Microsoft

Team Development with Visual Studio Team Foundation Server

BETA 1.0

Feedback/Comments: TFSGuide@microsoft.com

patterns & practices

Team Development with Visual Studio Team Foundation Server

patterns & practices

J.D. Meier
Jason Taylor
Alex Mackman
Prashant Bansode
Kevin Jones

Introduction

This guide shows you how to get the most out of Visual Studio 2005 Team Foundation Server to help improve the effectiveness of your team-based software development. Whether you are already using Team Foundation Server or adopting from scratch, you'll find guidance and insights you can tailor for your specific scenarios.

The information in this guide is based on practices learned from customer feedback and product support, as well as experience from the field and in the trenches. The guidance is task-based and presented in the following parts.

- **Part I, "Fundamentals,"** gives you a quick overview of team development with Team Foundation Server. You'll see the big picture in terms of your software development environment, including the development and test environment. You'll also learn the basic architecture of Team Foundation Server.
- **Part II, "Source Control,"** shows you how to structure your source code and manage dependencies. It also shows you how to determine a branching and merging strategy if you need isolation for your development efforts.
- **Part III, "Builds,"** shows you how to set up team builds, how to produce continuous integration builds for your development team, and how to drop scheduled builds to your test team. It also discusses common problems and how to work around them.
- **Part IV, "Large Project Considerations,"** show you additional considerations you need to deal with when working with large projects.
- **Part V, "Project Management,"** shows you how to use Team Foundation Server work items, areas and iterations to streamline your development process regardless of what project management approach you use.
- **Part VI, "Process Templates,"** shows you how to get the most out of the process templates and process guidance that is supplied with Team Foundation Server out of the box. It also shows how you can customize the process templates, and make modifications to work items and workflow to map to the software engineering process your team is already using.
- **Part VII, "Reporting,"** shows you how all of the other Team Foundation Server components integrate their data store into a common reporting mechanism. You'll learn how to use the default reports as well as how to build your own custom reports.
- **Part VIII, "Setting Up and Maintaining the Team Environment,"** removes the mystery from Team Foundation Server deployment. You'll learn how to choose between a single server and multiple server deployment. You'll also learn how to support remote development teams and how to maximize Team Foundation Server performance.

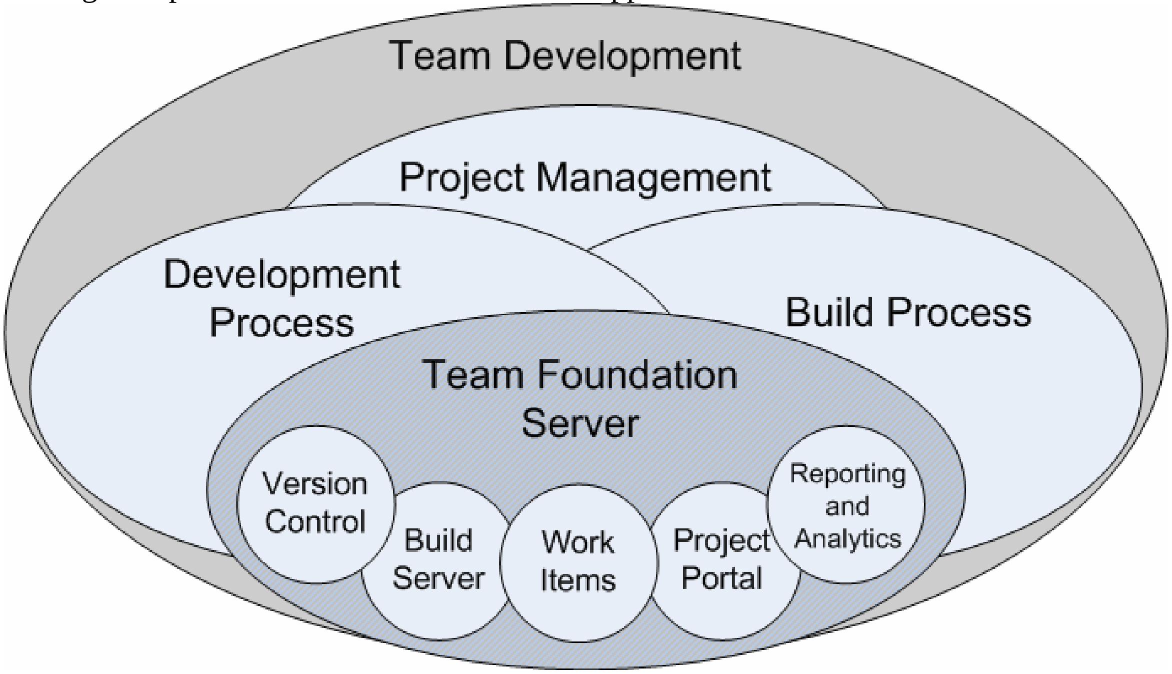
Team Development

There are many elements, processes, and roles that combine to enable successful team-based software development projects. This guide focuses on:

- The development process

- The build process
- The project management process

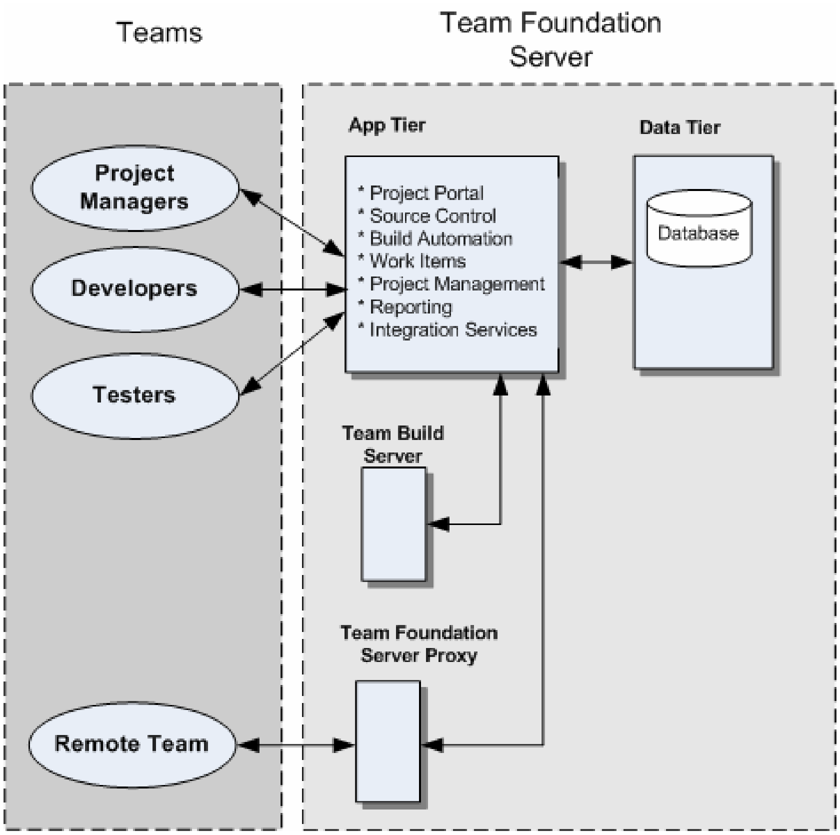
The following diagram illustrates the relationship between typical software development processes relating to team development and how Team Foundation Server can be leveraged to provide horizontal foundational support for these initiatives.



Scope of This Guide

This guide is focused on deploying Team Foundation Server and using it effectively for source control, build automation, work item management, and process management.

The following diagram outlines a sample logical implementation of Team Foundation Server as it relates to the roles most common to the software engineering and development lifecycle.



Why We Wrote This Guide

From our own experience with Team Foundation Server and through conversations with customers and Microsoft employees who work in the field, we determined there was demand for a guide that would show how to use Team Foundation in the real world. While there is information in the product documentation, in blog posts and in forums, there has been no single place to find proven practices for the effective use of Team Foundation Server in the context of a development project under real world constraints.

Who Should Read This Guide

This guide is targeted at providing individuals involved in the software development process with the resources, patterns and practices for creating an effective team development environment. The following are examples of roles that would benefit from this guidance:

- A development team that wants to adopt Team Foundation.
- A project manager looking to get the most out of Team Foundation, with regard to managing projects and development efforts, providing status of software development initiatives and providing feedback to business stakeholders.
- Interested parties investigating the use of Team Foundation but don’t know how well it would work for their development scenarios and team constraints..

- Individuals tasked with planning a deployment and installing Team Foundation.

How To Use This Guide

The guide is divided into parts based on the order we see most teams think about and adopt Team Foundation. If you are in the process of adopting Team Foundation you'll probably want to read the entire guide from start to finish. If you are interested in using Team Foundation for a particular use, such as Source Control or Team Build, you can restrict your reading to just those sections. Use the main chapters to learn concepts and guiding principles. Use the appendix of "Guidelines", "Practices", "How To" articles and "Questions and Answers" to dive into implementation details. This separation allows you to understand the topics first and then dive into details as you see fit.

Organization of This Guide

You can read this guide from end to end, or you can read the chapters you need for your job.

Parts

The guide is divided into eight parts:
- Part I, Fundamentals
- Part II, Source Control
- Part III, Builds
- Part IV, Big Team Considerations
- Part V, Project Management
- Part VI, Process Guidance
- Part VII, Reporting
- Part VIII, Setting Up and Maintaining the Team Environment

Part I, Fundamentals
- Ch 01 - Introducing the Team Environment
- Ch 02 - Team Foundation Server Architecture

Part II, Source Control
- Ch 03 - Structuring Projects and Solutions in Source Control
- Ch 04 - Structuring Projects and Solutions in Team Foundation Source Control
- Ch 05 - Defining Your Branching and Merging Strategy
- Ch 06 - Managing Source Control Dependencies in Visual Studio Team System

Part III, Builds
- Ch 07 - Team Build Explained
- Ch 08 - Setting Up Continuous Integration with Team Build
- Ch 09 - Setting Up Scheduled Builds with Team Build

Part IV, Large Project Considerations
- Ch 10 - Large Project Considerations

Part V, Project Management
- Ch 11 - Project Management Explained
- Ch 12 - Work Items Explained
- Ch 13 – MSF Agile Projects

Part VI, Process Templates
- Ch 14 - Process Templates Explained

Part VII, Reporting
- Ch 15 - Reporting Explained

Part VIII, Setting Up and Maintaining the Team Environment
- Ch 16 - Team Foundation Server Deployment
- Ch 17 - Providing Internet Access to Team Foundation Server

Guidelines

- Guidelines: Visual Studio 2005 Team Foundation Server Source Control

Practices

- Practices at a Glance: Team Foundation Server Source Control

Questions and Answers

- Questions and Answers: Team Foundation Server Source Control and Versioning

Resources

- Visual Studio 2005 Team Foundation Server

"How To" Articles

- How To: Add a New Developer To Your Project in Visual Studio Team Foundation Server
- How To: Automatically Run Code Analysis with Team Build in Visual Studio Team Foundation Server
- How To: Create a Custom Report for Visual Studio Team Foundation Server
- How To: Create a Risk Over Time Report for Visual Studio Team Foundation Server
- How To: Customize a Report in Visual Studio Team Foundation Server
- How To: Customize a Process Template in Visual Studio Team Foundation Server
- How To: Manage Projects in Visual Studio Team Foundation Server
- How To: Migrate Source code to Team Foundation Server from Visual Source Safe
- How To: Perform a Baseless Merge in Visual Studio Team Foundation Server
- How To: Setup a Continuous Integration Build in Visual Studio Team Foundation Server

- How To: Setup a Scheduled Build in Visual Studio Team Foundation Server
- How To: Create Custom Check-in Policies for Visual Studio Team Foundation Server
- How To: Create Your Source Tree in Visual Studio Team Foundation Server
- How To: Structure ASP.NET Applications in Visual Studio Team Foundation Server
- How To: Structure Windows Applications in Visual Studio Team Foundation Server
- How To: Structure Your Source Control Folders in Visual Studio Team Foundation Server

Feedback and Support

We have made every effort to ensure the accuracy of this guide and its companion content.

Feedback on the Guide

If you have comments on this guide, send e-mail to *TFSguide@microsoft.com*.

We are particularly interested in feedback regarding the following:

- Technical issues specific to recommendations
- Usefulness and usability issues

Technical Support

Technical support for the Microsoft products and technologies referenced in this guide is provided by Microsoft Product Support Services (PSS). For product support information, please visit the Microsoft Product Support Web site at http://support.microsoft.com .

Community Support

MSDN Newsgroups:
http://forums.microsoft.com/MSDN/default.aspx?ForumGroupID=5&SiteID=1

Forum	Address
Team Foundation Server - General	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=22&SiteID=1
Team Foundation Server - Setup	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=68&SiteID=1
Team Foundation Server - Administration	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=477&SiteID=1
Team Foundation	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=481&SiteID=1

Server - Build Automation	
Team Foundation Server - Power Toys	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=930&SiteID=1
Team Foundation Server - Process Templates	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=482&SiteID=1
Team Foundation Server - Reporting & Warehouse	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=480&SiteID=1
Team Foundation Server - Team System Web Access	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=1466&SiteID=1
Team Foundation Server - Version Control	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=478&SiteID=1
Team Foundation Server - Work Item Tracking	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=479&SiteID=1

The Team Who Brought You This Guide

This guide was created by the following team members:

- **J.D. Meier**
- **Jason Taylor**
- **Alex Mackman**
- **Prashant Bansode**
- **Kevin Jones**

Contributors and Reviewers

- **External Contributors/Reviewers**. David P. Romig, Sr; Eric Blanchet; Leon Langleyben; Martin Woodward; Quang Tran; Sarit Tamir; Tushar More; Vaughn Hughes; Michael Ruminer
- **Microsoft Contributors / Reviewers**. Ajay Sudan; Ajoy Krishnamoorthy; Alan Ridlehoover; Alik Levin; Buck Hodges; Clark Sell; Doug Neumann; Edward Jezierski; Eric Charran; Graham Barry; Janet Williams Hepler; Jeff Beehler; Julie MacAller; Ken Perilman; Mario Rodriguez; Marc Kuperstein; Mario Contreras; Mark Nichols; Matthew Mitrik; Michael Puleio; Nobuyuki Akama; Paul Goring; Pete Coupland; Peter Provost; Randy Miller; Rob Caron; Robert Horvick; Rohit Sharma; Sajee Mathew; Siddharth Bhatia; Tom Hollander; Venky Veeraraghavan

Tell Us About Your Success

If this guide helps you, we would like to know. Tell us by writing a short summary of the problems you faced and how this guide helped you out. Submit your summary to: *MyStory@Microsoft.com* .

PART I

Fundamentals

In This Part:

- **Introducing the Team Environment**
- **Team Foundation Server Architecture**

Chapter 1 - Introducing the Team Environment

Objectives

- Describe how Team Foundation Server supports the software development lifecycle.
- Describe how a typical development team uses Team Foundation Server.
- Describe how a typical test team uses Team Foundation Server.
- Describe the physical environment of a development team and test team.

Overview

This chapter describes how Team Foundation Server (TFS) and Visual Studio Team System (VSTS) are used in a team-based development environment. It introduces the core features of TFS and VSTS and describes the workflow between development and test teams during a software development project. TFS integrates source control, work tracking, reporting, project management and an automated build process, By doing so, TFS enables a development team to work together more effectively.

A successful team-based software development project has many processes that must work together smoothly to ensure an efficient working environment. The core processes include:

- Development
- Test
- Build
- Deployment
- Release

This chapter introduces you to typical functions that the development and test teams can perform with TFS and how you can use TFS to manage the workflow to support efficient collaboration across teams.

How to Use This Chapter

Use this chapter to learn how TFS is designed to support the software development lifecycle. Use this chapter to learn about the TFS workflow and how TFS enables you to improve team collaboration.

For more detailed information about TFS architecture, and the TFS core components, see Chapter 2, “Team Foundation Server Architecture.”

Logical Workflow of Team Foundation Server

Team Foundation Server enables a development team to store code in a centrally managed source code repository. You can create builds from this repository by using the Build Server and you can then distribute these to your test team.

Figure 1.1 shows the logical workflow of TFS and how the development and test environments are connected.

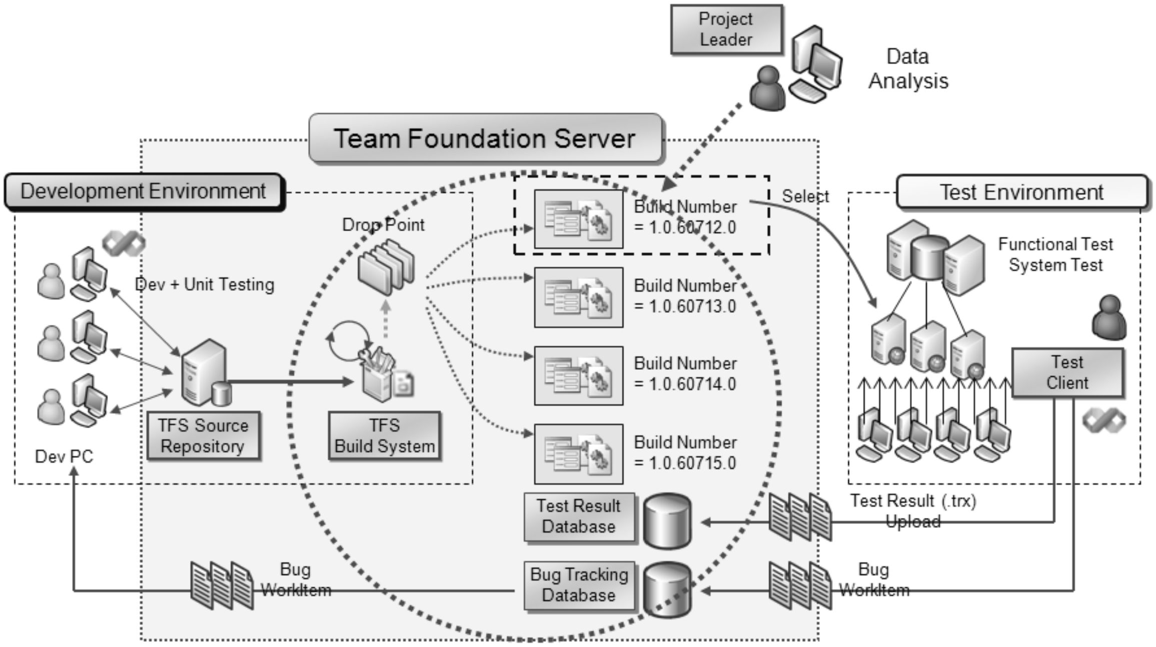


Figure 1.1 *Team Foundation Server Logical Workflow*

The test team picks up builds from a drop location and runs them through their testing environment by performing a combination of manual and automated tests. Test results are stored by TFS and are used to provide quality feedback on the build. The test team can also create work items and bugs (a specific type of work item) that the development team need to action. These work items allow the test team to track the work of the development team.

Logical Workflow of Development, Test and Production Environments

In larger organizations with multiple development teams, each development team maintains a separate TFS including separate source code repositories and team build servers. Figure 1.2 shows an example of the logical workflow that results from two development teams delivering application builds to an integration test team.

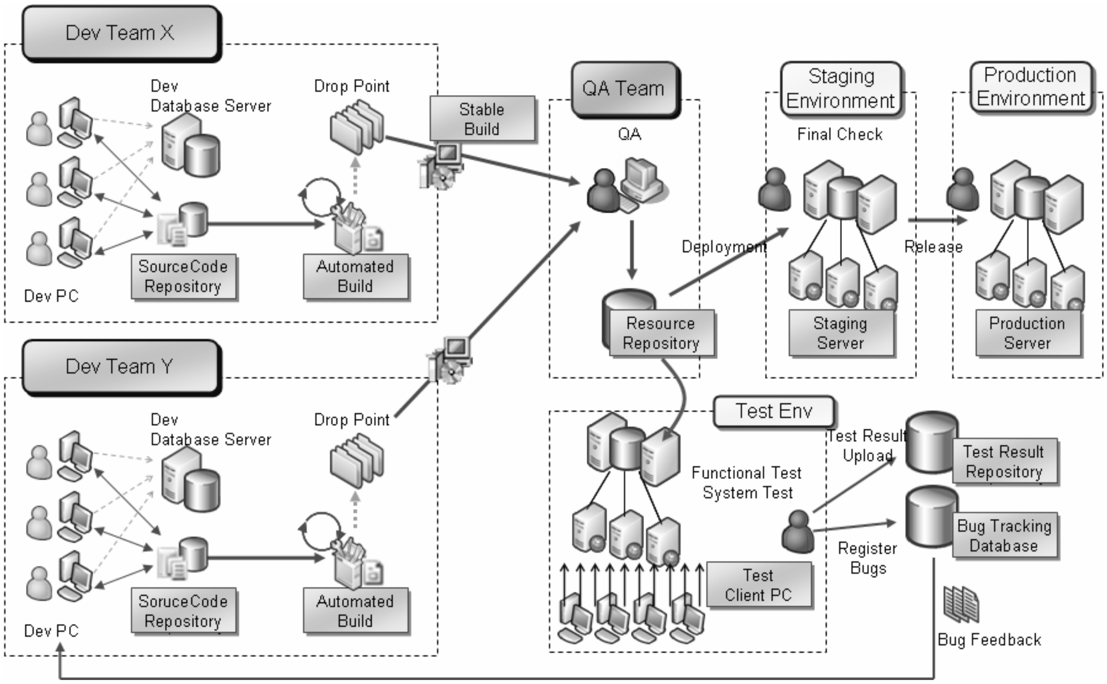


Figure 1.2 *Logical Workflow Showing Two Development Teams and an Integration Test Team*

Each development team delivers scheduled builds to a drop point such as a network share. These are picked up by the test team and tested to measure the quality of the build. When test quality gates are passed the applications are deployed to a staging server for final checks and user acceptance before ultimately being deployed to a production server.

Development Processes

Developers perform a number of key interactions with Team Foundation Server throughout the duration of a software development project. For example, as a developer you interact with TFS in the following ways:

- You access bugs and task work items from TFS to determine what work you need to do. For example, work items might have been assigned by your project manager, by another developer or by the test team.
- You use the VSTS Source Control Explorer to access the TFS source control repository and pull the latest source code into a local workspace or your development computer.
- After performing the work identified by the work item, you check your code back into the source control database.
- The check-in event might trigger a Continuous Integration build which uses Team Build.
- If the build fails a new work item is created to track the build break.

Test Processes

As a member of a test team, you might interact with TFS in the following ways:

- You pick up the output of a scheduled build from a specific drop location.
- You perform manual and automated testing including security testing, performance testing, and Web testing by using various Visual Studio Team Systems tools.
- You upload the results from the tests to the TFS Test Result Database for future reference.
- You log bugs identified by your testing into TFS as new work items.
- You resolve existing bugs, if the latest build fixes bugs logged previously.

Physical Environments of Development and Test

The size and number of computers associated with your development and test environments varies depending upon the size of your teams and projects. Figure 1.3 shows a typical development and test physical environment.

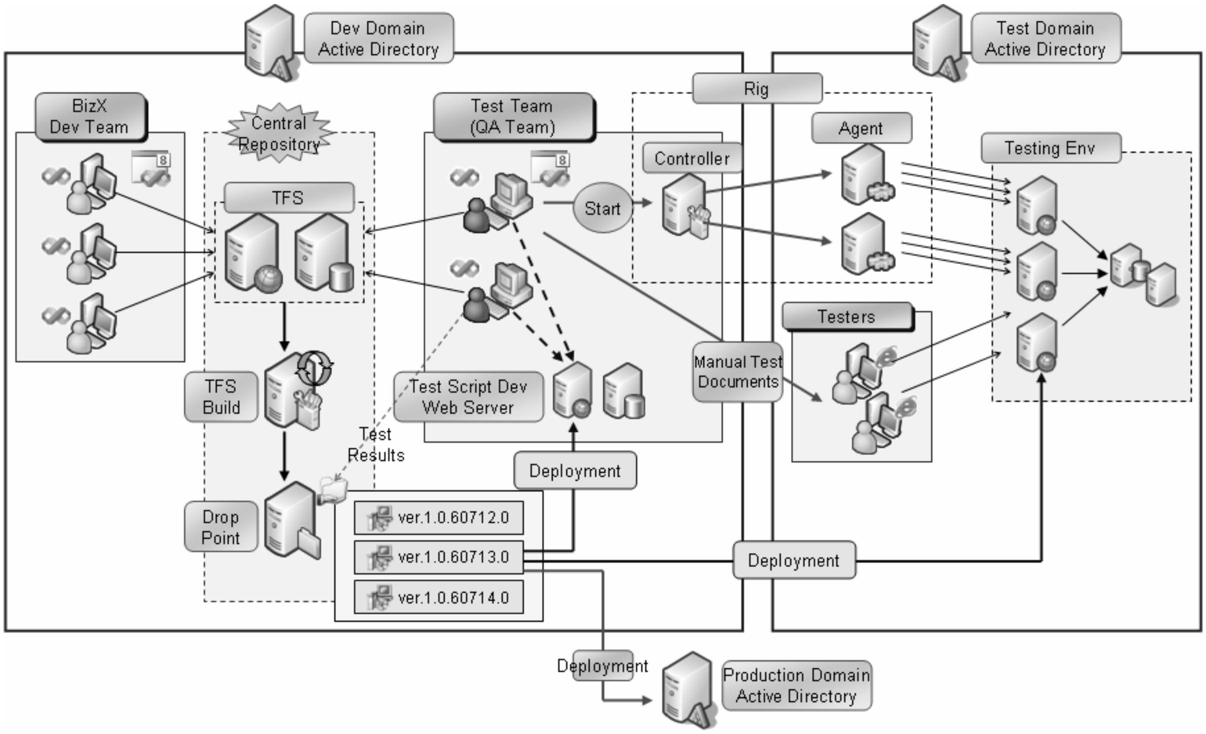


Figure 1.3 *Development and Test Physical Environment*

Development Environment

The development environment supports your development and build processes. The development environment contains the following computers:

- A Team Foundation Server,
- A Build Server
- A server to store the drops from the build server.
- Developer workstations.

If you have a remote development team that accesses TFS remotely, or you have a particularly large team that causes performance issues on your TFS then you can also setup a Team Foundation Server proxy to help improve performance.

Test Environment

The test environment consists of one or more test workstations with Visual Studio 2005 Team Edition for Software Testers installed. This is used to manage the test life cycle, perform functional testing, system testing, security testing, performance testing and Web testing. Members of the test team use Team Foundation Server to manage work items, bugs, and test results.

Additionally the test environment might also include Visual Studio Team Test Load for performance testing.

Chapter 2 - Team Foundation Server Architecture

Objectives

- Describe Team Foundation Server architecture.
- Identify the components that make up the client, application and data-tiers.
- Highlight the differences between a single server and multi-server deployment.

Overview

This chapter introduces you to Team Foundation Server (TFS) architecture and basic deployment topologies. TFS has a logical three-tiered architecture. It includes a client tier, an application-tier, and a data-tier. TFS clients interact with the application tier through various Web services, and the application-tier uses various SQL Server databases in the data-tier.

You can choose to install the application-tier and data-tier on the same physical server or you can install them on separate servers. Your choice depends largely on the size of your team. A single-server deployment works best for teams with fewer than 50 team members. A dual-server deployment can scale up to around 500 users.

How to Use This Chapter

Use this chapter to learn about the core TFS components and how they interact with one another. By reading this chapter, you will also learn the purpose of each of these components and how they are most commonly deployed.

- **Read Chapter 1, "Introducing the Team Environment"** If you are new to TFS, read Chapter 1 before this chapter to learn about the ways in which development and test interact with TFS and use it to improve collaboration and to improve the overall efficiency of your software development efforts.

Team Foundation Server Architecture

Team Foundation Server employs a logical three-tiered architecture, including *client*, *application*, and *data* tiers. TFS clients interact with the application tier through various Web services; the application tier in-turn is supported by various databases in the data tier. Figure 2.1 shows the components of each TFS tier as well as their interactions.

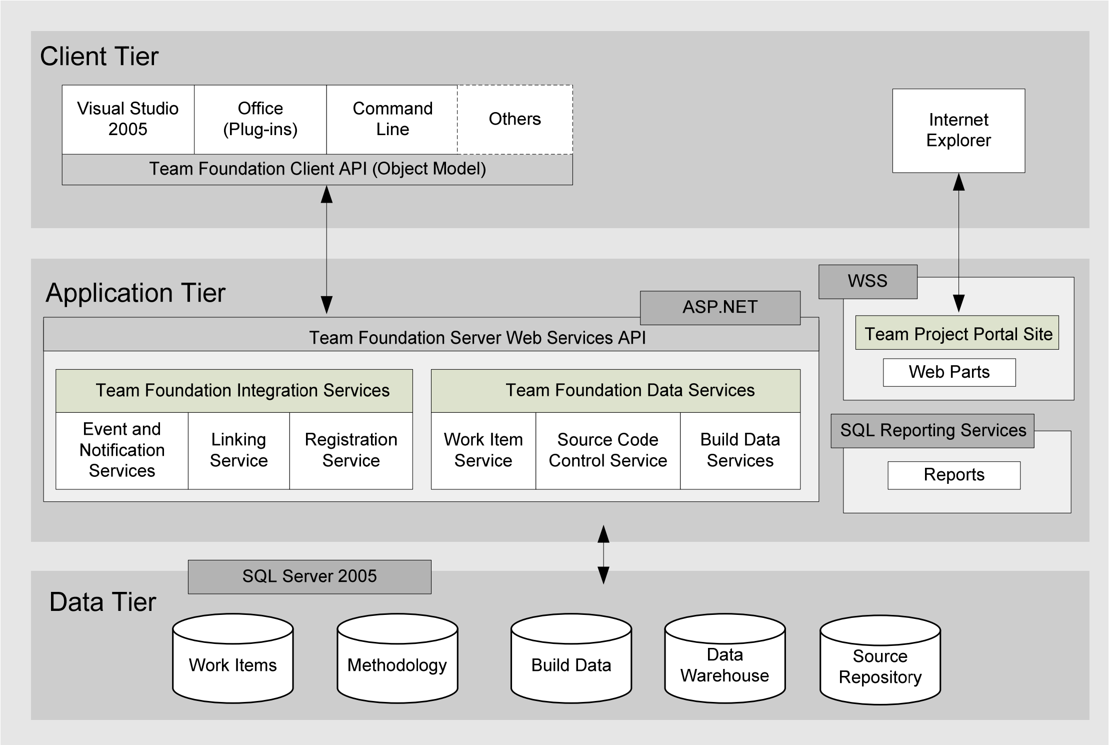


Figure 2.1 *TFS Components and Tiers*

Client Tier

The client tier contains the following important components

- **Team Foundation Server object model.** This is the public API used to interact with Team Foundation Server. You can use the object model to create your own client side applications that interact with TFS.
- **Visual Studio Industry Partners (VSIP) components**. These are third-party tools, add-ins and languages for use within Visual Studio.
- **Microsoft Office integration**. This consists of a set of add-ins for Microsoft Excel and Microsoft Project that enables you to query and update work items in the TFS work item database. This is particularly useful for project managers who already use these tools extensively.
- **Command-line tools**. These are the tools that enable you to interact with TFS from the command line. The majority of these tools provide source control functionality and they are useful for automating repetitive tasks and for scheduling tasks.
- **Check-in policy framework**. This supports the check-in policy feature, which is an extensible mechanism that enables you to validate code during the check-in process.

Application Tier

The application tier exposes the following ASP.NET Web services accessed by the client tier. These Web services are not intended for third-party integrators to program against, but are described here for completeness. Web services are grouped into the following collections:

- Team Foundation Data Services
- Team Foundation Integration Services

Team Foundation Data Services

This set of Web services is primarily concerned with manipulating data in the data tier. These services include:

- **Version Control Web service**. The client tier uses this Web service to execute various TFS source control features and to interact with the source control database.
- **Work Item Tracking Web service**. The client tier uses this Web service to create, update and query work items in the Work Item Tracking database.
- **Team Foundation Build Web service**. The client tier and the MSBuild framework uses this Web service to execute build processes.

Team Foundation Integration Services

The Team Foundation Integration Web services provide integration and automation functionality. They do not interact with the data tier. These services include:

- **Registration Web service**. This service is used to register various other TFS services. It maintains information in a registration database. The information is used by the services to discover each other and determine how to interact with one another.
- **Security Web service**. This consists of the Group Security Service and the Authorization Service. The Group Security Service is used to manage all TFS users and groups. The Authorization Service provides an access control system for TFS.
- **Linking Web service**. This service enables tools to establish loosely coupled relationships (or "links") between the data elements they hold. For example, the relationship between a defect work item and the source code that was changed to fix the defect is held by TFS using a link.
- **Eventing Web service**. This service enables a tool or service to register event types. Users can subscribe to those events and receive notification through e-mail or by invocation of a Web service. For example, you can use a check-in event to trigger a continuous integration build.
- **Classification Web service**. This service works together with the linking Web service to enable TFS artifacts to be classified according to predefined taxonomies. This helps support cross-tool reporting even for artifacts that do not share a common taxonomy to organize their data. For example, if work items are normally organized by team, while tests are normally organized by component, you can also organize tests by team, to enable them to be reported alongside work items.

Data Tier

TFS does not support direct access to data stored on the data tier from client applications. All requests for data must be made through the Web services on the application tier. The TFS data tier consists of the following data stores corresponding to data services on the application tier.

- **Work item tracking.** This stores all the data related to work items.

- **Version control.** This stores all the data related to source control.
- **Team Foundation Build.** This stores all the information related to the TFS Team Build feature..
- **Reporting warehouse**. This stores information related to all the tools and features in TFS. The reporting warehouse simplifies the creation of reports that combine data from multiple tools..

Deployment Topology

You can deploy TFS by using a variety of different topologies ranging from single server installations to more complex multiple server topologies. Regardless of topology, you need to be aware of a number of key requirements.

Key Requirements

Regardless of your selected deployment topology:

- You must install the application tier and the data tier in the same domain, although they can be on the same or separate server nodes.
- You must install TFS computers with Windows Server 2003 with Service Pack 1 (SP1) or later.
- You must install all TFS application-tier Web services to the same server.
- You must install single TFS instances on a single physical computer/
- You cannot install more than one instance of TFS per physical server.
- You cannot distribute TFS databases across multiple database servers. All projects must reside on one Team Foundation server group, and cannot be deployed across groups.
- You cannot use an existing SharePoint Portal Server infrastructure to host the team project portal. Consider using a dedicated server to host Team Foundation Server SharePoint Portals.
- You should not install TFS on a server configured as a domain controller because this is not supported.
- For dual-server deployments, you have to prepare some domain accounts to use to run TFS services. For example, you need to create accounts such as DOMAIN\TFSSERVICE, and DOMAIN\TFSREPORTS.

Single-Server Deployment

Single-server deployment is the simplest topology and is appropriate for small development organizations or for pilot projects, with up to 50 users. With this approach, you install all of the application tier and data tier components on a single server and access them from the same domain.

If you need to install test rig components for performance testing, you can install them on the server node or on one or more clients. Figure 2.2 shows the single server topology.

