TFS. To access things like `WorkItemHistory` or test related data, you can implement them as OData extensions.
- **Implement Query options**
The TFS OData project only supports a limited set of Query options, like `$filter`. To implement a specific query option, such as `$expand`, extend the OData project.

In this walkthrough, we'll add a feed with all revisions for all work items in a team project. This is very useful if you plan to do reporting on trends or historical data, like we are planning to do as part of our reference solution.

Customization walk-through

NOTE

You can find the complete code samples in the appendix on page 39 and as part of the reference solution.

Decide on the URI

For our reference solution, we need to get access to all revisions of all work items in a team project. We also need to get access to a pre calculated Fact table for the work item revisions. For that purpose we decided on the following URIs:

- [http://localhost/DefaultCollection/Projects\('projectName'\)/WorkItemHistory](http://localhost/DefaultCollection/Projects('projectName')/WorkItemHistory)
- [http://localhost/DefaultCollection/Projects\('projectName'\)/WorkItemFact](http://localhost/DefaultCollection/Projects('projectName')/WorkItemFact)

It provides direct access to the **WorkItemHistory** and the precalculated **WorkItemFact** table for a selected team project.

Adding WorkItemHistory and WorkItemFacts to Project

As the request is executed, the OData project will instantiate a Project entity and look for a member `WorkItemHistory` to return. What we need to do is then add that member to the Project entity.

The data we want to return is a flat list of all revisions.

- Navigate to the TFS.OData.Model project, open the Entities folder and open the Project.cs file.
- Add the following member to the Project class.

```
[ForeignKeyProperty]
public IEnumerable<WorkItemRevision> WorkItemHistory { get; set; }
public IEnumerable<WorkItemFact> WorkItemFacts { get; set; }
```

⁴¹ <http://blogs.msdn.com/b/briankel/archive/2013/01/07/odata-service-for-team-foundation-server-v2.aspx>

Returning a list of WorkItemRevisions and WorkItemFacts

Now the execution of the OData request will look for a list of WorkItemRevisions we need to make the project aware of the existence of WorkItemRevisions.

- Navigate to the Entities folder of the TFS.OData.Model project and add a new class `WorkItemRevision`. See page 39 for a sample implementation.
- Open the `TFSDData.cs` file in the root of the TFS.OData.Model project and add the following members to the `TFSDData` class.

```
public IQueryable<WorkItemRevision> WorkItemHistory
{
    get { return CreateQuery<WorkItemRevision>(); }
}
public IQueryable<WorkItemFact> WorkItemFact
{
    get { return CreateQuery<WorkItemFact>(); }
}
```

- Inject the following code into the `public override object RepositoryFor(string fullTypeName)` method of the `TFSDData` class

```
if (fullTypeName == typeof(WorkItemRevision).FullName
    || fullTypeName == typeof(WorkItemRevision[]).FullName)
{
    return new WorkItemHistoryRepository(this.tfsProxyFactory.TfsWorkItemHistoryProxy);
}
if (fullTypeName == typeof(WorkItemFact).FullName
    || fullTypeName == typeof(WorkItemFact[]).FullName)
{
    return new WorkItemFactRepository(this.tfsProxyFactory.TfsWorkItemFactProxy);
}
```

- Open the `TfsProxyFactory.cs` file in the serializations folder of the TFS.OData.Model project and add the following members to the `TFSPROXYFactory` class.

```
public ITfsWorkItemHistoryProxy TfsWorkItemHistoryProxy
{
    get { return new TFSWorkItemHistoryProxy(this.tfsUri, this.tfsCredentials); }
}
public ITfsWorkItemFactProxy TfsWorkItemFactProxy
{
    get { return new TFSWorkItemFactProxy(this.tfsUri, this.tfsCredentials); }
}
```

Fetching the WorkItemRevisions

Now we need to implement the logic to fetch WorkItemRevisions.

- Add the following interface (preferable place it in the Serializations folder):

```
public interface ITfsWorkItemHistoryProxy
{
    IEnumerable<WorkItemRevision> GetWorkItemHistoryByProject(string projectName,
                                                            FilterNode rootFilterNode,
                                                            ODataQueryOperation operation);
}
```

- Add a `TFSWorkItemHistoryProxy` class to the Repositories folder, for implementation refer to the `TFSWorkItemProxy` class.
- Add code to the `ExecuteWIQLRequest` method that loops through the query result and create new `WorkItemRevision` instances for each revision of a work item.

```
List<WorkItem> wilist = this.QueryWorkItems(wiql).Cast<WorkItem>()
    .Skip(operation.SkipCount).Take(operation.TopCount)
    .Select(w => w).ToArray();
List<WorkItemRevision> lst = new List<WorkItemRevision>();
foreach (WorkItem w in wilist) {
    // Get All work item revisions
    foreach (Revision revision in w.Revisions) {
        WorkItemRevision wrkRev = new WorkItemRevision();
        wrkRev.Set(w.ToModel(this.GetTfsWebAccessArtifactUrl(w.Uri)));
        wrkRev.Revision = revision.Index;
        // Get value of fields in the work item revision
    }
}
```

```

        foreach (Field field in w.Fields) {
            wrkRev.SetField(field.Name, revision.Fields[field.Name].Value);
        }
        lst.Add(wrkRev);
    }
}

```

Fetching the WorkItemFacts

Now we need to implement the logic to fetch WorkItemFacts.

- Add the following interface (preferable place it in the Serializations folder):

```

public interface ITfsWorkItemFactProxy
{
    IEnumerable<WorkItemFact> GetWorkItemFactByProject(string projectName,
                                                    FilterNode rootFilterNode,
                                                    ODataQueryOperation operation
                                                    );
}

```

- Add a TFSWorkItemFactProxy class to the Repositories folder, for implementation refer to the TFSWorkItemProxy class.
- Add code to the ExecuteWIQLRequest method that loops through the query result and create new two WorkItemFact instances for each revision of a work item, except the last one.

```

IEnumerable<TeamFoundation.WorkItemTracking.Client.WorkItem> wilist = this.QueryWorkItems(wiql)
.Cast<TeamFoundation.WorkItemTracking.Client.WorkItem>()
.Skip(operation.SkipCount).Take(operation.TopCount)
.Select(w => w).ToArray();
List<WorkItemFact> lst = new List<WorkItemFact>();
foreach (TeamFoundation.WorkItemTracking.Client.WorkItem w in wilist)
{
    TeamFoundation.WorkItemTracking.Client.Revision prevRev=null;
    // Get All work item revisions
    foreach (TeamFoundation.WorkItemTracking.Client.Revision revision in w.Revisions)
    {
        if (prevRev!=null)
        {
            WorkItemFact wiFactPrev = new WorkItemFact();
            wiFactPrev.Project = w.Project.Name;
            wiFactPrev.Id = w.Id;
            wiFactPrev.Revision = (int)prevRev.Fields["System.Rev"].Value;
            wiFactPrev.RevisionDate = System.Convert.ToDateTime(revision.Fields[
                "Changed Date"].Value).Date; // Take the date for the next revision
            wiFactPrev.RevisionCount=null;
            wiFactPrev.RecordCount=-1;
            wiFactPrev.SurrogateKey = wiFactPrev.Id + ":" + wiFactPrev.Revision;

            // add numeric values to the fact
            foreach (TeamFoundation.WorkItemTracking.Client.Field field in prevRev.Fields)
            {
                switch (prevRev.Fields[field.Name].FieldDefinition.FieldType)
                {
                    case TeamFoundation.WorkItemTracking.Client.FieldType.Integer:
                        wiFactPrev.SetField(field.Name,
                            System.Convert.ToInt64(field.Value) * -1);
                        break;
                    case TeamFoundation.WorkItemTracking.Client.FieldType.Double:
                        wiFactPrev.SetField(field.Name,
                            System.Convert.ToDouble(field.Value) * -1);
                        break;
                }
            }
            lst.Add(wiFactPrev);
        }
        prevRev=revision;
        WorkItemFact wiFact = new WorkItemFact();
        wiFact.Project = w.Project.Name;

        wiFact.Id = w.Id;
        wiFact.Revision = (int)revision.Fields["System.Rev"].Value;
        wiFact.RevisionCount = 1;
        wiFact.RecordCount = 1;
    }
}

```

Part 1: Hosted Service & OData – Customize OData for trending

```
wiFact.RevisionDate = System.Convert.ToDateTime(revision.Fields[
    "Changed Date"].Value).Date;
wiFact.SurrogateKey = wiFact.Id + ":" + wiFact.Revision;
// add numeric values to the fact
foreach (TeamFoundation.WorkItemTracking.Client.Field field in revision.Fields)
{
    switch (revision.Fields[field.Name].FieldDefinition.FieldType)
    {
        case TeamFoundation.WorkItemTracking.Client.FieldType.Integer:
            wiFact.SetField(field.Name, System.Convert.ToInt64(field.Value));
            break;
        case TeamFoundation.WorkItemTracking.Client.FieldType.Double:
            wiFact.SetField(field.Name, System.Convert.ToDouble(field.Value));
            break;
    }
}
lst.Add(wiFact);
}
retWorkItems = lst;
```

Stitching it together

In order to stitch it together we need to implement some OData plumbing.

