

TFS Version Control

Part 2

TFVC Gems

Visual Studio ALM Rangers



Microsoft



Visual Studio

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, you should not interpret this 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.

© 2014 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
Team Foundation Version Control	6
Version Control Folder Structure.....	6
Workspaces	9
Merging	12
Tracking and Administration	12
Baseless Merging	13
Hands-on Lab (HOL) – Workspaces & Merging	16
Prerequisites.....	16
Exercise 1: Create a new local workspace	17
Exercise 2: Use the local workspace in online mode.....	18
Exercise 3: Use the local workspace in offline mode.....	21
Exercise 4: Explore the diff and merge tools	22
Appendix: Environment Setup	27
FAQ	28
In Conclusion	29

Foreword

Since the first writing of the TFS branching guide, a lot has changed in the world of version control. Hosted version control solutions are everywhere, and many of them include integration with build, project tracking, and other services. Distributed Version Control is no longer a niche, and has changed the way that many developers think about what it means to version their code. More developers are using version control than ever before – and this is a great thing for the billions of end users of those software development projects.

More developers are using version control. This means developers need solid, practical, and easy-to-digest guidance that is industry proven now more than ever. This guide, and those that came before it, strive to do just that – provide the version control guidance that development teams need to be effective while being approachable and flexible. In this latest edition, we have streamlined the guidance and simplified concepts with the goal of helping teams of all shapes and sizes to develop strategies that enable the flexibility and agility that modern development teams demand.

I also need to mention that this guide would not be in its current form without the readers. Thanks to all of you whom have contributed your feedback and ideas that have helped shape this guide over the years. As is in the past, if you see something in this guide that you would like to see changed or improved, please let us know!

Happy versioning!

Matthew Mitrik – Program Manager, Cloud Dev Services

Introduction

This guide aims to provide an insight and practical guidance on how to use Team Foundation Version Control (TFVC) features, such as workspaces, merging, and new features such as Code Lens. It is one of a set of companion guides as outlined in the “What’s New” chapter of **TFS Version Control Part 1 – Branching Strategies**.

Intended audience

In this guide, we primarily address Garry, the development lead, Doris, the developer, and Dave, the TFS administrator. See [ALM Rangers Personas and Customer Profiles](#)¹ for more information on these and other personas.

Visual Studio ALM Rangers

The Visual Studio ALM Rangers provide professional guidance, practical experience, and gap-filling solutions to the ALM community. They are a special group with members from the Visual Studio Product group, Microsoft Services, Microsoft Most Valuable Professionals (MVP), and Visual Studio Community Leads. Membership information is available [online](#)².

Contributors

Anna Galaeva, Gordon Beeming, James Waletzky, Malcolm Hyson and Michael Fourie.

A special thank you to the ALM Ranger teams who laid the foundation with v1 and v2: Anil Chandr Lingam, Bijan Javidi, Bill Heys, Bob Jacobs, Brian Minisi, Clementino de Mendonca, Daniel Manson, Jahangeer Mohammed, James Pickell, Jansson Lennart, Jelle Druyts, Jens Suessmeyer, Krithika Sambamoorthy, Lennart Jansson, Mathias Olausson, Matt Velloso, Matthew Mitrik, Michael Fourie, Micheal Learned, Neno Loje, Oliver Hilgers, Sin Min Lee, Stefan Mieth, Taavi Koosaar, Tony Whitter, Willy-Peter Schaub, and the ALM Community.

Using the sample source code, Errata and support

All source code and revisions of this guide are available for download from the CodePlex site [Visual Studio Version Control Guide](#)³. You can contact the team under the Discussions tab there.

Additional ALM Rangers Resources

[Understanding the ALM Rangers](#)⁴

[Visual Studio ALM Ranger Solutions](#)⁵

¹ <http://vsarguidance.codeplex.com/releases/view/88001>

² <http://aka.ms/vsarindex>

³ <http://aka.ms/treasure18>

⁴ <http://aka.ms/vsarunderstand>

⁵ <http://aka.ms/vsarsolutions>

Team Foundation Version Control

Version Control Folder Structure

Folders are an organizational tool in TFS that provide a container for branches, other folders, or files. It is important to have a consistent folder structure across the source tree. A clear set of conventions helps team members find their way around in unfamiliar areas of the product. It also lets you use the same build practices and scripts across features and components.

The best folder structure depends which branching model your team chooses. (We discussed branching models in Part 1 of this guide.) For simplicity, this section assumes you are using Service and Release Isolation, one of the most common branching structures, although we will mention some of the others as well.

We can consider two levels of folder structure: global across the whole project, and local to individual features and components.

Global Folder Structure

NOTE

“Global” folder structure refers to the general organization of your source repository as viewed in Source Control Explorer. This discussion is specific to TFS Version Control and does not apply to Git.

Keep it simple. There are lots of different possible folder structures, and no single right answer. Be thoughtful about how you organize the repository, and regularly reevaluate the structure to ensure it is meeting the needs of the users. Two common pivot points for folder structures are release and branch purpose.

Arranging By Release

Assuming a product called “Product A” with two previous releases and a team working on the current release, arranging by release would yield a structure similar to the following:

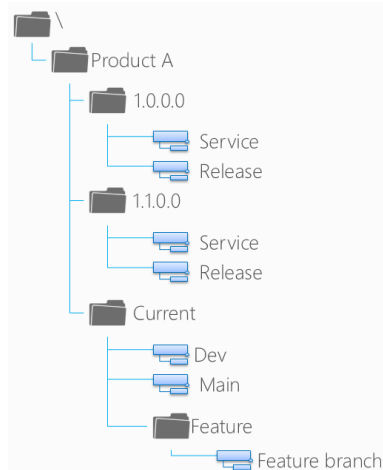


Figure 1 – Source control folders arranged by release

Advantages

- Easy to find release branches (closely mirrors the branch structure)
- If a developer is not working on a particular release, it is easy to cloak the whole folder, saving bandwidth and local disk space on “get” operations
- Reflects the branch structure a little more intuitively as people generally think release-first

Disadvantages

- The Release folder is not retrieved very often, so you may have to cloak a bunch of release branches
- Feature branches, if used, get quite deep down the hierarchy. You can run into the limitation on TFS path lengths.

NOTE

The “Current” folder is not strictly necessary, but keeps consistency with folder purposes. The “Feature” folder would only be necessary for feature isolation (e.g. risky feature that might not ship). You may also consider putting the “Feature” folder at the same level as current if they are not release-dependent.

Arrange By Purpose

Assuming a product called "Product A" with two previous releases and a team working on the current release, arranging by purpose would yield a structure similar to the following:

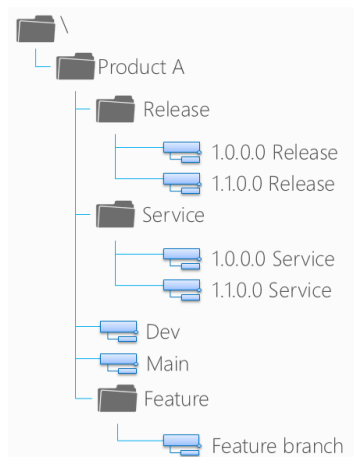


Figure 2 – Source control folders arranged by purpose

Ignore the "Product A" level of the tree if you only have one product in source control.

In general, arranging projects either by release or by purpose works well at the global level. Understand the advantages and disadvantages of the model, and consider what works best for your team.

Local Folder Structure

NOTE

"Local" structure refers to the organization of folders under a particular source project or feature. This discussion applies to both TFS Version Control (TFVC) and Git unless otherwise noted.