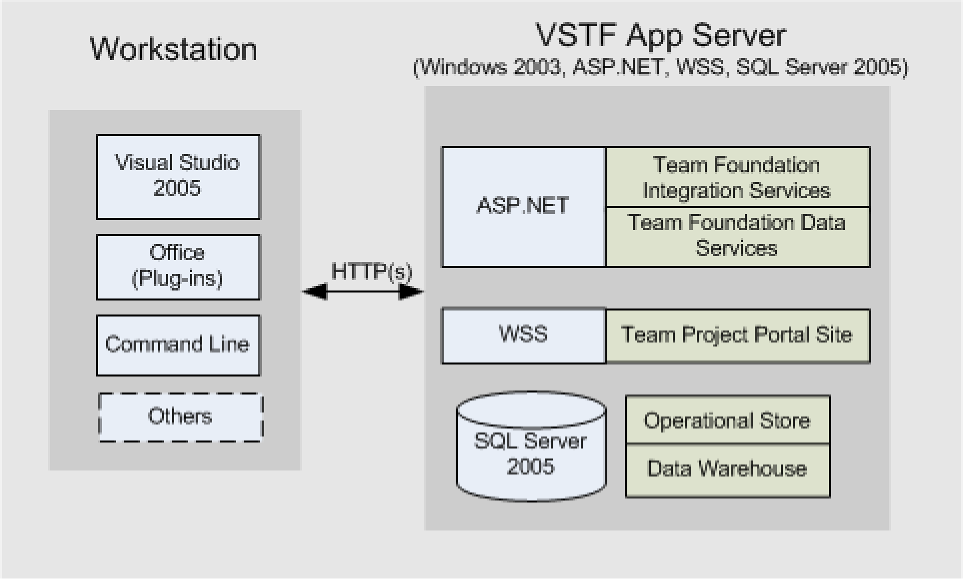


Figure 2.2 *Single Server Topology*

Dual-Server Deployment

The dual-server deployment topology is useful for large development teams in the range of 50 to 500 users. In this deployment topology you install the application tier on a separate server node from the data tier.

You can install Team Foundation Build services on the application tier, but you are recommended to setup one or more dedicated build server for large teams. If your project needs performance testing, you can deploy the test rig (controller and agents) to additional sever nodes. Figure 2.3 shows the dual server topology.

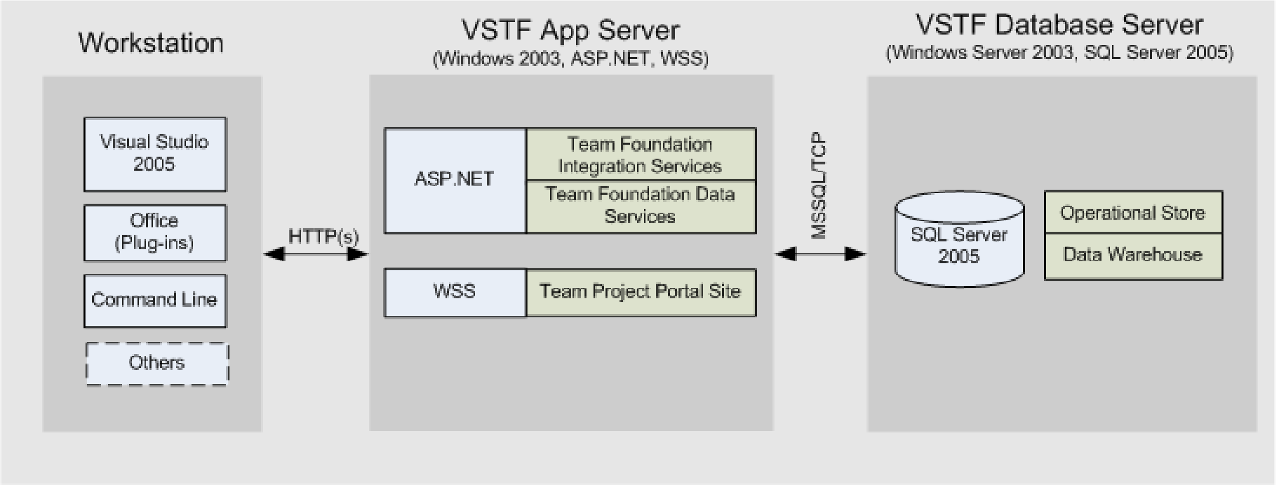


Figure 2.3 *Dual Server Topology*

PART II

Source Control

In This Part:

- Structuring Projects and Solutions
- Structuring Projects and Solutions in Team Foundation Server
- Defining Your Branching and Merging Strategy
- Managing Source Control Dependencies in Visual Studio Team System

Chapter 3 - Structuring Projects and Solutions in Source Control

Objectives

- Structure your Visual Studio solutions and projects appropriately.
- Know when to use multiple solutions and when to use a single solution.
- Identify appropriate structures for small, medium and very large teams.

Overview

This chapter explains various options for structuring Visual Studio solution and project files in a manner appropriate for team development. Visual Studio uses solution (.sln) files to group together related Visual Studio project (.csproj and .vbproj) files. Deciding how to structure your projects and solutions is an important decision. Your chosen pattern has a number of consequences. For example, it impacts how easily members of your development teams can push and pull solutions and projects to and from source control, the mechanism you use to reference dependencies, and also your build processes.

If you are working on a small project you can use a single solution to contain all of your project files. If you are working on a software development project with a large number of project files, you should use multiple solution files to group related projects that correspond to subsets of functionality within your overall team project. In this scenario, depending on your specific scenario you may also need a single solution file to group together all of your project files.

How to Use This Chapter

Use this chapter to select an approach for structuring your Visual Studio solutions and projects.

- **Use the strategies list**. Use the initial list of strategies: Single Solution, Multiple Solutions with Master and Multiple Solutions to quickly evaluate the best approach for your scenario.
- **Read the scenario section relevant to you**. Read the section describing how to implement the option you have chosen.
- **Read Chapter 4, “Structuring Projects and Solutions in Team Foundation Server”** next. Chapter 4 introduces you to important considerations to keep in mind when storing your code in Team Foundation Server source control.
- **Read Chapter 6, “Managing Source Control Dependencies”.** Project structure impacts the strategies available to you when managing dependencies across projects and solutions. For more information about how to manage dependencies, read Chapter 6.
- **Read the companion How Tos.** Read the following companion How Tos for a step-by-step walk through of various procedures discussed in this chapter.
    - How To: Structure ASP.NET Applications for Team Foundation Server
    - How To: Structure Windows Applications for Team Foundation Server

        - How To: Structure Your Source Control Folders in Team Foundation Server

Strategies for Solution and Project Structure

The three most common strategies used to structure solution and project files are:

- **Single Solution**. If you work on a small system, create a single solution and place all of your projects within it.
- **Partitioned Solution**. If you work on a large system, use multiple solutions to group related projects together. Create solutions to logically group subsets of projects that a developer would be most likely to modify as a set, and then create one master solution that contains all of your projects. This approach reduces the amount of data that needs to be pulled from source control when you only need to work on specific projects.
- **Multiple Solutions**. If you are working on a very large system that requires dozens of projects or more, use multiple solutions to work on sub-systems but for dependency mapping and performance reasons do not create a master solution that contains all projects.

In general:

- Use a single solution strategy unless the resulting solution is too large to load into Visual Studio.
- Use multiple solutions to create specific views on sub-systems of your application.
- Use multiple solutions to reduce the time it takes to load a solution and reduce build time for developers.

Keep the following considerations in mind when designing a project and solution structure:

- Each project generates an assembly at build time. Start by determining what assemblies you want to create and then use this to decide what projects you need – this will determine how your codebase is factored into projects.
- Start with the simplest single solution structure; only add complexity to your structure when it is really necessary.
- When designing a multi-solution structure:
    - Consider project dependencies. Try to group those projects that have dependencies on one another as part of the same solution. This enables you to use project references within your solution. By using project references instead of file references, you enable Visual Studio to keep build configurations synchronized (debug/release), and to track versioning to determine when projects need to be rebuilt. Try to minimize the number of cross solution project references.
    - Consider source sharing. Place projects that share source with each other together in the same solution.

    - Consider team structure. Structure your solutions to make it easy for teams to work on a set of related projects together.
- Keep a flat project structure so that it is easy for you to group projects into solutions without needing to make file system or source control folder structure changes.

Single Solution

If you work on a small system, consider using a single Visual Studio solution to contain all of your projects. This structure simplifies development since all of the code is available when you open the solution. This strategy also makes it easy to set up references, because all references are between projects in your solution. You may still need to use file references to reference third party assemblies, such as purchased components, that are outside your solution. Figure 3.1 shows the single solution approach.

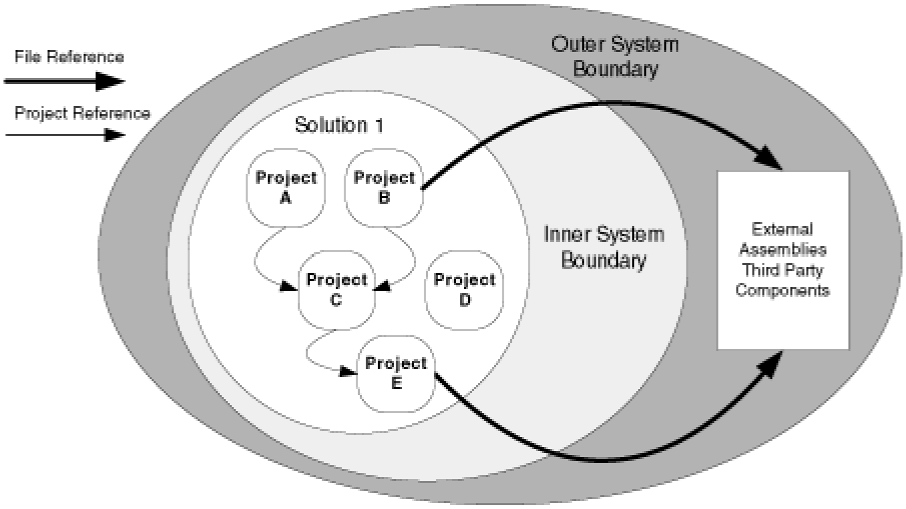


Figure 3.1 *Single Solution Approach*

The main reasons to choose this structure include:
- You can keep build scripts simple.
- You can easily map dependencies across projects within the solution.

Reasons not to use this structure include:
- All developers use the same solution and have the same set of projects. This could be a problem for large systems where you want to organize projects by sub-system or feature.

Partitioned Solution

If you work on a large system, consider using multiple solutions, each representing a sub-system in your application. These solutions can be used by developers in order to work on smaller parts of the system without having to load all code across all projects. Design your solution structure so any projects that have dependencies are grouped together. This enables you to use project references rather than file references. Consider creating a master solution file that contains all of the projects, if you want to use this to build the entire application. Figure 3.2 shows the partitioned solution approach.

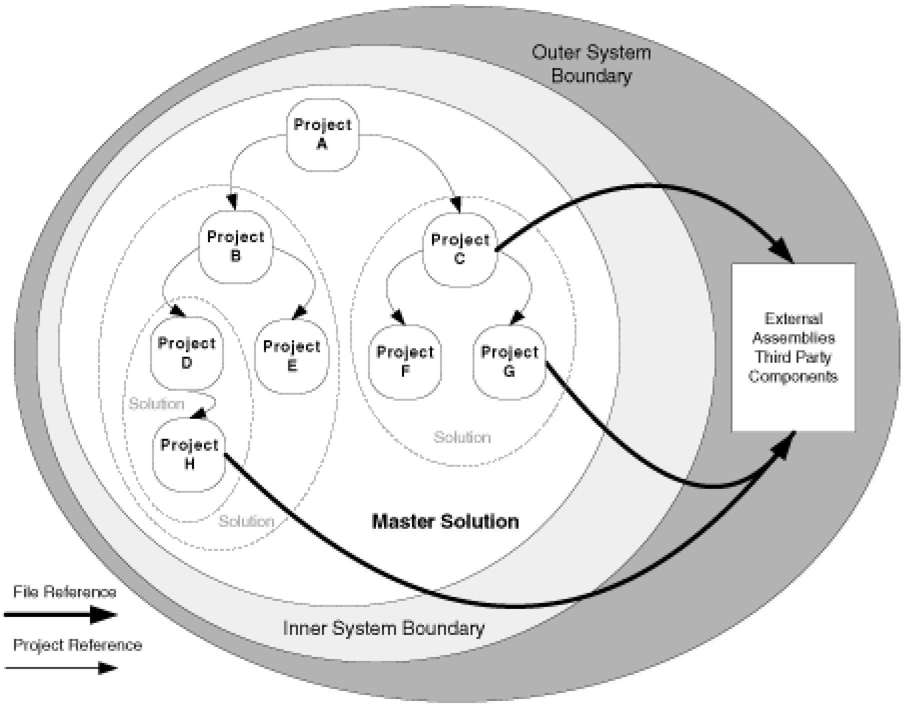


Figure 3.2 *Partitioned Solution Approach*

When working with multiple solutions use a flat file structure for all of your projects. A typical example is an application that has a Windows Forms project, an ASP.NET project, a Windows service, and a set of class library projects shared by some or all of those projects.

You can use the following flat structure for all projects:
- /Source
    - /WinFormsProject
    - /WebProject
    - /WindowsServiceProject
    - /ClassLibrary1
    - /ClassLibrary2
    - /ClassLibrary3
    - Web.sln
    - Service.sln
    - All.sln

Keeping the structure flat provides a lot of flexibility and then you can use solutions for presenting different views on the projects. Designing the physical folder structure around solution files is very hard to change, especially if you realize that you need to reuse a class library from another solution.

Reasons to use this structure include:

- Improved performance when loading and building application sub-solutions.
- Sub-solutions can be used to create views on sets of projects based on development sub teams or boundaries of code sharing.
- The master solution can be used to build the entire application.
- You can easily map dependencies across projects in each sub-solution.

Reasons not to use this structure include:

- Increased solution maintenance costs. Adding a new project might necessitate multiple solution file changes.
- Increased solution complexity. It may be harder for a new developer to understand which solutions to work on.

Multiple Solutions

If you work on a very large solution requiring many dozens of projects you may hit solution scalability limits. In this scenario, break your application into multiple solutions but do not create a master solution for the entire application. In this scenario, all references inside each solution are project references. References to projects outside of each solution (for example to third party libraries or projects in another sub-solution) are file references. This means that there can be no “master” solution. Instead, a script must be used that understands the order in which the solutions must be built. One of the maintenance tasks associated with a multiple-solution structure is ensuring that developers do not inadvertently create circular references between solutions. This structure requires complex build scripts and explicit mapping of dependency relationships. In this structure it is not possible to build the application in its entirety within Visual Studio. Instead, you use TFS Team Build. Figure 3.3 shows the multiple solutions approach.

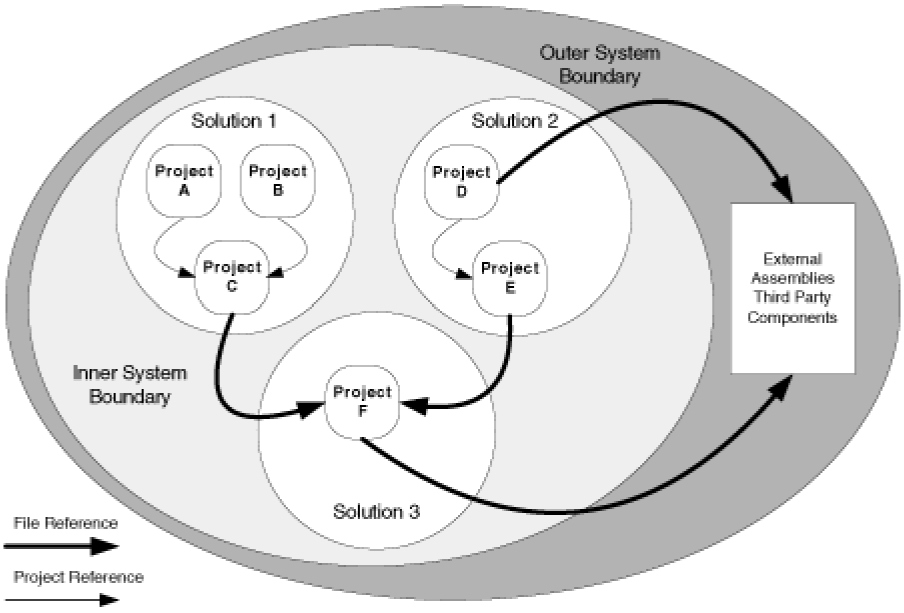


Figure 3.3 *Multiple Solution Approach*

Reasons to use this structure include:

- It works around Visual Studio IDE performance and scalability limits for very large applications.

Reasons not to use this structure include:

- It requires a complex build script to handle dependencies across the sub-solutions by building solutions in the right order.

Large Project Considerations

Large development teams are likely to distinguish themselves from smaller teams in the following ways:

- They require a more complex branching and merging structure.
- They are more likely to manage dependencies across solutions and team projects.
- They are more likely to maintain multiple builds for components and teams.

A Partitioned Solution approach works well for most large projects because it provides solution flexibility while maintaining a single solution that can be used to build the application. If your application is large enough that you hit solution scalability limits, use the Multiple Solutions approach.

Chapter 4 - Structuring Projects and Solutions in Team Foundation Source Control

Objectives

- Structure projects for effective team development in Team Foundation Server source control.
- Keep server-side and client-side folder structures synchronized.
- Choose a strategy for unit test structure.
- Create a folder structure that supports various branching scenarios.
- Learn what a workspace is and how it maps local files to source control.
- Understand what files are added to source control.

Overview

Many of the default folder conventions used by Visual Studio 2005 when you create new solutions and projects are not optimized for team development and for use with Team Foundation Server (TFS) Source Control. Rather than accepting the defaults when you create new Visual Studio projects and solutions, you should give careful consideration to your local and server-based folder structure when creating new Visual Studio solutions and projects.

This chapter starts by explaining how you should structure solutions and projects on your development computer (the client-side) and how you should structure your folders within TFS Source Control (the server-side). It provides example folder structures for a variety of application types including Window Forms, smart client and Web applications. The chapter then explains how workspaces are used to manage the mappings between client and server folder structures.

How to Use This Chapter

Use this chapter to learn sample folder structures suitable for team development projects of various sizes and complexity.

- **Use the server-side structure suggestions**. Use the suggested server side folder structures to organize your project source code within TFS source control.
- **Use the client-side structure suggestions**. Use the suggested client side folder structures to organize your project source code in your local development workspace.
- **Read the companion How Tos**. These provide a step-by-step walk through of some of the processes discussed in this chapter:
  - How To: Step Through Creating Your Source Tree in Team Foundation Server
  - How To: Structure ASP.NET Applications for Team Foundation Server
  - How To: Structure Windows Applications for Team Foundation Server
  - How To: Structure Your Source Control Folders in Team Foundation Server

Server-Side Structure

Most team projects contain one or more Visual Studio solutions, each of which contains one or more Visual Studio projects, When you require branching to support isolated development paths, you use a root level folder called **Main** (on client and server) to group together your Visual Studio projects. A sample folder structure within Team Foundation Server source control is shown here:

```
$MyTeamProject1
    /Main                                 → Can contain solution (.sln) files
        /Source
            /MyApp1                       → Contains MyApp1.sln file
                /Source                   → Contain folder for all source
                    /ClassLibrary1        → Contains ClassLibrary1.csproj
                    /MyApp1Web            → Contains Default.aspx
                /UnitTests                → Container folder for unit tests
                    /ClassLibrary1Tests   → Contains test project and code
                    /MyApp1WebTests       → Contains test project and code
            /SharedBinaries               → Shared binaries e.g. libraries
            /SharedSource                 → Shared source code
        /Build                            → Contains build output (binaries)
        /Docs                             → Contains product documentation
        /Scripts                          → Contains build scripts
        /Tests                            → Container for tests
            /FunctionalTests
            /PerformanceTests
            /SecurityTests
```

Main is a container folder for the source files and other related artifacts such as build output, design documentation, and test cases. An application folder (such as **MyApp1** in the preceding example) contains the Visual Studio solution (.sln) file used to group together a related set of Visual Studio projects. Each project file (.vcproj or .vbproj) is contained in a dedicated project folder, located beneath /Main/Source/MyApp1/Source. Unit tests that accompany each source project are located beneath the **UnitTests** folder. You can place additional Visual Studio solution (.sln) files in the **Main** folder to allow you to work with multiple different groupings of projects.

The folders **Build**, **Docs**, **Scripts** and **Test** are used to hold additional artifacts associated with the team project including the binary output of builds, documentation such installation guides, help files, build scripts and tests.

For more information about project groupings and solution structure, see “Chapter 03 - Structuring Projects and Solutions in Source Control”

Storing Unit Tests

You can store unit tests beneath a folder called **UnitTests** at the same level as **Source** as shown here.

```
…
        /MyApp1                        → Contains MyApp1.sln file
            /Source                    → Contain folder for all source
                /ClassLibrary1         → Contains ClassLibrary1.csproj
                /MyApp1Web             → Contains Default.aspx
            /UnitTests                 → Container folder for unit tests
                /ClassLibrary1Tests    → Contains test project and code
                /MyApp1WebTests        → Contains test project and code
```

This scenario treats unit tests as first class citizens. However, it does so at the expense of project level branching compatibility. An alternate structure is shown here:

```
        /MyApp1
            /Source
                /ClassLibrary1
                /ClassLibrary1Tests
                /MyApp1Web
                /MyApp1WebTests
```

The following pros and cons apply to each approach:

UnitTests as a Peer to the Source folder

- Pro: You can find unit tests in one place.
- Pro: You separate shipping code from non-shipping code.
- Pro: Your build process can easily run all unit tests across all projects.
- Con: It is harder for developer to run unit tests just for their project.
- Con: When you branch source it will not include unit tests.

UnitTests in Each Project

- Pro: Developers can easily run unit tests on a single project.
- Pro: When you branch, your branched folders include unit tests so they can stay tightly bound to the source in each branch.
- Con: You mix shipping with non-shipping code in the source folder.
- Con: It is generally harder to run all unit tests at once at build time across all projects.

Storing Documents

The **Docs** folder is for product related documentation. To help determine what documents to store in TFS source control and what to store in a document library on your SharePoint team site, consider the following:

- Use SharePoint for internal team documents such as use cases, scenario and requirements documentation, and design documentation.

- Use TFS source control for product-related documentation that you ship to your customers. This could include installation and deployment guides, operations guides, and help files.

Most documents are binary files, so consider using exclusive locks to avoid manual merges. This way, you get notification when a file is in use and you help avoid having to perform manual merges.

Use caution when using SharePoint because strict management of document versions is required. It is easier to overwrite changes in SharePoint compared to TFS source control. By default, SharePoint enables the "overwrite existing file" option selected when files are uploaded.

Client-Side Structure

The local folder structure on your development workstations should be identical to the server folder structure. Keep source artifacts well organized on your workstations by placing all source from all team projects together beneath a single root folder, such as C:\DevProjects. Create one sub-folder for each team project as shown in this example:

```
C:\DevProjects                    → Root container folder for all team projects
        \MyTeamProject1           → Container folder for TeamProject1
        \MyTeamProject2           → Container folder for TeamProject2
```

Beneath each team project folder, use a copy of the application folder structure used on the source control server.

```
\MyTeamProject1                         → Container folder for TeamProject1
   \Main                                → Contains .sln files that span projects
        \Source
                \MyApp 1                → Contains MyApp1.sln
                   \Source
                        \ClassLibrary1  → Contains ClassLibrary1.csproj
                        \MyApp1Web      → Contains Default.aspx
                   \UnitTests           → Contains unit test projects and source
                        \ClassLibrary1Tests
                        \MyWinApp1Tests
                \SharedBinaries         → Shared binaries e.g. libraries
                \SharedSource           → Shared source code
        \Build                          → Container folder for build output
        \Docs                           → Contains product documentation
        \Scripts                        → Contains build scripts, database scripts
        \Tests                          → Container for tests
                \FunctionalTests
                \PerformanceTests
                \SecurityTests
```

Note: The client-side structure automatically mirrors server-side structure if you create a workspace mapping from the application root to your local computer. However, in very large projects this can result in slow workspace load times. To optimize your approach for very large projects, create workspace mappings below the root to only retrieve the files you need for development.

Branched Folders

To support development isolation with branches, create additional folders as siblings of **Main**. You can also place additional Visual Studio solution (.sln) files in **Main** to enable developers to work with various groupings of projects. Branches created from the **Main** source folders can be used to support ongoing maintenance of product releases or parallel streams of development.

In the following sample structure, in addition to the **Main** root folder, a **Development** folder (branched from **Main**) is used to provide isolation for features or for teams. A **Maintenance** folder (again branched from **Main**) provides isolation for released builds that require ongoing maintenance and support.

```
$MyTeamProject1
    /Development
        /FeatureBranch1
            /Source
                /MyApp
        /FeatureBranch2
            /Source
                /MyApp
    /Main
        /Source
    /Release
        /Source
    /Maintenance
        /Release1
            /Source
                /MyApp
        /Release2
            /Source
                /MyApp
```

Note: Do not branch unless you need to. You can label a release and branch at a later time if required.

For more information about project groupings and solution structure, see “Chapter 03, Structuring Projects and Solutions in Source Control”

For more information about branching scenarios and related folder structures, see “Chapter 5, Defining Your Branching and Merging Strategy.”

Workspaces Explained

A Team Foundation Server workspace is a client-side copy of the files and folders in TFS source control. A workspace maps source control folders to local file system directories. When you make changes to files within the workspace on your local computer, the local changes, referred to as *pending changes* are isolated in your workspace until you check them into the server as an atomic unit. The collective set of changes, checked in as a batch is referred to as a *changeset*.

A single workspace can contain references to multiple team projects. You can also use multiple workspaces to isolate files or versions for your use only. Workspaces are per computer, per user account. Therefore, you can have different workspace definitions for each computer you use. Also, as a single user, you can have multiple workspaces on a single computer.

Note: You can only map each physical location on the local file system by using a single workspace. However, you can map each *server* directory to entirely different local directories by using different workspaces.

Creating a New Workspace Mapping

Because mappings are recursive, when you create a new workspace mapping and you perform a *Get Latest Version* operation at the root of that workspace, the entire local folder structure is created automatically. The newly created local folder structure matches the server structure.
Keep the following recommendations in mind when you create new workspace mappings:

- The project owner must ensure that the correct folder structure is used locally prior to adding the solution to source control for the first time.
- When establishing a workspace mapping for a team project for the first time and performing a get latest operation, be sure to map the root team project folder into an appropriate local folder such as C:\DevProjects\TeamProject1.

Where Are Workspace Mappings Stored?

Workspace information is maintained on the client and on the server. On the client, workspace information is held in VersionControl.config which is located in the following folder:
\Documents and Settings\[user]\Local Settings\Application Data\Microsoft\Team Foundation\1.0\Cache.

The VersionControl.config file maps the name of the workspace to a local directory on your computer. It does not hold the mapping between individual source control folders and your local directories. That information is held on the server in several tables (including tbl_Workspace and tbl_workingfolder) in the TfsVersionControl database.

What Files Should Be Version Controlled?

The following list identifies the key file types that you should add to source control. These are the file types that are added when you click **Add Solution to Source Control**.

- **Solution files (*.sln).** Solution files maintain a list of constituent projects, dependencies information, build configuration details, and source control provider details.
- **Project files (*.csproj or *.vbproj)**. Project files include assembly build settings, referenced assemblies (by name and path), and a file inventory.
- **Visual Studio Source Control Project Metadata (*.vspscc)**. These files maintain project bindings, exclusion lists, source control provider names and other source control metadata.
- **Application configuration files (*.config)**. XML configuration files contain project and application specific details used to control your application's run time behavior. Web applications use files called Web.config. Non-Web applications use files called App.config.

  Note: At run time, the Visual Studio build system copies App.config to your project's Bin folder and renames it as <YourAppName>.exe.config. For non-web applications, a configuration file is not automatically added to a new project. If you require one, add it manually. Make sure you call it App.config and locate it within the project folder.

- **Source files (*.aspx, *.asmx, *.cs, *.vb, …)**. Source code files, depending on application type and language.
- **Binary dependencies (*.dll).** If your project relies on binary dependencies such as third party DLLs, you should also add these to your project within source control. For more information about managing dependencies, see Chapter 6, "Managing Source Control Dependencies in Visual Studio Team System."

What Files Should Not Be Source Controlled?

The following files are specific to each developer and should therefore not be added to version control:

- **Solution user option files (*.suo).** These contain personalized customizations made to the Visual Studio IDE by an individual developer.
- **Project user option files (*.csproj.user or *.vbproj.user)**. These files contain developer specific project options and an optional reference path that is used by the Visual Studio to locate referenced assemblies.
- **WebInfo files (*.csproj.webinfo or *.vbproj.webinfo)**. This file keeps track of a project's virtual root location. This is not added to source control to allow individual developers to specify different virtual roots for their own working copy of the project. While this capability exists, you and all team members are recommended to use a consistent (local) virtual root location when you develop Web applications.
- **Build outputs.** These include assembly dynamic-link libraries (DLLs), interop assembly DLLs and executable files (EXEs). (Note though that assemblies such as

third-party binaries that are not built as part of the build process should be placed under version control as described above).

Additional Resources

- Team Foundation Server Branching Guidance: http://www.CodePlex.com/BranchingGuidance
- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx

Chapter 5 - Defining Your Branching and Merging Strategy

Objectives

- Know when and when not to branch.
- Choose a branching and merging strategy for your project.
- Describe how a very large team changes your branching strategy.
- Identify appropriate folder structures for different branching scenarios.

Overview

This chapter introduces branching and merging strategies for a range of common scenarios. Most often, you need branches either to support releases or to support parallel development

For many simple scenarios, you do not need to branch and labeling your builds is sufficient. For example, by using labels you can recreate a build at any point in the future or find out which versions of a source file were used to create a particular build. You should consider branching if you need isolation for parallel teams, to work on separate but overlapping features, or to support a release.

How to Use This Chapter

Use this chapter to learn when to branch, and when not to branch. If you decide that branching is appropriate for your situation, use this chapter to learn how to branch and merge effectively.

If you want an overview of branching and merging, then read the chapter from top to bottom to learn about the different strategies for branching and about the different scenarios where branching is appropriate. If you are interested in a particular scenario, jump directly to the relevant section.

- **Use the scenarios list**. Use the scenarios list to locate the scenario closest to your own. Read the recommendations to create a branched folder structure appropriate for your needs.
- **Use the companion guidance**. Refer to the branching and merging guidelines in “Guidelines: Visual Studio 2005 Team Foundation Server Source Control” for a summary of branching and merging guidelines.

Scenarios for Branching and Merging

The following are examples of when you might need to create branches and perform merges:

- If you are having regular problems with broken builds, then you should create a development branch to isolate parallel development efforts.

- If you have features that are causing stability issues or teams causing stability issues among each other, then create separate feature or team branches beneath a development container folder in source control.

If you have one stream of development, or are performing incremental and continuous releases, you might not need to create branches unless you frequently experience breaking changes that are destabilizing your development efforts.

Common Scenarios in Practice

The following are the most common branching scenarios:

- **Scenario 1 – No Branches.** Your team works only from the main source tree. In this case, you do not create branches and you do not need isolation. This scenario is generally for smaller or medium sized teams that do not require isolation for teams or for features, and does not need the isolation for releases.
- **Scenario 2 – Branch for Release.** Your team creates branches to support an ongoing release. This is the next most common case where you need to create a branch to stabilize for a release. In this case, your team creates a branch before release time to stabilize the release and merges changes from the release branch back into the main source tree after the software is released.
- **Scenario 3 – Branch for Maintenance.** Your team creates a branch to maintain an old build. In this case, you create a branch for your maintenance efforts, so that you do not destabilize your current production builds. You may or may not merge changes from the maintenance branch back into the main tree. For example, you may work on a quick fix for a subset of customers that you do not want to include in the main build.
- **Scenario 4 – Branch for Feature.** Your team creates branches by features or teams. In this case, you create a development branch, you perform work in the development branch, and then you merge back into your main source tree. You may create separate branches for specific features or for specific teams.
- **Scenario 5 – Branch for Team**. You branch to isolate sub teams so they can work without being subject to breaking changes or can work towards unique milestones.

You may encounter one or more of these scenarios. Use these scenarios as a reference point to see what guidance may or may not apply to you.

Example Folders and Their Purpose

The following folders are examples of folders you might create in Team Foundation Server source control when structuring your source tree for branching and merging scenarios.

- **Development**. Branched from **Main** and used to isolate active development.
- **Main.** Contains your main source tree.
- **Release.** Branched from **Main** and contains the release you are currently locking down prior to release.

- **Maintenance** (container). Contains branches you have already shipped but now need to maintain for customers. Provides isolation from the active development occurring in your development branch.

Key Notes

Note the following important points about these folders:

- **Development**. This is branched from **Main** to provide isolated development.
- **Main**. This contains your most recent development builds.
- **Release**. This contains your release candidate. You work in this branch to prepare your software for release, while others continue to work in the **Development** branch working on new features.
- **Maintenance**. As soon as you release your software, you create a **Maintenance** folder and move your **Release** branch into it. You use it to perform maintenance work. Use labels to mark pre-maintenance builds you may want to return to.

The following sections show the use of branching in each of the preceding scenarios with concrete examples.

Scenario 1: No Branches

This generally applies to smaller teams where you do not need to worry about isolation. By labeling builds you are able to retrieve the source corresponding to a particular build. There is no need to introduce branching complexity. You work directly from Main.

My Project
  Main
    Source

Scenario 2: Branch for Release

In this scenario your team creates a branch to stabilize the release and then merges the release branch back into the main source tree after the software is released. The following is a physical view showing branching for releases.

Main → Main integration branch
  Source
Release 1 → Release branch
  Source

Scenario 3: Branch for Maintenance

In this scenario, you create a branch for your maintenance efforts, so that you do not destabilize your current production builds. The following is a physical view showing maintenance branches:

Main →Main integration branch
  Source
Maintenance →Maintenance branch container

Release 1 →Maintenance branch
    Source
    Other Asset Folders
Release 2 → Maintenance branch
    Source
    Other Asset Folders

Scenario 4: Branch for Feature

In this scenario, you create a development branch, perform work in that branch, and then merge your work back into your main source tree. You organize your development branches by product features. The following is a physical view showing branching for feature development:

Development → Isolated development branch container
    Feature A → Feature branch
        Source
    Feature B → Feature branch
        Source
    Feature C → Feature branch
        Source
Main → Main Integration branch
    Source

Scenario 5: Branch for Team

This scenario is similar to the proceeding branch by feature scenario except that you organize your development branches by sub team, rather than by product feature. There might be a one to one correspondence between team and feature but in some cases a team might work on multiple features. The following is a physical view showing branching for sub team development:

Development → Isolated development branch container
    Team 1 → Team branch
        Feature A → Isolated branch for development
            Source
        Feature B → Isolated branch for development
            Source
    Team 2 → Team branch
        Feature A → Isolated branch for development
            Source
        Feature B → Isolated branch for development
            Source
Main → Main Integration branch
    Source

Physical Structure

The physical structure is the view you see in Source Control Explorer in Visual Studio. The following example shows the use of feature branches to isolate development, a main branch to integrate changes and produce a scheduled integration build, a release branch used to stabilize an upcoming release, and older maintenance branches used to support previous releases.