- Add a `WorkItemRevisionRepositories` class to the `Repositories` folder implementing `IRepository<WorkItemRevision>`
- Add a `WorkItemFactRepositories` class to the `Repositories` folder implementing `IRepository<WorkItemFact>`

We also need to set access rules on our new `WorkItemRevision` and `WorkItemFacts`. This is done by editing the `void InitializeService(DataServiceConfiguration config)` method of the `TFSService` class located in the `TFSService.cs` file in the root folder of `ODataTFS.Web` project.

Add the following code:

```
config.SetEntitySetAccessRule("WorkItemHistory", EntitySetRights.AllRead);
config.SetEntitySetAccessRule("WorkItemFact", EntitySetRights.AllRead);

config.SetEntitySetPageSize("WorkItemHistory", Constants.DefaultEntityPageSize);
config.SetEntitySetPageSize("WorkItemFact", Constants.DefaultEntityPageSize);
```

Build and Deploy

Build and deploy the services as described in **Setting up TfsOData**, page 11.

Test the extension

Now you can execute a request to fetch the `WorkItemHistory`, open a browser and navigate to [http://localhost/DefaultCollection/Projects\('projectName'\)/WorkItemHistory](http://localhost/DefaultCollection/Projects('projectName')/WorkItemHistory)



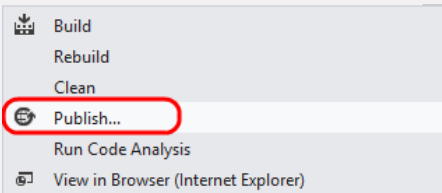

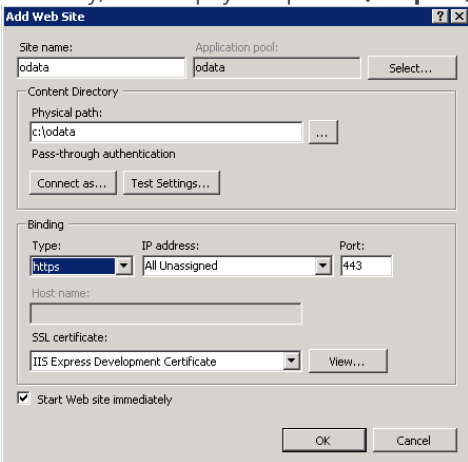
NOTE

This is only an extract of the customized code. Contact the ALM Rangers or Brian Keller for the complete code.

Customized OData Service deployment to IIS walk-through

NOTE

This walkthrough was verified for Windows Server 2008 R2 Standard SP1, Windows Server 2012 Standard and Windows Server 2012 R2. You may need to adapt the steps to match up with your environment if there are any differences.

Step	Instructions
1 Build OData to Server  - Done	<ul style="list-style-type: none"> Open the customized ODataTFS.sln solution. Select Build Rebuild Solution to ensure everything is up-to-date.
2 Publish OData Webproject to disk  - Done	<ul style="list-style-type: none"> Right-click on the ODataTFS.Web project in Solution Explorer and select the Publish option. You can use any publish method that you want, but the remaining instructions in the section are based on the File System method. 
3 Copy published website to server  - Done	<ul style="list-style-type: none"> Copy the directory to c:\inetpub\wwwroot\TfsOdataAlmRangers <div data-bbox="337 972 365 1037" data-label="Text">NOTE</div> <p>If you have completed Setting up TfsOData, page 11 you can skip the rest of this walk-through.</p> <ul style="list-style-type: none"> In IIS manager, create a new Website. Name it tfsoadata and point it to the newly created directory. Make sure that you use HTTPS and a SSL certificate for the binding. Preferably, use the physical path c:\inetpub\wwwroot\TfsOdataAlmRangers, in place of c:\odata.  <div data-bbox="337 1667 365 1732" data-label="Text">NOTE</div> <p>For testing purposes, you can use a self-signed certificate. One way to do this using IIS Manager is to navigate to the machine node, open Server Certificates, and select Create Self-Signed Certificate. To learn more about SSL and IIS, please see http://learn.iis.net/page.aspx/144/how-to-set-up-ssl-on-iis-7.</p>

Part 1: Hosted Service & OData – Customize OData for trending

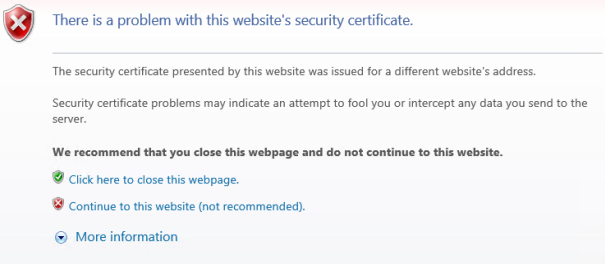
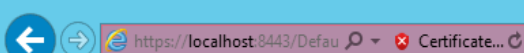
Step	Instructions
4 Import the certificate <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Browse to https://localhost:433/ and you will be presented with the following warning <div data-bbox="365 241 990 525">  <p>There is a problem with this website's security certificate.</p> <p>The security certificate presented by this website was issued for a different website's address.</p> <p>Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.</p> <p>We recommend that you close this webpage and do not continue to this website.</p> <p> <input checked="" type="checkbox"/> Click here to close this webpage. <input checked="" type="checkbox"/> Continue to this website (not recommended). <input type="checkbox"/> More information </p> </div> Click the Continue to this website link. The URL field will then turn red and show a certificate error. <div data-bbox="365 556 901 630">  <p>← → https://localhost:8443/Default.aspx Certificate...</p> </div> Click the error, choose View certificate and then import the certificate.
6 Add certificate <input type="checkbox"/> - Done	<ul style="list-style-type: none"> At this point, the OData Service is ready to go.

Table 11 – Walkthrough: Deploying the OData Service to IIS Server

Customize Reports for trending

Reporting on data over time is a paramount for process improvement. Without knowing the impact of process changes or knowing what's coming based on what has happened in the past, there's no way to know whether changes are having a positive or negative effect. The approach taken with the trend reporting from the hosted service is different from the point-in-time approach taken earlier.

NOTE

In order to create trend reports, the OData feed had to be extended because the necessary data was not present in the existing OData feeds. The point in time reporting described earlier can be accomplished with the existing feeds. To implement this solution, you must implement the extended OData feed as introduced on page 24.

Understanding the data produced by the extended OData feed

WARNING

If there is no data in the system, nothing is returned by the extended OData feed – not even the schema, so the update will fail.

The extended OData feed provides two new feeds required for trend reporting (and which can be used for point in time reporting using more sophisticated capabilities than are available in the original solution). The first feed is the **Work Item History** feed and the second is the **Work Item Facts** feed and both are used together to be able to provide trend reporting. Figure 2 shows the resulting PowerPivot model from using the required feeds and a date table.

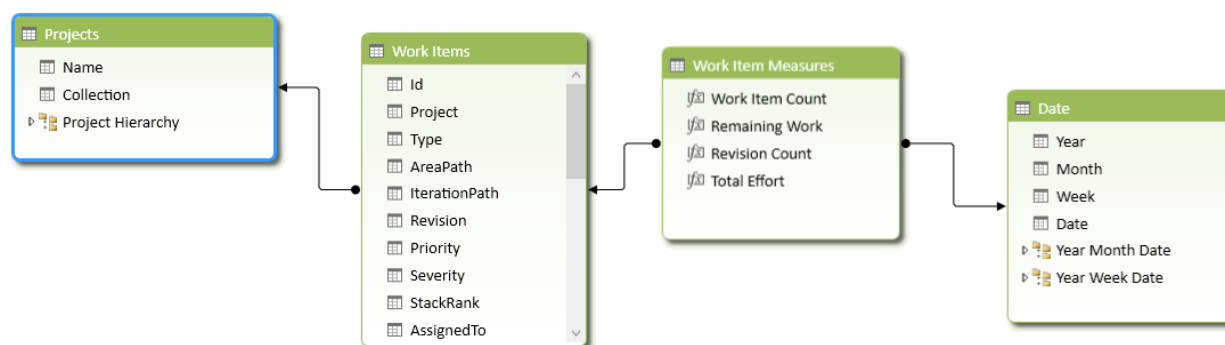


Figure 2 - Trend Reporting PowerPivot Model

The "Date" table comes from the Excel file in the sample solution and is described in more detail below.

Work Item History feed

The Work Item History feed is a dump of the history of all work items in a given collection. It identifies the point in time at which a work item was in a given state, assigned to a given user and various other generic attributes such as iteration and area paths, blocked or not, the stack rank and other data.

NOTE

In the model shown in Figure 2, the WorkItemHistory feed has been renamed to "Work Items"

This data feed, once pulled into the PowerPivot model, requires just one change – an alternate (or surrogate) key has to be added to the feed. This column is a simple calculated column. To add it, do this:

1. Click the first row in the empty column at the end of the table.
2. Enter the formula as: =[Id] & ":" & [Revision] and press ENTER.
3. Rename the column from CalculatedColumn1 to AlternateKey.