One should also consider local folder structures under each individual project or feature in the source tree. For example, Dev and Main/Master branches each likely have various features or components stored under them and each should have a standard folder structure within. This approach makes it easier to get familiar with and work across projects in the tree. An example structure may look like the following:

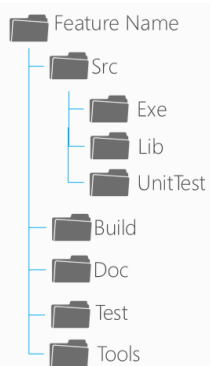


Figure 3 – Source control folders for a feature

TFVC Gems – Team Foundation Version Control

Folder	Purpose
<Feature Name>	In TFVC, this is the root folder for some component of work, which may be a feature or component in the product, an architectural layer, or anything else relevant to your structure. In Git, distinct branches typically represent features, so a folder like this may not exist.
Src	All source code relating to the feature that is included with the product or service.
Exe	[Optional] Code that builds the executable for an application or component. Note that a good architecture is to have a stub executable with helper libraries/DLLs that make up the product. The separation of concerns improves overall testability, thus the recommendation of a stub EXE that uses helper libraries.
Lib	[Optional] Individual components of the feature that are testable on their own. This folder may include, for example, product assemblies or native code libraries.
UnitTest	All unit tests associated with the feature. Unit tests should always live alongside the main source code and you should never cloak the unit test folder. We do not recommend keeping unit tests in a separate tree outside the product source code, as the development team owns them. Unit tests should be of high quality and build alongside product code. Solution files should include the unit test projects as well as the feature source code.
Build	[Optional] Any build or deployment scripts specifically related to this feature.
Test	[Optional] Any QA-owned integration tests, load tests, performance tests, and so on that validate this feature. Although it is acceptable to keep QA-owned tests in a separate tree, it is better to keep them with the feature source to form a cohesive whole. However, shared libraries between teams should be stored in a separate location in the source tree and not stored in context of any one feature.
Tools	Any tools specific to the feature, such as a code generator.

Table 1 – Suggested sub-folders under a feature or component.

Workspaces

Visual Studio Team Foundation Server 2012 and later provides two choices of workspaces – a **server workspace** and a **local workspace**. It is an administrative choice that you make for a whole team project collection. This section briefly reviews what a workspace is and provides considerations on choosing whether to use a server or local workspace.

NOTE

Also refer to [Team Foundation Server – Trying to understand Server versus Local Workspaces](#)⁶ and the supporting quick reference Branching and Merging Guide - Terminology Cheat Sheet for reference information.

What is a Workspace?

According to the [documentation](#)⁷, “Your workspace is a local copy of your team’s codebase. Your workspace enables you to develop and test your code in isolation on your dev machine until you are ready to check in your work.”

A Workspace maps Server folders to Local folders. This also includes:

- A list of all the files in your workspace
- The version of each file
- A list of the pending changes
- Active and cloaked items

Server workspaces

Team Foundation Server 2010 and earlier stored the mapping of Server folders to Local Folders on the Team Foundation Server. We call this approach **Server workspaces**. There is a local cache for this information, but we manage this centrally to allow coordination of multiple persons while working in either connected or disconnected mode.

Server workspaces allow you to perform operations such as branching directly on the server side, with minimal interaction with the client. Because the server stores this information, other users who have the correct permissions on the server and can see the same folders in source control can duplicate your working environment on their computers. However, they will not be able to see the details of any pending changes.

Server workspaces can be Private (can only be used by owner), Public (any valid user has read/write access) or Public Limited (any valid user has read access but only owner has write access). Team Foundation Server also keeps this information on the server.

NOTE

With the server workspace model, you cannot start any kind of change unless you have a connection to the server. Adding, editing, renaming, deleting files – all of these things require a server connection before you can begin them. Once you pend a change, then you must check- in the change to the server in order to share it with others.

Local Workspaces

With Local workspaces, you make changes in your local file system (no server connection required) using whatever tools you like, and then synchronize with the server later. It does not matter if you are offline while you are updating your code – you can work on the bus and sync later. Team Foundation Server works out the changes that you made and handles them appropriately.

⁶ http://blogs.msdn.com/b/willy-peter_schaub/archive/2011/11/30/team-foundation-server-trying-to-understand-server-versus-local-workspaces.aspx

⁷ <http://msdn.microsoft.com/en-us/library/cc138514.aspx>

In configuration management parlance, we call this **Modify-Merge-Commit** style version control (informally also known as “**Subversion style**”). Features:

- Files are not read-only. You can edit them at will and there is no need to do an explicit checkout operation. When you edit files, TFS automatically detects changes and sets the file status to “checked out.”
- If you create new files, Team Foundation Server will detect them and allow you to add them to your project.
- If you delete files locally, Team Foundation Server will notice and give you the option of either deleting them from Team Foundation Server or restoring the local copy from the server.
- You no longer need to use a tool with version control integration for the right thing to happen (because the local file system is the master) so you can use Notepad or any other tool and Team Foundation Server will work the way you’d expect.
- Diff and other file operations are still possible as you have a cached copy of the workspace.

Server Workspaces vs. Local Workspaces

The following table outlines the differences between Server and Local workspaces in Team Foundation Server.

Criteria	Server workspace	Local workspace
Network connection	Required*	Not required
Pending change visibility to team members	Visible	Not visible
Changes automatically pended	No	Yes
High performance with large projects	Yes	May degrade with large number of files
Compatibility with Visual Studio 2010 and earlier	Yes	No
Enforceable check-out locks	Yes	No

Table 2 - Server vs. Local Workspace

NOTE

*You can still work offline with a server workspace, but Source Control Explorer is unavailable. Consider using a Local workspace instead if working offline is a frequent scenario.

Deciding between Local vs. Server Workspaces

We recommend Local workspaces in most scenarios. Use a Server workspace if you:

- Need to scale Team Foundation Server source control to very large projects, and local workspaces are causing performance issues due to the large amount of data kept locally for local workspaces.
- Need to maintain VSS-inherited practices (single check out, get latest on checkout)

Configuring Workspaces

You can specify if you want to use Local vs. Server workspaces. However, this you can only make this choice at the Team Project Collection level. This requires coordination among the teams that belong to a collection so they can transition to the chosen type.

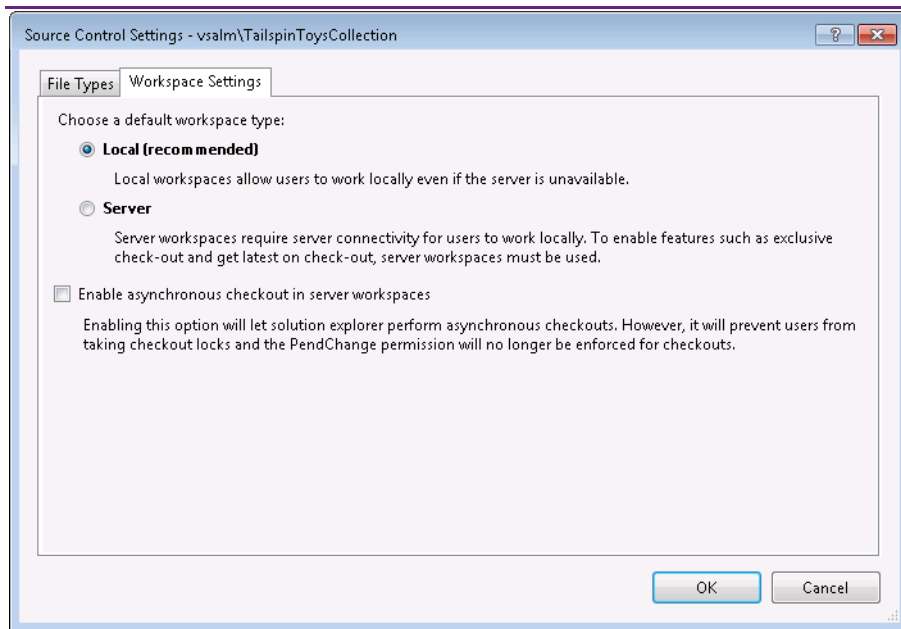


Figure 4 – Workspace settings

NOTE

Server workspaces are still available but local workspaces are now the default for new installations of TFS. When upgrading from TFS 2010, the default setting is to keep server workspaces for backwards compatibility. You can change that default via the Team Project Collection Source Control Settings in Team Explorer. Once you make this change, users of that collection receive a prompt to convert their existing server workspace to a local workspace when they navigate to Pending Changes for a loaded project connected to a server workspace. This is a one-time operation and may take some time on large source bases.