- **My Project**
    - **Development –** Container to isolate active development branches
        - **Feature A –** Branch for isolated development
            - **Source**
        - **Feature B –** Branch for isolated development
            - **Source**
    - **Main –** Main integration and build branch. All changes come together here.
        - **Source**
        - **Other Asset Folders**
    - **Release 3 –** Branch containing the code currently being locked down to release
        - **Source**
    - **Maintenance –** Container for maintenance branches
        - **Release 1 –** Active maintenance branch
            - **Source**
            - **Other Asset Folders**
        - **Release 2 –** Active maintenance branch
            - **Source**
            - **Other Asset Folders**

Note: Each branch contains source files and folders under it. In the preceding example, this is labeled Source. This is the set of source files that were branched from Main.

Logical Structure

The logical structure consists of the parent/child relationships for each branch. This may be different from the physical structure you can see in the Source Control Explorer. For example, in the preceding physical structure, **Release 2** and **Main** appear to be peers. **Release 2** is actually a child of **Main**. Figure 5.1 shows an example of this logical relationship and how branches and merges flow through the structure.

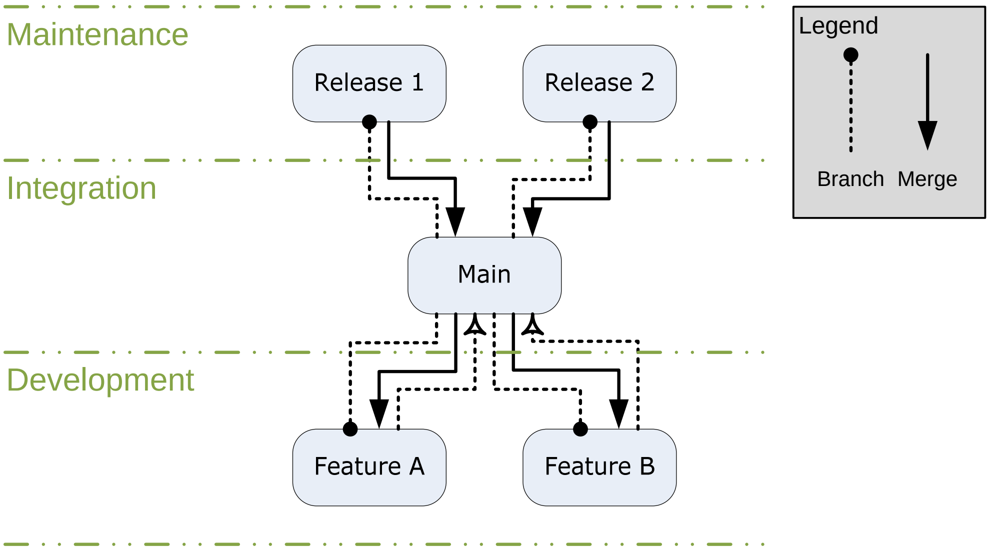


Figure 5.1 *Logical Relationship Showing Branch and Merge Flow*

Release Process

Figure 5.2 shows a typical timeline when branching for a release:

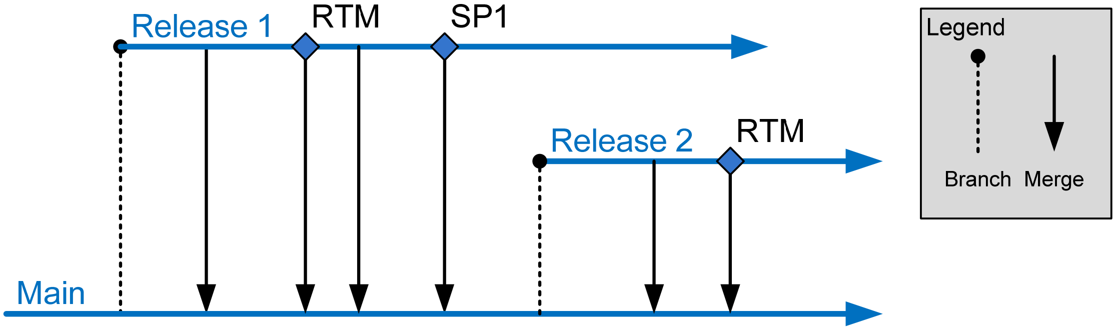


Figure 5.2 *Branching for Release Timeline*

The sequence of events is as follows:

1. The **Release 1** branch is created from **Main** once the release is ready to be locked down.
2. Periodic merges into **Main** ensure bug fixes from the release are moved into the main integration branch
3. The release build is labeled in the **RTM** branch and merged back into **Main**.
4. A service pack, SP1, is released. The build is labeled and changes are merged into **Main**.
5. The **Release 1** branch lives on in support of SP1 and to enable future service packs.

This process repeats for future releases.

Note: Successful use of release branches requires identifying which branch the fix should be applied to before implementing it. If you release a fix as a hotfix or service pack, you should make the change first on the appropriate Release branch, and then merge it into **Main** to ensure it gets applied to future releases.

Isolated Development Process

Figure 5.3 shows a typical timeline when branching for development isolation.

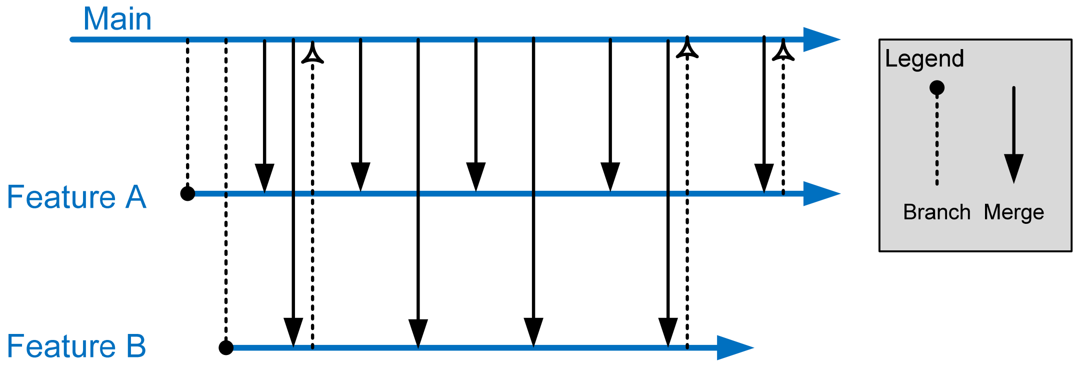


Figure 5.3 Branching for Development Isolation TimeLine

The sequence of events is as follows:

1. A feature branch is created to isolate development for Feature A.
2. A feature branch is created to isolate development for Feature B.
3. Each team merges their changes to Main as feature milestones are complete so that they can be integrated into the main build and picked up by other teams.
4. On a periodic schedule each team merges the latest changes from Main in order to stay in sync with the work of other teams and reduce the number of conflicts per merge.

Branching Considerations

When branching, consider the following:

- Structure your branch trees so that you only need to merge along the hierarchy (up and down the branch tree) rather than across the hierarchy. Branching across the hierarchy requires that you use a baseless merge which requires more manual conflict resolution.
- The branch hierarchy is based upon the branch parent and branch child, which may be different than the physical structure of the source code on disk. When planning your merges, keep the logical branch structure in mind rather than the physical structure on disk.
- Do not branch too deeply. Because there is latency in merging branches, a deep branching structure can mean that changes in a child branch may take a very long time to propagate to the main branch. This can negatively impact project schedules and increase the time to fix bugs.
- Branch at a high-level and include configuration and source files.
- Allow your branching structure to evolve over time.

- Do not branch unless your development team needs to work on the same set of files concurrently. If you are unsure, you can label a build and create a branch from that at a later point. Merging branches, especially if there are significant changes between them, can be costly.
- Merging requires one or more developers to execute the merge and sort out conflicts. The merged source must be thoroughly tested as it is not uncommon to make bad merge decisions that can destabilize the build.
- Merging across the branch hierarchy is especially difficult and requires you to manually handle many conflicts that could otherwise be handled automatically.

The decision whether to create a branch can be boiled down to the following:

- Will the cost of merging conflicts in real time be higher or will the overhead cost of merging conflicts between branches be higher?

Large Project Considerations

Large development teams are likely to distinguish themselves from smaller teams in the following ways:

- Require a more complex branching and merging structure
- More likely to manage dependencies across solutions and team projects
- More likely to maintain multiple builds for components and teams

Large teams are likely to branch both by team and by feature and are also more likely to have one or more branches designed to integrate external dependencies. Since the branching structure is deeper there is more lag between a change in a development branch and getting that change into the main integration branch. For this reason it pays to think through your merging strategy when creating branches.

For example, will your merges be scheduled or event driven?

1. Reverse integration is generally scheduled, for example from a development branch to the main integration branch.
2. Forward integration, such as from main integration branch to a development branch, is generally based on an event (such as feature milestone complete). This event is instigated when the team behind the development branch feels they are ready to merge changes from their parent branch.

Rationalize your branch/merge strategy with the frequency with which you want to produce builds. A deeper branching structure results in more time to merge from child branches to the main integration branch. Symptoms of this causing a problem for your development team include:

- Broken builds where the fix takes too long to propagate up to the main integration branch.
- Missed milestones because features take too long to propagate up to the main branch.

- Your team is spending large amounts of time merging changes between various branches.

If this becomes a problem for your team consider reducing the depth of your branching structure.

The following is an example of a branching structure for a large team:

- **My Project**
    - **Development** – Container to isolate active development branches
        - **Team 1**
            - **Feature A** – Isolated branch for development
                - **Source**
            - **Feature B** – Isolated branch for development
                - **Source**
        - **Team 2**
            - **Feature A** – Isolated branch for development
                - **Source**
            - **Feature B** – Isolated branch for development
                - **Source**
    - **Main –** Main integration and build branch. All changes come together here.
        - **Source**
        - **Other Asset Folders**
    - **Release 4 –** Branch containing the code currently being locked down to release
    - **Maintenance –** Container for maintenance branches
        - **Release 2**– Active maintenance branch
            - **Source**
            - **Other Asset Folders**
        - **Release 3 –** Active maintenance branch
            - **Source**
            - **Other Asset Folders**
    - **Safe Keeping**
        - **Release 1 –** Old release in safe keeping
            - **Source**
            - **Other Asset Folders**

This structure includes:

- Feature branches for multiple teams to work in isolation.
- A main branch to gather changes by all teams as well as bug fixes from the release branches
- A release branch used to lockdown the current release
- A maintenance branch for an old release that is being actively maintained and supported

- Safekeeping branches for older releases that are no longer being maintained

Additional Resources

- To see more Visual Studio Team System Guidance, visit the CodePlex site – http://www.codeplex.com/VSTSGuidance
- Team Foundation Server Branching Guidance: http://www.CodePlex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Chapter 6 - Managing Source Control Dependencies in Visual Studio Team System

Objectives

- Manage source control dependencies in Visual Studio Team System.
- Reference projects and assemblies from different solutions in the same team project.
- Reference projects and assemblies from other team projects.
- Reference third party assemblies.
- Manage Web service references in a team environment.
- Manage database references in a team environment.

Overview

This chapter explains how you should handle source control dependencies both within and across Visual Studio solutions. A consistent approach to managing dependencies in a team environment is necessary in order to reduce build instability and ongoing source control maintenance costs.

Dependencies include other projects, external assemblies, Web services and databases. Dependencies inevitably change over time and as a result they impact the build process (and the build order) of your application. A good dependency management approach improves the integration process while making builds as seamless as possible.

How to Use This Chapter

Use this chapter to learn about managing dependencies in a team environment. You can either read this chapter from top to bottom or read the section that addresses your specific dependency management requirement. Use the “Scenarios and Solutions” section to understand the general dependency management scenarios in a team environment. Use this section as a jumping off point to following sections that describe each dependency scenario.

- **Use the “Referencing Projects” section** to learn how to manage dependencies on other projects both inside and outside of your current team project.
- **Use the “Referencing Third Party Assemblies” section** to learn how to manage dependencies on third party assemblies for which you do not own the source.
- **Use the “Referencing Web Services” section** to learn how to reference shared Web services in a team environment.
- **Use the “Referencing Databases” section** to learn how to reference and connect to shared databases in a team environment.

Scenarios and Solutions

The following scenarios are common when managing dependencies:

1. You want to reference an assembly generated by another project in the same solution
2. You want to reference an assembly generated by another project in a separate solution.
3. You want to reference an assembly contained in another team project.
4. You want to reference a 3rd party assembly.

Referencing Assemblies Generated by Another Project in the Same Solution

If you need to reference another assembly in the same Visual Studio solution use a Visual Studio project reference. By using project references, you enable Visual Studio to do a few things automatically for you, such as keeping building configuration synchronized (debug/release) and tracking versioning and rebuilding components as necessary when assembly versions change.

Referencing Assemblies Generated by Projects in a Separate Solution

If you need to reference an assembly generated by a project in a different Visual Studio solution, you have two choices:

- Use a file reference to point to the binary assembly.
- Add the Visual Studio project (project and source files) to your solution and then use a project reference.

File references are more fragile than project references, do not adhere to your build configuration, and are not tracked by Visual Studio build dependencies. This means if the assembly that you have referenced changes, the Visual Studio build system does not automatically know that a rebuild is required.

Alternatively you can branch the external project into your solution, build the binary and then use a project reference. The reference will be more robust, however you will need to merge from the source branch regularly in order to pick up changes.

Referencing an Assembly From Another Team Project

If you share source or binaries across team projects you have two options:

- **Branching.** With this approach you branch the source from the other team project into your current solution. This creates a configuration that unifies the source from the shared location and their project on the server-side.
- **Workspace Mapping**. With this approach you map the source from the other team project into a workspace on your development computer. This creates a configuration that unifies the source from the other team project and your project on the client-side.

Branching is the preferred approach because it stores the dependency relationship on the source control server. Workspace mapping is a client-side only approach, which means that you and every developer must create the mapping on your own computers and also on the build server in order to be able to build the application. Branching adds additional merge overhead but it allows you to make the decision to pick up updated binaries or source more explicit.

Referencing a 3rd Party Assembly

This scenario is very similar to referencing across team projects except that you only share binaries, not source code. The choice between branching and workspaces is very similar with the added caveat that the overhead is likely to be less because 3rd party assemblies are less likely to change as frequently.

Referencing Projects

If you have a Visual Studio project, for example containing shared library code that is used by multiple team projects, you can either manage the project within the owning team's project or you can create a separate team project specifically for the shared project.

If you choose the latter approach and use a common shared project, the folder structure in Team Foundation Server source control looks like Figure 6.1:

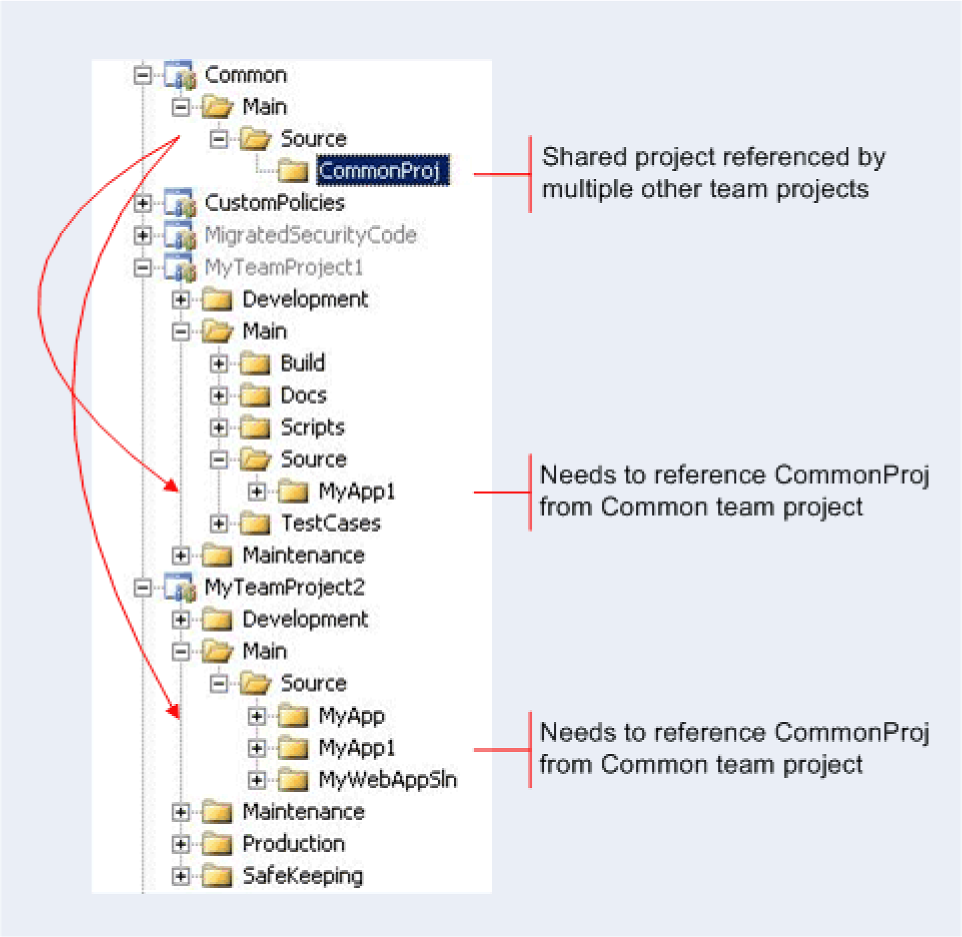


Figure 6.1 *Using a Common, Shared Project Folder Structure*

To consume the common shared project from your own team project you have two options:

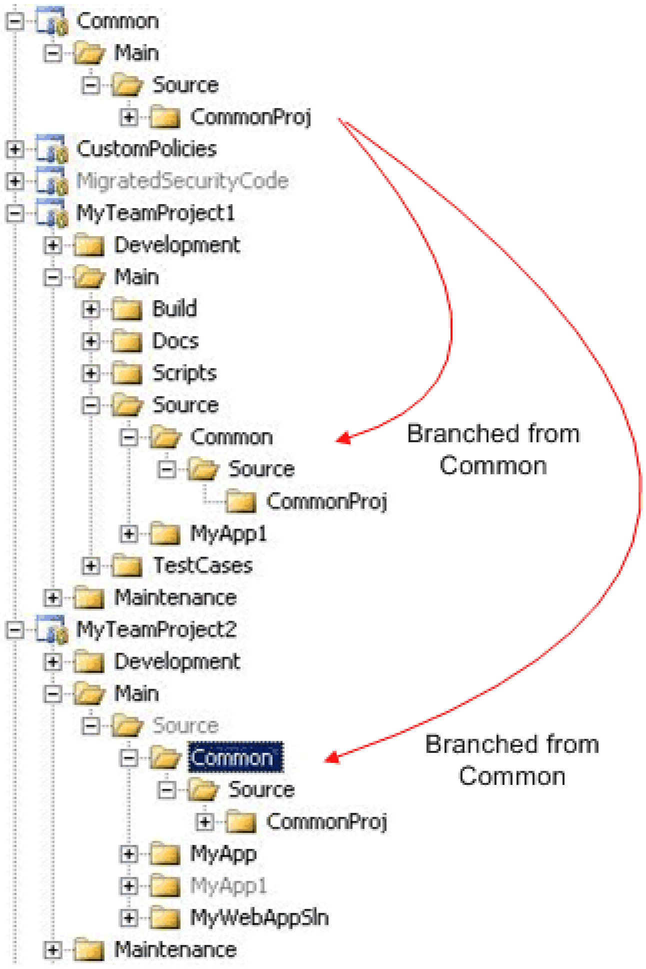
- **Branching**
- **Workspace Mapping**

Branching

Branching is the preferred method for most shared source scenarios. It enables you to pull the shared source into your project and check it into your team project's source control. In this scenario, you branch the source from the common shared location into your team project. This creates a configuration that unifies the source from the shared location and their project on the server-side.

Shared source changes are picked up as part of a merge process between the branches. This makes the decision to pick up changes in the shared source more explicit, easier to test, but also adds some overhead. It also makes the use of Team Build much simpler since the mapping is done server side, there is no client-side mapping that needs to be duplicated on the build server.

For example if you have two team projects $TeamProject1 and $Common, and Common is the shared project source, you create a branch from the shared location to the project which is referencing it. The TFS folder structure should resemble the one shown in Figure 6.2.



Branched from Common

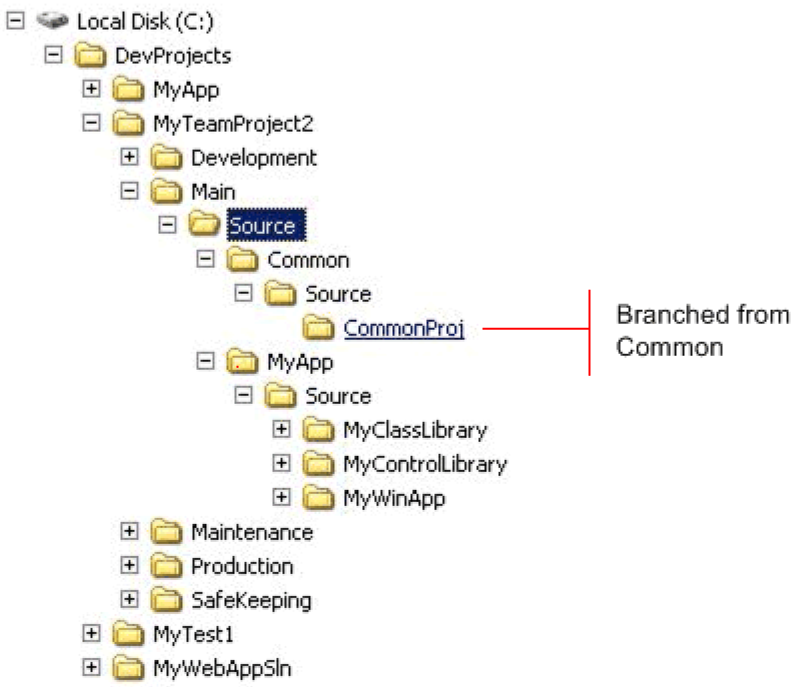
Branched from Common

Figure 6.2 *Using Branches*

Your workspace mapping should resemble the following:

Source Control Folder	**Local Folder**
$/MyTeamProject1/Main/Source/	C:\MyTeamProject\Main\Source

The client side workspace folder structure should resemble the one shown in Figure 6.3:



Branched from Common

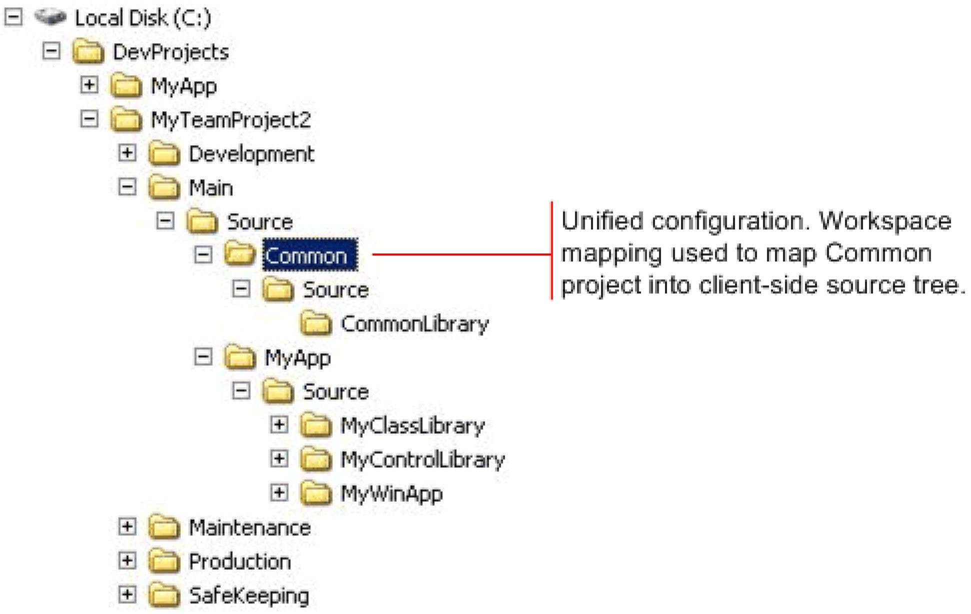
Figure 6.3 *Client Side Workspace Mapping*

Workspace Mapping

If you want your developers to instantly pick up any code changes from the shared source without the overhead of branching and merging you can map the shared source from the common project into the workspace on your development computer. This creates a configuration that unifies the source from the shared location and your project on the client-side.

The advantage of this approach is that shared project changes are picked up every time you retrieve the latest source into your workspace. However, this makes the use of Team Build more complex since the workspace mapping is a client-side construct.

For example if you have two team projects $MyTeamProject and $Common, and Common is the shared project source, for referencing the code from the shared location, these projects share a common path on the client’s hard drive. The client side workspace folder structure should resemble the one shown in Figure 6.4.



Unified configuration. Workspace mapping used to map Common project into client-side source tree.

Figure 6.4 *Using Workspace Mapping*

The workspace mappings should resemble the following:

Source Control Folder	**Local Folder**
$/MyTeamProject2/Main/Source/	C:\DevProjects\MyTeamProject2\Main\Source\
$/Common	C:\DevProjects\MyTeamProject2\Main\Source\Common

For more information see “Working with multiple team projects in Team Build” at http://blogs.msdn.com/manishagarwal/archive/2005/12/22/506635.aspx

Referencing Third Party Assemblies

If you cannot use a project reference and you need to reference an assembly outside of your current solution's project set, such as a third party library, and you do not wish to or cannot create a branch from the originating project into yours, you must set a file reference.

Managing shared binaries is similar to managing shared project source, and you must decide where you want to store the binaries and how you want your team to access the binaries. If you have binaries that are used by multiple team projects, you can either manage the binaries within the team project of the owning team or create a team project specifically for the shared binaries.

For the teams consuming the shared binaries, the same two options available for referencing projects are available for binaries.

- **Branching**

- **Workspace Mapping**

Branching

In this scenario, you branch the binaries from the common shared location into your team project. This creates a configuration that unifies the binaries from the shared location and their project on the server-side.

The difference is that any changes to the binaries such as new versions are picked up as part of a merge process between the branches. This makes the decision to pick up changed shared binaries much more explicit.

For example if you have two team projects $TeamProject1 and $Common, and Common contains the shared binaries, you create a branch from the shared location to the project which is referencing it. The TFS folder structure should resemble the one shown in Figure 6.5.

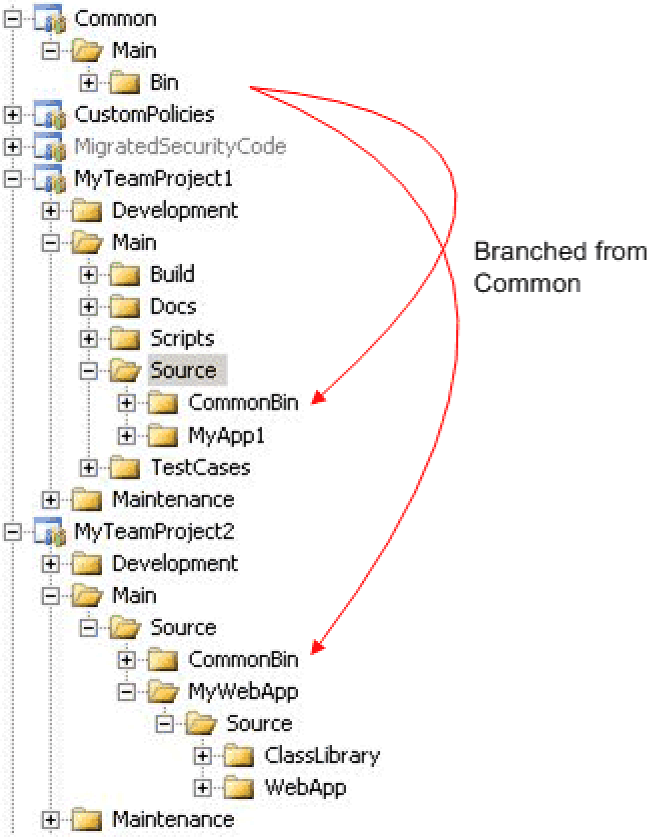


Figure 6.5 *Branching from Common*

Your workspace mapping should resemble the following:

Source Control Folder	**Local Folder**
$/MyTeamProject1/Main	C:\MyTeamProject1\Main

The client side workspace folder structure should resemble the one shown in Figure 6.6.

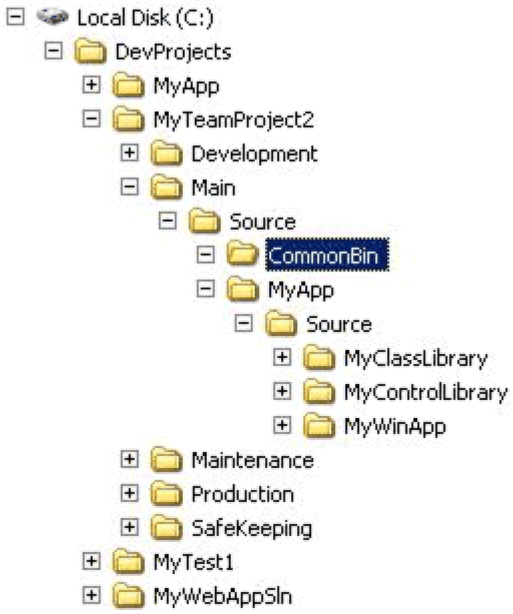


Figure 6.6 *Client-Side Workspace Folder Structure*

Workspace Mapping

In your team project that consumes the shared binaries, you reference the binaries from the shared location into your workspace on your development computer. This creates a configuration that unifies the binaries from the shared location and your project on the client-side.

The advantage of this approach is that shared binaries changes are picked up every time you retrieve the latest source into your workspace.

For example if you have two team projects $TeamProject1and $Common, where reference the binaries in Common from TeamProject1, then the client side workspace folder structure should resemble the one shown in Figure 6.7.

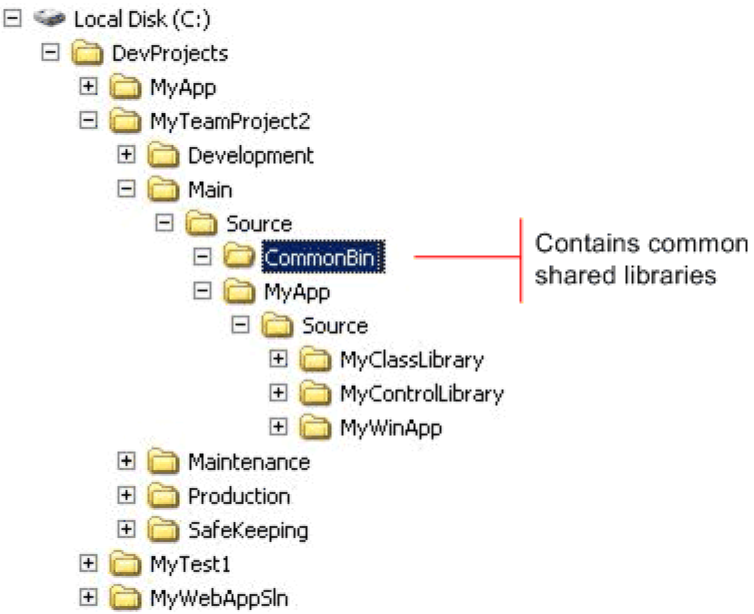


Figure 6.7 *Storing Common Shared Libraries*

The workspace mappings should resemble the following:

Source Control Folder	**Local Folder**
$/MyTeamProject2/Main/	C:\DevProjects\MyTeamProject2\Main\
$/Common/Main/Bin	C:\DevProjects\MyTeamProject2\Main\Source\CommonBin

For more information see “Working with multiple team projects in Team Build” at http://blogs.msdn.com/manishagarwal/archive/2005/12/22/506635.aspx

Guidelines for Referencing Projects and Assemblies

You can set a file reference in one of two ways:

- To reference a .NET Framework assembly, you select the assembly from the list displayed on the **.NET** tab of the **Add References** dialog box.
- You can use the **Browse** button in the **Add Reference** dialog box.

Assemblies such as System.XML.dll are located in the Global Assembly Cache (GAC). However, you never directly refer to an assembly within the GAC. Instead, when you select an assembly on the **.NET** tab of the **Add References** dialog box, you actually reference a copy of the assembly, located within the %windir%\Microsoft.NET\Framework\<version>\ folder

Project references are preferable to file references, so when managing assembly references, keep the following guidelines in mind:

- **Use project references whenever possible**
- **Use file references only where necessary**
- **Use Copy Local = True for project and file references**

Use Project References Whenever Possible

You should use a project reference whenever possible because they provide the following advantages:

- They work on all development workstations where the solution and project set are loaded. This is because a project Globally Unique Identifier (GUID) is placed in the project file, which uniquely identifies the referenced project in the context of the current solution.
- They enable the Visual Studio build system to track project dependencies and determine the correct project build orders.
- They avoid the potential for referenced assemblies to be missing on a particular computer.
- They automatically track project configuration changes. For example, when you build using a debug configuration, any project references refer to debug assemblies generated by the referenced projects, while they refer to release assemblies in a release configuration. This means that you can automatically switch from debug to release builds across projects without having to reset references.
- They enable Visual Studio to detect and prevent circular dependencies.

You can use a project reference if the assembly is already within your solution's project set. If the assembly is outside of your solution's project set and you still wish to you a project reference you can branch the dependency from the originating project into your project. When you wish to pick up a new version of the dependency perform a merge from the originating project into your branch.

Use File References Only Where Necessary

If you cannot use a project reference because you need to reference an assembly outside of your current solution's project set, and you do not wish to create a branch from the originating project into yours, you must set a file reference.

Use Copy Local = True for Project and File References

Every reference has an associated copy local attribute. Visual Studio determines the initial setting of this attribute (true or false) when the reference is initially added. It is set to false if the referenced assembly is found to be in the GAC; otherwise, it is set to true. You should not change this default setting.

With copy local set to true, the Visual Studio build system copies any referenced assembly (and any dependent downstream assemblies) to the client project's output folder when the reference is set. For example, if your client project references an assembly called Lib1, and Lib1 depends on Lib2 and Lib3, then Lib1, Lib2, and Lib3 are copied to your project's local output folder automatically by Visual Studio at build time.

Automated Dependency Tracking

Each time you build your local project, the build system compares the date and time of the referenced assembly file with the working copy on your development workstation. If the referenced assembly is more recent, the new version is copied to the local folder. One of the benefits of this approach is that a project reference established by a developer does not lock the assembly dynamic-link library (DLL) on the server and does not interfere in any way with the build process.

Referencing Web Services

In simpler systems where all of the projects for the system are contained within the same team project, all developers end up with local working copies of all Web services because they are defined by Visual Studio projects within the team project. When you open a solution from source control for the first time, all projects (including any Web services) are installed locally. Similarly, if a Web service is added to the solution by another developer, you install the Web service the next time you refresh your solution from source control. In this scenario, there is no need to publish Web services on a central Web server within your team environment.

For larger systems, Web services can be published through Internet Information Server (IIS) on a centrally accessed Web server and not all developers need to locally install the Web service. Instead developers can access the Web service from their client projects although you need to reference the Web service appropriately as discussed below.

Use Dynamic URLs When Referencing Web Services

If you want to call a Web service, you must first add a Web reference to your project. This generates a proxy class through which you interact with the Web service. The proxy code initially contains a static Uniform Resource Locator (URL) for the Web service, for example http://localhost or http://SomeWebServer.

Important - For Web services in your current solution that execute on your computer, always use http://localhost rather than http://MyComputerName to ensure the reference remains valid on all computers.