Work Item Fact Feed

This feed is the more interesting feed and allows a user to perform calculations in a simple but powerful way. For those unfamiliar with data warehousing, the critical columns are explained here as well as the rationale for the construction of those columns and the relationship with the Work Item History table.

Once the feed has been imported, the table was renamed to the "Work Item Measures" table, which is a bit more descriptive to an end user than the Work Item Fact table. In addition, every column in this table is hidden from client tools because users should only be using it for measures. All other substantive work item data comes from the Work Items table.

There is one added column on this table – the **DateSK** column. This column strips the time off of the Revision Date column because the measure of granularity that this is limited to is a given day, not an hour or minute within the day. More granular reporting is possible but it requires you to expand the date table considerably to include every hour of every day of every year listed. The reason for calling this the DateSK column is simply naming conventions – the primary key of the Date table is DateSK and the join is on this column.

Column/Measure	Description																																
RevisionCount	This column contains a single value – 1 – for each revision count shown in the Revision column. You will notice that each revision is duplicated (see the RecordCount explanation).																																
RecordCount	<p>The Record Count is the number of work items which exist on a given day for the purposes of trend reporting. Each revision of a work item will have one or two rows in this table. The first row will show the RecordCount as a “1”. The second row (if there is a second row) will show a “-1”. Why? Consider the following situation:</p> <ul style="list-style-type: none">• A user adds a work item on 10/1/2013 which has a revision of 1 and an ID of 560• Another user makes a change to this work item on 10/5/2013 which has a revision of 2• Another user makes a revision on 10/8/2013 which has a revision of 3 <p>This scenario would show up in this table with the following structure:</p> <table><tr><th>ID</th><th>RevisionDate</th><th>Revision</th><th>RecordCount</th></tr><tr><td>560</td><td>10/1/2013</td><td>1</td><td>1</td></tr><tr><td>560</td><td>10/5/2013</td><td>1</td><td>-1</td></tr><tr><td>560</td><td>10/5/2013</td><td>2</td><td>1</td></tr><tr><td>560</td><td>10/8/2013</td><td>2</td><td>-1</td></tr><tr><td>560</td><td>10/8/2013</td><td>3</td><td>1</td></tr></table> <p>What this allows the PowerPivot model to do is to calculate the intervening dates correctly for trend reporting. When the cube is created and the end-user adds a date to their pivot table, the rows for this record show that work item 560 existing from 10/1/2013 to 10/8/2013 (let’s say this was created on the 8th).</p> <p>Now, this table relates back to the Work Items table in the model and all of the other related data can be pulled from that table. Let’s make another assumption for this scenario that the states for this work item were as follows:</p> <table><tr><th>ID</th><th>RevisionDate</th><th>Revision</th><th>State</th></tr><tr><td>560</td><td>10/1/2013</td><td>1</td><td>To Do</td></tr></table>	ID	RevisionDate	Revision	RecordCount	560	10/1/2013	1	1	560	10/5/2013	1	-1	560	10/5/2013	2	1	560	10/8/2013	2	-1	560	10/8/2013	3	1	ID	RevisionDate	Revision	State	560	10/1/2013	1	To Do
ID	RevisionDate	Revision	RecordCount																														
560	10/1/2013	1	1																														
560	10/5/2013	1	-1																														
560	10/5/2013	2	1																														
560	10/8/2013	2	-1																														
560	10/8/2013	3	1																														
ID	RevisionDate	Revision	State																														
560	10/1/2013	1	To Do																														

Part 1: Hosted Service & OData – Customize Reports for trending

Column/Measure	Description																																																
	<table><tr><td>560</td><td>10/5/2013</td><td>2</td><td>In Work</td></tr><tr><td>560</td><td>10/8/2013</td><td>3</td><td>Done</td></tr></table> <p>These tables are joined on the alternate key which is the ID:Revision.</p> <p>Now, picture the pivot table with the date on the row, state on the column and work item count as the measure (filtered by this work item). It would look like this:</p> <table><tr><th colspan="4">State</th></tr><tr><th>Date</th><th>To Do</th><th>In Work</th><th>Done</th></tr><tr><td>10/1/2013</td><td>1</td><td></td><td></td></tr><tr><td>10/2/2013</td><td>1</td><td></td><td></td></tr><tr><td>10/3/2013</td><td>1</td><td></td><td></td></tr><tr><td>10/4/2013</td><td>1</td><td></td><td></td></tr><tr><td>10/5/2013</td><td></td><td>1</td><td></td></tr><tr><td>10/6/2013</td><td></td><td>1</td><td></td></tr><tr><td>10/7/2013</td><td></td><td>1</td><td></td></tr><tr><td>10/8/2013</td><td></td><td></td><td>1</td></tr></table> <p>This is generated correctly because of the RecordCount value. The cube says that between 10/1/2013 and 10/4/2013, the record count for this work item was 1. On 10/5/2013 however, the revision changed. If the -1 were not there, when the count of work items was calculated, it would be a count of 2.</p> <p>However, that's not the case – there is only ever one work item with the ID of 560. It's in a different state. Therefore, what this structure is telling us is that no work item with an ID of 560 existed on 10/5 with a state of "To Do", but a work item with ID 560 existed on 10/5 with a state of "In Work". What if the end user decides not include the states in the pivot table? Then it becomes a straight list of dates with every date having a "1" next to it because on any given day, the sum of the RecordCount field always equals one.</p> <div><div>NOTE</div><div>This logic applies to all other measure fields such as the Completed and Remaining Work fields.</div></div> <p>See below for the Work Item Count measure to understand the calculation that makes this work.</p>	560	10/5/2013	2	In Work	560	10/8/2013	3	Done	State				Date	To Do	In Work	Done	10/1/2013	1			10/2/2013	1			10/3/2013	1			10/4/2013	1			10/5/2013		1		10/6/2013		1		10/7/2013		1		10/8/2013			1
	560	10/5/2013	2	In Work																																													
	560	10/8/2013	3	Done																																													
	State																																																
	Date	To Do	In Work	Done																																													
	10/1/2013	1																																															
	10/2/2013	1																																															
	10/3/2013	1																																															
	10/4/2013	1																																															
	10/5/2013		1																																														
10/6/2013		1																																															
10/7/2013		1																																															
10/8/2013			1																																														
Work Item Count	<p>The calculation for the count is: <code>CALCULATE(sum([RecordCount]), DATESBETWEEN('Date'[DateSK], BLANK(), LASTDATE('Date'[DateSK])), ALL('Date'))</code></p> <p>Essentially this is a sum of the RecordCount column but by using the DAX CALCULATE function, it provides context to the Sum function. If the user changes the scope of the date, the count will be correctly updated. Without the CALCULATE function the record count would remain static.</p> <p>The calculation itself is very fast because it just adds the total of "1" and "-1" for every row in the table. For any given work item, the total is always 1. For a given set of work items, the total is always the sum of work items which were in a given revision in the selected time period. Using the example above, if the user changed the scope of the date range in the pivot table to 10/7 then the table would never display the "Done" column because based on the given range, the work item has no record count up to the date of 10/7.</p>																																																
Remaining Work	<p>The calculation for the work is: <code>CALCULATE(Sum([RemainingWork]), DATESBETWEEN('Date'[DateSK], BLANK(), LASTDATE('Date'[DateSK])), ALL('Date'))</code></p>																																																

Part 1: Hosted Service & OData – Customize Reports for trending

Column/Measure	Description
	This is identical to the Work Item Count. Fact tables constructed in this manner provide a uniform method for calculating measures.
Revision Count	<code>CALCULATE(Sum([RevisionCount]), DATESBETWEEN('Date'[DateSK], BLANK(), LASTDATE('Date'[DateSK])), ALL('Date'))</code>
Total Effort	<code>CALCULATE(Sum([Effort]), DATESBETWEEN('Date'[DateSK], BLANK(), LASTDATE('Date'[DateSK])), ALL('Date'))</code>

Table 12 – Work item fact feed

Date table

The Date table provides the range of dates for which the cube can be calculated against. It is a continuous list of dates from the start date until today (although it can be extended out as far as is needed). It allows the cube to fill in values for which there are no specific dates. This is accomplished by the measures themselves, which reference the date table and the fact that the Work Item Measures table is related to the date table.

The date table is not provided through a feed – it must be provided by you through an alternate mechanism. In the standard TFS reporting mechanism, the data warehouse contains a date table constructed through code. Because there is no warehouse in the hosted service, we provided a solution template with a hidden Date tab (to unhide it select any tab, Unhide and select the Date tab). The data from this tab is used to create a linked table in PowerPivot.

For consistency with the on-premises warehouse, the structure of this sheet is identical to the date table in the warehouse even though not all columns are needed. To extend the data (the date range only goes up until around the publication of the sample project), simply highlight all of the cells in the last five rows and continue to drag down. This can be done using VBA code as well, in addition to actually loading it in a SQL table and scheduling a job to update the table once a day.