Usage Scenarios

Offline scenario

For Server workspaces, Team Foundation Server 2010, 2012 and 2013 have the same functionality.

For Local workspaces:

- Responsiveness is quick since the file system is the master and you do not need Team Foundation Server to checkout anything. Many operations are asynchronous, such as starting to edit a file and comparing files.
- TFS caches files that you are working on locally, and supports some offline file operations. For example, you can perform the diff operation offline so you can see what changes you have made and undo – so you can revert your changes if you want to start over.
- Team Foundation Server 2012 added the concept of local pending changes so you can still do *tf status* (and see the status in Visual Studio) while offline and see all the files that you've edited (even though you never told Team Foundation Server you were editing them).
- You can also perform *tf delete*, *tf add* or *tf rename* while in a disconnected state, and when you connect back to the server, everything automatically synchronizes and it was as if you always in a connected state.

NOTE

Certain operations still require you to be online:

- You cannot check-in while you are offline. (Not full Distributed Version Control such as Git)
- You cannot view history, branch and merge.

For those operations, you will immediately get error messages that tell you need to be online.

Merging

Tracking and Administration

Tracking and administration of changesets and defects is the highest cost associated with creating additional branches. It is critical that the team remain on top of the merge 'debt' in their branch plan, and install processes to measure and address this debt.

If you have a development and release isolation plan (formerly known as the basic branch plan) like the one illustrated below, then keeping track of merge debt is easy. All you do in this case is ensure that you merge any fixes through main and into Dev where you can be integrate them with the upcoming functionality. There will be little need to change any out-the-box process templates or create any automation to report debt. You can use standard Visual Studio IDE features to administer this scenario.

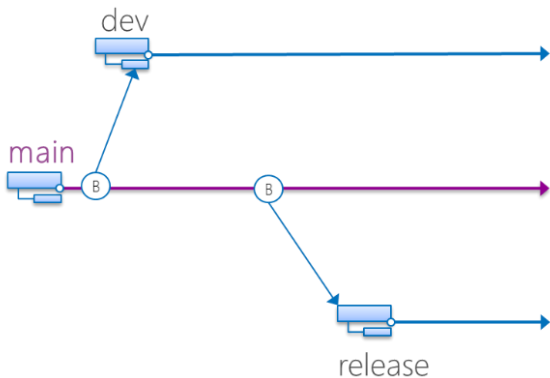


Figure 5 - Tracking basic merge debt

In the advanced branching scenario below, REL A is in production; while you work on a service pack in SER A. REL B1 is with the customer for early industry testing. SER B is taking feedback from REL B1 as well as implementing new functionality that you will merge with changes coming from fixes to REL A and the service pack features in SER A before you branch and deliver these as REL B2. While this is all happening, you will need to merge (forward integrate) anything that merges through Main into the feature branch FEA 1. FEA1 will ship to the customer some time as an unsupported release before being reverse integrated to main later for formal production release. You cannot manage this level of complexity without some level of automation or changes to process templates.

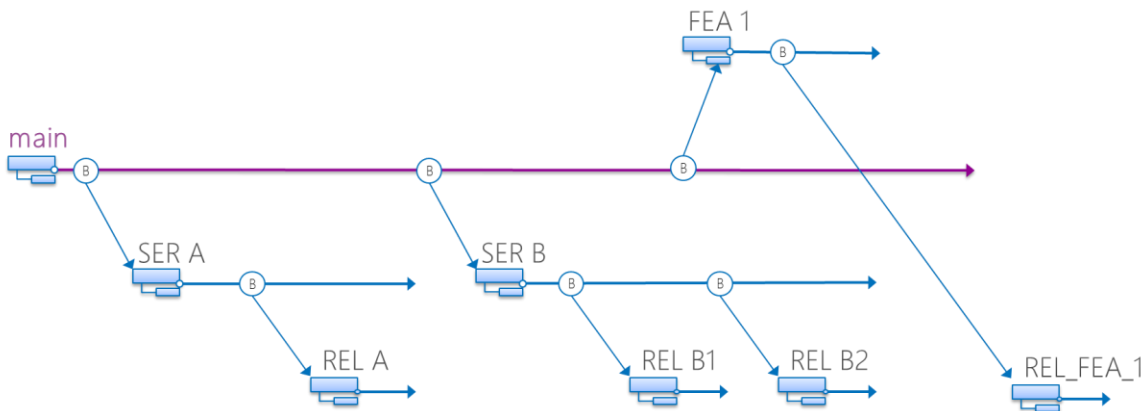


Figure 6 - Tracking advanced merge debt

If you have multiple active release vehicles, one approach to defect management is to create a duplicate bug work item for each branch where the fix is required. However, it is easier to keep a single bug, with additional fields to record which releases require the fix. Update the template for the Bug work item with the additional fields. For example, you may have an entry for each active branch: DEV, MAIN, RELEASE(S); and values of Merged, Merge Required, and Merge Not Required. This allows your defect manager to easily report on the status of defects across active release vehicles.

In complex branching scenarios, we recommend you automate the diagnosis and reporting of merge debt. You can do this in various ways. A simple way is to use tf.exe to run across all active merge paths to report the state of affairs. The sample below provides an MSBuild based solution to this. It runs a merge preview over a collection of merge paths.