The static URL that is embedded within the proxy is usually not the URL that you require in either the production or test environments. Typically, the required URL varies as your application moves from development to test to production. You have three options to address this issue:

- You can programmatically set the Web service URL when you create an instance of the proxy class.
- A more flexible approach that avoids a hard coded URL in the proxy, is to set the **URL Behavior** property of the Web service reference to **Dynamic**. This is the preferred approach. When you set the property to **Dynamic**, code is added to the proxy class to retrieve the Web service URL from a custom configuration section of the application configuration file, Web.config for a Web application or SomeApp.exe.config for a Windows application.

- You can also generate the proxy by using the WSDL.exe command line tool and specifying the /urlkey switch. This works in a similar way to setting the **URL Behavior** property in that it adds code to the proxy to retrieve the Web service URL, but in this case the URL is stored in the <appSettngs> section of the application configuration file.

The dynamic URL approach also lets you provide a user configuration file, which can override the main application configuration file. This allows separate developers (and members of the test team) to temporarily redirect a Web service reference to an alternate location.

How to Use Dynamic URLs and a User Configuration File

Set the **URL Behavior** property of your Web service reference to **Dynamic** to gain maximum configuration flexibility both within the development and production environments. By default Visual Studio sets the value of this property to **Dynamic** when you add a Web reference. To check that this value is still set to **Dynamic**:

1. In the Solution Explorer, expand the list of Web references.
2. Select each Web reference in the list,
3. For each Web reference, check that the value of the **URL Behavior** property is set to **Dynamic**.

To specify a Web Service URL in a user configuration file

1. When you first add a Web reference Visual Studio will automatically generate the app.config for you. This file contains the Web service reference and the the configuration setting in the app.config file looks like the following:

```
<configuration>
    <configSections>
        <sectionGroup name="applicationSettings"
type="System.Configuration.ApplicationSettingsGroup, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name=" SomeService.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" />
        </sectionGroup>
    </configSections>
    <applicationSettings>
        <YourProject.Properties.Settings>
            <setting name="SomeService_ localhost _Service"
serializeAs="String">

<value>http://localhost/someservice/Service.asmx</value>
            </setting>
        </ YourProject.Properties.Settings>
    </applicationSettings>
</configuration>
```

This file contains a new configuration section that is used by the generated proxy. This configuration section contains the address of the Web service that Visual Studio found when generating this proxy.

Visual Studio also places the default value of the URL into the code generated for this proxy. This value lives in a file called Settings.Designer.cs. To see this file,

1. Right-click on the Web service reference in the **Solution Explorer**
2. Select **View in Object Browser**. In the Object Browser look for the entry that says **YourProject.Properties**, where YourProject is the project name that contains Web service reference.
3. Expand **YourProject.Properties** and you then double-click **Settings.** This opens the Settings.Designer.cs file which contain a line like this

```
[global::System.Configuration.DefaultSettingValueAttribute("http://loca
lhost:/webservice/Service.asmx")]
```

This is the default value used for the URL of the Web service if no configuration information is found.

You often need to change the URL of the Web service you are calling. For example you may want to test against the Web service running locally on your computer or against a version of the Web service running in a test environment. It is also highly likely that the URL of the production Web service is not the same as the URL used during development. To manage each of these URLs the URL value should be specified in a user configuration file, which is referenced from the main App.config file. The name of the user configuration file is arbitrary. The following example uses user.config as the file name to be clear that this is where the user's configuration would go.

To create a user.config file perform the following steps

1. In Solution Explorer, right-click the project that contains the Web service reference, point to **Add** and click **New Item**
2. Select **Application Configuration File**, change the name to User.config and click **Add.**
3. Copy the <YourProject.Properties.Settings> element setting from your application configuration file (app.config) to the user.config file. This file should just contain the element for which the runtime is redirected. There should be no other configuration element present in the user.config file. Delete the <?xml> directive and the <configuration> element if present. Here is an example

```
<YourProject.Properties.Settings>
    <setting name="SomeService_localhost_Service" serializeAs="String">
       <value>http://localhost/someservice/Service.asmx</value>
    </setting>
</YourProject.Properties.Settings>
```

Individual developer should set the contents of their user.config file as needed to reference the appropriate URL. You now need to specify that the configuration system should use this user.config file for this configuration data rather than the app.config file. To do this you must update the app.config file as follows

1. Add a **configSource**="user.config" attribute to the <YourProject.Properties.Settings> element of your main application configuration file. This silently redirects the runtime to the named user configuration file when it accesses information from this section.
2. Delete the content of the <YourProject.Properties.Settings> element.

Your app.config should now look like this

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <sectionGroup name="applicationSettings"
type="System.Configuration.ApplicationSettingsGroup, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name="SomeService.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" />
        </sectionGroup>
    </configSections>
  <applicationSettings>
    <yourProject.Properties.Settings  configSource="user.config">
    </YourProject.Properties.Settings>
   </applicationSettings>
</configuration>
```

In the preceding example YourProject is the name of the project that contains the Web service reference.

Important - If you use the **configSource** attribute then the user configuration file must be present, with only the <SomeService.Properties.Service> element. You must also ensure that when you add the configSource=”user.config” attribute you remove the XML content from the <YourProject.Properties.Service> element.

When you use the User.config approach:

- Make sure that you deploy your user.config file along with the application code. To do this in the Solution Explorer, right click the user.config file and click **Properties** option. Set the **Copy To Output Directory** property to **Copy if newer**.

- Don't add your User.config file to source control. In this way, each developer (and the test team) can explicitly bind to specific URLs by using their own User.config files. Source control may contain other User.config files, for example for testing and for production. These files should be managed by the users responsible for managing the testing and production environments. These test and production User.config files should not be stored as part of the Web service projects but should be in different areas of the source control system.

- Store a global user.config file in source control. This could either contain only the root element (no <setting> element) or it could specify the default location of the web service. The user.config file must be present for the configuration system to work.

Tip: By default, the user configuration file is automatically added to source control when you add the solution. To prevent this, when you first check in the file, de-select the user.config file so it is not checked in. You can then right-click on the file in Solution Explorer and select **Under Pending Changes** option, to ensure that the file is subject to source control.

Important - If the user.config file that specifies the URL only contains the root element (i.e. there is no setting element in the user.config) then the Web service proxy will use its default value for the URL. This default value is hard coded into the proxy in a file called Settings.Designer.cs. The value is specified as a **DefaultValueSettings** attribute and will look like this

```
[global::System.Configuration.DefaultSettingValueAttribute("http://loca
lhost/webservice/Service.asmx")]
```

Important - For Web applications that use a user configuration file, by default, any changes made to the file result in the Web application being automatically recycled. If you do not want to recycle the application when a value changes, add the **restartOnExternalChanges**="false" attribute to the configuration section definition.

```
<configSections>
      <sectionGroup name="applicationSettings"
type="System.Configuration.ApplicationSettingsGroup, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
            <section name="Test.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" restartOnExternalChanges="true"/>
      </sectionGroup>
</configSections>
```

If you add this attribute to the configuration section in the web.config file then changes to the external user.config configuration file will not cause the Web application to recycle, however those changes will still be visible to the application.

It is important to realize that if you are using this mechanism then the User.config file must be present. Somebody must be responsible for ensuring the environment is correct when creating builds for production releases and for any test environments. As part of this build setup the appropriate User.confg file must be retrieved from the source control system and copied into the correct location for MSBuild to be able to find it.

Referencing Databases

Database references in the form of connection strings can also be managed by using an external configuration file. The advantage of this is that each developer can easily specify his or her own connection string is their own private User.config file. Any changes made by one developer, such as redirecting the connection to a local database for unit testing purposes, does not affect other developers.

User configuration files can also be used to control environmental-specific settings, such as those required by a test environment. The test environment can also use a User.config file which references the test database.

The procedure is similar to the preceding Web references example, except that in that example the Web service proxy contains the code to retrieve the Web service URL from the configuration file. For database connection strings, you must provide the code to read the connection string.

How to Use User Configuration Files for Database Connection Strings

The following procedure explains how to store and then reference a database connection string within a user configuration file.

To use a user configuration file to store database connection strings

1. Add a configSource="user.config" attribute to the <connectionStrings> element of your main application configuration file.

```
<configuration>
   <connectionStrings configSource=”user.config”/>
</configuration>
```

2. To override the main application configuration file, create a User.config file (located in the same folder as the application configuration file), and then add a similar <connectioStrings> entry to the file. Notice that the following connection string references a local database.

```
<connectionStrings>
      <add name="DBConnStr"
     connectionString="server=localhost;Integrated 
Security=SSPI;database=Accounts"/>
   </connectionStrings>
</configuration>
```

3. Within your project, use the following code to obtain the connection string from the user configuration file. This code uses the static **ConnectionStrings** property of the **System.Configuration.ConfigurationManager class.** In Win Form application you must add a reference to **System.Configuration.dll** explicitly.

```
using System.Configuration;
```

```
private string GetDBaseConnectionString()
{
  return
ConfigurationManager.ConnectionStrings["DBConnStr"].ConnectionString;
}
```

4. Ensure that the User.config file is deployed along with the application code. To do so, in the Solution Explorer right click the User.config file and click the **Properties**. In the properties pane, set the **Copy To Output Directory** property to **Copy if newer**.

5. Do not add the User.config file to source control. In this way, each developer (and the test team) can explicitly specify the connection string through their own User.config file. Source control may contain other User.config files, for example for testing and for production. These files should be managed by the users responsible for managing the testing and production environments. These test and production User.config files should not be stored as part of the database projects but should be in different areas of the source control system.

6. In source control you should have a User.config file for each of the environments that you use, such as production and test. These configuration files should specify the connection string for the database. The User.config file must be present for the configuration system to work.

Tip: By default, the user configuration file is automatically added to source control when you add the solution. To prevent this, when you first check in the files, de-select the User.config file so it is not checked in. You can then right-click on the file in Solution Explorer and select **Under Pending Changes** to ensure that the file never comes under source control.

It is important to realize that if you are using this mechanism then the User.config file must be present. Somebody will be responsible for ensuring the environment is correct when creating builds for production releases and for any test environments. As part of this build setup the appropriate User.confg file will need to be retrieved from the source control system and copied into the correct location for MSBuild to be able to find it.

Additional Resources

- To see more Visual Studio Team System Guidance, visit the CodePlex site – http://www.codeplex.com/VSTSGuidance
- For more information on project references, see “Project References”: http://msdn2.microsoft.com/en-us/library/ez524kew(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on workspaces, see: http://msdn2.microsoft.com/en-us/library/ms181383(VS.80).aspx

PART III

Builds

In This Part:

- ▶ **Team Build Explained**
- ▶ **Setting Up Continuous Integration with Team Build**
- ▶ **Setting Up Scheduled Builds with Team Build**

Chapter 7 - Team Build Explained

Objectives

- Understand Team Build architecture.
- Learn the components that make up Team Build.
- Understand what functionality Team Build provides.
- Select an appropriate build strategy.
- Understand how you may need to change your build strategy if working on a large project

Overview

This chapter begins explains how you can use Team Build to automate the build process. It outlines a number of common build-related stumbling blocks and compares various approaches to builds ranging from daily scheduled builds to continuous integration builds.

Team Build is built on top of the Microsoft Build Engine (MSBuild) and you can use it to retrieve the necessary source code for a build, compile solutions and, if required, execute unit tests and static code analysis tools as part of the build process. You can also release build output to a specified shared location.

Team Build labels the source code used for a particular build with build numbers so that you can retrieve the source used to create a specific build at some point in the future. In case of failures, you can configure Team Build to create work items and notify the users about the build failures encountered.

How to Use This Chapter

Use this chapter to learn about the functionality that Team Build provides for automating and managing the build process and to understand the different strategies for scheduling builds.

- **Read Chapter 8, “Setting up Continuous Integration with Team Build”** if you want to learn more about using continuous integration with Team Foundation Server.
- **Read Chapter 9, “Setting up Scheduled Builds with Team Build”** if you want to learn more about using scheduled builds.
- **Read the companion How Tos** to help perform build-related tasks:
  - How To: Automate Running Unit Test and Code Analysis with Team Build
  - How To: Create Build Notifications
  - How To: Setup a Continuous Integration Build
  - How To: Setup a Scheduled Build.

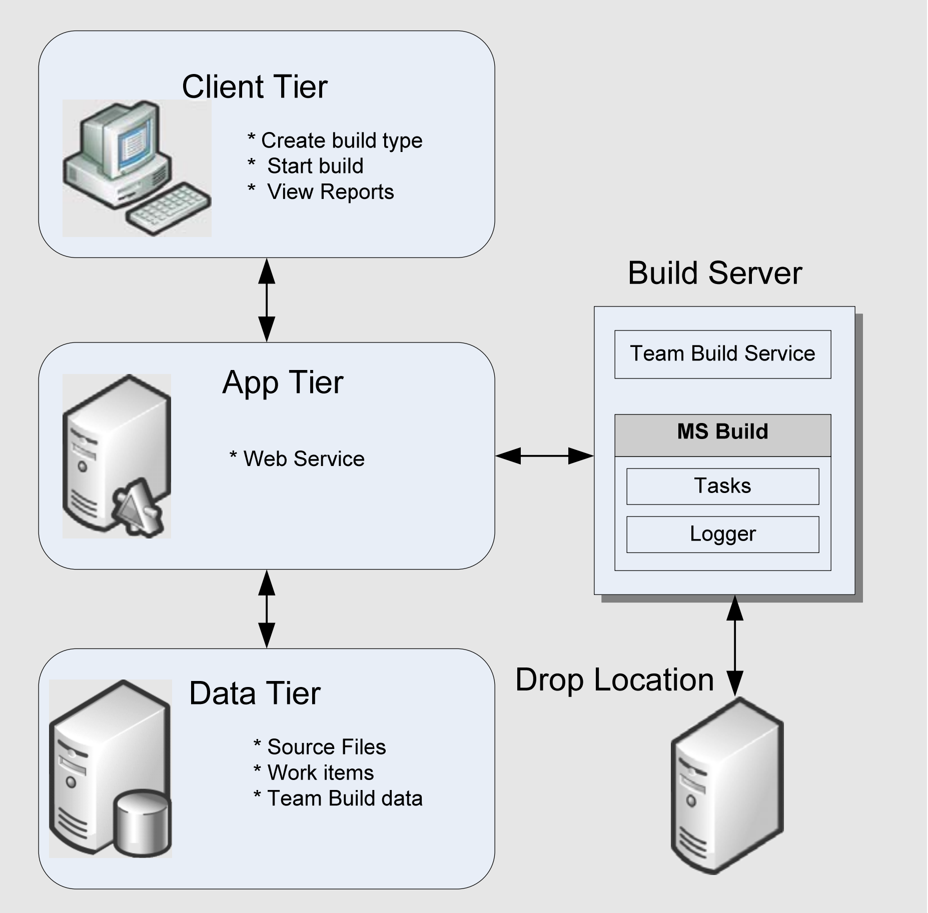
Architecture

This section introduces the architecture of Team Foundation Team Build through a description of its physical architecture and logical workflow.

Physical Architecture

- **New Team Build Type Creation Wizard –** This is a client side component accessed from Team Explorer. You can create new build types by using the New Team Build Type Creation Wizard.
- **Team Build Browser –** This is a client side component accessed from Team Explorer. You can view the Team Build reports and build progress information in Team Explorer by using the Team Build Browser
- **Source Control Web Service –** This is the application tier component used by the build service for accessing the source code.
- **Team Foundation Build Web Service –** This is the application tier component that accepts requests from the client and coordinates the execution of build steps.
- **Build Service –** This component can be on a separate build server or on the application tier itself. This service runs the build steps on instructions from the Team Build Web service.
- **Team Foundation Build Store –** This is a data tier component used to hold records related to Team Build processes.
- **Source Code database –** This is a data tier store of actual source which is accessed buy build service during the build processes.

Logical Workflow



Team Build integrates with Team Foundation Server through the Application Tier and interacts with work items, code coverage, code analysis, test cases and reporting.

The TFSBuild.proj file controls the build, including what projects to build, configurations, drop locations, code analysis, and tests to run. This file is generated by the Team Build wizard when the build is first created and then can be edited directly in order to make changes.
Team Build makes use of Team Foundation Server's eventing system. You can use events fired by Team Build to create custom build steps or to generate notifications of build status changes or when the build is complete.

Key Points

- Team Foundation Build is built on top of MSBuild.
- TFSBuild.proj file contains all of the build settings. You generate it by using wizard and then can directly edit it.
- The TFSBuild.rsp file contains command line options for MSBuild. This can be modified to further customize the build.
- Event notification allows custom build steps or notifications through the **BuildStatusChangeEvent** and **BuildCompletionEvent**.

- Team Build integrates with work items, code coverage, code analysis and test cases.

How Team Build Works

Team Foundation Build consists of the Team Build Service layered on top of the MSBuild build system. MSBuild is responsible for the build itself, Team Build Service is responsible for communication with the Team Foundation Server application tier. Team Builds are created from the Visual Studio client and can be started from either the client by an event on the build server, or from the command line for example as a scheduled task. A build, once started, consists of the following steps:

1. Get sources from source control into the build directory
2. Compile the source and generate binaries
3. Optional: Run code analysis
4. Create a work item if there is a build failure
5. Optional: Run tests
6. Optional: Calculate code coverage
7. Log build details
8. Copy build to the drop location

Once the build is complete, the following is available:

- Build details. Details can be viewed from any client or from reports.
- Build binaries. Binaries are placed in the drop location.
- BuildLog. The log can be reviewed for errors and warnings.
- Work item. If the build fails a work item is created to track the build break.

How to Think About Builds

Use the following steps to determine your build strategy:

1. Consider your build consumers
2. Review solution scenarios
3. Understand common stumbling blocks

Consider Your Build Consumers

Most development teams have one or more of the following build consumers:

- Development team
- Test team
- Internal dogfood adopters
- External beta adopters
- Release to customers

Each consumer has different build quality and build frequency needs. In general, they can be broken into two groups. Consumers who need predictable scheduled builds and consumers who need rapid event-driven builds. Scheduled builds are most commonly daily builds but can occur more or less frequently. Event-driven builds are usually initiated by a check-in and are designed to give rapid feedback to the development team. If your development team has problems with breaking builds, consider using a gated check-in model in which check-ins are tested before they are allowed into the source tree and are rejected if the tests fail.

Review Solution Scenarios

Choose solution scenarios based upon how closely the purpose and build consumers match your team's situation. When in doubt, use the simplest build scenario possible, such as a scheduled build, and add more complex scenarios only if necessary.

Schedule builds

Scheduled builds provide consistent binaries at specific intervals. For example, these can be builds for test teams, release or users. Examples of schedule builds would include daily or nightly builds or weekly drops.

Continuous Integration

Continuous integration builds provide feedback to the developers by building as soon after a check-in as possible.

Build Scenarios

- **Team with single daily builds.** Most teams rely on a scheduled build in order to provide consistent binaries to their test team and other users.
- **Team with daily builds and continuous integration builds.** Some teams choose to use continuous integration build in order to provide rapid feedback to their developers on every check-in. A pure continuous integration build works well for small teams but larger teams are likely to overload their build server due to high frequency check-ins and longer build times. When this happens, a rolling build can be used to build less frequently without giving up too much time between check-in and build feedback.
- **Team with multiple build labs each performing daily build and continuous integration builds.** Very large teams are likely to require more complex solutions to improve build quality and timeliness. Gated check-ins can be used to reduce build breaks by rejecting bad check-ins before they make into source control. Teams that branch can use multiple build servers, one per branch, to keep the builds focused on each branch's purpose. For instance integration branches would create integration builds while development branches would create development builds.

Scheduled Builds

A scheduled build is a time-based build that is created on a fixed time frequency. The purpose of a scheduled build is to produce consistent reliable builds that can be used to gain feedback on the quality of the build.

The consumers of a scheduled build are most often:

- Test team
- Internal dogfood adopters

- External adopters

Continuous Integration Build

A continuous integration is triggered any time there is a check-in. The purpose of a continuous integration build is to gain rapid feedback on build quality as soon after a check-in as possible.

The consumers of a continuous integration build are most often:
- Development team

Gated Check-ins

Gated check-ins is a process that requires each check-in to meet some quality bar before it can be added to the source tree. The purpose of gated check-ins is to reduce the prevalence of build breaks and improve the quality of the build by requiring each check-in is run through a set of tests before it can be admitted.

Understand Common Stumbling Blocks

There are several common scenarios that Team Build does not fully support by default. Review the following scenarios and if your team will be faced with one of them:
- **Building a Project with External Dependencies**. If you use branching to bring external dependencies into your team project then you can use project references and the build will work on the build server without any additional steps. If you use client-side workspace mapping to reference external dependencies, you'll need to maintain the mapping on the build server in order for the build to complete successfully. For more information, see Explained - Managing Source Control Dependencies in Visual Studio Team System
- **Building a Setup Project**. Team Build does not support setup projects by default. Use a custom post-build step to compile the setup project and copy the binaries to the build drop location. For more information, see How To – Build a Setup Project
- **Building .NET 1.1 Applications**. Team Build does not support .NET 1.1 applications by default. The MS Build Toolkit Extras for .NET 1.1 (MSBee) allows .NET 1.1 builds but requires that your projects and solutions are upgraded to VS 2005. If you cannot upgrade to VS 2005 projects and solutions you can use a custom post-build step to compile the .NET 1.1 application. To download MSBee, see http://www.codeplex.com/MSBee For more information on creating a custom post-build step to compile .NET 1.1 application, see Nagaraju's blog entry http://blogs.msdn.com/nagarajp/archive/2005/10/26/485368.aspx
- **Building VB6 Applications**. Team Build does not support VB6 applications by default. Use a custom post-build step to compile the VB6 application. To see similar post build step that can be used to compile a .NET 1.1 application, see Nagaraju's blog entry http://blogs.msdn.com/nagarajp/archive/2005/10/26/485368.aspx

Scheduled Build

The purpose of a scheduled build is to produce a consistent reliable build that can be used by test teams and others who will provide you quality feedback. Scheduled builds are usually daily builds, but they could occur more or less frequently depending upon your needs.

Use the following steps to create a scheduled build:
1. Create a batch file with a command line build
2. Use Windows Scheduled Tasks to execute the batch file on a schedule

For more information, see: ”Ch 09 - Setting Up Scheduled Builds with Team Build”

Continuous Integration Build

The purpose of a continuous integration build is to gain rapid feedback on build quality as soon after a check-in as possible. Depending on the size of your team, the length of your build, and the number of check-ins, choose one of the following strategies:
1. Continuous Integration build immediately following each check-in
2. Rolling build after a specified number of check-ins, or a specified time period – whichever comes first

For more information, see: “Ch 08 - Setting Up Continuous Integration with Team Build”

Large Project Considerations

If you work on a large development team, there are additional considerations to keep in mind. Large development teams are likely to distinguish themselves from smaller teams in the following ways:
- Require a more complex branching and merging structure
- More likely to manage dependencies across solutions and team projects
- More likely to maintain multiple builds for components and teams

Keep the following considerations in mind for large teams:
1. Large teams are likely to have longer build times. The frequency of CI builds should be less than the build time to avoid too much queuing and loading of the build server.
2. If your team branches, you can set up a build server per branch. This allows each build to be focused on the purpose of the branch – eg. integration or development.
3. Integration branches are likely to have scheduled builds only. Development branches may have continuous integration and scheduled builds.

Build Customization

TFSBuild.proj contains your team build settings, TFSBuild.rsp contains the command line options that are passed to MSBuild in order to execute the build on the build server.

Common scenarios for customizing the build include:

- You want to modify the build number that is generated for your build.
- You want to change the workspace mapping used on the build machine to change the files the build server gets from source control before the build. For instance, your project may include workspace dependencies through workspace mapping on the developer’s client machines. This mapping needs to be applied to the build machine as well.
- You want to change the compilation behavior of your build. For instance, to better support build and deployment of a line of business ASP.NET application.

Customize the Build Number

In order to customize the build number you need to generate the replacement build number and the write it to the assemblyinfo file.

Use the following steps to customize the build number:

1. Override the $(BuildNumber) in BuildNumberOverride target.
2. Override the BeforeCompile target to write the AssemblyInfo.cs/vb.

Example

```
<Target Name="BuildNumberOverrideTarget">
  <Message Importance="High" Text="$(BuildNumber)" />

  <ConvertTFSBuildNumberToSolutionBuildNumber
    MajorAndMinorVersion="1.0"
    TFSBuildNumber="$(BuildNumber)"
    TFSLastBuildNumber="$(LastBuildNumber)">
    <Output TaskParameter="SolutionBuildNumber" PropertyName="SolutionBuildNumber" />
    <Output TaskParameter="TFSBuildNumber" PropertyName="BuildNumber" />
  </ConvertTFSBuildNumberToSolutionBuildNumber>
  <Message Importance="High" Text="$(SolutionBuildNumber)" />
  <Message Importance="High" Text="$(BuildNumber)" />
</Target>

<Target Name="BeforeCompile">
  <Message Importance="High" Text="$(SolutionBuildNumber)" />
  <CreateItem Include="$(SolutionRoot)\**\AssemblyInfo.cs">
    <Output TaskParameter="Include" ItemName="AssemblyInfoFiles"/>
  </CreateItem>
  <CreateItem Include="$(SolutionRoot)\**\AssemblyInfo.vb">
    <Output TaskParameter="Include" ItemName="AssemblyInfoFiles"/>
  </CreateItem>
  <RewriteFileVersions
    AssemblyInfoFiles="@(AssemblyInfoFiles)"
    AssemblyVersionNumber="$(SolutionBuildNumber)"
    AssemblyFileVersionNumber="$(SolutionBuildNumber)"
    AssemblyInformationalVersionNumber="$(SolutionBuildNumber)" />
</Target>
```

Change Workspace Mapping

In order to change the workspace mapping for your build you need change the get location to match that on a developer’s machine.

Use the following steps to change the workspace mapping:

1. Change the $(SkipGet) property to true.
2. Change the $(SolutionRoot) property to such as "C:\DevProjects".
3. Change the $(WorkspaceName) property to build machine name, such as "TFSBuild".
4. Override the BeforeGet target to change the behavior of source code download.

Example

```
 <PropertyGroup>
   <SolutionRoot>C:\DevProjects</SolutionRoot>
   <WorkspaceName>VSWS-BB</WorkspaceName>
 </PropertyGroup>

 <PropertyGroup>
   <SkipGet>true</SkipGet>
 </PropertyGroup>
 <Target Name="BeforeGet">
   <Get Workspace="$(WorkspaceName)" Recursive="true" Force="true"
        FileSpec="$/DevProjects/MCS.Japan.SCSystem/*" />
 </Target>
```

Change Compilation Behavior

In order to change the compilation behavior of your build you need to modify the default behavior of CoreCompile.

1. Override the CoreCompile target for the normal compilation tasks.
2. In CoreCompile target, do the following tasks.
    a. Build the solution.
    b. Precompile the ASP.NET Web applications.
    c. Create the instrumentation binary.
    d. Change the web.config to encrypted web.config.
    e. Copy the build outputs to the $(BinariesRoot).

Example

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="DesktopBuild"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

 <ItemGroup>
   <SolutionToBuild Include="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.sln"
/>
 </ItemGroup>

 <ItemGroup>
   <ConfigurationToBuild Include="Release|Mixed Platforms">
     <FlavorToBuild>Release</FlavorToBuild>
```

```
      <PlatformToBuild>Mixed Platforms</PlatformToBuild>
    </ConfigurationToBuild>
    <ConfigurationToBuild Include="Debug|Mixed Platforms">
      <FlavorToBuild>Debug</FlavorToBuild>
      <PlatformToBuild>Mixed Platforms</PlatformToBuild>
    </ConfigurationToBuild>
  </ItemGroup>

  <Target Name="CoreCompile"
        DependsOnTargets="$(CoreCompileDependsOn)"
        Inputs="%(ConfigurationToBuild.PlatformToBuild);%(ConfigurationToBuild.FlavorToBuild)"
        Outputs="%(ConfigurationToBuild.PlatformToBuild);%(ConfigurationToBuild.FlavorToBuild)"
>

    <TeamBuildMessage Tag="Configuration" Condition=" '$(IsDesktopBuild)'!='true' "
                      Value="%(ConfigurationToBuild.FlavorToBuild)" />
    <TeamBuildMessage Tag="Platform" Condition=" '$(IsDesktopBuild)'!='true' "
                      Value="%(ConfigurationToBuild.PlatformToBuild)" />

    <CallTarget Targets="CoreCompileForDebug"
                Condition="'%(ConfigurationToBuild.FlavorToBuild)'=='Debug'" />
    <CallTarget Targets="CoreCompileForRelease"
                Condition="'%(ConfigurationToBuild.FlavorToBuild)'=='Release'" />

    <OnError ExecuteTargets="OnBuildBreak;"/>
  </Target>

  <Target Name="CoreCompileForDebug">
    <MSBuild
          Projects="@(SolutionToBuild)"
          Properties="Configuration=%(ConfigurationToBuild.FlavorToBuild);Platform=%(Configurati
onToBuil
d.PlatformToBuild);SkipInvalidConfigurations=true;TeamBuildConstants=$(TeamBuildConstants);
RunCodeAnalys
is=$(RunCodeAnalysis)"
          Targets="Rebuild" RunEachTargetSeparately="true" ContinueOnError="true" />

    <AspNetCompiler
        VirtualPath="/WebSvc"
        PhysicalPath="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc"
        TargetPath="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc_deplo
y"
        Clean="True" Debug="True" Force="True" />

    <Copy SourceFiles="$(MSBuildProjectDirectory)\Web.debug.config"
          DestinationFiles="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc
_deploy\web.conf
ig" />

    <CreateItem
Include="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc_deploy\bin\Ap
p_*.d
ll">
      <Output TaskParameter="Include" ItemName="DllFilenameForInstrument"/>
```

```xml
    </CreateItem>
    <Exec Command="&quot;\Program Files\Microsoft Visual Studio 8\Team Tools\Performance
Tools\vsinstr.e
xe&quot; -coverage %(DllFilenameForInstrument.Identity)"
          ContinueOnError="true"  />
    <Exec Command="&quot;\Program Files\Microsoft Visual Studio 8\Team Tools\Performance
Tools\vsinstr.e
xe&quot; -coverage
$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc_deploy\bin\MCS.Japan.
SCS
ystem.BizLogic.dll"
          ContinueOnError="true"  />

    <DirectoryCopy
From="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc_deploy"
                  To  ="$(BinariesRoot)\Debug\WebSvc" />
    <DirectoryCopy
From="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WinUI\bin\Debug"
                  To  ="$(BinariesRoot)\Debug\WinUI" />
  </Target>

  <Target Name="CoreCompileForRelease">
    <MSBuild
          Projects="@(SolutionToBuild)"
          Properties="Configuration=%(ConfigurationToBuild.FlavorToBuild);Platform=%(Configurati
onToBuil
d.PlatformToBuild);SkipInvalidConfigurations=true;TeamBuildConstants=$(TeamBuildConstants);
RunCodeAnalys
is=$(RunCodeAnalysis)"
          Targets="Rebuild" RunEachTargetSeparately="true" ContinueOnError="true" />
    <AspNetCompiler
        VirtualPath="/WebSvc"
        PhysicalPath="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc"
        TargetPath="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc_deplo
y"
        Clean="True" Debug="False" Force="True" />
    <Copy SourceFiles="$(MSBuildProjectDirectory)\Web.release.config"
          DestinationFiles="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc
_deploy\web.conf
ig" />
    <DirectoryCopy
From="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WebSvc_deploy"
                  To  ="$(BinariesRoot)\Release\WebSvc" />
    <DirectoryCopy
From="$(SolutionRoot)\MCS.Japan.SCSystem\MCS.Japan.SCSystem.WinUI\bin\Release"
                  To  ="$(BinariesRoot)\Release\WinUI" />
  </Target>

  <Target Name="AfterCompile">
    <Exec Command="cabarc.exe -r -p n MCS.Japan.SCSystem.cab
$(SolutionRoot)\MCS.Japan.SCSystem\*.*" />
    <Copy SourceFiles="$(MSBuildProjectDirectory)\MCS.Japan.SCSystem.cab"
          DestinationFiles="$(BinariesRoot)\Sources\MCS.Japan.SCSystem.cab" />
    <OnError ExecuteTargets="OnBuildBreak;"/>
  </Target>
```

Chapter 8 - Setting up Continuous Integration with Team Build

Objectives

- Understand the purpose of a continuous integration build.
- Setup continuous integration with Team Build.
- Optimize your continuous integration build to reduce bottlenecks.

Overview

This chapter explains how you can setup continuous integration builds with Team Build and Team Foundation Server. Use continuous integration builds to gain rapid feedback on build quality as soon after a check-in as possible. During team development it is important to get rapid feedback on the quality of check-ins, especially if they combine with other developer's changes and break the build. A continuous integration build gives you the chance to fix code quickly, to unblock other team members improving the consistency and quality of your build.

How to Use This Chapter

Read this chapter to learn strategies for continuous integration and about how to setup and configure continuous integration builds with Team Build. For a step-by-step walkthrough to help you setup continuous integration, see "How To: Setup a Continuous Integration Build."

If you are new to Team Foundation Server and Team Build or want to learn more about the options available for automating and scheduling builds, read Chapter 7, "Team Build Fundamentals" first..

Strategy for Continuous Integration Build

Continuous integration (CI) is the process of creating builds whenever a developer checks code into Source Control. Following are various strategies used for continuous integration build.

Build on Each Check-in

Building immediately after every check-in is the simplest continuous integration strategy and will generally give you the most rapid feedback. However, if check-ins occur rapidly enough to overwhelm the build server you should use a rolling build in which you build after a specified number of check-ins or after a specified time period. To determine if you need to use a rolling build determine the following:

- Length of your team build in minutes
- Average frequency of check-ins in minutes
- Time window during which frequent check-ins occur

If the length of the build is longer than the average frequency of check-insthen your builds will start to queue up since one build will not complete before the next check-in occurs. If the build queue is long enough, it will impact the performance of build server and start to delay other builds, such as scheduled builds. Review the time window during which frequent check-ins occur and determine if the queue will have room to unwind after the busiest check-in period is over.
For more information, see How to Produce a Continuous Integration Build.

Rolling Build after Specific Number of Check-ins

If your build server is overloaded by building after each check-in you can choose to only build after a specified number of check-ins have completed. While this is the simplest rolling build solution it has a significant drawback. Since the build server must see a specific number of check-ins before a build is started, the last check-in of the day is virtually guaranteed not to get a build and so build feedback will be delayed.
For more information, see “How to Produce a Rolling Build”.

Rolling Build after Specific Time Interval

Producing a build a specific time interval after each check-in is an improvement over building after a specific number of check-ins. This approach is guaranteed to produce a build within a specific timeframe of each check-in made. The downside is that each build may have a widely varying number of associated check-ins. Some builds may only have one check-in while others may have hundreds. The more check-ins in each build the more difficult it is to determine which check-in created the breaking change.

Rolling Build after Specific Number of Check-ins or Time Interval

Producing a build after a time interval or number of check-ins, whichever occurs first, will result in the most consistency in your rolling builds while still reducing build server load. Use a rolling build if a continuous integration build will create a build queue that is very long and results in builds that occurs a significant time after the check-in. Use a check-in interval to specify the number of check-ins between each build. Use the timeout period to ensure a build occurs even if there are no additional check-ins.