On the date table, two hierarchies are created. These hierarchies match the date hierarchies in the standard TFS cube. These hierarchies are Year – Month – Date and Year – Week – Date and all you to produce drill downs for these time structures.

Projects table

This table is from the same feed as that in the previous sample, page 15.

Creating Trend Reports Walkthrough

NOTE

Refer to **ALMRangersTrendDataReferenceSolution.xlsx** which is a completed reference sample workbook done by completing this walkthrough.

Trend reporting is similar to point in time reporting except that the row is usually divided by date to produce charts that look like this:

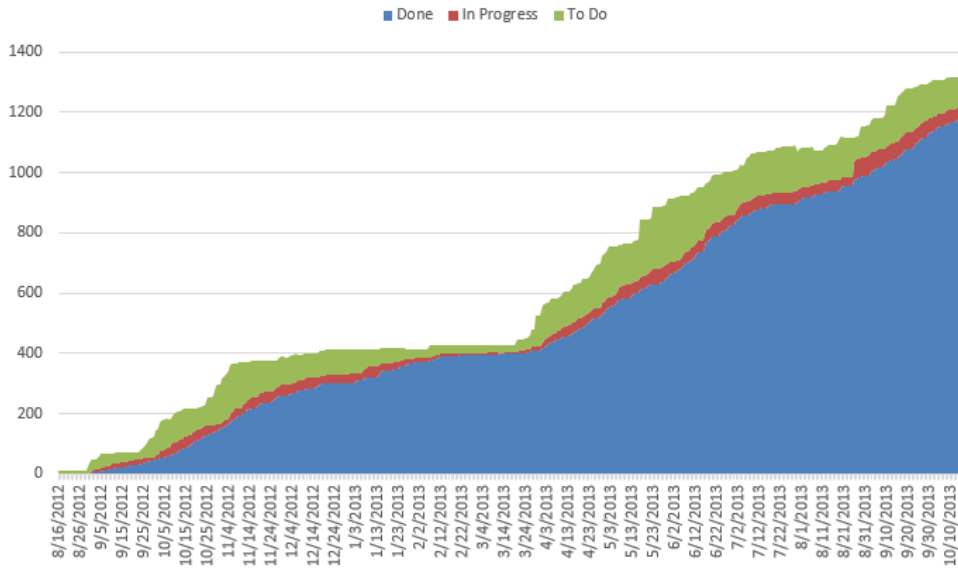
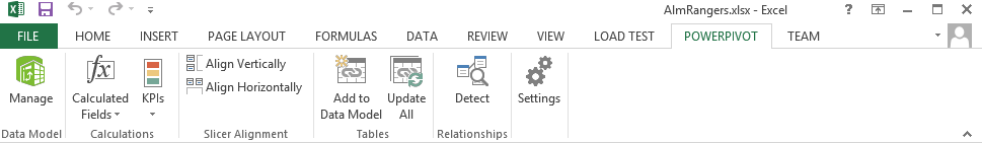
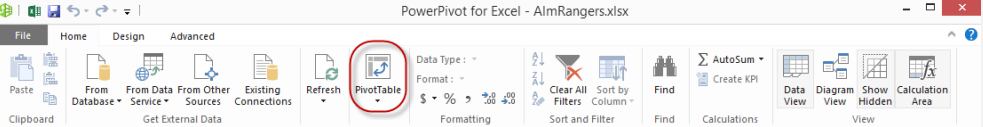
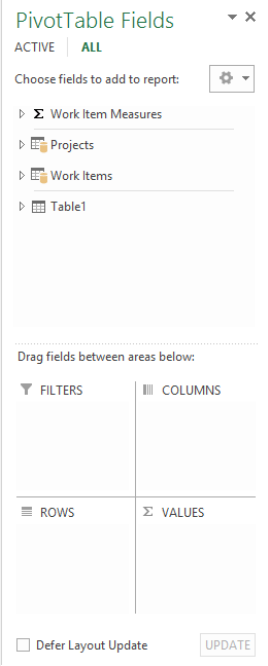




Figure 3 - Trend Reporting Example

The example in Figure 2 is a standard Cumulative Flow Diagram (CFD) on the tasks of the ALM Rangers project since its inception on August 16th 2012. To create a report similar to this, complete the following walk through:

Step	Instructions
1 Select PowerPivot [- Done]	<ul style="list-style-type: none"> Select PowerPivot, Manage from the main menu in the Excel sample project. 
2 Select worksheet [- Done]	<ul style="list-style-type: none"> In the PowerPivot window on the Home tab select PivotTable.  <ul style="list-style-type: none"> Select New Worksheet (or use existing and select the starting range – usually A1) and click OK.

Part 1: Hosted Service & OData – Customize Reports for trending

Step	Instructions
	<ul style="list-style-type: none"> This will display the PivotTable Fields list to the right of the Excel window as show.  <p>NOTE The Date table shows up as Table 1 instead of Date but expanding it will show the correct data. There seems to be no way to correct this at this time</p>
3 Define measure  - Done	<ul style="list-style-type: none"> Expand the Work Item Measures section and click the Work Item Count measure. This will add the Work Item Count measure to the Values section of the Fields pane and show the work item count on the pivot table.
4 Define dimension  - Done	<ul style="list-style-type: none"> Expand the Work Items dimension and drag the Type field to the Filters box of the Fields pane In cell B2 change the Type filter from All to Task. Expand the Table1 dimension and the More Fields folder and drag the Date field to the Rows box of the Fields pane. Next, under Work Items, drag the State field to the Columns box of the Fields pane At this point, and assuming you are using your data, you might need to change the order of the State columns and/or remove columns. To remove a state, click the drop-down list next to Column Labels (Cell B3 if following this example) and clear the checkbox next to the state to remove (or add). To reorder the columns, simply grab a column label (for example, cell B4) and drag it to the correct location. In a CFD diagram using Excel, the bottom value starts at the left and the top value is at the right in terms of column order.

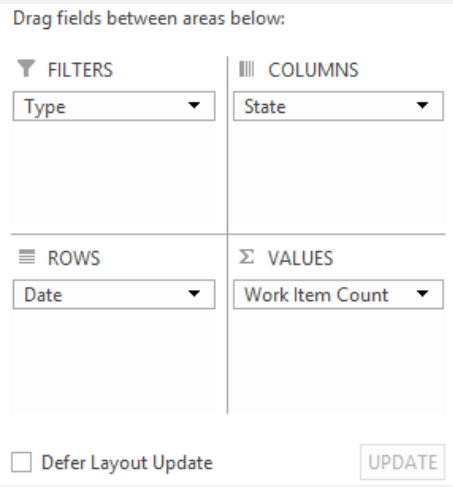

Step	Instructions
	<ul style="list-style-type: none">The PivotTable Fields boxes should look like this: 
5 Create chart  - Done	<ul style="list-style-type: none">Finally, to create the chart, select Analyze from the ribbon, select PivotChart and select Stacked Area.This data can be replaced with any other type of data for the breakdown you need. The dates can also be used to constrain a range rather than being used as the row data.