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- Create a collection of active merge paths -->
  <ItemGroup>
    <MergePath Include="DevToReleaseA">
      <SourceBranch>$/ProjectName/Development</SourceBranch>
      <TargetBranch>$/ProjectName/Releases/RelA</TargetBranch>
    </MergePath>
    <MergePath Include="ReleaseBtoDev">
      <SourceBranch>$/ProjectName/Releases/RelB</SourceBranch>
      <TargetBranch>$/ProjectName/Development</TargetBranch>
    </MergePath>
  </ItemGroup>
  <!-- Set which workspace to operate in and whether to ignore conflicts or not -->
  <PropertyGroup>
    <WorkspacePath>C:\YourWorkspacePath</WorkspacePath>
    <IgnoreConflicts>true</IgnoreConflicts>
  </PropertyGroup>
  <Target Name="Report" Inputs="@ (MergePath)" Outputs="% (Identity) ">
    <Message Text="% (MergePath.SourceBranch)" to "% (MergePath.TargetBranch) "' />
    <Exec Command='tf merge "% (MergePath.SourceBranch)" "% (MergePath.TargetBranch)" /preview
/recursive' WorkingDirectory='$(WorkspacePath)' IgnoreExitCode="$(IgnoreConflicts)"/>
  </Target>
</Project>
```

Alternatively, Team Foundation Server provides a rich API to report (VersionControlServer.GetMergeCandidates) and action (Workspace.Merge) merge debt. You could write a simple UI to help monitor and manage the debt. It would allow a member of the team to determine how many changes are pending and how many conflicts; auto merge, create a merge task and assign it to a member of the team if conflicts are found; and perhaps allow you to discard changesets which are not intended for merging. By keeping the merge paths 'clean' you can be more confident that fixes and new work have been integrated into their peer branches.

Baseless Merging

Normally, developers perform bi-directional merges between branches that share a parent-child relationship. On occasion, you need to perform a merge between two branches that do not yet have a merge relationship. We call these types of merges baseless merges. A baseless merge creates a merge relationship between a source and target that is not part of a direct parent-child branching relationship.

Baseless merging should be the exception rather than the rule. If you find yourself frequently using baseless merges, revise the overall branching strategy. However, there are legitimate reasons for doing a baseless merge, so it you should not rule them out completely. For example, you might urgently need a hotfix in production just when they are performing a testing cycle in MAIN: so you want to move the fix without going through MAIN.

Disadvantages of Baseless Merging

Since there is no base version, 3-way merging is not possible in a baseless merge. A 3-way merge compares both source and target files to the base version allowing a better merge experience such as auto-merging. Auto merging is not possible with baseless merges, so you need to resolve any conflict manually.

Prior to Visual Studio 2010 SP1, TFS does not merge files that are in a delete status in the source branch to the target branch.

Merge Scenario

In this scenario shown below, there are three DEV branches from MAIN. Within this structure, if you need to merge a change from DEV FT2 to DEV FT1, the normal path would be a reverse integration merge from DEV FT2 to MAIN, followed by a forward integration merge from MAIN to DEV FT1. However, if it is not possible to merge the code to MAIN, the only way to move it from DEV FT2 to DEV FT1 is through a baseless merge.

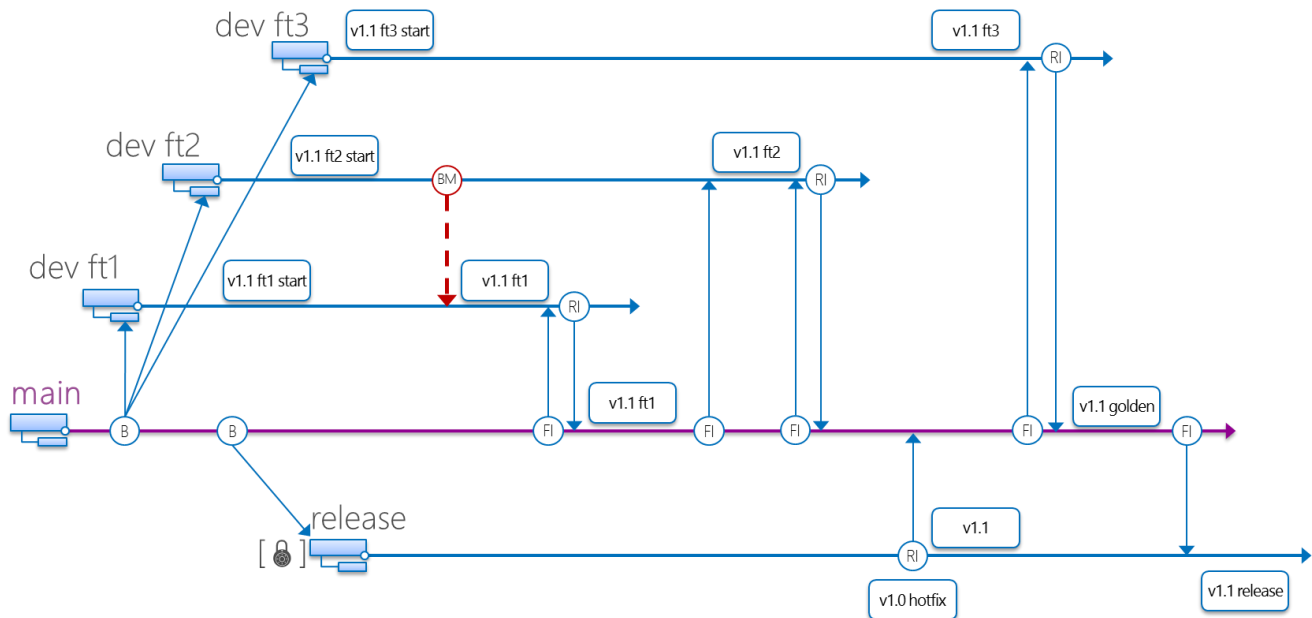


Figure 7 - Baseless Merge between DEV FT1 and DEV FT2

Since there is no direct branching relationship between DEV FT2 and DEV FT1, you must first perform a baseless merge and manually resolve any merge conflicts. Once a baseless merge is completed, Team Foundation Server records a merge history and establishes a branching relationship. In future merges, you will have a 3-way merge, resulting in fewer conflicts.

Baseless Merging in Visual Studio

In Visual Studio 2013, you can do baseless merges in Source Control Explorer. In the merge window, use the Browse button to select the target branch. If your selected source and target don't have a direct branching relationship, a warning icon and message will indicate that a baseless merge will happen.




Select the source and target branches for the merge operation

Select the branch that contains the changes you would like to merge.

Source branch:

Select the source branch changes you would like to merge:
☒ All changes up to a specific version
☐ Selected changesets

Select the target branch for the merge operation. The drop-down list contains all target branches applicable to the selected source branch.

Target branch: 

A merge relationship does not exist between the source and the target. A baseless merge will be performed.

< Previous **Next >** Finish Cancel

Figure 8 - Baseless Merge Warning

Since In Team Foundation Server 2010, you could not do a baseless merge using Source Control Explorer. In the Visual Studio command prompt, type:

```
tf merge /baseless <<source path>> <<target path>> /recursive
```

You can also specify which changesets to apply. For example, to do a baseless merge of changeset 150 from DEV FT2 to DEV FT1:

```
tf merge /baseless /v:C150~C150 "$/TeamProjectA/DEV FT2" "$/TeamProjectA/DEV FT1" /recursive
```

To do a baseless merge of all changesets up to 150 from DEV FT2 to DEV FT1:

```
tf merge /baseless /v:C150 "$/TeamProjectA/DEV FT2" "$/TeamProjectA/DEV FT1" /recursive
```

You can do subsequent merges between these branches by using Source Control because their relationship will exist.

If you perform a baseless merge with "All changes up to a specific version," when checking in a baseless merge every file will have a merge status next. This is because a baseless merge creates a branching relationship for every file in the parent folder (DEV FT1). If you selected "Selected changesets," the Pending Changes window will only show the files included in the changeset. TFS will not establish a branching relationship between DEV FT2 and DEV FT1, because TFS will only establish a branching relationship between the files that are included in the changeset.

The Figure below shows the branching relationship between DEV FT2 and DEV FT1 after you perform a baseless merge using the "All changes up to a specific version" option in the Merge wizard.

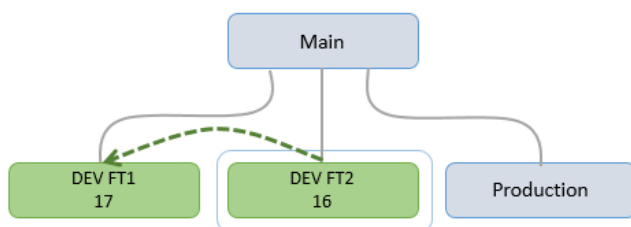


Figure 9 – Tracking shows Baseless Merge

Hands-on Lab (HOL) – Workspaces & Merging

Using **Visual Studio 2013**, we will create a new local workspace and will explore its operation in both online and offline modes. We will then investigate the functionality of the diff and merge tools and then finally dive into the new Code Lens feature.

IMPORTANT

We have based this Hands-on Lab (HOL) on the new HOL template, which assumes that you are an experienced Visual Studio ALM user.

- “How to” instructions, for example starting Visual Studio and creating a Team Project, are no longer included. We encourage you to visit <http://msdn.microsoft.com> and other product documentation for that information.
- We have reduced and focused the screenshots, to reduce the maintenance effort as the products evolve.

Prerequisites

To complete this Hands-On Lab you need the following environment:

- [Visual Studio 2013 Application Lifecycle Management Virtual Machine from Brian Keller](#) ⁸

Or

- A single server environment (physical or virtual) that has the following software installed and configured:

Software	Version
Operating System	Windows 8, Windows Server 2012 (or later)
Visual Studio	2013
Visual Studio Team Foundation Server	2013

Table 3 - Hands-On Lab Prerequisites

Please refer to the Appendix on page 27 for additional information about configuring the environment for this Hands-On Lab.

Suggested Focus and Time Map

If you intend to follow this Hands-On Lab (HOL) step-by-step, use these times as a guideline. If, however, you are intending to investigate each step in detail, then double these times as a bare minimum.

Topic	Duration in minutes	Page
Exercise 1 – Create a new local workspace	5	17
Exercise 2 – Use the local workspace in online mode	15	18
Exercise 3 – Use the local workspace in offline mode	15	21
Exercise 4 – Explore the new diff and merge tools	15	22
	TOTAL 50 min	

Table 4 - Suggested Focus and Time Map

⁸ <http://aka.ms/almvms>

Exercise 1: Create a new local workspace

GOAL

In this exercise, we connect to Team Foundation Server and create a local workspace.

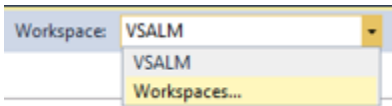
Task 1: Logon to your environment

Step	Instructions
Logon ☐ - Done	<ul style="list-style-type: none"> If you are logging into an instance of BK's VM, i.e. at TechReady, login using the Administrator P2ssw0rd credentials. Alternatively, login using your own environment credentials.

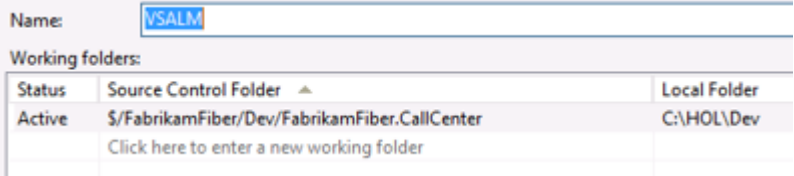
Task 2: Connect to Team Foundation Server

Step	Instructions
1 Open Select Team Project ☐ - Done	<ul style="list-style-type: none"> In Team Explorer click on Select Team Projects
2 Select Team Foundation Server ☐ - Done	<ul style="list-style-type: none"> Select the server localhost, If a server doesn't currently exist for localhost add a new server to the location http://localhost:8080/tfs
3 Connect Team Project ☐ - Done	<ul style="list-style-type: none"> Choose FabrikamFiberCollection Check FabrikamFiber Click Connect

Task 3: Create the local workspace

Step	Instructions
1 Open Workspaces ☐ - Done	<ul style="list-style-type: none"> In Team Explorer, click the Source Control Explorer link. At the top of Source Explorer, select the Workspace drop down and then Workspaces. 
2 Select Working Folder ☐ - Done	<ul style="list-style-type: none"> Click Edit Under Working folders click <i>Click here to enter a new working folder</i> Click the ellipses (...) Expand localhost\FabrikamFiberCollection Expand FabrikamFiber Expand Dev Select FabrikamFiber.CallCenter Click OK
3	<ul style="list-style-type: none"> Click in the space directly underneath Local Folder Click the ellipses (...)

TFVC Gems – Hands-on Lab (HOL) – Workspaces & Merging

Step	Instructions
Select Local Folder <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Navigate to and select the folder c:\hol\Dev (create this folder if it does not already exist) Click OK 
4 Map and Get latest <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Click OK Click Yes when prompted that the workspace has been modified Finally, click Close

REVIEW


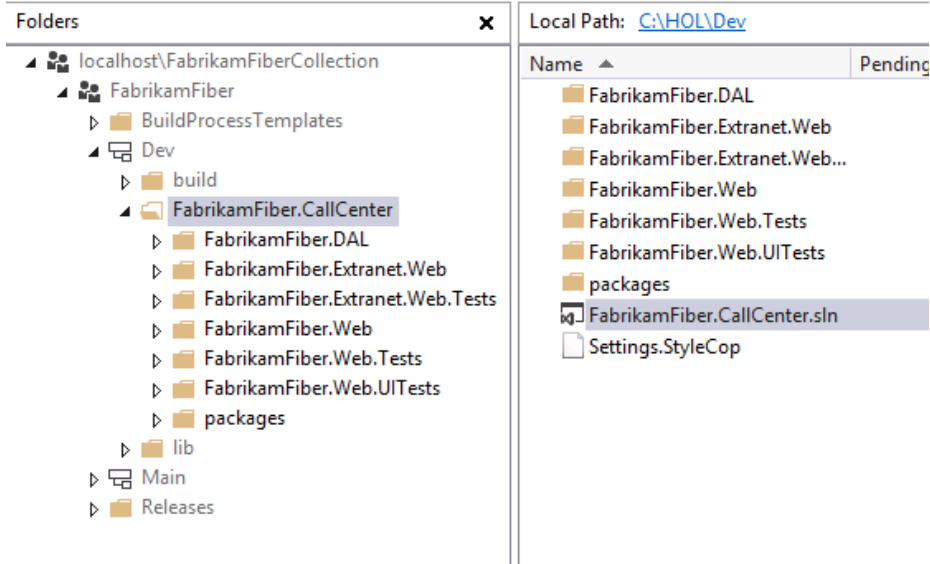
We have connected to Team Foundation Server, opened the FabrikamFiber team project, and created a local workspace.