Determining Your Rolling Build Interval

To determine the ideal rolling build interval, divide the average frequency of check-ins by the length of your build. For instance if you have a build that takes 10 minutes and you average a check-in every 5 minutes you could set a check-in interval of 2 check-ins and a timeout period of 10 minutes. This would help ensure that the build is complete before the next build is kicked off. If you notice excessive load on your build server you can increase these values.

Continuous Integration Build in Team Foundation Server

The Team Foundation Server 2005 does not provide a CI solution out of box, but does provide the framework for you to implement your own CI build solution.

For more information about setting up a continuous integration build with Team Foundation Server 2005, see “How To: Setup a Continuous Integration Build with Team Foundation Server 2005”. This “how to” uses the solution provided by the Visual Studio Team System development team. The solution installs a Web service that runs under an account which has access to the TFS server. Team Foundation Server is able to send an email or call a web service when specific events occur. This event mechanism is used by the CI solution to register a Web service with the **CheckinEvent,** so whenever a check in occurs, the Web service initiates a Team Build.

Chapter 9 - Setting up Scheduled Builds with Team Build

Objectives

- Understand the purpose of a scheduled build.
- Setup a scheduled build with Team Build.

Overview

This chapter explains how you can setup scheduled builds with Team Build and Team Foundation Server. The purpose of a scheduled build is to automate the process of creating a reliable build on a consistent schedule. This is the type of build most often used by test teams, internal adopters, external beta users..

Scheduled builds are the simplest form of build automation. You can configure them to run hourly, daily, weekly or at any time interval that works best for your team.

How to Use This Chapter

Read this chapter to learn strategies for scheduled builds and about how to setup and configure scheduled builds with Team Build. For a step-by-step walkthrough to help you setup a scheduled build see, “How To: Setup a Scheduled Build.”

If you are new to Team Foundation Server and Team Build or want to learn more about the options available for automating and scheduling builds, read Chapter 7, “Team Build Fundamentals” first.

If you are concerned about build instability caused by the quality of the code that your development team checks in, you should consider using continuous integration builds. For more information about continuous integration, see Chapter 8, “Setting up Continuous Integration with Team Build.”

Strategy for Scheduled Build

This section discusses the various strategies used when deciding on Scheduled Builds.

Frequency of Scheduled Build

The frequency of the builds is one of the important points to decide when doing a scheduled build.

Hourly Builds

If you are working on a project for which there are enough check-ins that significant changes may occur within an hour and you don’t have continuous integration builds you can use an hourly build frequency. You use the hourly build to provide feedback to developers and you can also make them available to testers and others for their quality feedback.

Nightly Builds

This is the most common scheduled build frequency. This scheduled gives your test and development team a new build every morning incorporating the changes from the previous day, ready to be tested.

Weekly Builds

If you are working on a large project, where your development team had moderate velocity owing to the complexity of the project and the build time runs into days, you should opt for weekly builds. This ensures that your test team, at the start of each week, has a build incorporating the changes from the previous week, ready to be tested.

Scheduled Build in Team Foundation Server

Team Build feature in Team Foundation Server does not support scheduled builds from the user interface. Instead you can use the Windows Task Scheduler to run the TFSBuild command utility to start builds at predetermined time.

Use the following steps to create a scheduled build:

1. Create a TFSBuild command line.
   ```
   TfsBuild start <<TeamFoundationServer>> <<TeamProject>>
   <<BuidlTypeName>>
   ```
2. Place the command line in a batch file.
3. Create a Windows Scheduled Task that runs the batch file at your desired interval

For more information see How To: Setup a Scheduled Build with Team Foundation Server 2005

PART IV

Large Project Considerations

In This Part:

- **Large Project Considerations**

Chapter 10 - Large Project Considerations

Objectives

- Understand large project workflow in Team Foundation Server.
- Learn how to optimize source control and build for large teams.
- Learn how source control is impacted by large projects.
- Learn how your branch and merge strategy may need to change if you are in a large project.
- Learn how build strategy is influenced by large projects.

Overview

This chapter describes additional considerations for large-scale development efforts with Team Foundation Server (TFS). You can distinguish a large project from a smaller project in the following ways. Large teams working on large projects:

- Require a more complex branching and merging structure
- Must deal with a greater number of dependencies across solutions and team projects.
- Are more likely to maintain multiple builds for components and teams.

For example, in a large project you might need to support multiple branches to support the parallel development efforts of multiple feature teams. You are likely to need to manage dependencies across solutions and team projects and to share common Web services and databases. Each sub-team may need to maintain its own build server and output builds to a specific drop point.

How to Use This Chapter

Read this chapter if you need to manage, support or participant in large-scale development project. Chapters 3, 5, and 7 also provide specific “Large Project Considerations” sections. Use this chapter to review all of the large project considerations in a single place.

Logical Workflow for Large Projects

Figure 10.1 outlines a common scenario of multiple sub teams working together to produce a complex application.

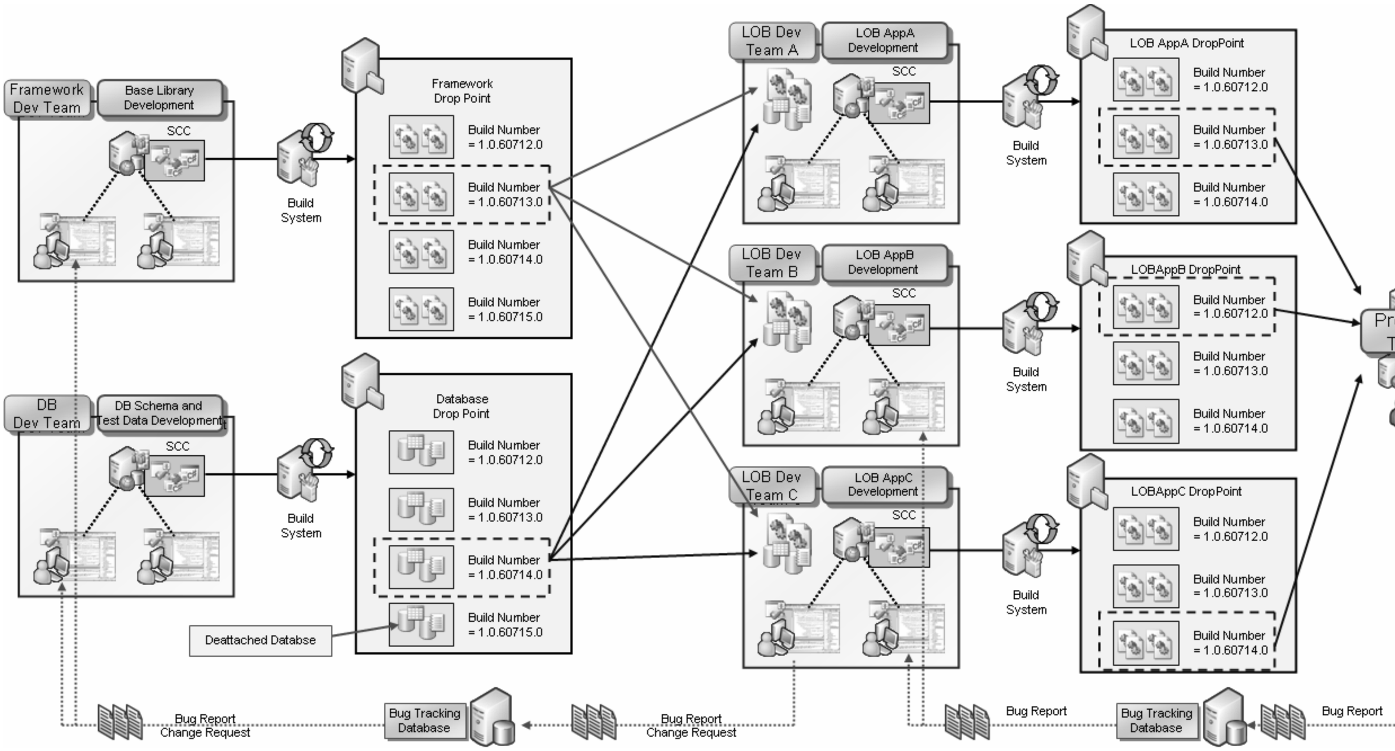

Framework Dev Team
Base Library Development
SCC
Build System
Framework Drop Point
Build Number = 1.0.60712.0
Build Number = 1.0.60713.0
Build Number = 1.0.60714.0
Build Number = 1.0.60715.0
DB Dev Team
DB Schema and Test Data Development
SCC
Build System
Database Drop Point
Build Number = 1.0.60712.0
Build Number = 1.0.60713.0
Build Number = 1.0.60714.0
Build Number = 1.0.60715.0
Deattached Databse
LOB Dev Team A
LOB AppA Development
SCC
Build System
LOB AppA DropPoint
Build Number = 1.0.60712.0
Build Number = 1.0.60713.0
Build Number = 1.0.60714.0
LOB Dev Team B
LOB AppB Development
SCC
Build System
LOBAppB DropPoint
Build Number = 1.0.60712.0
Build Number = 1.0.60713.0
Build Number = 1.0.60714.0
LOB Dev Team C
LOB AppC Development
SCC
Build System
LOBAppC DropPoint
Build Number = 1.0.60712.0
Build Number = 1.0.60713.0
Build Number = 1.0.60714.0
Bug Report Change Request
Bug Tracking Database
Bug Report Change Request
Bug Report
Bug Tracking Database
Bug Report

Figure 10.1 *Large Project Scenario*

Figure 10.1 highlights a number of key points about the way in which a large team project operates:

- Each sub team maintains its own build server and produces builds to a drop point.
- Application team's pick up builds from the component teams and reuse them as part of their solutions and include them in their builds.
- Component and integration testing occurs on each build, bugs are filed in the appropriate bug database.

Source Control Considerations

If you work on a very large solution requiring many dozens of projects you may hit Visual Studio solution (.sln) scalability limits. For this reason, you should organize your source control structure by sub systems or sub applications and use multiple solution files. You divide your application into multiple solutions but do not create a master solution for the entire application. Figure 10.2 shows the multiple solutions approach used by large teams.

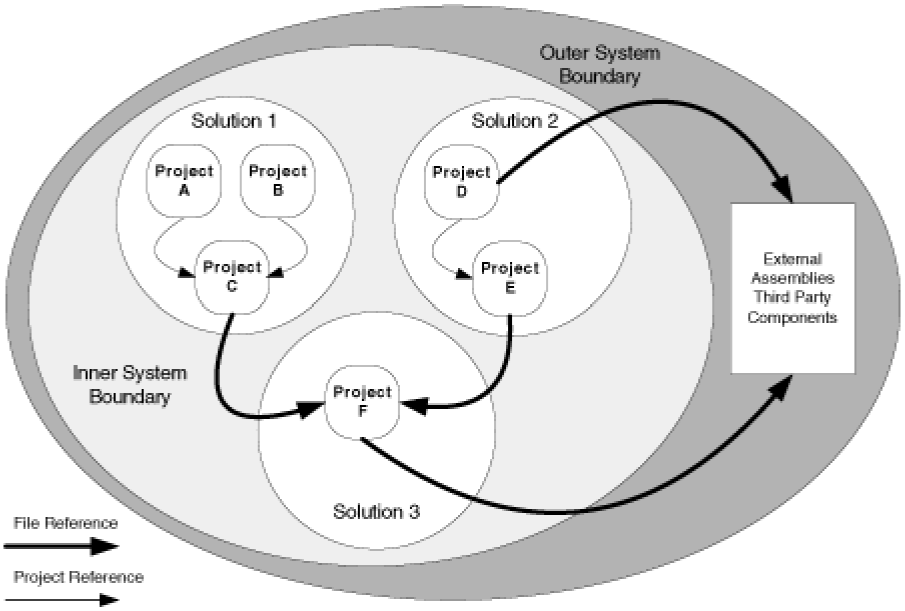


Figure 10.2 *Multiple Solution Approach*

In this scenario, all references inside each solution are project references. References to projects outside of each solution (for example to third party libraries or projects in another sub-solution) are file references. This means that there can be no “master” solution. Instead, you must use a script that understands the order in which the solutions must be built. One of the maintenance tasks associated with a multiple-solution structure is ensuring that developers do not inadvertently create circular references between solutions. This structure requires complex build scripts and explicit mapping of dependency relationships. In this structure it is not possible to build the application in its entirety within Visual Studio. Instead, you use TFS Team Build.

For more information about this multiple solution approach see Chapter 3, “Structuring Projects and Solutions in Source Control.”

Branching and Merging Considerations

Large teams are likely to branch both by team and by feature and are also more likely to have one or more branches designed to integrate external dependencies. Since the branching structure is deeper there is more lag between a change in a development branch and getting that change into the main integration branch. For this reason it pays to think through your merging strategy when creating branches.

For example, will your merges be scheduled or event driven?

- Reverse integration is generally scheduled, for example from a development branch to the main integration branch.
- Forward integration, such as from main integration branch to a development branch, is generally based on an event (such as feature milestone complete). This event is instigated when the team behind the development branch feels they are ready to merge changes from their parent branch.

Rationalize your branch/merge strategy with the frequency with which you want to produce builds. A deeper branching structure results in more time to merge from child branches to the main integration branch. Symptoms of this causing a problem for your development team include:

- Broken builds where the fix takes too long to propagate up to the main integration branch.
- Missed milestones because features take too long to propagate up to the main branch.
- Your team is spending large amounts of time merging changes between various branches.

If this becomes a problem for your team consider reducing the depth of your branching structure.

The following is an example of a branching structure for a large team:

- **My Project**
    - **Development** – Container to isolate active development branches
        - **Team 1**
            - **Feature A** – Isolated branch for development
                - **Source**
            - **Feature B** – Isolated branch for development
                - **Source**
        - **Team 2**
            - **Feature A** – Isolated branch for development
                - **Source**
            - **Feature B** – Isolated branch for development
                - **Source**
    - **Main** – Main integration and build branch. All changes come together here.
        - **Source**
        - **Other Asset Folders**
    - **Release 4 –** Branch containing the code currently being locked down to release
    - **Maintenance** – Container for maintenance branches
        - **Release 2**– Active maintenance branch
            - **Source**
        - **Release 3** – Active maintenance branch
            - **Source**
    - **Safe Keeping**
        - **Release 1** – Old release in safe keeping
            - **Source**

This structure includes:

- Feature branches for multiple teams to work in isolation.

- A main branch to gather changes by all teams as well as bug fixes from the release branches
- A release branch used to lockdown the current release
- A maintenance branch for an old release that is being actively maintained and supported
- Safekeeping branches for older releases that are no longer being maintained

Build Considerations

Larger teams are likely to have longer build times. The frequency of Continuous Integration (CI) builds should be less than the build time to avoid too much queuing and loading of the build server. A build system for a large team is typically organized as follows:

- If the team uses multiple team or feature branches, a build server is dedicated to each branch. This enables the team to focus each build on the purpose of the branch, for example integration or development.
- Build frequency is determined based on the needs of team. The build infrastructure is designed and built to support these needs.
- Build frequency is chosen first, and based on that goal, the merge frequency is determined.
- Scheduled builds are generally used for integration branches.
- A combination of continuous integration and scheduled builds are used for development branches. Integration builds are distributed to the integration test team. The output from the development branch scheduled build goes to the associated test team and the development branch continuous integration build provides feedback specific to the particular development team.

Another key question is whether you should have build, testing and quality gates at each integration point? Alternatively, you could save that for your main integration branch. Ideally you have a build server for each branch and you have quality gates along with a supporting development and test team. You should be pragmatic though, and if that does not exist, you may want to remove quality gates or remove branches to simplify your structure.

PART V

Project Management

In This Part:

- ▶ **Project Management Explained**
- ▶ **Work Items Explained**
- ▶ **MSF Agile Projects**

Chapter 11 - Project Management Explained

Objectives

- Learn what functionality Visual Studio Team System provides for project managers.
- Choose a strategy for team project creation.
- Create and manage projects with Visual Studio Team System

Overview

This chapter introduces you to the project management features provided by Visual Studio Team System and how they help you to overcome many of the common problems and challenges associated with managing software development projects.

Visual Studio Team System and Team Foundation Server provide tools, templates and reports to help you monitor and support your software development processes. Team communication is made easier, handoffs required between team members are automated, work items such as tasks are more easily assigned and tracked, and metrics that indicate project health and progress are more easily monitored.

How to Use This Chapter

Use this chapter to learn about the specific features of TFS and VSTS relevant to project managers.

- Read Project Management Summary and Traditional Project Management Issues sections to understand the problems that TFS project management is attempting to solve.
- Use the Strategies for Team Projects section to identify your strategy for Team Project creation and organization.
- Use the remaining sections to better understand process templates, work items and other project management features.

Project Management Summary

Planning a project generally consists of some variation of the following steps:

1. **Generating vision statement**. This involves creating a shared view of the end in mind in terms of outcomes.
2. **Generating scenarios.** This includes figuring out the initial set of usage scenarios for the software. This involves using customers. It also involves validating the scenarios to make sure they represent customer value.
3. **Generating set of features to support those scenarios**. This includes breaking the scenarios down into chunks of customer value that you can talk to customer about.

4. **Generating a set of work items**. This involves breaking the scenarios and features down into specific tasks. Put another way, when the work items are completed, the relevant feature or scenario is implemented
5. **Breaking tasks into areas**. This involves breaking tasks out into areas. These areas could be functional or based on how teams are organized.
6. **Scheduling the work.** This could involve scheduling all the work up front, or breaking the work into iterations.

Traditional Project Management Issues

Most project managers today use a variety of different tools to manage the software development process and these tools usually have little if any integration with the tools used by software developers to do their job. This lack of integration and cohesion across tools and teams leads to significant challenges for the project manager. Typical issues include:

- **Dealing with disparate silos of information**. Project management tools are usually used in isolation, leading to silos of information that are not easily integrated. It's also usually difficult to integrate the data maintained by project management tools and the data maintained by other team members such as bugs tracked within a separate bug tracking system in order to generate meaningful metrics.
- **Difficultly capturing project-related metrics**. Obtaining critical metrics about a project is needed to track status, to make decisions and to answer fundamental questions such as "will the project be delivered on time and to budget?" To answer these key questions, project managers typically rely on data obtained from Microsoft Project or the bug tracking system used by the development and test teams. Consolidating data from these disparate systems is difficult, time consuming and error prone. Most of the metrics generated by tools are not stored or accessed in a uniform manner. Creating reports is usually a manual and intensive effort that requires much copy and pasting of information from different tools.
- **Difficultly ensuring that requirements are met**. There is often a gap between the work planed for the development team and customer requirements and key non-functional requirements identified for your system. There is another gap between scheduled work and actual work. Vital information gets lost in these gaps which results in requirements not being met.
- **Managing processes and process change**. Which policies should be applied? Which processes should be followed by your teams? Communicating processes that your team should be following is a challenging task. Implementing changes to address systematic problems, without impacting team productivity is more difficult still.
- **Lack of auditable communication and task tracking**. Collaboration and team cohesion is normally addressed by holding team meetings, and allocating task lists for developers to help them focus on the right priorities. Keeping track of the progress of individual tasks is difficult. Also, project managers often spend a lot of time gathering status from different plans and lists. Team members also spend time completing status reports and updating a variety of different documents and forms.

- **Quality assurance**. Predicting the number and severity of bugs in the software you are producing is difficult and as a result schedule estimates and costs are usually “best guesses”. These are traditionally supplemented with contingency, the amount of which usually depends on how confident the project manager is about the current health of the project.

Microsoft Visual Studio Team System is designed to help address many of these traditional problems faced by project managers. It does so by providing a set of integrated tools to help teams improve their software development activities and to help project managers better support the software development processes.

Project Management Features in Team Foundation Server

The key project management features provided by Visual Studio Team System include:

- **Centralized work item management**. Work items including bugs, risks, tasks, scenarios and quality of service requirements are centrally recorded, managed and maintained in the Team Foundation Server work item database. Centralizing their storage makes it easy for all team members to view and access them.
- **Office Excel and Office Project integration**. By using the Office Excel and Office Project integration features, project managers can continue to access the work item repository and schedule information by using the tools they already know.
- **Metrics and reporting**. Team System provides a reporting service which transforms operational data such as work items, build results, and test results into metrics stored within Team System’s data warehouse. Predefined reports allow you to query a variety of project health and quality metrics.
- **Project portals**. For every team project, Team System creates an associated project portal that uses Microsoft Windows SharePoint Services. You use the portal to manage project related documentation, and to quickly view key reports and assess the current status of the project.

Benefits

The project management features of VSTS result in the following benefits:

- Centralized management
- High traceability
- Integrated project planning and scheduling
- Improved process control
- Improved team communication and cohesiveness
- Accurate progress reporting

Creating and Managing Projects with Team Foundation Server

This is a summary of the general approach for creating team projects in Team Foundation Server:

1. **Choose a process template**. You can either use a default template out of the box or you can customize your own.
2. **Create a team project**. Your team project will be based on your process template.
3. **Add appropriate groups and members to your team project**.
4. **Decide on your iteration cycle duration for the project**. This includes creating iterations in your team project.
5. **Capture your scenarios as work items in Team Foundation Server**.
6. **Determine which scenarios need to be completed for each iteration**.
7. **Define the quality of service requirements**. Link the quality of service requirements to your scenarios.
8. **Break your scenarios down into stories to provide manageable work items for developers**. Obtain estimates from the developers for each work item.
9. **Create a project schedule.** Create a project schedule (by using Microsoft Project) and adds it to the Team Project.
10. **Define acceptance criteria**. Define acceptance criteria for the work items (correlated to the QoS requirements).
11. **Define reporting requirements**. Define your projects key performance indicators and identify reporting requirements.

For more information on creating a Team Project and Managing it please see, “How To - Manage Projects in Team Foundation Server”

Strategies for Team Projects

You need at least one Team Project to start working with Team Foundation Server. When a project manager creates a new Team Project a team project Web site is created containing document libraries with document templates, and stock reports. A work item database is created for tracking all effort on the project and a methodology template is installed that determines rules, policies, security groups, and queries for all work effort. Additionally a source code branch is created for source control.

The structure of your Team Project should be guided by your requirements and might change as your software development process evolves. There are three common strategies for creating new team projects. You might use one of these strategies or a combination of several. The three common strategies are:

- **Team Project per application**
- **Team Project per release**
- **Team Project per team**

Team Project per Application

This is the most common strategy for creating team projects. This approach is useful for both large and small applications, as well as multiple releases of applications being developed in parallel. With this approach, you create one project for each application under development.

Team Project per Release

This approach is useful for large teams who are working on long running projects. After every major release, you create a new project and have a clean start. With this approach you don't have to worry about carrying the pervious release baggage including work items. Also, this approach provides you with the opportunity to improve the process templates or use new ones based on your newly acquired experience and learning.

Team Project per Team

This approach is useful for large projects that span multiple teams, where central control and activity monitoring is important. With this approach, you create a project for each team. This approach closely aligns a team with the workflows defined in TFS work item types and provides a unit of reporting that spans the entire team.

Key Project Management Features in Team Foundation Server

Team Foundation Server consists of the following components:

- **Process Management**. Team Foundation Server process management includes MSF process guidance as well as process templates that set up new team projects with work item types, reports, a project SharePoint portal, and source control settings.
- **Security and Permissions**. New projects contain default groups and permissions that map to common development team roles.
- **Work Item Management**. Each process template includes a set of work item types customized for that process. For instance MSF CMMI contains many more work item types than MSF Agile since it is a more formal process with more roles and activities.
- **Progress and Reporting**. Each process template includes a set of default reports that you can use to track project progress and health.

Process Management

With Visual Studio Team System, the software development lifecycle and its associated processes has have been made an integral part of software project development efforts. By integrating the software process the lifecycle processes into Visual Studio Team System, the process and handoffs are methodology for development controls team interaction and activities automated between team members.

Process Templates

Visual Studio Team System uses process templates to define the set of instructions and artifacts for setting up a new project. A process template is a self-contained set of instructions that provide development teams with the methodology for a software development imitative. A process template contains the following elements:

- **Process guidance.** This is supplied for each template, provides contextual information, help and direction for team members who need assistance following or understanding a particular activity. Process guidance is integrated into the Visual Studio Help System.
- **Document templates.** These enable team members to create new documents such as specifications, risks and project plans in a consistent manner.
- **Work items and workflow.** Work items have their own set of fields and rules which determine the workflow of the work item and how team members can assign and perform work.
- **Security groups.** These are used to define who can control and manipulate reports, work products such as source code and documentation, and work items. Project managers can administer security groups without needing to be Windows administrators.
- **Check-in policies.** These can be used to enforce rules and quality gates for all code checked into source control. For example, you can enforce that checked-in code meets specific criteria, such as conforming to your corporate coding standards, or passing unit tests. For more information about how to create and configure custom check-in policies, see “How To: Create a Custom Check-in Policy.”
- **Reports.** These are used to monitor the ongoing performance and status of the project. VSTS ships with many predefined reports including code quality reports, schedule progress reports, test effectiveness reports and many others. You can create your own report and customize existing reports.

MSF and CMMI Process Templates

There are two process templates available out-of-the box.

- **MSF for Agile Software Development**. This is a lightweight process for small, agile or informal software projects. It is based on scenarios and context driven actions and is project and team member (people) centric.
- **MSF for CMMI® Process Improvement**. This is designed for more mature software projects. It extends MSF for Agile Software Development by providing support for auditing, verification and formal processes. It relies on process and conformance to process and is organization centric.

If the provided templates do not match your specific process requirements and methodology closely enough, you can add new process templates to the system, and you can customize the supplied templates to fit the needs of your organization. For more information about customizing existing templates, see “How To: Customize a Process Template.”

Security and Permissions

When you create a project in Team Foundation Server, four default groups are created for that project regardless of your choice of process template. By default, each of these groups has a set of permissions defined for them that govern what members of those groups are authorized to do.

- Project Administrator
- Contributor
- Reader
- Build Services.

You can create security groups for your team project to better meet the security requirements of your organization. Creating a security group is an efficient way to grant a specific set of permissions to a group of users on your team project. Make sure that you allow only the minimum permissions necessary for the group, and add only those users or groups who must belong to this new team project group

After you have created a team project group, you must add the new group, give the group the appropriate permissions, and add members to the group. By default, a team project group is created without any permission granted.

Work Item Management

Work Items are used as units of work for communication and collaboration between the team. Your selected process template provides an initial set of work item types and project managers then create and manage additional work items that need to be completed on a development project. A work item might define a task, risk, scenario, bug, or quality of service requirement. You can link work items together to provide better traceability. For example, you could associate a specific work item task with the related scenario or quality of service requirement to which that work items relates.
The process template provides the definitions for the work item types including the set of fields defined for each work item type. Hence selection of the process template is important, because it can't be modified during the project. If required you should customize the process template to include any additional work item types other then the once provided by the base templates.

There are number of work-items predefined in both the out of box process templates.

MSF for Agile Software Development

The work item types provided by this process template include:

- **Scenario –** Used to represents a user interaction with the application system. It records the specific steps that need to reach a goal. When writing scenarios, be specific as there may be many possible paths.

- **Task** – Used to represents a unit of work that needs to be performed. Each role has its own requirements for a task. For example, a developer uses development tasks to assign work.
- **Quality of Service Requirement** – Used to document the characteristics of the system such as performance, load, availability, stress, accessibility and serviceability
- **Bug** – Used to communicate a potential problem in the system.
- **Risk** – Used to identify and manage the inherent risks of a project.

MSF for CMMI® Process Improvement

The work item types provided by this process template include:

- **Requirement** – Used to capture the requirements captured during the requirements gathering phase.
- **Change Request** – Used to capture any change requests subsequent to requirements gathering.
- **Issue** – Used to capture issues to be tracked in the projects.
- **Task** – Used to represents a unit of work that needs to be performed. Each role has its own requirements for a task. For example, a developer uses development tasks to assign work.
- **Review** – Used to represent the review work unit with in the projects, like code review, design review etc.
- **Bug** – Used to communicate a potential problem in the system.
- **Risk** – Used to identify and manage the inherent risks of a project.

Microsoft Project Integration

Installing Visual Studio Team System or the Team Explorer application provides extensions for Microsoft Project. For large projects where you have large number of resources, you can use Microsoft Project, to manipulate schedule data within Team Foundation Server.

For example, you can use Microsoft Project to manage and plan, assign, balance and track work, and then publish the updates back to the work item database when they are ready to be used by other team members.

For more information see “Working with Work Items in Microsoft Project” at http://msdn2.microsoft.com/en-us/library/ms244368(VS.80).aspx

Microsoft Excel Integration

Installing Visual Studio Team System or the Team Explorer application provides extensions for Microsoft Excel. For large projects where there huge number of work items, you can use the Excel integration feature to create work items in the excel sheet and upload to the work item database for to be used by other team members.

For more information see “Working with Work Items in Microsoft Project” at http://msdn2.microsoft.com/en-us/library/ms244368(VS.80).aspx

Progress and Reporting

The reports provided with Team Foundation Server help you quickly assess the status of the team project, the quality of the software under development, and the progress toward project completion. These reports are generated from data maintained within the Team Foundation data warehouse and they summarize metrics derived from work items, source control, test results, and builds.

For example, you can use reports to tell you how fast your team is working from week-to-week, based on their actual activities. The goals of the TFS reporting system consist of providing integrated data from across team system components to enable project managers and team members to understand the state of the software development project and take action to ensure its success.

The process template you use to create your team project determines which reports are available by default, but you can also add your own custom reports. The content and use of each report created by the process template is explained in the process guidance for that template. Team Foundation Server is built on SQL Server 2005 and uses SQL Server to store everything about work items, quality attributes, testing, test results, and build results. Team Foundation Server then uses SQL Server Analysis Services to aggregate and analyze the data and drive the reports. The reports that are created by the process template or by individual team members by using Microsoft Excel or Visual Studio 2005 Report Designer are made available through SQL Server 2005 Reporting Services and the team SharePoint portal site.

For more information about customizing reports, see “How To: Create a custom TFS Report.”

Additional Resources

- For more information see “Visual Studio 2005 Team System: Software Project Management” at http://msdn2.microsoft.com/en-us/library/aa302181.aspx

Chapter 12 - Work Items Explained

Objectives

- Learn about the purpose and structure of work items.
- Describe work item workflow.
- Customize work items to meet your team's need.

Overview

This chapter introduces you to work items and explains how you can use them to help manage your software development projects. Each work item represents a unit of work for your development team. A set of work item types are defined in the process template you choose when creating a new team project.

Once the project has started you can create any of the available work item types to track work. While default work item types and behaviour are defined in the process templates, any aspect of a work item type can be modified to better fit the way your team works.

How to Use This Chapter

- Read the "Work Item Structure" section to learn about predefined work item types and how work item workflow is defined.
- Read the "Customizing Work Items" section to learn how and why you would customize a work item type.

Scenarios and Solutions

Work items are the primary way that project managers and team leaders track the work that remains to be done as well as the completed work. Team members use work items to track their personal work queue and assign work to each other, for example in the form of bugs or tasks.

The following are common uses of work items in team projects:

- Create user requirements or quality of service requirements for an application.
- Track development and testing against requirements.
- Create development tasks to represent the work necessary in order to implement application components and features.
- Create bugs to represent defects in the implementation.
- Triage bugs and tasks so that they are appropriately prioritized and balanced across the team.
- Track development tasks to determine progress toward code complete.
- Track bugs, along with other quality metrics, to determine quality of the application and ship-readiness.

How you use work items depends upon the work item types that are defined for your project. The work item definitions are stored in the process template you choose when the team project is first created. You can choose from the default templates, MSF for Agile Software Development or MSF for CMMI Process Improvement, or you can customize work items to meet your team's specific needs and process.

Work Item Structure

Each work item type can be defined as follows:

- It has a purpose and intended use. For instance, bugs are used to track quality defects, tasks are used to track scheduled work, quality of service requirements are used to capture critical non-functional aspects such as security and performance requirements.
- It has a workflow defined by states and transitions. For example, the steps from opened to resolved to closed.
- A set of fields that can be set, queried, and reported on. For example, priority, status and, iteration.

Work Item Types

MSF Agile and MSF CMMI each define a set of work items that map to the roles and activities defined in the process guidance.

MSF Agile

MSF Agile contains the following work item types:

- **Bug**. Represents a problem or potential problem in your application.
- **Risk**. Represents a possible event or condition that would have a negative impact on your project.
- **Scenario**. Represents a single path of user interaction through the system.
- **Task**. Represents the need for a member of the team to do some work.
- **Quality of Service Requirement**. Represents a requirement constraining how the system should work.

MSF CMMI

MSF CMMI contains the following work item types:

- **Bug**. Represents a problem or potential problem in your application.
- **Change Request**. Represents a proposed change to your application.
- **Issue**. Represents a situation that may block work or is currently blocking work.
- **Requirement**. Represents a description of what your application should do to solve a customer problem.
- **Review**. Represents the results of a code, design, or deployment review.
- **Risk**. Represents a possible event or condition that would have a negative impact on the project.
- **Task**. Represents the need for a member of the team to do some work.

Work Item Workflow

Each work item has a pre-defined workflow representing each state the work item can be in as well as transitions between states. Each state is naturally associated with a role in TFS. For example when a tester opens a new bug in MSF Agile the state is **Active**. When a developer fixes the bug the state is changed to **Resolved**. When the tester verifies the fix the bug state is changed to **Closed**.

Workflow Examples

The following examples show workflow for two common work item types.

MSF CMMI Task

An MSF CMMI task has the following possible states:

- **Proposed.** For example, proposed by a developer, tester, or architect.
- **Active.** For example, accepted by a lead or manager.
- **Resolved.** For example, resolved by a developer.
- **Closed.** For example, tested and closed by a tester.

Figure 12.1 shows each state along with the possible transitions between the states.

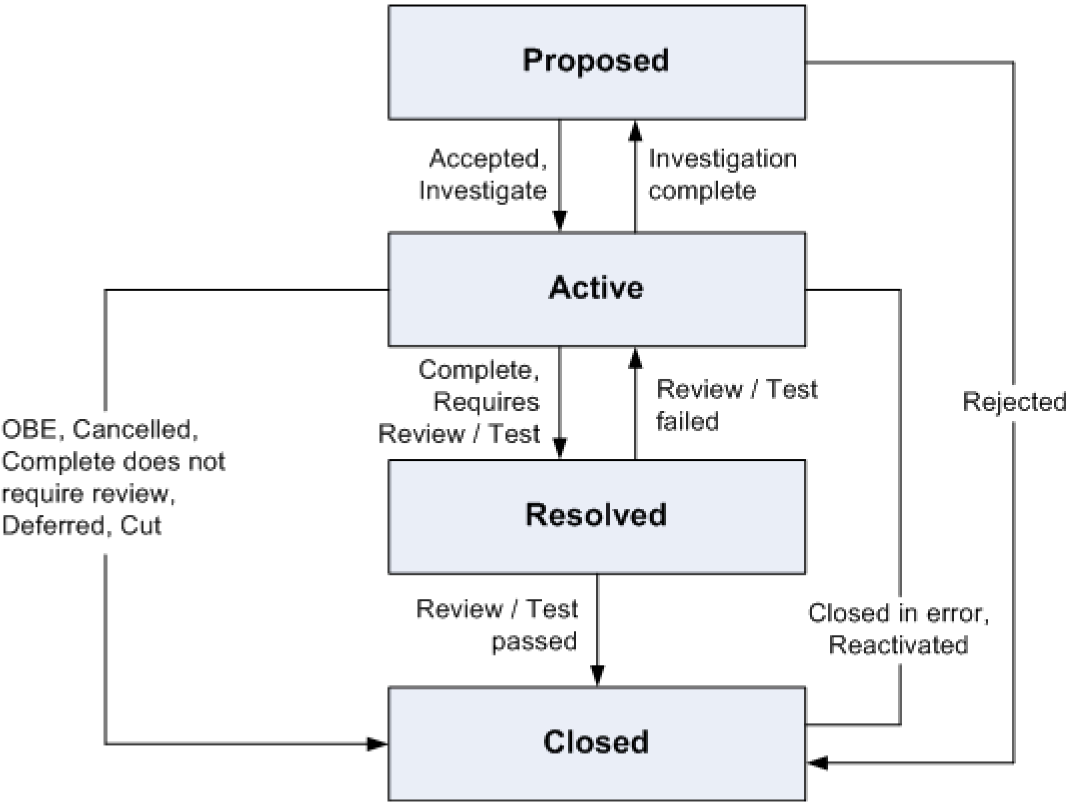

Proposed
Accepted, Investigate
Investigation complete
Active
Complete, Requires Review / Test
Review / Test failed
OBE, Cancelled, Complete does not require review, Deferred, Cut
Rejected
Resolved
Review / Test passed
Closed in error, Reactivated
Closed

Figure 12.1 *Transition States for MSF CMMI*

MSF Agile Bug

An MSF Agile Bug has the following possible states:

- **Active**. For example, opened by a tester.
- **Resolved**. For example, resolved by a developer.
- **Closed**. For example, tested and closed by a tester.