Table 13 – Walkthrough: Trend reporting example

Sample Reports built using the trending walkthrough

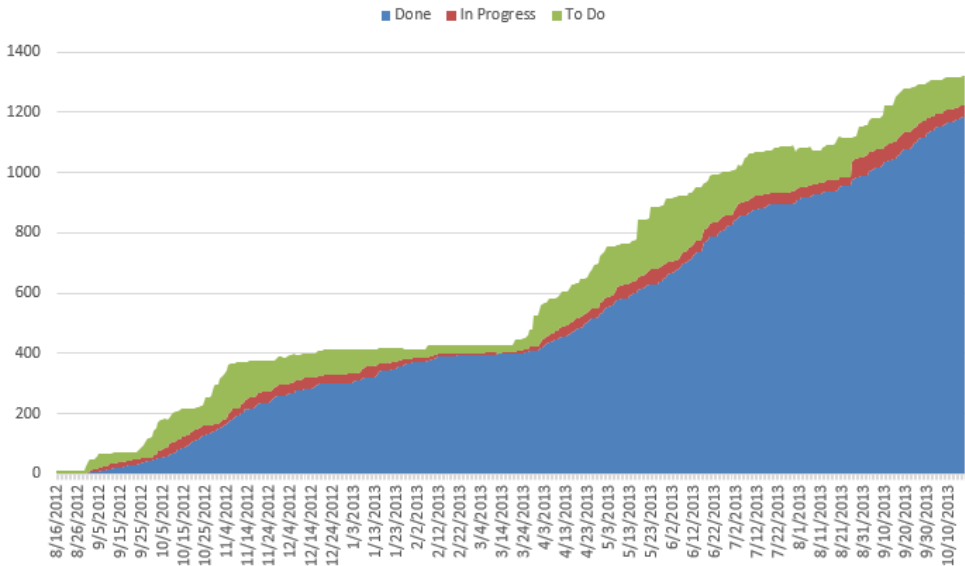


Figure 4 – Example showing overall trend data

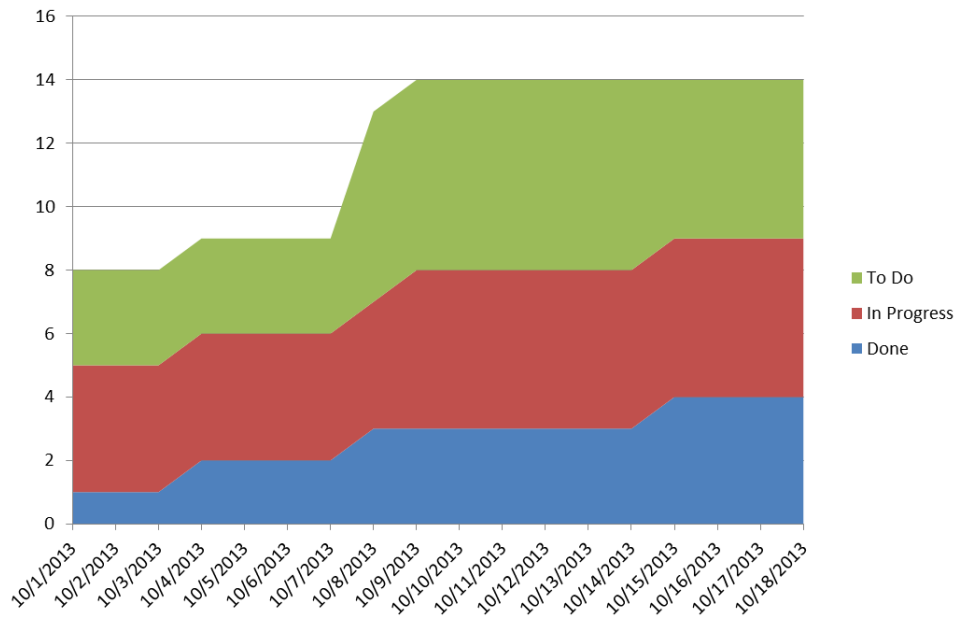


Figure 5 – Example showing work item count for a selected sprint

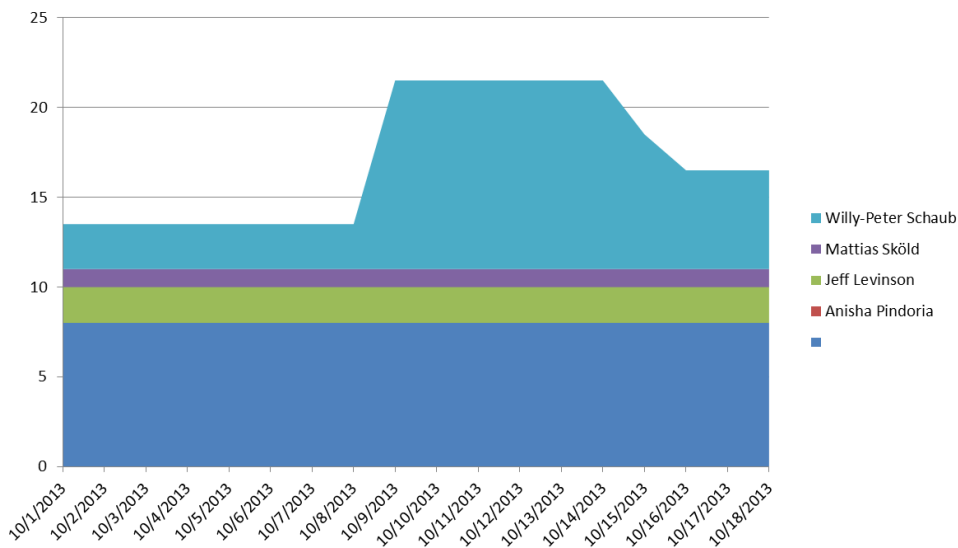


Figure 6 – Example showing tasks assigned to team members for a selected sprint and team

The workbook is your oyster ☺ Slice, format and report the trending data as needed for your team.

In Conclusion

This concludes our adventure into the Tabular Store Model. We have touched on theory, introduced you to the Tabular Store Model, PowerPivot, OData and Reporting, and illustrated when to use this technology. We have covered various exercises in the walk-throughs, guided you through the theory, design and practical usage of the PowerPivot and OData.

In the final pages of this guide, you will find our **Quick Reference** poster(s). You might find it useful to print these and hang them up in the team area.

We are at the beginning of the reference solution and sample reports. We hope you find it a valuable technology to invest in and that you have found this guide useful.

Sincerely

The Microsoft Visual Studio ALM Rangers



Appendix

Sample Code - OData Customization

WorkItemRevision

Code 1 – WorkItemRevision

```
// -----
// Microsoft Developer & Platform Evangelism
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//
// THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
// EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES
// OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.
// -----
// The example companies, organizations, products, domain names,
// e-mail addresses, logos, people, places, and events depicted
// herein are fictitious. No association with any real company,
// organization, product, domain name, email address, logo, person,
// places, or events is intended or should be inferred.
// -----

namespace Microsoft.Samples.DPE.ODataTFS.Model.Entities
{
    using System;
    using System.Collections.Generic;
    using System.Data.Services;
    using System.Data.Services.Common;
    using System.Reflection;
    using Microsoft.Data.Services.Toolkit.QueryModel;

    //Field references:
    //http://msdn.microsoft.com/en-us/library/ms194971.aspx
    [DataServiceKey("Id")]
    [ETag("ChangedDate")]
    [EntityPropertyMapping("ChangedDate", SyndicationItemProperty.Updated,
        SyndicationTextContentKind.Plaintext, true)]
    [EntityPropertyMapping("Title", SyndicationItemProperty.Title,
        SyndicationTextContentKind.Plaintext, true)]
    [EntityPropertyMapping("Description", SyndicationItemProperty.Summary,
        SyndicationTextContentKind.Plaintext, true)]

    public class WorkItemRevision
    {
        //Base information to provide in addition to WorkItem fields
        public int Id { get; set; }
        public string Project { get; set; }
        public string Type { get; set; }
        public string WebEditorUrl { get; set; }

        //Base fields
        public string AreaPath { get; set; }
        public string IterationPath { get; set; }
        public int Revision { get; set; } //aka Rev
        public string Priority { get; set; }
        public string Severity { get; set; }
        public double StackRank { get; set; }
        public string AssignedTo { get; set; }
        public DateTime CreatedDate { get; set; }
        public string CreatedBy { get; set; }
        public DateTime ChangedDate { get; set; }
        public string ChangedBy { get; set; }
        public string ResolvedBy { get; set; }
        public string Title { get; set; }
        //valid states and transitions specific to template and work item
        public string State { get; set; }
        public string Reason { get; set; }
    }
}
```

```

public double CompletedWork { get; set; }
public double RemainingWork { get; set; }
public string Description { get; set; }
public string ReproSteps { get; set; }
public string FoundInBuild { get; set; }
public string IntegratedInBuild { get; set; }
public int AttachedFileCount { get; set; }
public int HyperLinkCount { get; set; }
public int RelatedLinkCount { get; set; }

//Agile
public string Risk { get; set; }
public double StoryPoints { get; set; }

//Agile and CMMI
public double OriginalEstimate { get; set; }

//Scrum
public double BacklogPriority { get; set; }
public int BusinessValue { get; set; }
public double Effort { get; set; }

//Scrum and CMMI
public string Blocked { get; set; }

//CMMI
public double Size { get; set; }

[ForeignKey]
public IEnumerable<Attachment> Attachments { get; set; }

[ForeignKey]
public IEnumerable<Link> Links { get; set; }

public void Set(WorkItem w)
{
    foreach (PropertyInfo p in w.GetType().GetProperties())
    {
        SetField(p.Name, p.GetValue(w, null));
    }
}

public void SetField(string field, object value)
{
    if (this.GetType().GetProperty(field) != null)
    {
        if (value != null)
        {
            this.GetType().GetProperty(field).SetValue(this,
                System.Convert.ChangeType(value,
                    this.GetType()
                        .GetProperty(field)
                        .PropertyType), null);
        }
        else
        {
            this.GetType().GetProperty(field).SetValue(this, value, null);
        }
    }
}
}
}