Exercise 2: Use the local workspace in online mode

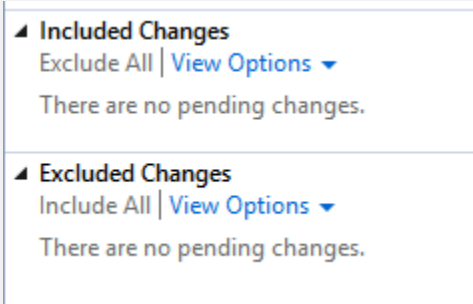
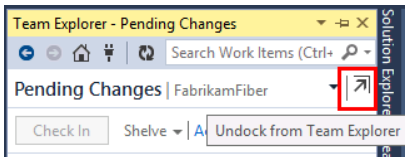
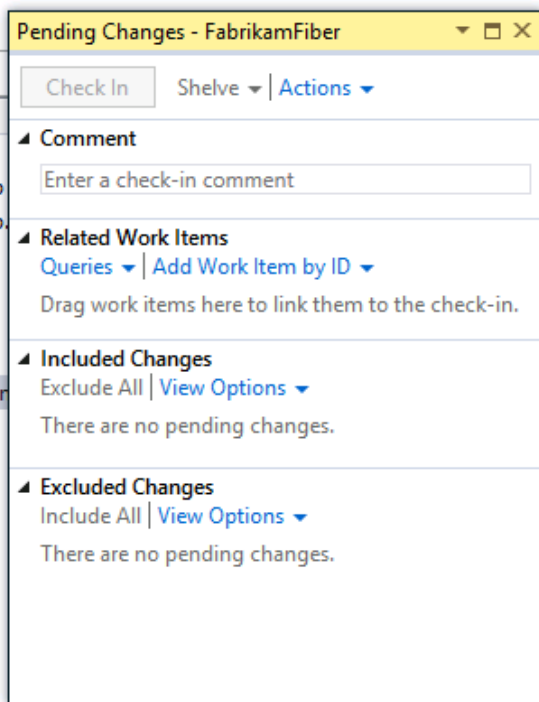
GOAL

In this exercise, we explore the functionality of local workspaces when working in online mode.

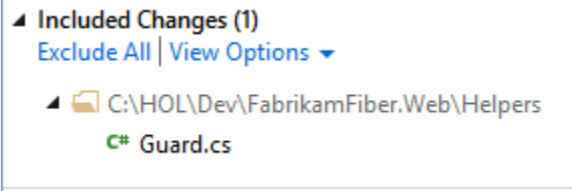
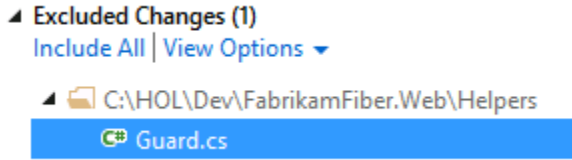
Task 1: Open the FabrikamFiber Call Center solution

Step	Instructions
Open FabrikamFiber.CallCenter.sln <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Open Source Control Explorer to FabrikamFiber\Dev\FabrikamFiber.CallCenter Double-click FabrikamFiber.CallCenter.sln. <p>Source location:  \$/FabrikamFiber/Dev/FabrikamFiber.CallCenter</p> 


Task 2: Make changes using Visual Studio

Step	Instructions
1 Open a C# source code file ☐ - Done	<ul style="list-style-type: none"> Open Solution Explorer Expand the FabrikamFiber.Web project Expand the Helpers folder Double-click Guard.cs to view the C# source code
2 Open Team Explorer ☐ - Done	<ul style="list-style-type: none"> Open Team Explorer Click the Pending Changes link. Note the collapsible sections for the Included Changes and the Excluded Changes 
3 You can Undock Pending Changes ☐ - Done	<p>Note that new to Visual Studio 2013 is the ability to undock the Pending Changes window from Team Explorer, you can do this by clicking the arrow in the top right corner of the pending changes section .</p>  <p>What the window looks like after being undocked</p> 

TFVC Gems – Hands-on Lab (HOL) – Workspaces & Merging

Step	Instructions
4 Modify the C# source code <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Add a comment to (or otherwise modify) the source code for the Guard class Save the file Note that TFS immediately lists the file in the Include Changes section. 
5 Exclude Guard.cs from the list of pending changes <input type="checkbox"/> - Done	<p>Drag the entry for Guard.cs from Included Changes and drop it onto <i>Drag changes here to exclude from the check-in</i> under Excluded Changes. Note that the file is now listed in the Excluded Changes section</p> 
6 Undo all pending changes <input type="checkbox"/> - Done	<ul style="list-style-type: none"> In Pending Changes click Actions, Undo All In Undo Pending Changes dialog click Undo Changes In Confirm Undo Checkout message box, click Yes to All. Note that the change is undone and the file is removed from Excluded Changes

Task 3: Make changes outside of Visual Studio

Step	Instructions
1 Open the FabrikamFiber.Web folder in Windows Explorer <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Open Source Control Explorer Browse to the FabrikamFiber.Web folder Click on the link next to the label Local Path to open the location to which FabrikamFiber.Web mapped in windows explorer, which should be c:\hol\Dev\FabrikamFiber.Web. 
2 Make changes to the project files without using Visual Studio <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Double-click readme.txt to open the file in Notepad then make a change and save it. Use Windows Explorer to delete the file Web.config. Use Windows Explorer to add a new file Notes.txt. Double-click the Helpers folder, right-click Guard.cs, choose Open with, Notepad, make and save a change.
3 Reload changes in Visual Studio <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Switch back to Visual Studio Click Yes when prompted to reload the modified version of Guard.cs
4 Review the detected changes <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Note that the project files that were modified are already listed under Included Changes, even though the changes were made outside of Visual Studio Note too that the Excluded Changes section now includes a link Detected: 1 add(s), 1 delete(s).

TFVC Gems – Hands-on Lab (HOL) – Workspaces & Merging

Step	Instructions
5 Promote Excluded Changes ☐ - Done	<ul style="list-style-type: none"> Click the Detected: 1 add(s), 1 delete(s) link. Note that Promote Candidate Changes lists the other files that were modified Click Promote. All the changes we made will now be in the Include Changes section
6 Undo Deleted file change ☐ - Done	<ul style="list-style-type: none"> In Included Changes right-click the entries for the deleted Web.config (and Web.Debug.config and Web.Release.config if they exists in the list) Click Undo In Undo Pending Changes dialog, click Undo Changes. The deleted Web.config file is restored and is removed from the list of Included Changes
7 Clean up ☐ - Done	<ul style="list-style-type: none"> In Pending Changes click Actions, Undo All In Undo Pending Changes dialog click Undo Changes In Confirm Undo Checkout, click Yes to All. All changes are undone and are removed from Included Changes Complete the clean-up by using Windows Explorer to delete the Notes.txt file that you created

REVIEW

We have opened the FabrikamFiber CallCenter solution and have made changes to it from inside and outside of Visual Studio.

Exercise 3: Use the local workspace in offline mode

GOAL

In this exercise, we explore the functionality of local workspaces when working in offline mode.

With local workspaces in offline mode, you can quickly begin editing a file when your network connection is unavailable or unreliable.