Figure 12.2 shows each state along with the possible transitions between the states.

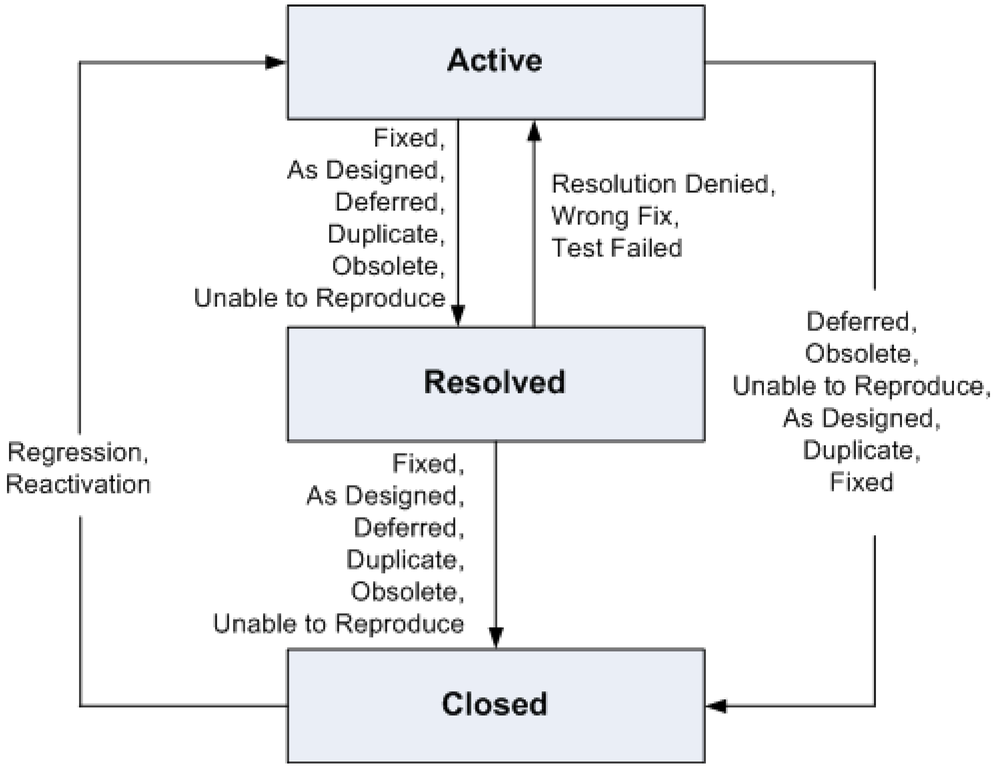


Figure 12.2 *Transition States for MSF Agile*

Customizing Work Items

There are a few scenarios in which you may want to modify the work item types defined in MSF Agile or MSF CMMI:

- A work item is missing a field that is important for your development process.
- A work item workflow does not match how your team works.
- You need a new work item type.

To support these scenarios, you can do the following in Team Foundation Server:

- Add/remove work item types.
- Modify fields in existing work items.
- Modify states and transitions in existing work items.

For more information about customizing work items, see “How To: Customize the Process Template by using the Process Editor.”

Chapter 13 - MSF for Agile Software Development Projects

Objectives

- Learn when to use the MSF for Agile Software Development (MSF Agile) process template.
- Understand how teams typically use the MSF Agile process template.
- Customize the MSF Agile process template to meet your team's needs.

Overview

The process defined by the MSF for Agile Software Development process template incorporates key ideas from the Agile software movement, along with principles and practices from the Microsoft Solution Framework (MSF). It is an agile software engineering process that uses multiple iterations and a scenario-based approach for building applications. The template provides the automation and guidance necessary to support your team development, including configuration management, project management, work item tracking, and a project portal for communication.

This chapter explains the workflow of a typical MSF Agile software development project, shows examples of teams using the MSF for Agile Software Development process, and describes the default template settings and what options you have to customize the supplied template.

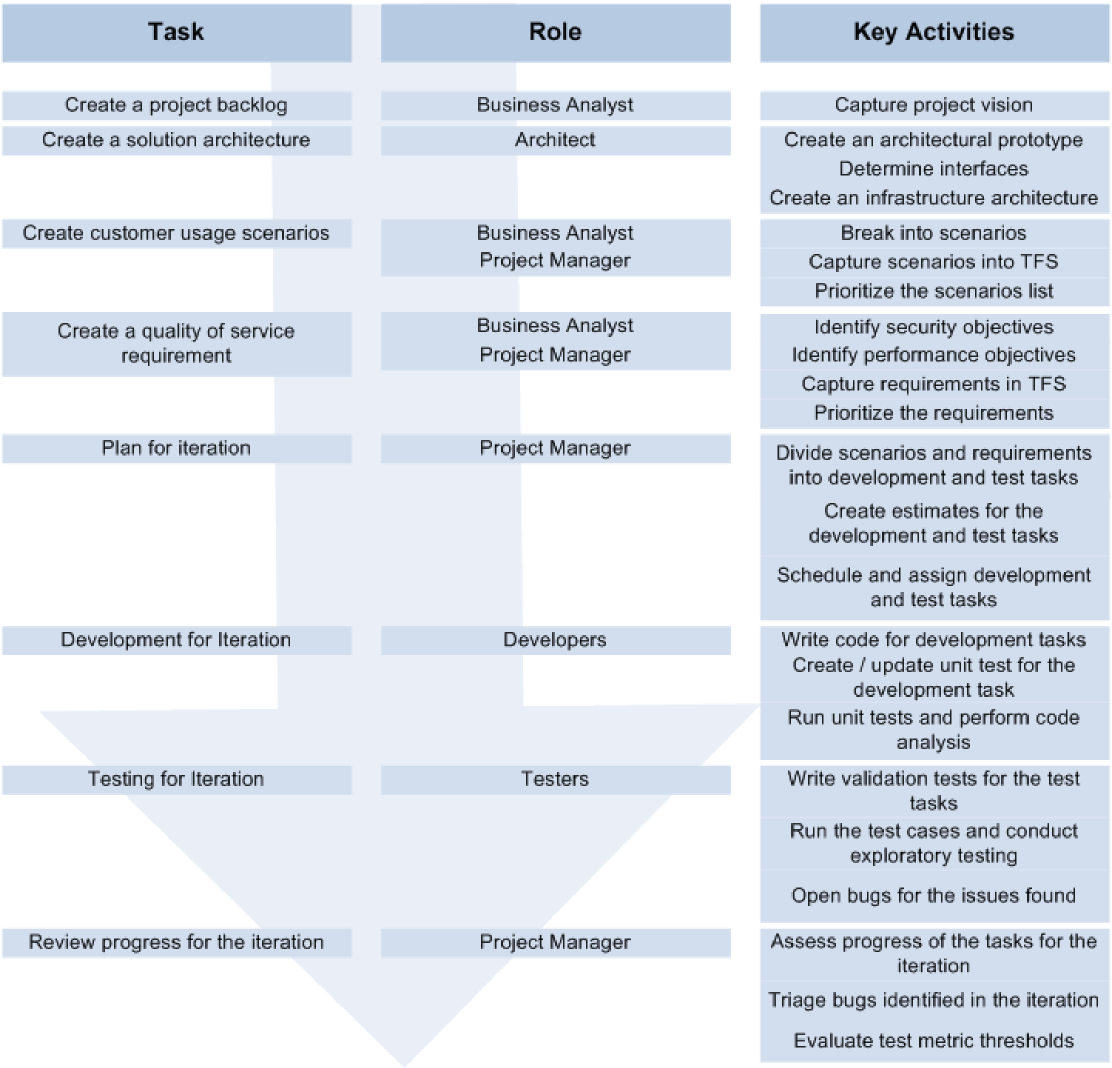
How to Use This Chapter

Read this chapter if you want to gain a better understanding of how the MSF Agile process template and process guidance works as well as how it is has been used successfully by various teams.

- **Use "MSF for Agile Software Development Defaults" section**. Read this to understand the details of the MSF Agile process template including default reports, work items, and permissions.
- **Use the example sections.** Read these to see how real teams have successfully used MSF Agile to develop and release applications.
- **Read "Chapter 14, Process Template Fundamentals."** If you would like to learn more about process templates in general, read "Chapter 14, Process Template Fundamentals."

Work Flow for MSF for Agile Software Development

The MSF for Agile Software Development process template defines a set of tasks to be performed during iterations by the various roles involved in a software development lifecycle including business analysts, architects, project managers, developers and testers. Figure 13.1 shows the key activities associated with each task.



Task	Role	Key Activities
Create a project backlog	Business Analyst	Capture project vision
Create a solution architecture	Architect	Create an architectural prototype Determine interfaces Create an infrastructure architecture
Create customer usage scenarios	Business Analyst Project Manager	Break into scenarios Capture scenarios into TFS Prioritize the scenarios list
Create a quality of service requirement	Business Analyst Project Manager	Identify security objectives Identify performance objectives Capture requirements in TFS Prioritize the requirements
Plan for iteration	Project Manager	Divide scenarios and requirements into development and test tasks Create estimates for the development and test tasks Schedule and assign development and test tasks
Development for Iteration	Developers	Write code for development tasks Create / update unit test for the development task Run unit tests and perform code analysis
Testing for Iteration	Testers	Write validation tests for the test tasks Run the test cases and conduct exploratory testing Open bugs for the issues found
Review progress for the iteration	Project Manager	Assess progress of the tasks for the iteration Triage bugs identified in the iteration Evaluate test metric thresholds

***Figure 13.1** MSF Agile Tasks, Roles and Key Activities*

MSF for Agile Software Development Defaults

When you create a new team project by using the MSF for Agile Software Development process template, a concepts page outlining the process guidance is displayed in the main window of Visual Studio. This is your initial viewpoint into the MSF for Agile Software Development process. You can also access this from the project portal home page.

The configuration of the tooling goes well beyond the description of the process and includes work items (such as scenarios, quality of service requirements, tasks, bugs, and risks), project reports, roles (groups and permissions) and a project portal. Key items supplied by the MSF Agile template include:

- Work items
- Groups and permissions
- Source control
- Areas and iterations
- Reports
- Portal

The following sections detail the important defaults available to you when using the MSF for Agile Software Development process template.

Work Items

The MSF for Agile Software Development process template contains the following work item types:

- **Bug**. Represents a problem or potential problem in your application.
- **Risk**. Represents a possible event or condition that would have a negative impact on your project.
- **Scenario**. Represents a single path of user interaction through the system you are building.
- **Task**. Identifies a specific item of work for a team member to perform.
- **Quality of Service Requirement**. Represents a non functional requirement such as a security, performance or manageability requirement.

When you create a new team project based on the MSF for Agile Software Development template, the following work items are created for you. This saves you work by providing you with a common set of tasks that you need to perform at project initiation.

- **Set up: Set Permissions**. Task to add team members to one of the four security groups: Build Services, Project Administrators, Contributors, or Readers.
- **Set up: Migration of Source Code.** Task to migrate your existing source code from Visual SourceSafe, if you moving an existing project to Team Foundation Server. You should complete migration of source code before you grant team members access to the team project.
- **Set up: Migration of Work Items.** If you are bringing an existing project into TFS you can migrate work items such as bugs and tasks from Clearquest or a CSV file. You should complete this migration of work items before you grant team members access to the team project.
- **Set up: Set Check-in Policies**. Task to setup the business rules or policies that surround source code check-ins.
- **Set up: Configure Build**. Task to create an initial source tree and set up the build to run on a periodic, usually daily, basis.
- **Set up: Send Mail to Users for Installation and Getting started**. Task to send an e-mail to team members that provides information about which Team Foundation Server they should connect to, and which team project they should use so they can get started working on the team project.
- **Create Vision Statement.** Task to create a vision statement for the project.
- **Set up: Create Project Description on Team Project Portal**. Task to change the default project description to better describe the new team project; such as describing project purpose, goals, or vision.
- **Create Personas**. Task to create personas interacting with the system.
- **Define Iteration Length.** Task to define the iteration cycle to be used by the project. This depends upon the size of the project and complexity.
- **Create Test Approach Worksheet including Test Thresholds**. Upfront test considerations.
- **Brainstorm and Prioritize Scenarios List**. Identify and prioritize key usage scenarios.

- **Brainstorm and Prioritize Quality of Service Requirements List.** Identify non functional QoS requirements such as security, performance and manageability scenarios.
- **Set up: Create Project Structure.** Task to create the project structure that captures what areas the development team will be working in.
- Create Iteration Plan. Decide how to divide your development efforts up into iterations.

Reports

The following reports are available by default with the MSF for Agile Software Development process template.

- **Bugs by Priority**. Are the correct bugs being found? This report shows you the rate of high priority bugs found versus low priority.
- **Bug Rates.** How effectively are bugs being found, fixed and closed? This chart shows trends over time for new bugs, bug backlogs, and bug resolution.
- **Builds**. What is the quality of a build? Provides a list of available builds including build quality and other detailed information.
- **Project Velocity.** How quickly is the team completing work? This report shows how quickly the team is completing planned work and show rates of change from day to day.
- **Quality Indicators.** What is the quality of the software? This report collects test results, bugs code coverage and code churn into a single report to track project health.
- **Load Test Summary**. This report shows test results for load testing on your application.
- **Regressions**. This report shows a list of all the tests that passed previously and now are failing.
- **Reactivations.** How many work items are being reactivated? This report shows work items that have been resolved or closed prematurely.
- **Related Work Items.** What work items are dependent on other work items? This report shows a list of work items that are linked to other work items so you can trace dependencies.
- **Remaining Work.** How much work is left to be done and when will it be completed? This report shows work remaining, resolved and closed over time. Projecting remaining work trends forward can enable you to predict the point at which you will be code complete.
- **Unplanned Work**. How much unplanned work is there? This report charts total work versus remaining work and distinguishes planned from unplanned.
- **Triage.** What work items need to be triaged? This report shows every work item that is still in the proposed state.
- **Work Items**. What are the active work items? This report lists every active work item.
- **Work Items by Owner.** How much work is assigned to each member of the team? This report shows work items per team member.
- **Work Items by State**. How many active, resolved and closed work items are there?

Groups and Permissions

The following groups are available by default in the MSF for Agile Software Development process template.

- **Readers**. Members of this group have read access to the team project.
- **Contributors**. Members of this group can add, modify, and delete items within the team project.
- **Build Services**. Members of this group have build service permissions for the team project. This group is used for service accounts only.
- **Project Administrators**. Members of this group can perform all operations in the team project.

Source Control

MSF Agile uses the following default source control settings:

- **Multiple Checkout.** By default MSF Agile allows multiple checkouts to enable multiple team members to work on the same file at the same time. Any resulting conflicts must be resolved at check-in time.
- **Permissions.** The default permissions on the source control is as follows:
  - **Project Administrators.** Has all available rights.
  - **Build Services**. Has rights to read, pend changes, check-in, label, start build, and edit build.
  - **Contributors.** Has rights to read, pend changes, check-in, check-out, label, and start build.
  - **Readers**. Has only read access to source control

Areas and Iterations

The out-of-box process template does not provide a classification structure for either areas or iterations. The recommend approach is to carve out the areas based on the component or features of the project. Iterations can be time based cycles for which you will repeat particular set of major activities such as plan, develop, and test.

Examples of MSF for Agile Software Development

The following examples show how the MSF for Agile Software Development process has been adopted and used by the *patterns & practices* team within Microsoft as well as by a development team outside of Microsoft.

Example 1: patterns & practices Teams

The following example shows how a typical *patterns & practices* project is executed by using the MSF for Agile Software Development process.

New project through iteration 0

The product manager:

- Interacts with customers and stake holders to gathers requirements. These are captured in a Microsoft Word document called the Project Back Log.
- Creates a vision statement for the project by using Microsoft PowerPoint.
- Brain storms with customers and various stake holders and defines the scenarios that will address the requirements and the vision of the project.

- Works with the project manager and other stake holders to prioritize the scenarios.

The project manager:
- Captures the scenarios as work items in Team Foundation Server.
- Decides the iteration cycle duration, depending upon the project size, and delivery capabilities.

Pre-Iteration Planning

The project manager:
- Decides depending upon the priority which scenarios should be worked on during the iterations.

The product manager together with the project manager:
- Creates Quality of Service Requirements (QoS) for the scenario. They then link the quality of service requirements to the scenarios.

Iteration Planning

The project manager:
- Together with developers and other members, breaks the scenarios into development tasks.
- Captures the development tasks in TFS and links them to the scenarios.
- Defines the acceptance criteria for each of the development tasks.
- Breaks the QoS into test tasks.
- Captures the test tasks in TFS and links them to the scenarios
- Defines the acceptance criteria for each of the test tasks.
- Schedules and assigns the tasks.

The developer:
- Estimates each development task.

Important – If it looks like the tasks (developer stories) might take longer than a day or two to implement, then you should break these down into sub-stories.

The tester:
- Provide estimation for each of the test tasks.

During the Iteration

The project manager:
- Guides the iteration.

The developer:
- Writes code for the development task, and closes the tasks once the acceptance criteria are met.

The tester:
- Executes the test tasks assigned to them and creates new bugs (work items) for any issues identified.

After the Iteration

The project manager:

- Assesses the project progress, re-prioritizes the scenarios which are incomplete from the current iteration.
- Provide a status report to stakeholders.
- Decides depending upon the priority which scenarios should be worked on during each iteration.

The product manager:

- Adds any newly discovered scenarios.
- Reprioritizes scenarios (where necessary).
- Together with the project manager creates Quality of Service Requirements (QoS) for the project. The product manager also links the quality of service requirements to the scenarios.

Example 2: Field Customer Engagement

The following example shows how the MSF for Agile process is used by a field customer engagement.

New project through iteration 0

The business analyst:

- Creates a short (one page) vision statement.
- Identifies an on-site customer who can be used to provide input and creates personas.
- Brainstorms scenarios (just the names) with the customer.
- Prioritizes scenarios with the customer.
- Writes scenarios for the upcoming iteration.

The project manager:

- Gathers the developers together and obtains their estimates. The estimates are rough orders of magnitude estimates.
- Checks if priorities change as a result of costs.
- Schedules scenarios for the upcoming iteration.

The architect:

- Divides scenarios into architecture tasks.

The developer:

- Divides scenarios into development tasks.
- Defines an appropriate build strategy (Continuous Integration if possible).

The tester:

- Divides scenario into test tasks.

During the Iteration

The project manager:

- Guides the iteration.
- Guides the project.

The architect:
- Defines the solution architecture.

The developer:
- Implements a development task.

The tester:
- Tests a scenario.

After iteration 0

The tasks change slightly at this point.

The business analyst:
- Updates personas (where necessary).
- Adds any newly discovered scenarios.
- Reprioritizes scenarios (where necessary).
- Writes scenarios for the upcoming iteration.

The project manager:
- Estimates any new scenarios.
- Schedules scenarios for the upcoming iteration.

The architect:
- Divides scenarios into architecture tasks.

The developer:
- Divides scenarios into development tasks.
- Updates the build process (Continuous Integration if possible).

The Tester
- Divides scenarios into test tasks.

Customizing MSF for Agile Software Development Process

You can use two methods to customize the MSF for Agile Software Development process template to suit your specific organizational requirements.

- **Manually customize the XML files**. This is manual process and hence is error prone but it gives you fine grained control on the customization of the process templates. For more information see “Customizing Process Templates” at http://msdn2.microsoft.com/en-us/library/ms243782(VS.80).aspx,
- **Process Editor tool available with Power Tools**. The latest version of the TFS Power Tools provides you with a user interface-based tool to view and customize process templates. When connected to Team Foundation Server, you can use this tool to customize work item type definitions and global lists on an active project. For more information see “How To: Customize the Process Template by using the Process Editor.”

PART VI

Process Templates

In This Part:

- ▶ **Process Templates Explained**

Chapter 14 – Process Templates Explained

Objectives

- Understand the purpose, content and structure of process templates.
- Learn the key differences between MSF Agile and MSF CMMI.
- Customize process templates to meet your team's needs.

Overview

This chapter explains the role of process templates in Team Foundation Server. It identifies the main features and main differences between the two supplied process templates: MSF Agile and MSF CMMI.

Software-development processes are complex, involving many levels of interdependent activities. Processes are generally available in documentation form but are not enforced by tools. This makes it difficult for development teams to learn and follow the processes consistently. Project managers might use a variety of different tools for project management, requirement management, bug tracking, or review management and they are often not integrated in any way. By using different tools you make it difficult to enforce a consistent methodology implementation across multiple projects and to generate consistent reports with common understanding. This makes the process analysis unreliable, which makes it difficult to identify process improvements over time.

Visual Studio Team System and Team Foundation Server provide an integrated environment that supports most of the process activities involved in a software-development project. TFS implements its life-cycle methodologies by using process templates. A process template is a set of XML files that provides specifications for different artifacts and processes of a methodology. This chapter explains the architecture of a process template and its components to give you a better understanding.

If these are not a good match for your team's development process you can create a new process template or modify the existing templates. For example you can add new reports or new work item types.

How to Use This Chapter

Use this chapter to understand process template architecture, structure and customization.

- **Read the "MSF for Agile and CMMI Process Templates" section**. Learn about the default process templates and how to choose between them.
- **Read the "Customizing Process Guidance" section**. Learn how to customize the existing process templates to better suit your team's needs.

MSF for Agile and CMMI Process Templates

Out of the box TFS comes with two process templates MSF for Agile (MSF) and CMMI. These are aimed at two different types of development. You use MSF for Agile if you are using an agile methodology to build your software. MSF for Agile encourages test first development and other agile practices. You use CMMI if you are following the Capability Maturity Model® Integration methodology. This is a formal process aimed at improving the existing development process. The templates differ in what they provide, for example, the default reports created are different as are some of the work item types. These templates can easily be edited to suit the needs of your project.

Customizing Process Guidance

The project you are creating may not fit the Process templates provided with VSTS. You may simply need a different work item type or you are using an entirely different process methodology. For example, if you are using SCRUM there is no mention of sprints in the current process template. In these case you need to amend or replace the existing process template.

Process Template Architecture

There are three key pieces to the process template architecture:

- **Process template plug-ins**
- **XML process definition files.**
- **New Team Project Wizard**

Process Template Plug-ins

Process template plug-ins are components that run when a new team project is created. A plug-in sets up required files or configures data for its area. Following are the plug-ins available out-of-box with the Team Foundation Server.

- **Classification** – It defines a team project’s initial iteration and areas
- **Groups and Permissions** – It defines a team project’s initial security groups and their permissions
- **Windows SharePoint services** – It defines the project portal for the team based on a SharePoint site template. Also defines template files and process guidance.
- **Work Item Tracking** – It defines a team project's initial work item types, queries, and work item instances.
- **Reports** – It defines a team project's initial reports and sets up the report site.
- **Version Control** – It defines a team project's initial version control security permissions, and check-in notes

Each plug-in can be modified to customize a process template. Except for the classification plug-in, plug-ins can also be deleted to customize a process template.

XML Process Definition Files

The XML process definition files are a set of XML files that define a set of tasks that must run to correctly configure a new team project for the process. When the New Team Project Wizard creates a team project, it runs the required plug-ins. Each plug-in reads its corresponding XML process definition file to obtain the list of tasks it must run. By using the XML process definition files, you specify what configurations and settings the plug-ins must implement. Following are the details about the xml files.

1. **Work Item Tracking XML** – It is named workitems.xml and is located in the Work Item Tracking folder in the process template folder hierarchy. There are three key types of tasks to specify: work item types, work item queries, and work item instances.
    a. **Work Item Types** – It defines the rules, fields, states, and transitions for an item of work that will be tracked on a team project; such as tasks, bugs, requirements, etc.
    b. **Work Item Queries** – It is used to find specific groupings of work items, such as tasks or active bugs. Work item queries are specified in work item query (WIQ) files in the **Queries** folder under the **Work Item Tracking** folder in the process template folder hierarchy.
    c. **Work Item Instances** – It is the initial set of work item instances that are already created when team members begin working.
2. **Classification XML** – It is named as classification.xml and is located in the Classification folder in the process template folder hierarchy. IT has two parts Iterations and Areas.
    a. **Iterations** – It is used to define how many times the team will repeat a particular set of major activities (such as plan, develop, test). Iterations affect work item queries and reports because iterations are used to group work items.
    b. **Areas** – It is used to organize work in the team project. For example, a team could organize the product work such as UI area, Application area, and Database area. Areas are used to group work items for queries and reports.
3. **Windows SharePoint Services XML** – It is named WssTasks.xml and is located in the Windows SharePoint Services folder in the process template folder hierarchy. There are three key tasks to specify: which site template to use, which document libraries to create, and which folders and files to copy into the document libraries.
    a. **Site Template** – You must specify a site template that the project portal is based on. The site template also must be available on the Team Foundation Server SharePoint Portal. Site templates are not included in the process template.
    b. **Document Libraries** – After the project portal is created, you can specify that additional document libraries be created.
    c. **Folders and Files** – After the project portal is created, you can specify additional folders to create. You can also specify files to copy such as template files.
4. **Version Control XML** – It is named VersionControl.xml and is located in the Version Control folder in the process template folder hierarchy. It defines a team project's initial version control security permissions, check-in notes, and whether exclusive check out is required.
    a. **Check-in Notes** – It is provided by the developer when code is checked in to describe how, or if, the code changes are related to team processes. For example, a check-in

    note can indicate if the change was part of a security review, and include details about the changes relative to the security review.
    b. **Exclusive Check-out** – It is used to control whether multiple people can check out a file at the same time.
    c. **Permissions –** Specifying permissions will define what actions security groups and individuals can perform on items under version control.
5. **Reports XML** – It is named ReportsTasks.xml and is located in the Reports folder in the process template folder hierarchy. It defines the team project's initial reports.
    a. **Reports Site** – The reporting site has a link to it on the project portal home page labeled Reports.
    b. **Folders** – You can create folders on the reporting site, folder appears on the project site and under the Reports folder in Team Explorer.
    c. **Reports** – Used to add reports using the .rdl files
6. **Groups and Permissions XML** – It is named GroupsandPermissions.xml and is located in the Groups and Permissions folder in the process template folder hierarchy. It is used to define the team project's initial security groups.
    a. **Groups** – Used to specify a new Team Foundation Server security group.
    b. **Permissions** – Each group you specify, you must also specify permissions.

New Team Project Wizard

The New Team Project Wizard is used to create new team projects. It uses the plug-ins and the xml process definition files for creating the project.

Customization Approach

The core methodology of customization of process template is

1. Review the out-of-box process templates and choose the one which is closet to your organizational process.
2. Download the chosen process templates
3. Customize the process templates for various components of the process templates
4. Upload the customized template to TFS
5. Verify the changes made suits your process requirements.

This core approach is adopted and used by following solutions for customizing the process templates

- **Manually customize the XML files**. This is manual process and hence is error prone but gives fine grained control on the customization of the process templates. For more information see “Customizing Process Templates” at http://msdn2.microsoft.com/en-us/library/ms243782(VS.80).aspx
- **Process Editor tool available with Power tools**. The latest version of power tool provides you a UI based tool for viewing and customizing process templates. When connected to Team Foundation Server, you can use this tool to customize work item type definitions and global lists on an active project. For more information see “How To - Customize the Process Template using Process Editor” at <<Add link here>>

Common Customization

Following are the set of the components which generally you will be customizing to adopt your custom process.

- **Groups and Permissions**. The out-of-box process templates have a set of groups with various permissions assigned to them. If the default groups and permissions associated with them are not adequate or appropriate for your process requirements, you can update those groups or create new groups. You can also add or remove individual users to a group or grant and revoke permissions for a group.
- **Source Control Check-In Notes and Policies**. The out-of-box process templates have a set of Source Control Check-In notes and Policies. If the default check-in notes are not adequate or appropriate for your process requirements, you can add or remove check-in note fields, or make some fields required and others not. If the default check-in policies are not adequate or appropriate, you can add, update, or delete individual check-in policies.
- **Areas and Iterations**. The out-of-box process templates do not provide a classification structure for either areas or iterations. You can customize the areas and iterations structure as per the process requirements you have. The recommend approach is to carve out the areas based on the component or features of the project. Iterations can be time based cycles for which you will repeat particular set of major activities (such as plan, develop, test).
- **Team Portal**. The out-of-box process templates provide a default team portal, which can be central focus for communication among team members and others in the organization. You can modify the team portal to change its appearance, behavior and content to suit your process requirements.
- **Process Guidance**. The out-of-box process templates provide relevant process guidance which explains the roles, forms, reports and work flow used in the team project. While customizing the process template for your requirement, you will have to edit the process guidance to reflect the changes for various components.
- **Reports**. The out-of-box process templates provide set of default reports. If the default reports are not appropriate or adequate you can create your own custom reports for your requirements.
- **Customizing Work Item Types and Queries**. The out-of-box process templates have a set of Work item types, default work item instances and queries. If the default Work item types, work item instances and queries are not adequate or appropriate for your process requirements, you can customize work item types to fit your workflow or different types of work items that you want to track. For example, you can:
    - Add New work item type
    - Remove existing Work item type
    - Add default work item type instances
    - Remove default work item type instances.
    - Create your own public or private queries.

You can also modify an existing work item type. For example:

- Add fields.
- Rename fields.

  - Restrict the list of allowed values for fields.
  - Change the states and supported state transitions.
  - Make fields required or read-only.
  - Make one field dependent on another.
  - Automatically populate field values.
  - Re-arrange the appearance of information on the form.
  - Change which Microsoft Project column a certain field is mapped to.

How Does It Work

- The user launches the New Team Project Wizard The Wizard asks for
  - The name of the team project.
  - The process template to be used when creating the team project.
- Which screens are displayed in the wizard are determined by which plug-ins are used. For example, if a process template does not include the Windows SharePoint Services plug-in, there is no screen presented to ask information about the project portal.
- After user finishes the wizard and clicks **Finish**, the wizard makes calls to the plug-ins to perform the work of creating the team project.
- The order in which the plug-ins are called is determined by the XML process definition files.
- The wizard reads the instructions contained in the process template and then creates and configures the items specified
- User do not need to specify in the wizard any details about the various types of work items to be created because the instructions are already provided by the process template.
- If the wizard encounters a problem while creating the new team project, you will see an error message describing the problem and suggesting corrective action.

PART VII

Reporting

In This Part:

- ▶ **Reporting Explained**

Chapter 15 - Reporting Explained

Objectives

- Describe Team Foundation Server reporting architecture.
- Identify the components that make up Team Foundation Server reporting.
- Describe the purpose of each available report.
- Know which process template contains which report.
- Customize and create new reports.

Overview

This chapter describes TFS reporting architecture and the common reports that you can use with new team projects. It also connects common reporting scenarios with reports available in Team Foundation Server (TFS), and describes common reasons to customize existing reports or create new reports. TFS reporting enables you to view aggregated data over many aspects of your team project. You can use this information to analyze project progress, project health, and the effectiveness of your development and test teams.

TFS reporting uses SQL Server 2005 Reporting Services to create, manage and run reports. Each process template contains a pre-defined set of reports that you can use from the start of your project. By using Reporting Services you can also amend these reports and create custom reports for your project. You can add new reports to an existing process template so that they are available for other team projects.

How to Use This Chapter

Use this chapter to understand how Team Foundation Server reporting works and how it can help you assess project health and status.

- **Read the "Scenarios and Solutions" section**. Understand common reasons to use TFS reporting and learn the purpose of each standard report.
- **Read the "Physical Architecture" section**. Learn what components make up the reporting system and how they interrelate.
- **Read the "Customizing Reports" section**. Learn the mechanisms available for report customization and creation.
- **Read the companion How Tos**. Read the following companion How Tos for a step-by-step walk through of various procedures discussed in this chapter.
    - How To - Customize a Report with Visual Studio 2005 Team Foundation Server"
    - How To - Create a Custom Report in Visual Studio 2005 Team Foundation Server
    - How To - Create a Risk Over Time Report with Visual Studio 2005 Team Foundation Server

Scenarios and Solutions

Reports are the primary way project managers and team leaders keep up to date with an ongoing project. After you create a new team project a set of reports are generated for you based upon the process template you choose. These reports are available to you from the project SharePoint portal or from the reports node in Team Explorer inside Visual Studio.

The following are common questions that can be answered by using TFS reports:

- When will my application be ready to ship?
- Is work proceeding according to plan?
- What is the quality of the build?
- What is the status of development against defined scenarios?
- How quickly is development work getting completed?
- Are bugs getting fixed?
- Are bugs regressing?

MSF Reports

Both the MSF for Agile Software Development and MSF for CMMI Process Improvement process templates each include a set of reports by default.

Bugs

The bug related reports enable you to see the kind of bugs being generated and fixed and to spot trends. The following bug related reports are available:

- **Bugs by Priority**. Are the correct bugs being found? This report shows you the rate of high priority bugs found vs. low priority. This report is both of the supplied process templates.
- **Bug Rates**. How effectively are bugs being found, fixed and closed? This report shows trends over time for new bugs, bug backlogs, and bug resolution. This report is available in both of the supplied process templates.

Release Management

Release management reports enable you to judge how close your software is to being fit for release. The following release management reports are available:

- **Actual Quality versus Planned Velocity**. How many scenarios can be completed before quality is unacceptable? This report presents the relationship, for each iteration, of estimated size to overall quality. This report is available in both process templates.
- **Builds**. What is the quality of a build? This report provides a list of available builds including build quality and other detailed information. This report is available in MSF for CMMI Process Improvement.
- **Quality Indicators**. What is the quality of the software? This report collects test results, bugs code coverage and code churn into a single report to track project health. This report is available in MSF Agile and MSF CMMI.
- **Velocity**. How quickly is the team completing work? This report shows how quickly the team is completing planned work and show rates of change from day to day. This report is available in MSF Agile and MSF CMMI.
- **Scenario Details**. What scenarios are we building the application against? This report provides information on each scenario including completion status, risks and testing progress.