```

WorkItemHistoryProxy

Code 2 – WorkItemHistoryProxy

```
// -----
// Microsoft Developer & Platform Evangelism
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//
// THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
// EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES
// OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.
// -----
// The example companies, organizations, products, domain names,
// e-mail addresses, logos, people, places, and events depicted
// herein are fictitious. No association with any real company,
// organization, product, domain name, email address, logo, person,
// places, or events is intended or should be inferred.
// -----
namespace Microsoft.Samples.DPE.ODataTFS.Model.Serialization
{
    using System;
    using System.Collections.Generic;
    using System.Globalization;
    using System.Linq;
    using System.Net;
    using System.Linq.Expressions;
    using System.Text;
    using System.Text.RegularExpressions;
    using System.Web;
    using Microsoft.Samples.DPE.ODataTFS.Model.Entities;
    using Microsoft.Samples.DPE.ODataTFS.Model.ExpressionVisitors;
    using System.Collections;
    using Microsoft.Data.Services.Toolkit.QueryModel;
    using Microsoft.TeamFoundation.WorkItemTracking;

    public class TFSWorkItemHistoryProxy : TFSBaseProxy, ITfsWorkItemHistoryProxy
    {
        public TFSWorkItemHistoryProxy(Uri uri, ICredentials credentials)
            : base(uri, credentials){}

        public WorkItem GetWorkItem(int workItemId)
        {
            var wiql = string.Format(CultureInfo.InvariantCulture,
                "SELECT [System.Id] FROM WorkItems WHERE [System.Id] = {0}",
                workItemId);
            return this.QueryWorkItems(wiql)
                .Cast<TeamFoundation.WorkItemTracking.Client.WorkItem>()
                .Select(w => w.ToModel(this.GetTfsWebAccessArtifactUrl(w.Uri)))
                .FirstOrDefault();
        }

        public IEnumerable<WorkItemRevision>
            GetWorkItemHistoryByProject(string projectName,
                                        FilterNode rootFilterNode,
                                        ODataQueryOperation operation)
        {
            var key = string.Format(CultureInfo.InvariantCulture,
                "TFSWorkItemProxy.GetWorkItemsByProject {0} {1}",
                projectName, this.GetFilterNodeKey(rootFilterNode));
            HttpContext.Current.Items[key] =
                this.RequestWorkItemsByProject(projectName,
                                                rootFilterNode, operation);
            return (IEnumerable<WorkItemRevision>)HttpContext.Current.Items[key];
        }

        private static string BuildWiql(FilterNode rootFilterNode,
                                        ODataQueryOperation operation)
        {
            var constrains = string.Empty;
            if (rootFilterNode != null)
            {
                foreach (var filterNode in rootFilterNode)
                {

```

```

        constrains +=
            AddComparisonConstrainToWiql(filterNode,
                Constants.TFS.FieldLookup(filterNode.Key));
    }
}

if (constrains.StartsWith(" OR ", StringComparison.OrdinalIgnoreCase))
{
    constrains = constrains.Substring(" OR ".Length);
}

if (constrains.StartsWith(" AND ", StringComparison.OrdinalIgnoreCase))
{
    constrains = constrains.Substring(" OR ".Length);
}

var wiql = string.Format(CultureInfo.InvariantCulture,
    "SELECT [System.Id] FROM WorkItems {0} {1}",
    string.IsNullOrEmpty(constrains) ?
        string.Empty : "WHERE", constrains).Trim();
string orderByWiql = string.Empty;
if (operation.OrderStack.Count > 0)
{
    orderByWiql = TFSWorkItemHistoryProxy.GenerateOrderByWiql(operation.OrderStack);
}
wiql = wiql + " " + orderByWiql;
return wiql;
}

private static string
GenerateOrderByWiql(Stack<ODataOrderExpression> orderExpressionStack)
{
    if (orderExpressionStack == null)
    {
        throw new ArgumentNullException("orderExpressionStack");
    }

    StringBuilder finalOrderBy = new StringBuilder("ORDER BY ");
    //When the $top param is used with WCF Data Services Toolkit, this appears to
    //also imply ordering by key ascending (in this case work item ID), so if the
    //user also issues an explicit $orderby param for Id there will be a duplicate.
    //It appears that the first $orderby value (top of stack) is the one provided
    //explicitly by the user, so we take that one and track that it has been seen
    //already. Subsequent order by fields are ignored.
    ISet<string> seen = new HashSet<string>();
    while (orderExpressionStack.Count > 0)
    {
        ODataOrderExpression expr = orderExpressionStack.Pop();
        UnaryExpression unExpr = ((UnaryExpression)expr.Expression);
        var opString = unExpr.Operand.ToString();
        var name = opString.Substring(opString.IndexOf('.') + 1);
        name = Constants.TFS.FieldLookup(name);

        if (!seen.Contains(name))
        {
            if (expr.OrderMethodName.StartsWith("ThenBy"))
            {
                finalOrderBy.Append(" , ");
            }
            finalOrderBy.Append(name);

            switch (expr.OrderMethodName)
            {
                case "OrderBy":
                case "ThenBy":
                    finalOrderBy.Append(" asc");
                    break;
                case "OrderByDescending":
                case "ThenByDescending":
                    finalOrderBy.Append(" desc");
                    break;
            }
            seen.Add(name);
        }
    }
}

```

```

    }
    return finalOrderBy.ToString();
}

private static string AddComparisonConstrainToWiql(FilterNode filterNode, string tfsFieldName)
{
    if (filterNode != null)
    {
        var sign = default(string);
        switch (filterNode.Sign)
        {
            case FilterExpressionType.Equal:
                sign = "=";
                break;
            case FilterExpressionType.NotEqual:
                sign = "<>";
                break;
            case FilterExpressionType.GreaterThan:
                sign = ">";
                break;
            case FilterExpressionType.GreaterThanOrEqual:
                sign = ">=";
                break;
            case FilterExpressionType.LessThan:
                sign = "<";
                break;
            case FilterExpressionType.LessThanOrEqual:
                sign = "<=";
                break;
            case FilterExpressionType.Contains:
                sign = filterNode.Key.Equals("AreaPath", StringComparison.OrdinalIgnoreCase)
                    || filterNode.Key.Equals("IterationPath",
                        StringComparison.OrdinalIgnoreCase) ?
                    "UNDER" : "CONTAINS";
                break;
            case FilterExpressionType.NotContains:
                sign = filterNode.Key.Equals("AreaPath", StringComparison.OrdinalIgnoreCase)
                    || filterNode.Key.Equals("IterationPath",
                        StringComparison.OrdinalIgnoreCase) ?
                    "NOT UNDER" : "NOT CONTAINS";
                break;
            default:
                throw new NotSupportedException(string.Format(CultureInfo.InvariantCulture,
                    "WorkItem {0} can only be filtered with equal, not equal, greater than, lower than, greater than or equal, lower than or equal operators",
                    filterNode.Key));
        }
        return string.Format(CultureInfo.InvariantCulture, " {0} {1} {2} '{3}' ",
            filterNode.NodeRelationship.ToString(), tfsFieldName, sign,
            filterNode.Value);
    }
    return string.Empty;
}

private IEnumerable<WorkItemRevision>
    RequestWorkItemsByProject(string projectName, FilterNode rootFilterNode,
        ODataQueryOperation operation)
{
    FilterNode newFilterNodeStartWithProject =
        new FilterNode() { Key = "Project", Sign = FilterExpressionType.Equal,
            Value = projectName };
    if (rootFilterNode != null)
    {
        newFilterNodeStartWithProject.AddNode(rootFilterNode);
    }
    var wiql = BuildWiql(newFilterNodeStartWithProject, operation);
    return this.ExecuteWiqlRequest(wiql, operation);
}

private IEnumerable<WorkItemRevision>
    ExecuteWiqlRequest(string wiql, ODataQueryOperation operation)
{
    IEnumerable<WorkItemRevision> retWorkItems = null;

```

```

if (!operation.IsCountRequest)
{
    if (operation.TopCount == 0)
    {
        //workaround for bug (I think) in WCF Data Services Toolkit.
        //It appears that ODataQueryOperation.TopCount will be 0 when $select
        // param is used and $top param is not explicitly sent by client
        //(normally TopCount would have whatever was set for entity page size in
        //TFSService.InitializeService()
        operation.TopCount = Constants.DefaultEntityPageSize;
    }

    IEnumerable<TeamFoundation.WorkItemTracking.Client.WorkItem>
        wilist = this.QueryWorkItems(wiql)
                .Cast<TeamFoundation.WorkItemTracking.Client.WorkItem>()
                .Skip(operation.SkipCount).Take(operation.TopCount)
                .Select(w => w).ToArray();
    List<WorkItemRevision> lst = new List<WorkItemRevision>();

    foreach (TeamFoundation.WorkItemTracking.Client.WorkItem w in wilist)
    {
        // Get All work item revisions
        foreach (TeamFoundation.WorkItemTracking.Client.Revision revision
            in w.Revisions)
        {
            WorkItemRevision wrkRev = new WorkItemRevision();
            wrkRev.Set(
                w.ToModel(this.GetTfsWebAccessArtifactUrl(w.Uri))
            );
            wrkRev.Revision = revision.Index;

            // Get value of fields in the work item revision
            foreach (TeamFoundation.WorkItemTracking.Client.Field field
                in w.Fields)
            {
                wrkRev.SetField(field.Name, revision.Fields[field.Name].Value);
            }
            lst.Add(wrkRev);
        }
    }
}
else
{
    var workItemServer = this.TfsConnection.GetService<TeamFoundation
        .WorkItemTracking.Client.WorkItemStore>();

    try
    {
        Microsoft.TeamFoundation.WorkItemTracking.Client.Query q =
            new TeamFoundation.WorkItemTracking.Client.Query(workItemServer,
                wiql, null, false);

        int cnt = q.RunCountQuery();

        List<WorkItemRevision> wiBlanks = new List<WorkItemRevision>(cnt);
        WorkItemRevision blank = new WorkItemRevision();
        for (int i = 0; i < cnt; i++)
        {
            wiBlanks.Add(blank);
        }
        retWorkItems = wiBlanks;
    }
    catch (Microsoft.TeamFoundation.WorkItemTracking.Client.ValidationException
        ex)
    {
        throw new System.Data.Services.DataServiceException(500,
            "Internal Server Error", ex.Message, "en-US", ex);
    }
}
return retWorkItems;
}
}