Task 1: Disconnect from Team Foundation Server

Step	Instructions
Stop IIS to disconnect from TFS ☐ - Done	<ul style="list-style-type: none"> Open a Command Prompt with administrative privileges Execute iisreset /stop from the administrative level Command Prompt. The message <i>Internet services successfully stopped</i> should be displayed

Task 2: Make changes inside and outside of Visual Studio

Step	Instructions
1 Make changes using Visual Studio ☐ - Done	<p>Repeat Exercise 2 Task 2 to make changes to Guard.cs from the FabrikamFiber.Web project using Visual Studio</p> <div> <div>NOTE</div> <p>If you receive a TF400324: Team Foundation services are not available from server localhost\DefaultCollection message then dismiss it; the message is simply notifying you that the Team Foundation Server cannot be contacted and that Visual Studio is working offline.</p> </div>
2	Repeat Exercise 2 Task 3 to make various changes outside of Visual Studio

Step	Instructions
Make changes outside of Visual Studio ☐ - Done	

NOTE

The first task in this step is to open the solution folder in Windows Explorer using the Local Path link in the Source Control Explorer. This link is not available because you have disconnected Source Control Explorer from Team Foundation Server so navigate to the solution folder manually.

REVIEW

We have disconnected from Team Foundation Server and made changes both inside and outside of Visual Studio.

Exercise 4: Explore the diff and merge tools

GOAL

In this exercise, we explore the diff and merge tools when working in both offline and online modes.

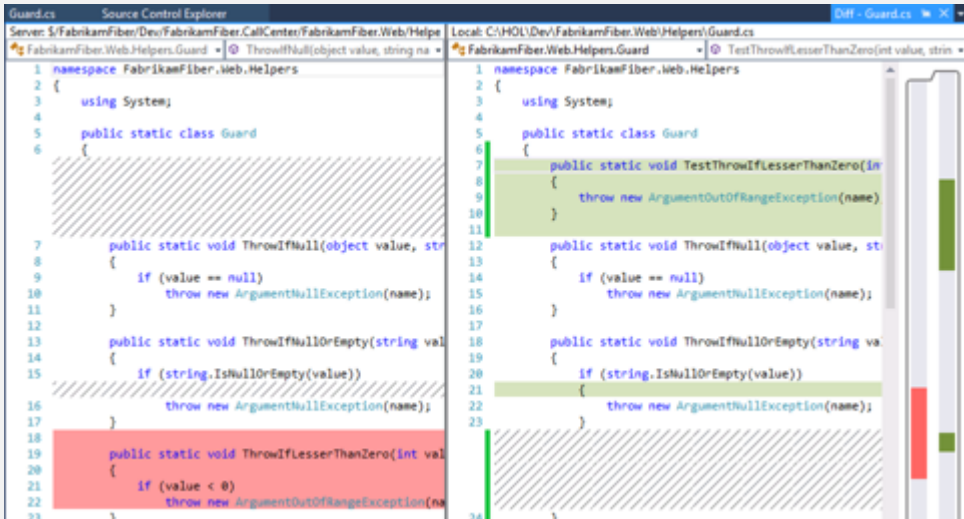
NOTE

This exercise assumes that you have just completed the previous exercise and that you are working in disconnected mode from Team Foundation Server. If this is not the case, please disconnect by completing Exercise 3 Task 1.

Task 1: Perform a diff in offline mode

Step	Instructions
1 Modify the C# source code ☐ - Done	Switch back to Visual Studio and make various insert, edit and delete changes to the source code for the Guard class and then save the changes
2 Compare with Workspace Version ☐ - Done	<ul style="list-style-type: none"> Open Pending Changes Right-click Guard.cs under Included Changes Click Compare with Workspace Version The Diff window should appear in the provisional tab
3 Changing the Compare mode ☐ - Done	<ul style="list-style-type: none"> Locate the Compare Files toolbar, click the first icon in the list Click side-by-side mode

TFVC Gems – Hands-on Lab (HOL) – Workspaces & Merging

Step	Instructions
4 Explore Compare Files Toolbar ☐ - Done	<p>Take a moment to explore the other options on the Compare Files toolbar and to explore the functionality of the Diff window. In particular, note the following about the Diff window:</p> <ul style="list-style-type: none"> It uses the Visual Studio editor and therefore supports features such as IntelliSense, syntax highlighting, undo, redo, and navigation. Visual Studio highlights changes within a line. There is a change map on the right.
	

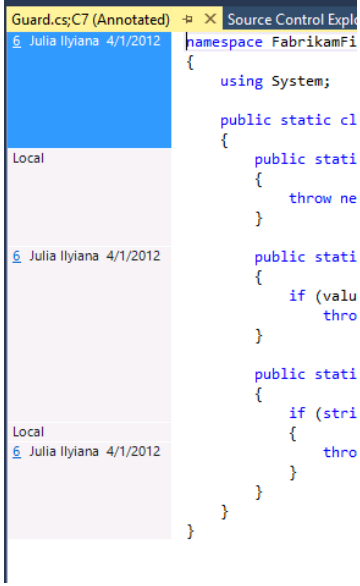
Task 2: Reconnect to Team Foundation Server

Step	Instructions
1 Start IIS ☐ - Done	Switch back to the administrative level Command Prompt and execute iisreset /start . The message <i>Internet services successfully started</i> should be displayed
2 Reconnect to Team Foundation Server ☐ - Done	<ul style="list-style-type: none"> Open Source Control Explorer Click the Refresh icon to reconnect to Team Foundation Server Source Control Explorer should reconnect and refresh

Task 3: Annotate the source code changes

You can annotate a file to learn who made changes and what changes they made in all earlier versions of the file

TFVC Gems – Hands-on Lab (HOL) – Workspaces & Merging

Step	Instructions
1 Open Annotate for Guard.cs <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Open Pending Changes Right-click Guard.cs under Included Changes Click Annotate  <ul style="list-style-type: none"> Review the annotations to the left of the source code
2 Log Off <input type="checkbox"/> - Done	<ul style="list-style-type: none"> Save all unsaved changes, close Visual Studio and then log off

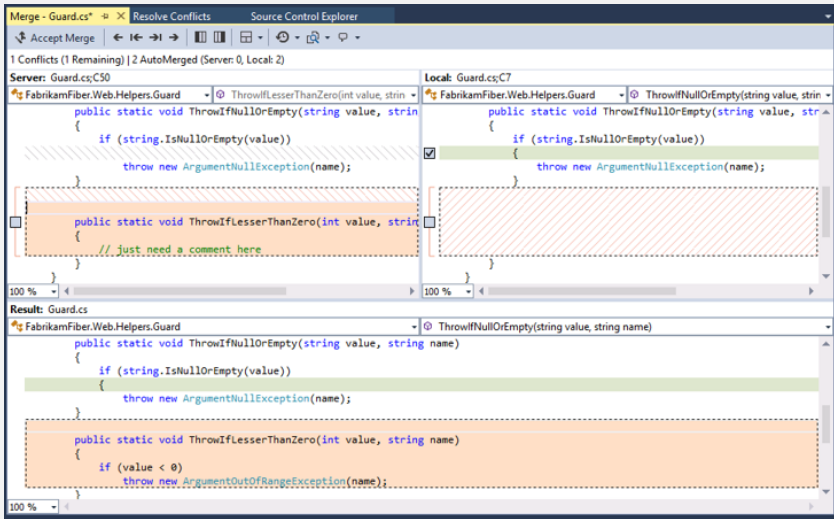
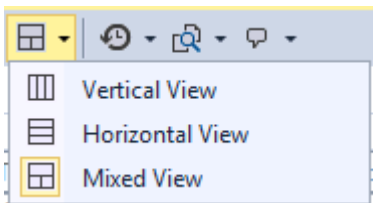
Task 4: Make changes as a different user

Step	Instructions
1 Log on <input type="checkbox"/> - Done	Repeat Exercise 1, Task 1 and Task 2 - Log on as Adam Barr using the same password
2 Map Folder <input type="checkbox"/> - Done	Repeat Exercise 1, Task 3 - Use the path c:\hol\Adam Barr for the Local Folder
3 Open the FabrikamFiber CallCenter solution <input type="checkbox"/> - Done	Repeat Exercise 2 Task 1 to open the FabrikamFiber CallCenter solution
4 Open a C# source code file <input type="checkbox"/> - Done	Open Solution Explorer if it is not already open and expand the FabrikamFiber.Web project, expand the Helpers folder and finally double-click Guard.cs to view the C# source code

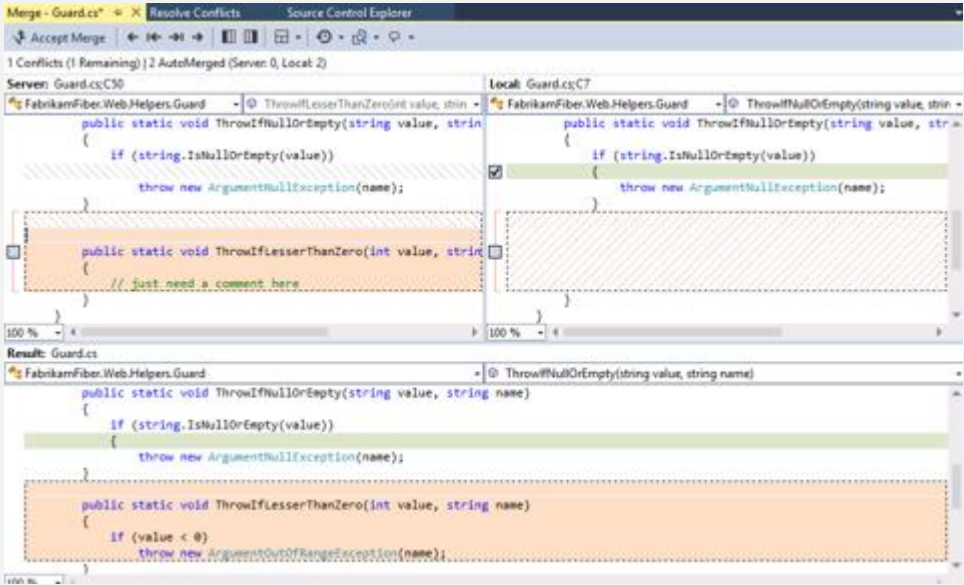
TFVC Gems – Hands-on Lab (HOL) – Workspaces & Merging

Step	Instructions