Testing

Testing reports enable you to monitor the effectiveness and progress of your testing. The following test reports are available:

- **Regressions**. What tests passed at one point but are now failing? This report shows a list of all the tests that passed previously and now are failing. This report is available in MSF CMMI.
- **Requirements Test History**. How well tested are my scenarios and requirements? This report shows the progress of testing against defined scenarios and requirements. This report is available in MSF CMMI.
- **Test Failure Without Active Bug**. Are there bugs to track every known defect? This report shows any tests that have failed and are not associated with an open bug. This report is available in MSF CMMI.
- **Test Passing With Open Bug**. Is the bug list up to date and consistent with application quality? This report shows stale bugs for which tests are now passing. This report is available in MSF CMMI.
- **Load Test Summary**. What are the results of load testing on application performance? This report shows test results for load testing on your application. This report is available in MSF Agile.

Work Items

Work item reports enable you to assess the current state of your project and current project progress. The following work item reports are available.

- **Open Issues and Blocked Work Items Trend**. How many open issues remain? This report shows the remaining open issues as well as the trend toward resolving them. This report is available in MSF CMMI.
- **Reactivations**. How many work items are being reactivated? This report shows work items that have been resolved or closed prematurely. This report is available in MSF Agile and MSF CMMI.
- **Related Work Items**. What work items are dependent on other work items? This report shows a list of work items that are linked to other work items so you trace dependencies. This report is available in MSF CMMI.
- **Remaining Work**. How much work is left to be done and when will it be completed? This report shows work remaining, resolved and closed over time. Projecting remaining work trends forward can enable you to predict the point at which you will be code complete. This report is available in MSF Agile and MSF CMMI.
- **Triage**. What work items need to be triaged? This report shows every work item that is still in the proposed state. This report is available in MSF CMMI.
- **Unplanned Work**. How much unplanned work is there? This report charts total work versus remaining work and distinguishes planned from unplanned. This report is available in MSF Agile and MSF CMMI.
- **Work Items**. What are the active work items? This report lists every active work item. This report is available in MSF CMMI.
- **Work Items by Owner**. How much work is assigned to each member of the team? This report shows work items per team member. This report is available in MSF CMMI.

- **Work Items by State**. How many active, resolved and closed work items are there? This report lists work items organized by state. This report is available in MSF CMMI.

Customizing Reports

You might want a report that does not exist in either of the MSF process templates. You can customize reports in one of three ways:

- **Use a filter on an existing report**. Many reports provide parameters that you can use to filter the report. For example date, area, iteration, and priority filters are available. Use these filters to review a subset of the data provided in the report. These filters are temporary and are lost when you browse away from the report.
- **Customize an existing report**. If a report you want is similar to an existing report, it is often easiest to make a copy of the existing report and modify it. For instance, you may want to plot risks over time to analyze how well you team is working with project risks.
- **Create a new report**. You can create a new report from scratch.

If you modify an existing report or create a new report from scratch you can publish it to the report server so that it is available to the rest of your team. If you want to modify an existing report or create a new report, you can use one of the following methods:

- You can use the report designer available from the team reporting site.
- You can use Excel to create pivot tables from data in the reporting databases.
- You can create a new report server project in Visual Studio and then create new reports or import existing reports.

Using a Report Server project in Visual Studio is the most powerful and flexible method for report creation and modification.

More Information

- For more information about customizing an existing report, see “How To: Customize a Report with Visual Studio 2005 Team Foundation Server”.
- For more information about creating a custom report, see “How To: Create a Custom Report in Visual Studio 2005 Team Foundation Server.”
- For a step-by-step guide to creating a Risk Over Time report, see “How To: Create a Risk Over Time Report with Visual Studio 2005 Team Foundation Server.”

Physical Architecture

Team Foundation Server is built on SQL Server 2005 and uses SQL Server Analysis Services to aggregate data and drive reports. You can create new reports by using the process template or individual team members can use Microsoft Excel or Visual Studio 2005 Report Designer. Reports are displayed through SQL Server 2005 Reporting Services and the team report site. Figure 15.1 shows the physical reporting architecture.

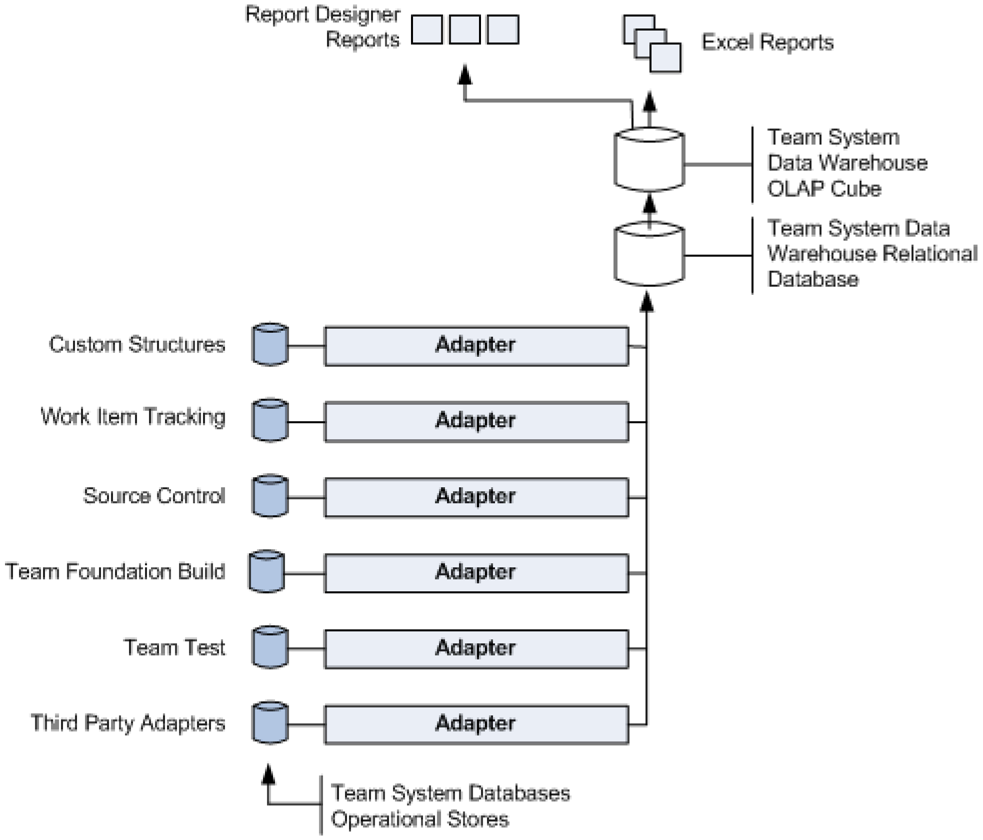


Figure 15.1 *Physical Reporting Architecture*

Each component of Team Foundation maintains its own set of transaction databases. This includes work items, source control, tests, bugs, and Team Build. To ease reporting, this data is aggregated into a relational database from which you can generate reports. The data is also placed in an OLAP cube to support trend-based reporting and more advanced data analysis.

TFSWarehouse is a data warehouse designed to be used with queries rather than transactions. Data is transferred from the various Team Foundation databases into this warehouse for reporting purposes. The warehouse is useful for seeing current state of project vs. seeing trends. TfsReportDS datasource points to TFSWarehouse. The Team System data cube is an OLAP database that is accessed through SQL Server Analysis Services. The cube is useful for data analysis of trends such as ‘how many bugs closed this month versus last month?’ TfsOlapReportDS points to the Team System data cube in the analysis services database.

Report System Components

The reporting system consists of the following server-side and client-side components.

Server-Side Components

Server-side components include:

- **Report server databases**. These contain the report definitions, historical reports and configuration data.

- **Report server Web service**. This provides programmatic access to the Report Server.
- **Report Manager Web site**. This provides user access to Report Server through a Web browser.
- **Windows service**. Provides the scheduling and delivery of report snapshots.

Client-Side Components

Client-side components include:

- **Browser**. To access the Report Manager Web site
- **Team Explorer**. To access reports through Visual Studio.
- **Business Intelligence Designer Studio (BIDS)**. This enables developers to design and deploy reports.
- **Report Builder**. This allows end users to design ad hoc reports.

Additional Resources

- Team Foundation Server Reporting: http://msdn2.microsoft.com/en-us/library/ms194922(VS.80).aspx

PART VIII

Setting Up and Maintaining the Team Environment

In This Part:

- **Team Foundation Server Deployment**
- **Providing Internet Access to Team Foundation Server**

Chapter 16 - Team Foundation Server Deployment

Objectives

- Learn the advantages and disadvantages of single-server and multi-server deployments.
- Select a deployment topology to suit your requirements.

Overview

This chapter outlines the main approaches and decision points when deploying Team Foundation Server into your organization. The chapter explains deployment options and describes how to choose between them.

There are two deployment options, single-server or dual-server. A single-server installation places the data-tier and application-tier on a single machine. A dual server splits the application and data tiers onto separate servers. Additionally, the build server and the source control proxy server may be installed on separate machines or as part of the data tier. Each client requires access to the servers and the appropriate client side tools must to be installed.

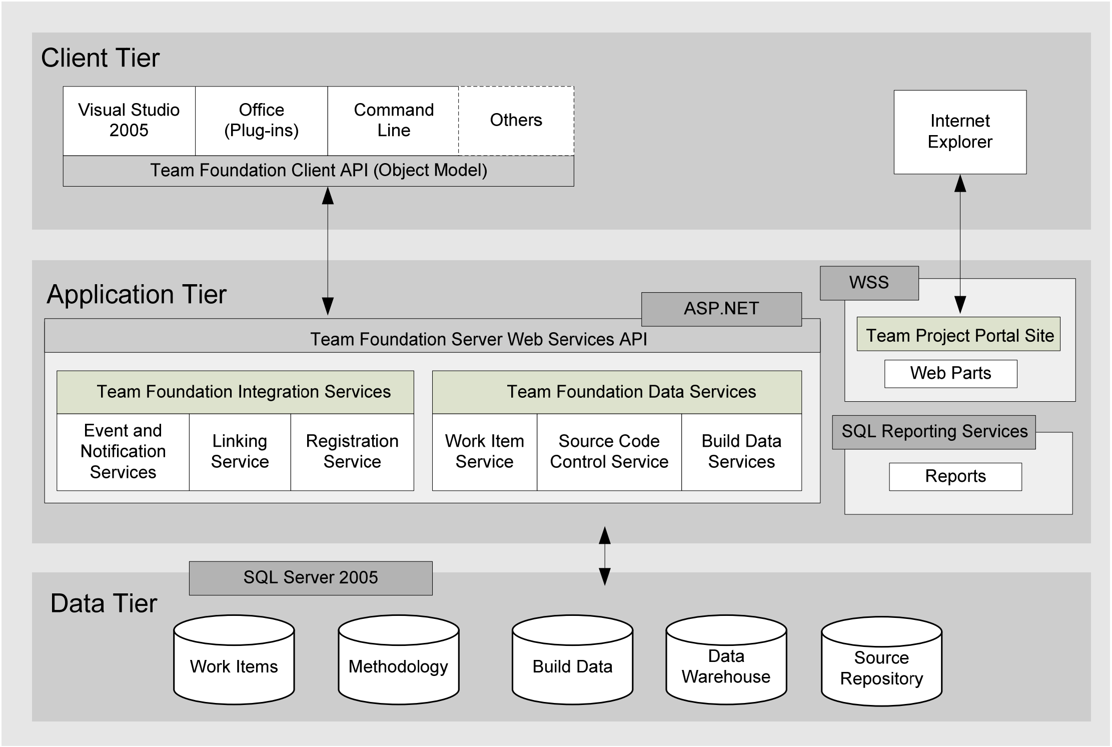
How to Use This Chapter

Use this chapter to determine your Team Foundation Server deployment strategy.

- **Understand TFS Architecture**. If you are unfamiliar with the Team Foundation Server architecture read the “TFS Architecture” section, or for more information read Ch 2 – Team Foundation Server Architecture.
- **Choose a deployment strategy**. Read the “Deployment Scenarios” section to determine which deployment strategy will work best for your team.

.

TFS Architecture



The TFS Architecture is made up of various tiers, the data tier, the application tier and the client tier. These are logical separations, the data tier, application tier and client tiers can be installed onto the same computer.

The Team Foundation data tier consists of Microsoft SQL Server 2005. There are a number of databases installed that store the work items, the version control system, test results and any reports.

The Application tier contains a Web based front end integrated into Internet Information Server (IIS), the Team Foundation Web services and Sharepoint services. The application tier also contains any build servers and source control proxy servers.

The client tier contains the applications that access Team Foundation Server. Developers use Team Explorer to connect to Team Server, installed either as a standalone tool or as part of Visual Studio 2005. Project Managers use Microsoft Excel or Microsoft Project. You can also use third party tools to connect to the server.

For more information, see “Chapter 2 – Team Foundation Server Architecture”

Deployment Scenarios

You can deploy Team Foundation Server in different ways

- Single Server Deployment

    - Within a Workgroup
    - Using Active Directory
- Dual Server Deployment

Workgroup Deployment

You create a Workgroup where you have no Active Directory server. You use this installation mode when you have a small team. If you use this installation mode each user will need a local account on the server machine to allow them to log on to the server.For this deployment you can only install onto a single server, dual server installation is not supported.

Deployment with Active Directory

If you have an Active Directory then you have two deployment choices. You can install both the data tier and the application tier on the same server, or you can install the data tier and the application tier on separate servers.

How to Think About Deployment

To decide if a single or dual server install is the right choice, consider the following questions:

- **How many users do I need to support?** If you plan to have more than 100 users, consider whether a dual-server deployment might better suit your needs.
- **How many projects will I be supporting with Team Foundation Server?** If you are supporting a large number of projects, consider whether a dual-server Team Foundation Server deployment might better suit your business needs.
- **Do I have a server I can dedicate to Team Foundation Server?** The server in a single-server Team Foundation Server deployment should be dedicated to Team Foundation Server functionality. The Team Foundation server should not serve any other purpose, such as a mail server, file server, or database server for other applications.

Advantages of Single-Server Deployments

- **Simplicity** You can manage all aspects of Team Foundation Server deployment on a single server. You can configure all access rights and permissions for users and groups on one server. You only have one server to schedule for backup and maintenance.
- **Availability** Because both the application tier and the data tier are present on a single server, you do not have to consider network restrictions or network latency between the application tier and data tier when planning your deployment.

Advantages of Dual-Server Deployments

- **Scalability.** Single-server deployments are designed for up to 100 users. A dual server deployment will allow you to scale beyond this limit up to 500 users.
- **Fail-over.** You can redirect the application-tier server to a different data-tier server in case of maintenance or repairs, and you cannot configure and deploy an additional server that can act as a standby or fail-over application-tier server.

Single Server Deployment

Figure 16.1 shows a typical single server installation. Installed on the server are the Team Foundation application and data tiers, along with SharePoint services and SQL Server 2005.

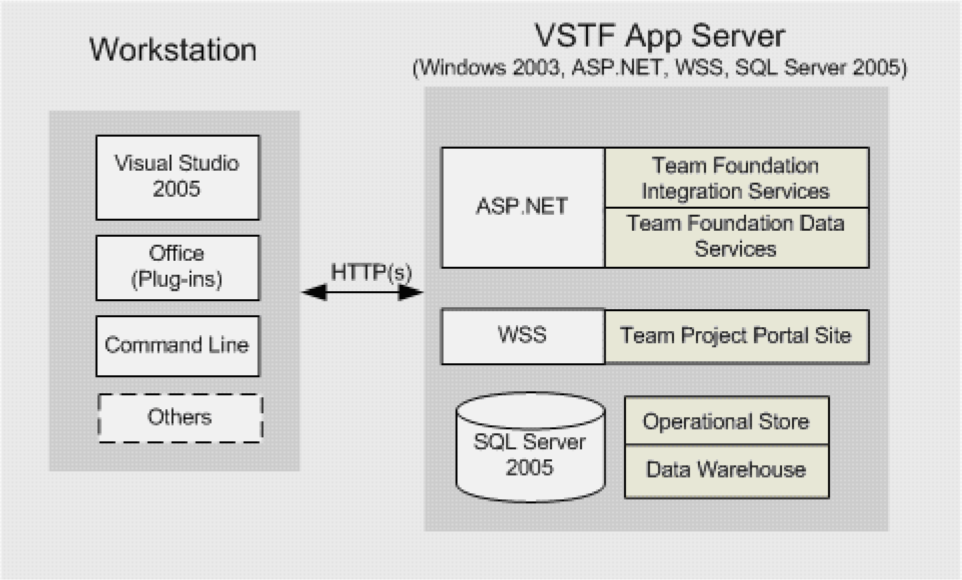


Figure 16.1 Single Server Installation

Dual Server Deployment

Figure 16.2 shows a typical dual server installation. The Team Foundation Application Tier is installed along with SharePoint services on one tier. The Team Foundation Data Tier is installed alongside SQL Server 2005 on another server.

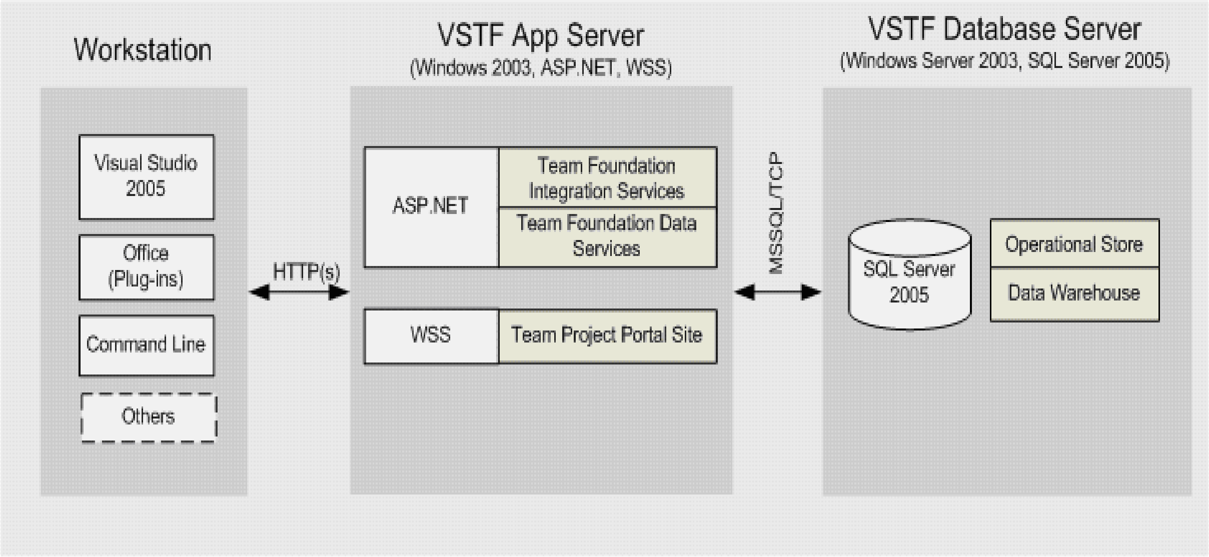


Figure 16.2 Dual Server Installation

Other Servers

In either the single or dual tier installation you can install a separate Build server and a Proxy server. These can be installed onto the same server as the application-tier or onto different servers.

Build Server Installation

Its possible to locate your build services on a separate server to improve the performance of your build and reduce load on your application tier. If your application tier server performance is impacted by frequent builds consider moving the build services to a separate server.

Team Foundation Proxy Server

The Team Foundation Proxy Server caches copies of source control files. You should use the proxy server if you are accessing the source control server over a network and there is high latency.

Team Foundation Server Topologies

Once you have decided on single or dual server installation there are several topologies you can use. Topologies range from simple to complex. Each topology is designed for teams of certain sizes.

Simple

Figure 16.3 shows a simple topology where the Team Foundation Server application-tier and data-tier components are deployed to a single server. The Team Foundation Server Proxy is deployed to a separate server. The server can be accessed from client workstations in the same domain.

This configuration is best suited to small development teams or to pilot projects.

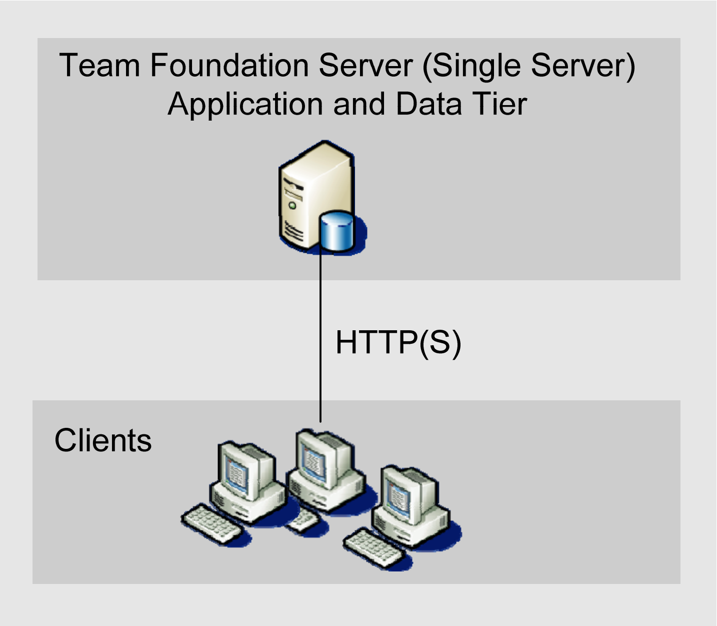

Team Foundation Server (Single Server)
Application and Data Tier
HTTP(S)
Clients

Figure 16.3 Simple Team Foundation Server Topology

Moderate

Figure 16.4 shows Team Foundation Server installed onto different tiers. The application services are deployed onto one application-tier node and the databases onto a separate data-tier node. The diagram also shows a test rig and Build servers being deployed to separate nodes. The client nodes are either in the same domain as the servers or in a domain that has a trust relationship with the servers.

Topologies of this complexity are targets for larger development teams in the range of 50 to 500 users.

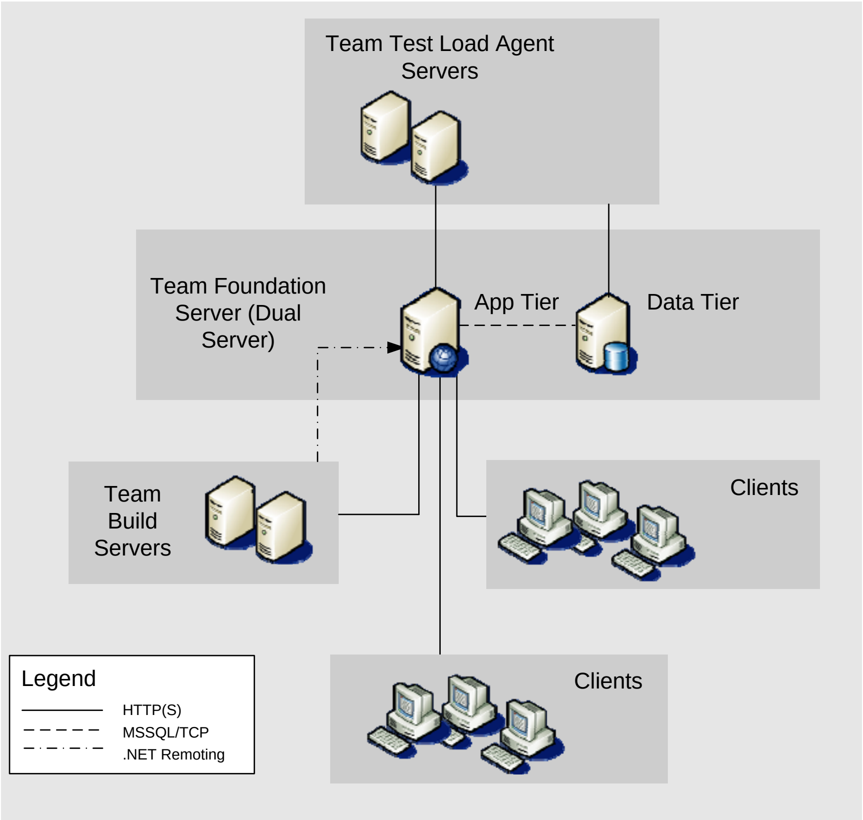


Figure 16.4 Moderate Team Foundation Server Topology

Complex

This complex topology shown in figure 16.5 is similar to the moderate topology. In this topology fail-over components have been added with an application-tier standby server and a data tier with SQL clustering technologies.

The diagram also shows a geographically distant child domain that uses a limited bandwidth connection. These clients use the Team Proxy server to improve access times to source control.

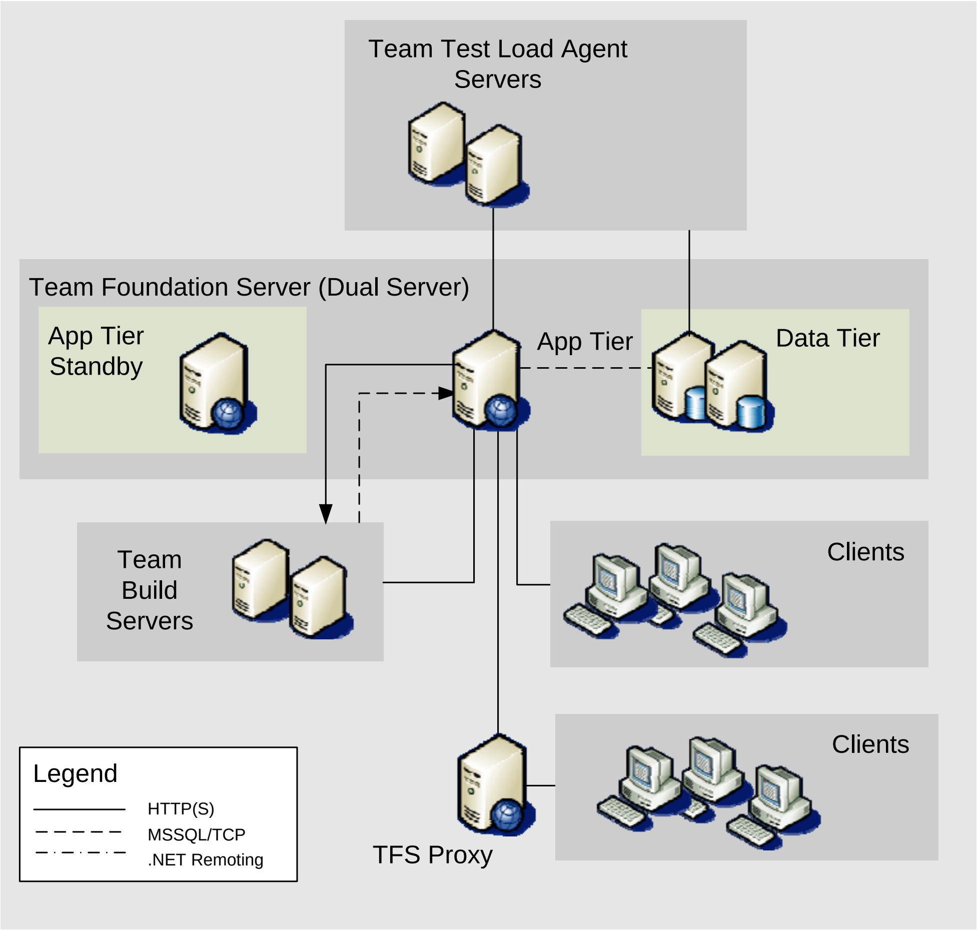


Figure 16.5 Complex Team Foundation Server Topology

Additional Resources

For more information in installing Team Foundation Server see the installation guide at http://go.microsoft.com/fwlink/?linkid=40042

Chapter 17 - Providing Internet Access to Team Foundation Server

Objectives

- Provide remote access to your Team Foundation Server over the Internet.
- Learn how to protect and restrict access to your Team Foundation Server.
- Improve remote access performance by using the Team Foundation Server Proxy.

Overview

This chapter explains how to provide access to remote users over the Internet to your Team Foundation Server (TFS). If you want to use Windows integrated authentication or if you are using a version of TFS earlier than Service Pack 1 (SP1), then you must setup a virtual private network (VPN) to provide external access to your server.

If you are using TFS SP1 and are satisfied with Basic or Digest authentication, you do not need to use a VPN and you can set up a proxy server to support more efficient remote access. By using a proxy server you improve performance for your remote users by caching commonly accessed source control files.

How to Use This Chapter

Use this chapter to setup your Team Foundation Server for remote access over the Internet.

- **Use the scenarios list**. Use this to quickly determine the correct approach you should adopt to provide external Internet access to your server.
- **Use the “Configuration Steps” section**. Use this section to understand the high level configuration steps required to configure your TFS installation for remote access using Basic or Digest authentication over HTTPS.
- **Use the referenced walkthroughs**. Use the following walkthroughs for step-by-step guidance to install and configure the certificates and SSL access to be used with Basic and Digest authentication.
  - “Walkthrough: Setting up Team Foundation Server with Secure Sockets Layer (SSL) and an ISAPI Filter” at http://msdn2.microsoft.com/en-us/library/aa833872(VS.80).aspx
  - “Walkthrough: Setting up Team Foundation Server to Require HTTPS and Secure Sockets Layer (SSL)” at http://msdn2.microsoft.com/en-us/library/aa833873(VS.80).aspx
- **Use the “Improving Remote Access Performance” section.** Use this section to identify ways of reducing the amount of traffic that needs to be sent over the Internet.

Scenarios

The version of TFS you have installed and the type of authentication you want to use to authenticate remote users governs the approach you should use to provide Internet access to your server. The main scenarios are:

- **Using a VPN Connection**. With this approach your remote development team uses a direction VPN connection to your TFS server on your internal network. The advantage of this approach is that it supports the use of Windows integrated authentication. The disadvantage is that a VPN solution might not be permitted by your company's security policy. If you have Team Foundation Server without SP1 or you require Windows integrated authentication, then using a VPN connection is your only option.
- **Locating TFS in an Extranet**. With this approach you install your TFS infrastructure inside your perimeter network. You configure TFS to only accept SSL requests, all users use Basic authentication, and all users use the same URL. In this scenario you cannot use Active Directory features for example to add users to TFS groups or add AD group to folders in source control.
- **Publishing TFS through Microsoft Internet Acceleration Server (ISA) 2006**. With this approach, you install TFS on your internal network use ISA Web publishing expose TFS to the external network. Remote users access TFS over SSL and use Basic authentication. An advantage of this approach is that ISA authenticates users and inspects traffic. The disadvantage of this approach is that not all TFS features can be enabled. With this approach, you cannot use the remote TFS proxy server, your remote users cannot remotely initiate builds or access the build log from the build report, they cannot create new team projects and they cannot manipulate Active Directory accounts in TFS.

Using a VPN Connection

If you are currently using TFS without SP1, you must use a VPN to provide remote access. To avoid using a VPN and to provide protected access by using Basic or Digest authentication over an HTTPS connection you must upgrade your installation to TFS SP1.

For more information about creating a VPN, see http://support.microsoft.com/kb/324747.

Using Basic or Digest Authentication

By using Team Foundation SP1, you can use either Basic or Digest authentication, by configuring Internet Information Services (IIS) on your TFS application tier. With Basic authentication, logon credentials are passed over the Internet using an unprotected Base64 encoded format. To protect your client's credentials you should only use Basic authentication over an HTTPS connection that uses Secure Sockets Layer (SSL).

Configuration Steps

To configure Basic or Digest authentication you must perform the following steps:

1. Install a server certificate on your Team Foundation Server. Before you can use SSL with TFS, you must install the server certificate and then configure HTTPS on the Team Foundation Server Web sites which include:
   - The Default Web Site
   - The SharePoint Central Administration Web site
   - The Team Foundation Server Web site.

    - If you installed Build Services on one or more servers, you must install the certificate on each of those servers. In order to perform builds over SSL, you must install the certificate in the trusted root store on both the build server for the account on which the build service is running and the computer that initiates the build.
2. Configure your Team Foundation Server Web site to require SSL.
3. Configure your firewall to allow SSL traffic. You must configure your firewall to allow for traffic on the SSL ports you specified in IIS for the various Team Foundation Server Web sites listed above.
4. Configure your Team Foundation Server Web site in IIS to use Basic or Digest authentication.
5. Configure the ISAPI filter to enforce the IP address rules. To do this you must create an ISAPI initialization file in the same directory as the AuthenticationFilter.dll file that is part of Team Foundation Server SP1. You must also add the ISAPI filter to the registry.
6. Install certificates on your Team Foundation client computers. You must install the server certificate (or a certificate higher in that certificate's trust chain) on every client computer that accesses Team Foundation Server. Additionally, if any client computer has previously accessed a team project, you must clear the client cache for every user who uses that computer to connect to your new Team Foundation Server.
7. Update the registry for SQL Report Server so that reports are displayed correctly on the team project portal sites.
8. Update SQL Server Management Studio with the HTTPS URL values for the Windows SharePoint Services and Reporting Services Web sites.

If you use the Team Foundation Proxy Server at the remote location, you must perform the following configuration:

1. Install a server certificate on each computer that you have installed Team Foundation Proxy Server.
2. If you have configured automatic e-mail notification alerts, update the Web.config file on your TFS Proxy Server computer to allow E-mail notification alerts.

For more information and details about how to follow this process see:

- "Walkthrough: Setting up Team Foundation Server with Secure Sockets Layer (SSL) and an ISAPI Filter" at http://msdn2.microsoft.com/en-us/library/aa833872(VS.80).aspx
- "Walkthrough: Setting up Team Foundation Server to Require HTTPS and Secure Sockets Layer (SSL)" at http://msdn2.microsoft.com/en-us/library/aa833873(VS.80).aspx

Configuring the ISAPI Filter

You can configure the ISAPI filter to enforce rules on any given set of IP addresses. The ISAPI filter uses an AuthenticationFilter.ini file for its configuration settings. You must configure this file with settings appropriate to your deployment. On the application tier computer create a file to hold the addresses you are managing. The file can use the following configuration keys and values:

```
[config]
```

```
RequireSecurePort=false
ProxyIPList=ProxyAddress;
SubnetList=SubnetMask;
```

- **ProxyIpList** is the semi-colon list of IP address where traffic will appear from that will use Basic/Digest authentication.
- **SubnetList** is the semi-colon separated list if IP address/subnet mask pairs that you do not want to enforce basic/digest for
- **RequireSecurePort** If you configure **RequireSecurePort** as False, only the connections using the addresses specified in **ProxyIPList** will be required to use HTTPS/SSL and either Digest or Basic authentication.

Important. Adding a **ProxyIPList** causes **SubnetList** to be ignored

Name this file AuthenticationFilter.ini and save it in the same directory as AuthenticationFilter.dll. By default, this directory is C:\Program Files\Microsoft Visual Studio 2005 Team Foundation Server\TF Setup

Next, you must add entries to the Windows registry to enable the filter. You can use the following commands:

- Reg.exe add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\TFS ISAPI Filter" /v EventMessageFile /t REG_SZ /d %windir%\Microsoft.NET\Framework\v2.0.50727\EventLogMessages.dll /f
- reg.exe add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\TFS ISAPI Filter" /v TypesSupported /t REG_DWORD /d 7 /f

Finally, add the ISAPI filter to your Team Foundation Server Web site. Set the name of the filter to **TFAuthenticationFilter** and the default location is C:\Program Files\Microsoft Visual Studio 2005 Team Foundation Server\TF Setup\AuthenticationFilter.dll.

Improving Remote Access Performance

To help improve the performance experienced by your remote team, you should install and configure Team Foundation Server Proxy in the remote office. This improves performance by caching source control files on the remote office’s proxy server.