```


WorkItemRevisionRepositories

Code 3 – WorkItemRevisionRepositories

```
// -----
// Microsoft Developer & Platform Evangelism
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//
// THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
// EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES
// OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.
// -----
// The example companies, organizations, products, domain names,
// e-mail addresses, logos, people, places, and events depicted
// herein are fictitious. No association with any real company,
// organization, product, domain name, email address, logo, person,
// places, or events is intended or should be inferred.
// -----
namespace Microsoft.Samples.DPE.ODataTFS.Model.Repositories
{
    using System;
    using System.Collections;
    using System.Collections.Generic;
    using System.Data.Services;
    using System.Globalization;
    using System.Linq;
    using Microsoft.Data.Services.Toolkit.QueryModel;
    using Microsoft.Samples.DPE.ODataTFS.Model.Entities;
    using Microsoft.Samples.DPE.ODataTFS.Model.ExpressionVisitors;
    using Microsoft.Samples.DPE.ODataTFS.Model.Serialization;

    public class WorkItemHistoryRepository : IRepository<WorkItemRevision>
    {
        private readonly ITfsWorkItemHistoryProxy proxy;

        public WorkItemHistoryRepository(ITfsWorkItemHistoryProxy proxy)
        {
            this.proxy = proxy;
        }

        public IEnumerable<WorkItemRevision> GetAll()
        {
            List<WorkItemRevision> lst = new List<WorkItemRevision>();
            return lst;
        }

        public WorkItemRevision GetOne(string id)
        {
            return null;
        }

        [RepositoryBehavior(HandlesFilter = true, HandlesSkip = true, HandlesTop = true,
            HandlesOrderBy = true)]
        public IEnumerable<WorkItemRevision>
            GetWorkItemHistoryByProject(ODataSelectManyQueryOperation operation)
        {
            if (operation == null)
            {
                throw new ArgumentNullException("operation");
            }
            var parameters = new
                WorkItemFilterExpressionVisitor(operation.FilterExpression).Eval();
            return this.proxy.GetWorkItemHistoryByProject(operation.Key, parameters,
                operation);
        }
    }
}
```

TFSWorkItemFactProxy

Code 4 – WorkItemFactProxy

```
// -----
// Microsoft Developer & Platform Evangelism
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//
// THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
// EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES
// OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.
// -----
// The example companies, organizations, products, domain names,
// e-mail addresses, logos, people, places, and events depicted
// herein are fictitious. No association with any real company,
// organization, product, domain name, email address, logo, person,
// places, or events is intended or should be inferred.
// -----

namespace Microsoft.Samples.DPE.ODataTFS.Model.Serialization
{
    using System;
    using System.Collections.Generic;
    using System.Globalization;
    using System.Linq;
    using System.Net;
    using System.Linq.Expressions;
    using System.Text;
    using System.Text.RegularExpressions;
    using System.Web;
    using Microsoft.Samples.DPE.ODataTFS.Model.Entities;
    using Microsoft.Samples.DPE.ODataTFS.Model.ExpressionVisitors;
    using System.Collections;
    using Microsoft.Data.Services.Toolkit.QueryModel;
    using Microsoft.TeamFoundation.WorkItemTracking;

    public class TFSWorkItemFactProxy : TFSBaseProxy, ITfsWorkItemFactProxy
    {
        public TFSWorkItemFactProxy(Uri uri, ICredentials credentials)
            : base(uri, credentials)
        {
        }

        public WorkItem GetWorkItem(int workItemId)
        {
            var wiql = string.Format(CultureInfo.InvariantCulture,
                                     "SELECT [System.Id] FROM WorkItems WHERE [System.Id] = {0}",
                                     workItemId);
            return this.QueryWorkItems(wiql)
                .Cast<TeamFoundation.WorkItemTracking.Client.WorkItem>()
                .Select(w => w.ToModel(this.GetTfsWebAccessArtifactUrl(w.Uri)))
                .FirstOrDefault();
        }

        public IEnumerable<WorkItemFact> GetWorkItemFactByProject(string projectName,
                                                                    FilterNode rootFilterNode,
                                                                    ODataQueryOperation operation)
        {
            var key = string.Format(CultureInfo.InvariantCulture,
                                     "TFSWorkItemFactProxy.GetWorkItemsFactByProject_{0}_{1}",
                                     projectName, this.GetFilterNodeKey(rootFilterNode));

            HttpContext.Current.Items[key] = this.RequestWorkItemFactsByProject(projectName,
                                                                                    rootFilterNode,
                                                                                    operation);

            return (IEnumerable<WorkItemFact>)HttpContext.Current.Items[key];
        }

        private static string BuildWiql(FilterNode rootFilterNode, ODataQueryOperation operation)
        {
            var constrains = string.Empty;

```

```

if (rootFilterNode != null)
{
    foreach (var filterNode in rootFilterNode)
    {
        constrains += AddComparisonConstrainToWiql(filterNode,
                                                    Constants.TFS.FieldLookup(filterNode.Key));
    }
}
if (constrains.StartsWith(" OR ", StringComparison.OrdinalIgnoreCase))
{
    constrains = constrains.Substring(" OR ".Length);
}

if (constrains.StartsWith(" AND ", StringComparison.OrdinalIgnoreCase))
{
    constrains = constrains.Substring(" OR ".Length);
}

var wiql = string.Format(CultureInfo.InvariantCulture,
                        "SELECT [System.Id] FROM WorkItems {0} {1}",
                        string.IsNullOrEmpty(constrains) ? string.Empty : "WHERE",
                        constrains).Trim();

string orderByWiql = string.Empty;
if (operation.OrderStack.Count > 0)
{
    orderByWiql = TFSWorkItemFactProxy.GenerateOrderByWiql(operation.OrderStack);
}

wiql = wiql + " " + orderByWiql;

return wiql;
}

private static string GenerateOrderByWiql(Stack<ODataOrderExpression> orderExpressionStack)
{
    if (orderExpressionStack == null)
    {
        throw new ArgumentNullException("orderExpressionStack");
    }

    StringBuilder finalOrderBy = new StringBuilder("ORDER BY ");
    //When the $top param is used with WCF Data Services Toolkit, this appears to also imply
    //ordering by key ascending (in this case work item ID), so if the user also issues an
    //explicit $orderby param for Id there will be a duplicate. It appears that the first
    //$oderby value (top of stack) is the one provided explicitly by the user, so we take that
    //one and track that it has been seen already. Subsequent order by fields are ignored.
    ISet<string> seen = new HashSet<string>();
    while (orderExpressionStack.Count > 0)
    {
        ODataOrderExpression expr = orderExpressionStack.Pop();
        UnaryExpression unExpr = ((UnaryExpression)expr.Expression);

        var opString = unExpr.Operand.ToString();
        var name = opString.Substring(opString.IndexOf('.') + 1);
        name = Constants.TFS.FieldLookup(name);

        if (!seen.Contains(name))
        {
            if (expr.OrderMethodName.StartsWith("ThenBy"))
            {
                finalOrderBy.Append(" , ");
            }
            finalOrderBy.Append(name);

            switch (expr.OrderMethodName)
            {
                case "OrderBy":
                case "ThenBy":
                    finalOrderBy.Append(" asc");
                    break;
                case "OrderByDescending":
                case "ThenByDescending":
                    finalOrderBy.Append(" desc");
                    break;
            }
        }
    }
}

```

```

    }

    seen.Add(name);
}
}
return finalOrderBy.ToString();
}

private static string AddComparisonConstrainToWiql(FilterNode filterNode, string tfsFieldName)
{
    if (filterNode != null)
    {
        var sign = default(string);

        switch (filterNode.Sign)
        {
            case FilterExpressionType.Equal:
                sign = "=";
                break;
            case FilterExpressionType.NotEqual:
                sign = "<>";
                break;
            case FilterExpressionType.GreaterThan:
                sign = ">";
                break;
            case FilterExpressionType.GreaterThanOrEqual:
                sign = ">=";
                break;
            case FilterExpressionType.LessThan:
                sign = "<";
                break;
            case FilterExpressionType.LessThanOrEqual:
                sign = "<=";
                break;
            case FilterExpressionType.Contains:
                sign = filterNode.Key.Equals("AreaPath", StringComparison.OrdinalIgnoreCase)
                    || filterNode.Key.Equals("IterationPath",
                        StringComparison.OrdinalIgnoreCase) ?
                    "UNDER" : "CONTAINS";

                break;
            case FilterExpressionType.NotContains:
                sign = filterNode.Key.Equals("AreaPath", StringComparison.OrdinalIgnoreCase)
                    || filterNode.Key.Equals("IterationPath",
                        StringComparison.OrdinalIgnoreCase) ?
                    "NOT UNDER" : "NOT CONTAINS";

                break;

            default:
                throw new NotSupportedException(string.Format(CultureInfo.InvariantCulture,
                    "WorkItem {0} can only be filtered with equal, not equal, greater than, lower than, greater than or equal, lower than or equal operators",
                    filterNode.Key));
        }

        return string.Format(CultureInfo.InvariantCulture, " {0} {1} {2} '{3}' ",
            filterNode.NodeRelationship.ToString(), tfsFieldName, sign,
            filterNode.Value);
    }

    return string.Empty;
}

private IEnumerable<WorkItemFact> RequestWorkItemFactsByProject(string projectName,
    FilterNode rootFilterNode, ODataQueryOperation operation)
{
    FilterNode newFilterNodeStartWithProject = new FilterNode()
    {
        Key = "Project", Sign = FilterExpressionType.Equal, Value = projectName
    };

    if (rootFilterNode != null)
    {
        newFilterNodeStartWithProject.AddNode(rootFilterNode);
    }
}