5 Modify the C# source code ☐ - Done	Make one minor change to the source code for the Guard class. You should make this change to one of the lines of code that you modified when logged on as the first user. However, this change should be a different change. Since you will check in this modification, the change should not materially affect the operation of the code. Save your change
6 Check-in changes ☐ - Done	<ul style="list-style-type: none"> Open Pending Changes if it is not already open and in the Comment section click on <i>Enter a check-in comment</i>. Enter a comment of your choice Click Check in In Check-in Confirmation message box click Yes You will see at the top of the Pending Changes windows a message confirming that the check was successful.
7 Logoff ☐ - Done	Log off and then log back on as Administrator

Task 5: Resolve conflicting changes

Step	Instructions
1 Get Latest Version ☐ - Done	<ul style="list-style-type: none"> Open Visual Studio Open Source Control Explorer Right-click on FabrikamFiber.CallCenter Click Get Latest Version
2 Resolve Conflicts ☐ - Done	<ul style="list-style-type: none"> The Resolve Conflicts should appear Click Merge Changes in Merge Tool The Merge tool will appear showing the Server version, the Local version and the Result 
3 Change View mode ☐ - Done	<p>You can change the merge view by click on the View icon</p> 

TFVC Gems – Hands-on Lab (HOL) – Workspaces & Merging

Step	Instructions
<p>4</p> <p>Explore Merge toolbar</p> <p><input type="checkbox"/> - Done</p>	<p>Take a moment to explore the other options on the Merge toolbar and to explore the functionality of the Merge window. In particular, note the checkboxes on the left hand side of the Server and Local panes for each difference and conflict. These checkboxes provide an easy to use mechanism for indicating which of the changes should be included and which should be excluded</p> 
<p>5</p> <p>Undo all pending changes</p> <p><input type="checkbox"/> - Done</p>	<ul style="list-style-type: none"> • Open Pending Changes • Click Actions, Undo All • In Undo Pending Changes dialog click Undo Changes • In Confirm Undo Checkout, click Yes to All. <p>All changes are undone and are removed from Included Changes</p>

REVIEW

We have performed a diff in offline mode, reconnected to Team Foundation Server, annotated the source code changes, made changes as a different user, and resolved conflicting changes.

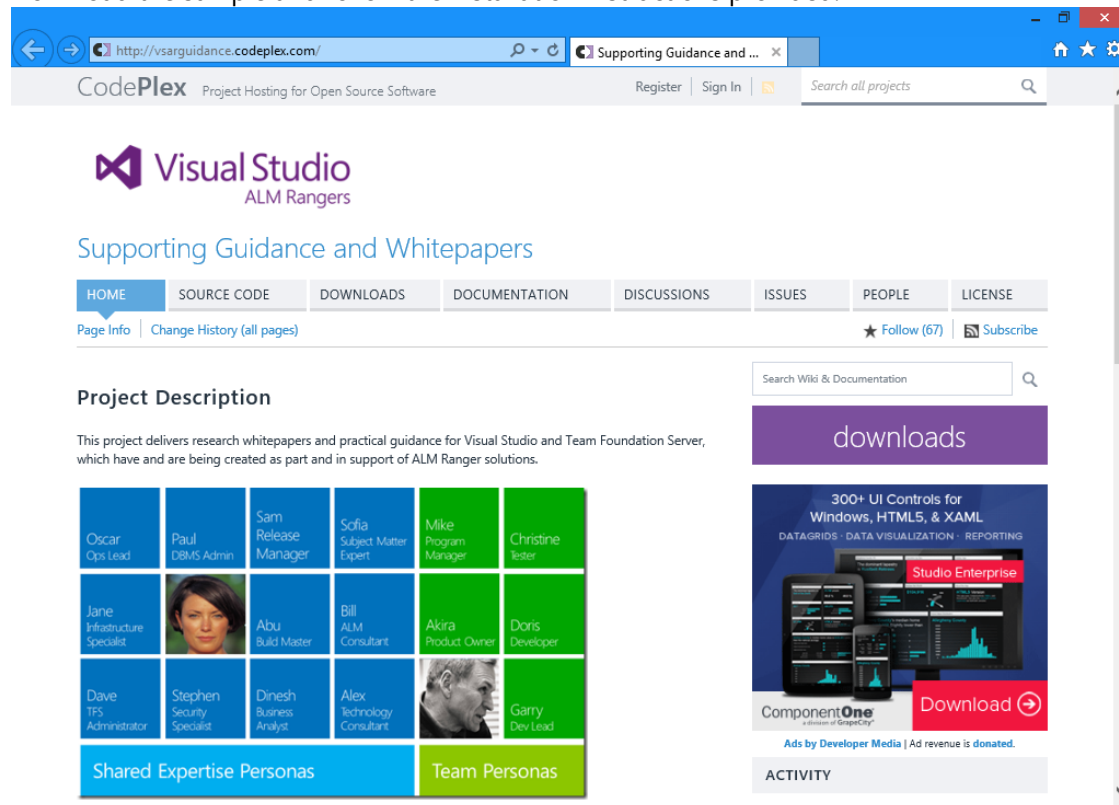
Appendix: Environment Setup

NOTE

This appendix contains information about configuring custom environments for use with this Hands-On Lab **if and only if** your environment does not include an instance of the FabrikamFiber solution as used on ALM Rangers Base VMs or Brian Keller's VMs (<http://aka.ms/almvms>).

1. Install and configure the FabrikamFiber sample solution.

This Hands-On Lab uses a team project called FabrikamFiber containing the FabrikamFiber.CallCenter solution within a Dev folder. You can download the FabrikamFiber sample solution from the Visual Studio ALM Rangers Supporting Guidance and Whitepapers site on CodePlex at <http://vsarguidance.codeplex.com>. Download the sample and follow the installation instructions provided.



If you are using the Visual Studio 2013 ALM Virtual Machine from Brian Keller, you will already have the project files but will be required to [remove workspaces mappings](#) as they are already setup when you first login.

NOTE

This Hands-On Lab does not use any specific features of the **FabrikamFiber** sample solution. If the **FabrikamFiber** sample is not available, you may substitute an alternative codebase and make appropriate allowances when following the steps.

FAQ

What are the disadvantages of cherry picking changes?

Cherry picking means that you select only certain files to merge back to main branch. However, suppose another developer had to change more files than you are aware of. Because you did not execute a full merge, you left those files behind. Now your target source is “broken,” might not even build correctly and you will probably spend hours trying to discover what went wrong

A cherry picked changeset can fall in the middle of a future merge range resulting in remerge of that changeset. This can result in merge conflicts.

With more complex projects, cherry picking is more likely to cause problems such as bugs that you have fixed but then appear again in new builds, or broken builds due to incorrect file versions.

What is baseless merge and how does it compare to regular merge?

A baseless merge allows merging two folders that do not already have merge relationship. Once you perform baseless merge, you can then perform ordinary merges between the branches (as if the two branches had a normal merge relationship between them).

What is the maximum file length supported by TFS?

TFS currently supports up to 260 characters in any client and 400 on the server. Examples of paths are below:

- Client: c:\code\src\...
- Server: \$/TP/SomeFolder/Code/Src...

In Conclusion

This concludes our adventure into TFVC Gems. We have touched on basic theory of folders, workspaces, and merging. We concluded with a practical Hands-on lab (HOL) exploring workspaces and merging.

We hope that you have found this guide useful.

Sincerely

The Microsoft Visual Studio ALM Rangers