```

```

    }

    var wiql = BuildWiql(newFilterNodeStartWithProject, operation);
    return this.ExecuteWiqlRequest(wiql, operation);
}

private IEnumerable<WorkItemFact> ExecuteWiqlRequest(string wiql,
                                                    ODataQueryOperation operation)
{
    IEnumerable<WorkItemFact> retWorkItems = null;

    if (!operation.IsCountRequest)
    {
        if (operation.TopCount == 0)
        {
            //workaround for bug (I think) in WCF Data Services Toolkit.
            //It appears that ODataQueryOperation.TopCount will be 0 when $select param is used
            //and $top param is not explicitly sent by client (normally TopCount would have
            //whatever was set for entity page size in TFSService.InitializeService()
            operation.TopCount = Constants.DefaultEntityPageSize;
        }

        IEnumerable<TeamFoundation.WorkItemTracking.Client.WorkItem> wilist =
            this.QueryWorkItems(wiql).Cast<TeamFoundation.WorkItemTracking.Client.WorkItem>()
                .Skip(operation.SkipCount).Take(operation.TopCount)
                .Select(w => w).ToArray();

        List<WorkItemFact> lst = new List<WorkItemFact>();

        foreach (TeamFoundation.WorkItemTracking.Client.WorkItem w in wilist)
        {
            TeamFoundation.WorkItemTracking.Client.Revision prevRev=null;

            // Get All work item revisions
            foreach (TeamFoundation.WorkItemTracking.Client.Revision revision in w.Revisions)
            {
                if (prevRev!=null)
                {
                    WorkItemFact wiFactPrev = new WorkItemFact();
                    wiFactPrev.Project = w.Project.Name;
                    wiFactPrev.Id = w.Id;
                    wiFactPrev.Revision = (int)prevRev.Fields["System.Rev"].Value;
                    wiFactPrev.RevisionDate =
                        System.Convert.ToDateTime(revision.Fields["Changed Date"].Value).Date;
                    // Take the date for the next revision
                    wiFactPrev.RevisionCount=null;
                    wiFactPrev.RecordCount=-1;
                    wiFactPrev.SurrogateKey = wiFactPrev.Id + ":" + wiFactPrev.Revision;

                    // add numeric values to the fact
                    foreach (TeamFoundation.WorkItemTracking.Client.Field field
                        in prevRev.Fields)
                    {
                        switch (prevRev.Fields[field.Name].FieldDefinition.FieldType)
                        {
                            case TeamFoundation.WorkItemTracking.Client.FieldType.Integer:
                                wiFactPrev.SetField(field.Name,
                                    System.Convert.ToInt64(field.Value) * -1);
                                break;

                            case TeamFoundation.WorkItemTracking.Client.FieldType.Double:
                                wiFactPrev.SetField(field.Name,
                                    System.Convert.ToDouble(field.Value) * -1);
                                break;
                        }
                    }
                    lst.Add(wiFactPrev);
                }
                prevRev=revision;

                WorkItemFact wiFact = new WorkItemFact();
                wiFact.Project = w.Project.Name;
                wiFact.Id = w.Id;
            }
        }
    }
}

```

```

wiFact.Revision = (int)revision.Fields["System.Rev"].Value;
wiFact.RevisionCount = 1;
wiFact.RecordCount = 1;
wiFact.RevisionDate = System.Convert.ToDateTime(
    revision.Fields["Changed Date"].Value).Date;
wiFact.SurrogateKey = wiFact.Id + ":" + wiFact.Revision;

// add numeric values to the fact
foreach (TeamFoundation.WorkItemTracking.Client.Field field in revision.Fields)
{
    switch (revision.Fields[field.Name].FieldDefinition.FieldType)
    {
        case TeamFoundation.WorkItemTracking.Client.FieldType.Integer:
            wiFact.SetField(field.Name, System.Convert.ToInt64(field.Value));
            break;

        case TeamFoundation.WorkItemTracking.Client.FieldType.Double:
            wiFact.SetField(field.Name, System.Convert.ToDouble(field.Value));
            break;
    }
}

lst.Add(wiFact);
}
}
retWorkItems = lst;
}
else
{
    var workItemServer = this.TfsConnection
        .GetService<TeamFoundation.WorkItemTracking
        .Client.WorkItemStore>();

    try
    {
        Microsoft.TeamFoundation.WorkItemTracking.Client.Query q =
            new TeamFoundation.WorkItemTracking.Client.Query(workItemServer, wiql, null,
                false);

        int cnt = q.RunCountQuery();

        List<WorkItemFact> wiBlanks = new List<WorkItemFact>(cnt);
        WorkItemFact blank = new WorkItemFact();
        for (int i = 0; i < cnt; i++)
        {
            wiBlanks.Add(blank);
        }

        retWorkItems = wiBlanks;
    }
    catch (Microsoft.TeamFoundation.WorkItemTracking.Client.ValidationException ex)
    {
        throw new System.Data.Services.DataServiceException(500, "Internal Server Error",
            ex.Message, "en-US", ex);
    }
}
return retWorkItems;
}
}
}

```

WorkItemFactRepository

Code 5 - WorkItemFactRepositories

```
// -----
// Microsoft Developer & Platform Evangelism
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//
// THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
// EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES
// OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.
// -----
// The example companies, organizations, products, domain names,
// e-mail addresses, logos, people, places, and events depicted
// herein are fictitious. No association with any real company,
// organization, product, domain name, email address, logo, person,
// places, or events is intended or should be inferred.
// -----

namespace Microsoft.Samples.DPE.ODataTFS.Model.Repositories
{
    using System;
    using System.Collections;
    using System.Collections.Generic;
    using System.Data.Services;
    using System.Globalization;
    using System.Linq;
    using Microsoft.Data.Services.Toolkit.QueryModel;
    using Microsoft.Samples.DPE.ODataTFS.Model.Entities;
    using Microsoft.Samples.DPE.ODataTFS.Model.ExpressionVisitors;
    using Microsoft.Samples.DPE.ODataTFS.Model.Serialization;

    public class WorkItemFactRepository : IRepository<WorkItemFact>
    {
        private readonly ITfsWorkItemFactProxy proxy;

        public WorkItemFactRepository(ITfsWorkItemFactProxy proxy)
        {
            this.proxy = proxy;
        }

        public IEnumerable<WorkItemFact> GetAll()
        {
            List<WorkItemFact> lst = new List<WorkItemFact>();
            return lst;
        }

        public WorkItemFact GetOne(string id)
        {
            return null;
        }

        [RepositoryBehavior(HandlesFilter = true, HandlesSkip = true, HandlesTop = true,
            HandlesOrderBy = true)]
        public IEnumerable<WorkItemFact>
            GetWorkItemFactsByProject(ODataSelectManyQueryOperation operation)
        {
            if (operation == null)
            {
                throw new ArgumentNullException("operation");
            }
            var parameters = new WorkItemFilterExpressionVisitor(operation.FilterExpression).Eval();
            return this.proxy.GetWorkItemFactByProject(operation.Key, parameters, operation);
        }
    }
}
```

Setting up Alternative Credentials for VS Online

The procedure of setting alternative credentials is described at <http://tfodata.visualstudio.com> and is a prerequisite for the walkthroughs in this guide.

Step	Instructions
1 Optional Enable and configure basic authentication credentials on tfs.visualstudio.com <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Navigate to the account that you want to use on https://tfs.visualstudio.com. For example, you may have https://account.visualstudio.com. In the top-right corner, click on your account name and then select My Profile. Select the Credentials tab. Click the Enable alternate credentials and set password link, Enter a password. It is suggested that you choose a unique password here (not associated with any other accounts), Click Save Changes.
2 Configure authentication against OData service <input type="checkbox"/> - Done	<ul style="list-style-type: none"> To authenticate against the OData service, you need to send your basic auth credentials in the following domain\username and password format: <ul style="list-style-type: none"> account\username password The account is from account.visualstudio.com, username is from the Credentials tab under My Profile, and password is the password created in step 1.

Table 14 – Setup alternative credentials