To configure and optimizing proxy performance:

- Make sure caching is enabled and monitor the performance of your cache. Monitor the performance counters (installed by default) and event logs (for errors/warnings) on your proxy server on a periodic basis, to see how your proxy is performing. Note that TFS proxy saves cache performance statistics to an XML file called ProxyStatistics.xml and you can change the interval for saving these statistics. The ProxyStatistics.xml file is located in the App_Data folder in the proxy installation directory.
- Run a scheduled task to retrieve the latest files on a periodic basis to the proxy server. This helps to ensure that the latest version of the files are available in the

proxy cache and to ensure that subsequent client requests for these files result in a cache hit.

- If you know that large files are going to be downloaded over a low bandwidth (< 3Mbps) network, then set the **executionTimeout** configuration to an appropriate value in Web.config. The default value is one hour <httpRuntime executionTimeout="3600"/>.
- Disable the proxy on the clients if the proxy is going to be down for an extended period of time. Do this to prevent fruitless reconnections. By default, these are attempted every 5 minutes.
- Consider using workspace cloaking to hide specific workspaces from being viewed and to prevent unnecessary file transfers. Cloaking hides specified workspace folders from view and also increases performance bandwidth and conserves local disk space by preventing folders and files that you do not currently require from being copied to your local workspace. Although you can cloak an existing folder mapping in your workspace, a better approach is to create a new folder mapping specifically intended to be cloaked.

For more information about these performance optimization guidelines, see “Distributed / Remote Development” in “Guidelines: Visual Studio 2005 Team Foundation Server Source Control.”

Additional Resources

- To learn more about the Team Foundation Server Proxy, see “Team Foundation Server Proxy and Source Control”: http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx
- For more information about examining TFS proxy performance, see: http://msdn2.microsoft.com/en-us/library/ms252455(VS.80).aspx
- To learn more about the TFS Proxy File Cache Server, see the MSDN Webcast: http://msevents.microsoft.com/CUI/WebCastEventDetails.aspx?culture=en-US&EventID=1032291120&CountryCode=US

Guidelines: Source Control

Index

Accessing Version Control

- *Consider using command line tools*
- *Use Team Foundation Power Tools to unshelve a change*
- *Use Team Foundation Power Tools to rollback a change*
- *Use Team Foundation Power Tools to work offline*
- *Use Team Foundation Power Tools to get a changeset*
- *Use Team Foundation Power Tools to remove pending edits*

Administration

- *Turn off inherited permissions on maintenance branches*
- *Deny check-in permissions to developers you don't yet trust to make changes to your source*

Branch / Label / Merge

- *Use labels to mark builds you might need to return to*
- *Use branches to isolate supported releases*
- *Plan your branching structure along merge paths*
- *Branch at a high level, including configuration and source files*
- *Do not branch too deeply*
- *Do not branch unless you need to*
- *Avoid baseless merges where possible*
- *Prefer full merges to cherry-pick merges*
- *Merge frequently*
- *Always create a top-level folder for a new team project to serve as a main branch*
- *Before merging consider using the candidate or preview switch to double check*
- *When renames are part of the merge pay close attention to the path that the tool recommends*
- *Be careful when resolving merge conflicts.*
- *Check-in the results of one merge at a time.*
- *Build and run tests after the merge and prior to checkin*

Check-in and Check-in Policies

- *Only check-in your code when you are ready to share it.*
- *Use shelvesets to backup or share pending changes*
- *Resolve one work item per check-in*

- *Use check in policies to enforce coding standards*
- *Use check in policies to enforce a code quality gate*
- *Detect when a policy has been overridden*
- *Plan to avoid conflicts*

Checkout, Get, and Lock

- *Get latest source before making changes.*
- *Use the Lock command with discretion.*
- *Communicate with your teammates when locking files*

Distributed / Remote Development

- *Make sure you get an appropriate sized disk drive for your proxy*
- *Create a scheduled task to pull the latest files on a periodic basis.*
- *Monitor proxy performance counters and event log periodically.*
- *Configure executionTimeout based on file sizes and bandwidth*
- *Don't add an AT to Proxy.Config, if that AT is going to be used only for a short period of time*
- *Disable the proxy if the proxy is going to be down for an extended period of time*
- *Consider workspace cloaking to reduce unnecessary file transfers*

Migration

- *Use the VSS converter to migrate to Team Foundation Source Control*
- *Migrate from other source control systems to Team Foundation Source Control*

Project / Workspace Management

- *To isolate a single developer use workspaces not branches*
- *Delete and rename files by using source control, not Windows Explorer.*
- *Only delete and rename with your solution open*
- *Create one team project per application if you want to move your assets between application versions*
- *Create one team project per version if you want to start fresh with each application version*
- *Use branching to share code and binaries that require integration testing*
- *Avoid workspace mapping to support cross-project dependencies*
- *Create workspace mappings at the team project root level*
- *On shared computers, use a unique local folder path*
- *Consider mapping only part of the source tree*
- *Structure your source tree to support branching*

Shelving

- *Use shelving to share pending changes for review or hand off*

- *Use shelving to backup pending changes to the server*
- *Use shelving if interrupted by higher priority work*

Accessing Version Control

- *Consider using command line tools*
- *Use Team Foundation Power Tools to unshelve a change*
- *Use Team Foundation Power Tools to rollback a change*
- *Use Team Foundation Power Tools to work offline*
- *Use Team Foundation Power Tools to get a changeset*
- *Use Team Foundation Power Tools to remove pending edits*

Consider Using Command Line Tools

For operations not available from Visual Studio or if you need to schedule operations, consider using command line tools such as Tf.exe provided with Team Foundation Server and the Team Foundation Power Tools tool (Tfpt.exe) available as a separate download. You can use the command line tools to schedule operations by using the Windows Task Scheduler.

To ensure the appropriate path and other environment variables are setup, run Tf.exe from the Visual Studio 2005 Command Prompt window or run the Vsvars32 batch file, which is normally located in *DriveLetter*:\Program Files\Microsoft Visual Studio 8\Common7\Tools. The Tf.exe tool supports most source control commands including Commands, Checkin, Checkout, Get, History, Shelve, Branch, Merge, Label, Status, Undelete, and Undo.

The following are common operations you may want to execute from the command line by using Tf.exe:

- Synchronize files from the server to your local machine – tf get
- Add a file to the server – tf add
- Check out a file for editing – tf checkout
- Check in pending changes – tf checkin
- Retrieve a particular changeset from the server – tf get /version

There are certain operations that you can only perform from the command line:

- Delete another user's workspace – tf workspace /delete
- Undo another user's check-in – tf undo
- Unlock another user's lock – tf lock
- Define label scope – tf label
- Perform a baseless merge – tf merge

More Information

- MSDN: Walkthrough: Working with Team Foundation Source Control from Command Line: http://msdn2.microsoft.com/en-us/library/zthc5x3f.aspx

- For more information on commands only available from the command line, see “Operations Available Only From the Command-Line (Team Foundation Source Control)”: http://msdn2.microsoft.com/en-us/library/ms194957(VS.80).aspx

Use Team Foundation Power Tools to Unshelve a Change

The Team Foundation Power Tools (TFPT) provides version control functionality that is not available from Visual Studio. For example, you can use it to help support offline working or to perform rollback operations to undo check-ins of a changeset. Consider using TFPT if you need to unshelve a change.

The **unshelve** operation supported by TFS does not allow shelved changes and local changes to be merged together. By using TFPT to unshelve a change from a change set, if an item in your local workspace has a pending change that is an edit, and the shelved change is also an edit, then TFPT can merge the changes with a three-way merge.

You run this command from the command line by using Tfpt.exe

More Information

- You can download Team Foundation PowerTools from here: http://go.microsoft.com/?linkid=5431080
- A forum to discuss Team Foundation Power Tools: http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=930&SiteID=1

Use Team Foundation Power Tools to Rollback a Change

Consider using TFPT if you need to rollback a change.The ability to undo a checkin of a changeset is not directly supported by TFS. By using the TFPT **rollback** command, you can attempt to undo any changes made in a specified changeset. Not all changes can be rolled back but the rollback works for most scenarios.

You run this command from the command line by using Tfpt.exe

More Information

- You can download Team Foundation PowerTools from here: http://go.microsoft.com/?linkid=5431080
- A forum to discuss Team Foundation Power Tools: http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=930&SiteID=1

Use Team Foundation Power Tools to Work Offline

Off-line working is not supported natively by Team Foundation Server. If you want to work off-line you must adhere to the following strict workflow:

1. Manually remove read-only flag
2. Edit files

3. Add or delete files
4. Run the Team Foundation Power Toy (TFPT) online command

Important: You must not rename any files while you're off-line.

1. **Manually remove read-only flag**
   By default, all files in the workspace that have not been checked out are marked as read-only. When you work without a server connection, you must manually remove the read-only flag from files before editing or deleting them. To do this, right-click the file in Windows Explorer, click **Properties**, uncheck the **Read-only** check box and click OK. Alternatively, you can use the DOS command “attrib -r”
2. **Edit files**
   You can now edit any files for which you have removed the read-only flag.
3. **Add or delete files**
   You can add or delete files for which you have removed the read-only flag. Do not rename files, because the TFPT **online** tool cannot distinguish a rename from a deletion paired with an add. Note that you have to specify an option to the Tfpt **online** command to get it to look for deletions, as this is a more time consuming operation.
4. **Run the TFPT online command**
   When you are back online, run the TFPT online command by typing "TFPT online" at the command line. This command scans your workspace for writable files and determines what changes should be pended on the server. If you have deleted any files, use the /**delete** switch. This tells the tool to scan for deleted files in your local workspace as well. The tool then displays the online window from which you can choose which changes to pend into the workspace.

More Information

- You can download Team Foundation PowerTools from here:
  http://go.microsoft.com/?linkid=5431080
- A forum to discuss Team Foundation Power Tools:
  http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=930&SiteID=1

Use Team Foundation Power Tools to Get Changeset

Consider using TFPT if you need to get a changeset.The TFPT **GetCS** command enables you to get all the items listed in a changeset at that changeset version. This is useful if a colleague has checked in a change which you need to have in your workspace, but you cannot update your entire workspace to the latest version.

You run this command from the command line by using Tfpt.exe

More Information

- You can download Team Foundation PowerTools from here: http://go.microsoft.com/?linkid=5431080
- A forum to discuss Team Foundation Power Tools: http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=930&SiteID=1

Use Team Foundation Power Tools to Remove Pending Edits

Consider using TFPT if you need to remove pending edits. The TFPT **Undo Unchanged** command removes pending edits from files which have not actually been edited. This is useful in a scenario where you check out a large number of files for edit, but only actually make changes to a small number of them. You can back out your edits on the unchanged files by running the TFPT UU tool, which compares hashes of the files in the local workspace to hashes the server has to determine whether or not the file has actually been edited

You run this command from the command line by using Tfpt.exe

More Information

- You can download Team Foundation PowerTools from here: http://go.microsoft.com/?linkid=5431080
- A forum to discuss Team Foundation Power Tools: http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=930&SiteID=1

Administration

- *Turn off inherited permissions on maintenance branches*
- *Deny check-in permissions to developers you don't yet trust to make changes to your source*

Turn off Inherited Permissions on Maintenance Branches

Once a branch is in maintenance, for example after you have shipped a version of your software, you can turn off inheriting permissions to lock down the tree. Once you have done this, you can grant individual users the pend-change and check-in permissions as needed for hot fixes.

More Information

- For more information on removing permissions, see "How to: Remove Access to Source Control Files": http://msdn2.microsoft.com/en-us/library/ms400718(VS.80).aspx

Deny Check-in Permissions to Developers You Don't Yet Trust to Make Changes to Your Source

You can deny check-in permissions on the source tree for un-trusted developers, such as new hires or interns. Make sure that you set your desired permissions (including permissions for your own account) before turning off inheritance. Rather than check-in directly they can make pending changes and then shelve these changes. A more experienced developer can unshelve the changes, review them, and then check them in.

More Information

- For more information on removing permissions, see "How to: Remove Access to Source Control Files": http://msdn2.microsoft.com/en-us/library/ms400718(VS.80).aspx

Branch / Label / Merge

- *Use labels to mark builds you might need to return to*
- *Use branches to isolate supported releases*
- *Plan your branching structure along merge paths*
- *Branch at a high level, including configuration and source files*
- *Do not branch too deeply*
- *Do not branch unless you need to*
- *Avoid baseless merges where possible*
- *Prefer full merges to cherry-pick merges*
- *Merge frequently*
- *Always create a top-level folder for a new team project to serve as a main branch*
- *Before merging consider using the candidate or preview switch to double check*
- *When renames are part of the merge pay close attention to the path that the tool recommends*
- *Be careful when resolving merge conflicts.*
- *Check-in the results of one merge at a time.*
- *Build and run tests after the merge and prior to checkin*

Use Labels to Mark Builds You Might Need to Return To

Use labels to mark daily builds so that you can easily view and retrieve the set of files used for a particular build. Team Build automatically assigns a label to the set of files associated with each build that it creates. Team build labels follow the format "ReleaseType_Build#", for instance, "Releasex86_20070226.1".

While every team build is labeled, you may want to create your own labels for builds that you are likely to want to return to, such as:

- An internal milestone, e.g. alpha release
- A build created after branch or external dependency integration

You should use a label naming convention that easily allows you to come back to it later and know what it represents. Use labels that are clear about what they represent. For example use a convention such as "AlphaBuild_20070112" composed of "LabelName_Date".

When you release a build to customer for which you will need to provide maintenance, use a branch instead of a label to isolate future maintenance work on the release. You can subsequently merge the branch with any fixes it might contain back into to your main source tree.

To locate an existing label, from the **File** menu, point to **Source Control**, **Label** and click **Find Label**. Having found the label you can modify it or delete it from within the **Find Label** dialog box.

More Information

- For more information, see “Working with labels”: http://msdn2.microsoft.com/en-us/library/ms181439(VS.80).aspx
- For more information on labels, see “How to: Apply Labels”: http://msdn2.microsoft.com/en-us/library/ms181440(VS.80).aspx

Use Branches to Isolate Supported Releases

Create a branch containing the source code used to generate a supported release build. This allows you to apply fixes and updates to the supported release version of your software without impacting the continued development of your source code base which continues along the main branch.

You continue to develop the next version of your application along your main branch in parallel with the supported release branch. You can merge any stabilization changes you made to the supported release branch into the main branch prior to releasing the next version of your product.

The following is an example of what your branch structure may look like after you have created a Maintenance branch used to support released versions of your software:

- **Main** – Integration Branch
    - **Source**
    - **Other Asset Folders**
- **Maintenance** – Container for maintenance branches
    - **Release 1** – Maintenance Branch
        - **Source**

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For more information about branching strategies, see “Explained: Branching in Team Foundation Server.”
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Plan Your Branching Structure Along Merge Paths

You can only merge along existing branch paths by using Source Control Explorer. You can do baseless merges along other paths from the command line, but this type of merge is less intelligent, resulting in more merge conflicts, and additional conflicts in future merges.

Keep the following in mind when you perform merges:

- Merging along the hierarchy, from parent to child or child to parent, results in fewer conflicts than merging across the hierarchy.
- The branch hierarchy is based upon the branch parent and branch child, which may be different than the physical structure of the source code on disk. For example:
    - **Physical Structure**:
        - **Development –** Folder for isolated development branches
            - **External** - External dependency branch
            - **Team 1 -** Team branch
            - **Team 2 -** Team branch
                - **Feature A –** Feature branch
                - **Feature B –** Feature branch
                - **Feature C –** Feature branch
        - **Main –** Integration branch
        - **Maintenance -** Folder for release branches
            - **Release 2** – Release Branch
            - **Release 2 Hot Fix 1** – Supported hot fix Branch
        - **Safe Keeping -** Folder for safe keeping branches
            - **Release 1 -** Safe keeping branch
    - **Logical Structure**:
        - **Main**
            - **External**
            - **Team 1**
            - **Team 2**
                - **Feature A**
                - **Feature B**
            - **Maintenance**
                - **Release 1**
                - **Release 2**
                    - **Release 2 Hot Fix 1**

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance

- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Branch at a High Level, Including Configuration and Source Files

Branch at a high enough level that the branch you create is still compilable. For instance, in the following source tree:

- **Main** - container for all assets you need to ship the product
  - **Source** - container for everything you need to build
    - **Code** - container for source code
    - **Shared Code** – container for source code that is shared from other projects
    - **Unit Tests** - container for unit tests
    - **Lib** - container for binary dependencies
  - **Docs** - container for documentation that will ship with the product
  - **Installer** - container for installer source code and binaries
  - **Builds** - container for team build scripts
  - **Tests** - container for test team test cases

Branch at the level of the **Source** folder to ensure the new branch contains all source and configuration files.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Do Not Branch Too Deeply

Do not branch too deeply as this will add to the time required to merge a change from a child branch up to the topmost parent. For example, in the following branching structure:

- **Development** – Container for development branches
    - **Development Branch**
        - **Sub Branch**
            - **Sub Sub Branch**
- **Main** – Integration Branch
    - **Source**
    - **Other Asset Folders**

Merges result in fewer conflicts when performed along the branch hierarchy. Therefore, if there is a change in the **Sub Sub Branch** that you would like to merge into **Main** you would first need to merge with the **Sub Branch** and the **Development Branch** before you can merge into **Main.** Each merge takes time to complete, resolve conflicts, build and test which multiplies merge time by the level of branches you've created in your branch structure.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Do Not Branch Unless You Need To

Do not branch unless your development team needs to work on the same set of files concurrently. If you are unsure, you can label a build and create a branch from that at a later point. Merging branches, especially if there are significant changes between them, can be costly.

Merging requires one or more developers to execute the merge and sort out conflicts. The merged source must be thoroughly tested as it is not uncommon to make bad merge decisions that can destabilize the build.

Merging across the branch hierarchy is especially difficult and requires you to manually handle many conflicts that could otherwise be handled automatically.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Avoid Baseless Merges Where Possible

Avoid baseless merges where possible. When performing a baseless merge Team Foundation Server has no information describing the relationship of files and folders in the branches being merged. This generally results in more merge conflicts, and can lead to more conflicts in future merges.

Structure your branch trees so that you only need to merge along the hierarchy (up and down the branch tree) rather than across the hierarchy. Branching across the hierarchy forces you to use a baseless merge.

Keep the following in mind when performing merges:

- Merging along the hierarchy, from parent to child or child to parent, results in fewer conflicts than merging across the hierarchy.
- The branch hierarchy is based upon the branch parent and branch child, which may be different than the physical structure of the source code on disk. For example:
    - **Physical Structure**:
        - **Development** – Folder for isolated development branches
            - **External** - External dependency branch
            - **Team 1 -** Team branch
            - **Team 2 -** Team branch
                - **Feature A –** Feature branch
                - **Feature B –** Feature branch
                - **Feature C –** Feature branch

        - **Main –** Integration branch
        - **Maintenance -** Folder for release branches
            - **Release 2** – Release Branch
            - **Release 2 Hot Fix 1** – Hot Fix Branch
        - **Safe Keeping -** Folder for safe keeping branches
            - **Release 1 -** Safe keeping branch
    - **Logical Structure**:
        - **Main**
            - **External**
            - **Team 1**
            - **Team 2**
                - **Feature A**
                - **Feature B**
            - **Maintenance**
                - **Release 1**
                - **Release 2**
                    - **Release 2 Hot Fix 1**

More Information

- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Prefer Full Merges to Cherry-pick Merges

Rather than picking individual changes to merge between branches (a cherry-pick merge), choose to merge the entire branch at once. It can be convenient to select individual changes in a branch to merge with another branch. However, if a cherry-picked changeset falls in the middle of a future merge range, that changeset will be re-merged, potentially resulting in additional merge conflicts.

This is especially true for /discard merges run from the command line. The discarded changeset can be picked up if it falls in the middle of a future merge range.

More Information

- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx

- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Merge Frequently

Eliminate the latency between branches, especially when teams are working on the same release but in their own isolated branch. This ensures compatibility between changes.

The merge schedule depends on the complexity of your branch structure as well as the individual needs of your development team. As an example, in a moderately complex branch structure such as the one below the development branches may merge up to the main branch daily and merge changes back from the integration branch every couple of days.

- **Development –** Folder for isolated development branches
  - **External** - External dependency branch
  - **Team 1 -** Team branch
  - **Team 2 -** Team branch
    - **Feature A –** Feature branch
    - **Feature B –** Feature branch
    - **Feature C –** Feature branch
- **Main –** Integration branch
- **Maintenance -** Folder for release branches
  - **Release 2** – Release Branch
  - **Release 2 Hot Fix 1** – Hot Fix Branch
- **Safe Keeping -** Folder for safe keeping branches
  - **Release 1 -** Safe keeping branch

More Information

- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Always Create a Top-level Folder for a New Team Project to Serve as a Main Branch

Start any new team project by creating a folder, frequently called **Main**, underneath your team project. Put all source for the **Main** branch in this folder. When you need to create a new branch, branch directly from the **Main** folder.

The following example shows a typical branch structure:

- **Development** – Folder for isolated development branches, branched from Main
    - **Feature A**
        - **Source**
    - **Feature B**
        - **Source**
- **Main** – Integration Branch
    - **Source**
    - **Other Asset Folders**
- **Maintenance** – Folder for release branches, branched from Main
    - **Release 1** – Maintenance Branch
        - **Source**

More Information

- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Before Merging Consider Using the Candidate or Preview Switch to Double Check

Prior to performing a merge use the candidate or preview switch to double check your merge results. Although this option is only available from the command line it is extremely beneficial to know which files and which versions will be merged when you perform the merge command. For example, this can help to ensure that the scope of your merge is not significantly greater than you were expecting and to make sure you understand the impact of your merge command. For large merges, the merge report also helps you to divide the work between individuals or teams.

To preview the results of a merge, use the Tf.exe command line with the **merge** command together with the **preview** or **candidate** switches. For example:

Tf merge main/source development/feature/source /preview

The **preview** switch shows a preview of the merge while the **candidate** switch prints a list of all changesets in the source that have not yet been merged into the destination. The list includes the changeset ID that has not been merged and other basic information about that changeset.

More Information

- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

When Renames are Part of the Merge Pay Close Attention to the Path That the Tool Recommends and Make Changes as Appropriate

When renames are part of the merge pay close attention to the path that the tool recommends and make changes as appropriate (all renames are marked as conflicts). The merge algorithm used by Team Foundation Server tries to calculate the best target path when a rename is being merged over. In some cases the default target path is not the desired one so you should always double check prior to committing the merged file.

More Information

- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Be Careful when Resolving Merge Conflicts

Be careful when merging, it is easy to make mistakes that may result in build instability When merging files:

- Double check the code you are merging before committing the merges.

- Check that you can compile the resulting merged file before checking it back into source control.
- Check that the associated unit tests run and succeed before checking the merged file back into source control.
- Make sure you understand the nature of the changes made by another developer. If you are in doubt about merging some lines, talk to the developer who made the other changes so that you are sure of their purpose and to get a second view on the impact of your merge.

Once you are done merging, compile the resulting source and run unit tests to test for major breaks.

More Information

- For more information about resolving merge conflicts, see “Resolving Conflicts” on MSDN at http://msdn2.microsoft.com/en-us/library/ms181432(VS.80).aspx

Check-in the Results of One Merge At a Time

Check in the results of one merge before performing a second merge that involves the same files. Having performed the first merge, compile the code and ensure that the unit tests succeed, and then check-in your pending changes. Then begin the second merge and repeat the process.

This helps to reduce complexity and by keeping the merges separate enables you to undo changes if the need arises.

More Information

- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Build and Run Tests After the Merge and Prior to Checkin

Having performed a merge, make sure that you compile the code and run the associated tests before checking in the merged file. Do this to avoid introducing build instability as a result of your merges.

The results of a merge are initially isolated within your workspace and are not uploaded to the server until you check-in the pending changes.

More Information

- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Check-in and Check-in Policies

- *Only check-in your code when you are ready to share it.*
- *Use shelvesets to backup or share pending changes*
- *Resolve one work item per check-in*
- *Use check in policies to enforce coding standards*
- *Use check in policies to enforce a code quality gate*
- *Detect when a policy has been overridden*
- *Plan to avoid conflicts*

Only Check-in Your Code When You Are Ready to Share it

Only check your updated code into source control when it is fully unit tested and ready to share with the rest of the team. Use shelvesets for intermediate work that is not yet complete.

When you check-in your code, it will be used as part of your next schedule build. Any incomplete code is likely to cause build instability. Also, when you check-in your code, any other developer who does a get latest operation will pick up your changes. If they are incomplete or inadequately tested they will cause problems for your colleague.

More Information

- For more information on check-ins, see “How to: Check In Pending Changes”: http://msdn2.microsoft.com/en-us/library/ms181411(VS.80).aspx

Use Shelvesets to Backup or Share Pending Changes

Use shelvesets to backup files that contain pending changes you are not yet ready to check in. You can also use shelvesets to share code for code review or if you are passing off a development task to another developer. By using shelvesets you can upload your pending changes to the server without checking in partially completed work that could potentially lead to build instability.

To shelve a set of pending changes, first view them, by right-clicking your solution in Solution Explorer and clicking **View Pending Changes**. Select the files you want to shelve and then click **Shelve**. Type a shelveset name, a comment that identifies the purpose of the shelveset and then click **Shelve**.

To retrieve a shelveset, from the **File** menu, point to **Source Control** and then click **Unshelve**. In the **Owner name** box, type the shelveset creator's name (for example, Contoso\JimB or simply JimB) and then click **Find**. In the **Results** pane, select the shelveset you want to unshelve into your workspace, and then click **Details**. If you want to delete the shelveset as you retrieve it from the Team Foundation source control server, deselect the **Preserve shelveset on server** option. Team Foundation restores each shelved revision into the destination workspace as a pending change as long as the revision does not conflict with a change that is already pending in your workspace. Optionally deselect the **Restore work items and check-in notes** option if you do not want to have the work items and check-in notes associated with the shelveset restored. When the **Details** dialog box appears, select the shelveset or shelveset items you want to unshelve into your workspace, and then click **Unshelve**.

More Information

- For more information see “How to: Shelve and Unshelve Pending Changes” at http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

Resolve One Work Item Per Check-in

Having resolved a work item, check-in your pending changes and update the work item status. This practice serves the following purposes:

- It provides a consistent record in the source control database that can be used to review changes per work item or back a work item out if necessary in the future.
- It ensures you don’t go too long between check-ins. The longer you go between check-ins, the more likely that you will have a conflict that requires a manual merge and additional testing.

More Information

- For more information on check-ins, see “How to: Check In Pending Changes”: http://msdn2.microsoft.com/en-us/library/ms181411(VS.80).aspx
- For more items on work items and changesets, see “How to: Associate Work Items with Changesets”: http://msdn2.microsoft.com/en-us/library/ms181410(VS.80).aspx
- For more information on reviewing work item details, see “How to: View Work Item Details from Pending Changes Window”: http://msdn2.microsoft.com/en-us/library/ms245467(VS.80).aspx

Use Check in Policies to Enforce Coding Standards

Use check-in policies to enforce coding standards across your development team. Use them to ensure that all source code checked into Team Foundation Server source control meets your defined coding standards.

The code analysis check-in policies that ship with Team Foundation Server enable you to automatically run FxCop static code analysis on code as it is checked in to ensure that the relevant rules are satisfied. You can fine tune the code analysis policy to check many different rules. For example you can check rules governing design, interoperability, maintainability, mobility, naming conventions, reliability and more.

To enforce a code analysis check-in policy for a team project:

1. Right-click your team project in **Team Explorer**, point to **Team Project Settings** and click **Source Control**.
2. Click the **Check-in Policy** tab and then click **Add**.
3. In the **Add Check-in Policy** dialog box, select **Code Analysis** and click **OK**.
4. In the **Code Analysis Policy Editor**, select either **Enforce C/C++ Code Anlysis (/analyze)** or **Enforce Code Analysis For Managed Code**. Select both if your project contains a combination of managed and unmanaged code.
5. If you select manage code analysis, configure your required rule settings for managed code analysis based on your required coding standards. This determines precisely which rules are enforced.

You can also create a custom check-in policy to perform checks that are not available by default. For example, you can disallow code patterns such as banned API calls, or you can write a policy to enforce your team's specific coding style guidelines, such as where braces should be positions within your source code.

More Information

- For more information about creating and using a custom check-in policy, see "How To: Step Through Creating Custom Check-in Policies for TFS."
- To learn how to customize a check-in policy, see "Walkthrough: Customizing Check-in Policies and Notes", http://msdn2.microsoft.com/en-us/library/ms181281(VS.80).aspx
- To see sample code that will disallow selected patterns on checkin, see Checkin Policy to Disallow Certain Patterns: http://blogs.msdn.com/jmanning/archive/2006/02/02/523125.aspx
- To see sample code that will enforce comments on checkin, see Sample Checkin Policy: Make Sure the Comment Isn't Empty: http://blogs.msdn.com/jmanning/archive/2006/01/21/515858.aspx

- To learn how to register a new checkin policy, see I've Made a New Check-In Policy! How Do I Add It?: http://blogs.msdn.com/jmanning/archive/2006/02/07/526778.aspx

Use Check in Policies to Enforce a Code Quality Gate

Use a combination of code analysis and testing policies to enforce a code quality gate. For example, use the supplied testing policy to ensure that specific tests are executed and passed prior to allowing source to be checked into Team Foundation Server source control. You can also configure a code analysis policy to help ensure that your code meets certain quality standards by ensuring that security, performance, portability, maintainability and reliability rules are passed.

By enforcing this type of check-in policy in addition to policies that enforce coding standards and guidelines, you ensure your code meets a specific code quality gate.

To enforce a code analysis check-in policy for a team project, you right-click your team project in **Team Explorer**, point to **Team Project Settings** and click **Source Control**. Click the **Check-in Policy** tab and then click **Add** and then select and configure the appropriate policy.

More Information

- For more information about creating and using a custom check-in policy, see “How To: Step Through Creating Custom Check-in Policies for TFS.”
- To learn how to customize a check-in policy, see “Walkthrough: Customizing Check-in Policies and Notes”, http://msdn2.microsoft.com/en-us/library/ms181281(VS.80).aspx
- To see sample code that will disallow selected patterns on checkin, see Checkin Policy to Disallow Certain Patterns: http://blogs.msdn.com/jmanning/archive/2006/02/02/523125.aspx
- To see sample code that will enforce comments on checkin, see Sample Checkin Policy: Make Sure the Comment Isn't Empty: http://blogs.msdn.com/jmanning/archive/2006/01/21/515858.aspx
- To learn how to register a new checkin policy, see I've Made a New Check-In Policy! How Do I Add It?: http://blogs.msdn.com/jmanning/archive/2006/02/07/526778.aspx

Detect When a Policy Has Been Overridden

Team Foundation Version Control does not prevent you overriding a policy. However, you can use the following steps to detect if a policy has been overridden:

- Use the Team Foundation Eventing Service (from the Team Foundation Core Services API) for hooking in to check-in events.

- Write a Notify method that parses the details of the changeset and then react to it if an override has occurred

Alternatively, you can manually scan changeset history to discover policy overrides.

More Information

- To learn more about overriding a check-in policy, see "How to: Override a Check-in Policy": http://msdn2.microsoft.com/en-us/library/ms245460(VS.80).aspx
- To learn more about the Team Foundation Eventing Service, see "Eventing Service": http://msdn2.microsoft.com/en-us/library/bb130154(vs.80).aspx
- To learn how to receive an e-mail when a policy has been violated, see: http://blogs.infosupport.com/marcelv/archive/2005/10/18/1635.aspx

Plan to Avoid Conflicts

Plan to avoid having multiple developers working in the same region of the same source file at the same time. Do this to avoid conflicts that are potentially difficult to correctly resolve. While automatic conflict resolution can resolve many conflicts, you should avoid situations where two or more developers are working on the same method, or on the same lines of code. Conflicts on the same lines of code require manual conflict resolution, which complicates the merge operation. Effective team communication is the key.

Before starting work on a file, make sure you have the latest version from source control and check if anyone else has the file checked out before you begin work. If a colleague has the file checked out, ask your colleague what he or she is working on and then decide if you can wait until they complete their changes or if it is safe to continue to work in parallel on the same file because you are working on separate functionality in separate source code regions within the file.

To check if someone else has a file checked out:

1. Double click **Source Control** from the Team Explorer window in Visual Studio.
2. Browse to the folder containing the file you want to check in the source control folder hierarchy. Any pending changes are listed together with the username of the user who owns those changes.

To check which files people currently have pending changes for, run the following command from a Visual Studio 2005 command prompt window.

```
Tf status /format:detailed /user:*
```

When you do begin working on a source file that you know others will work on in parallel, let other team members know you are working on the file and which aspects you will be updating.

More Information

- For more information on viewing pending changes in your workspace, see "How to: View and Manage All Pending Changes in Your Workspace": http://msdn2.microsoft.com/en-us/library/ms181400(VS.80).aspx
- For more information on viewing pending changes in other workspaces, see "How to: View Pending Changes in Other Workspaces": http://msdn2.microsoft.com/en-us/library/ms181401(VS.80).aspx

Checkout, Get, and Lock

- *Get latest source before making changes*
- *Use the Lock command with discretion*
- *Communicate with your teammates when locking files*

Get Latest Source Before Making Changes

To make sure you have the latest versions of all the source files for the project you are working on, run a get latest command before you checkout a file for edit. The danger of not doing so is that you locally build your code against non current source files. This increases the likelihood of your code causing build problems when you subsequently check it into the server.

To get the latest copies of the file set associated with a specific team project, right-click the team project in Source Control Explorer and click **Get Latest Version**. If you currently have writable files with pending edits in your workspace this command does not overwrite those files. You can run the same command from the command line by running the **Tf get /all** command from a directory mapped to the current workspace.

More Information

- For more information about the Get command, see the "Get Command" on MSDN at http://msdn2.microsoft.com/en-us/library/fx7sdeyf(VS.80).aspx

Use the Lock Command with Discretion

Use the **lock** command sparingly because locking a file while you work on it can lead to bottlenecks in the development process by preventing other people working on the same file at the same time. You should only lock a file while editing it if you are concerned that there will be a conflict resulting in a complicated manual merge operation. Where possible, prefer shared checkouts by selecting the lock type **None** when you check out the file.

The following are common scenarios where using a lock is appropriate to avoid conflicts:

- You are modifying a binary file format, such as an image or document, that cannot be easily merged
- You are making such widespread changes to a file that any other developer working on the same file would be very likely to make changes to the same lines as you.

If you do want to avoid any possibility of conflicts, use a check-out lock which prevents other people from checking out and checking in the same file. If you do lock a file, notify your teammates when you lock a file and tell them why you are doing so and how long your work on the file is likely to take. Also let them know when you remove the lock and check your changes back in.

You specify your lock type when you check out a file for edit or you can explicitly lock a file. To lock a file while checking it out, from Source Control Explorer, right-click the file and select **Check Out for Edit**. Then specify your lock type; **None**, **Check Out** or **Check In**. To explicitly lock a file, right-click the file and click **Lock**. Then choose the **Check Out** or **Check In** lock type.

Note that unlike Visual Source Safe (VSS) behavior, checking out a file does not implicitly get the latest version. Prior to locking a file make sure you have the latest version in your workspace by clicking **Get Latest Version**.

More Information

- For more information about locks, see “How to: Lock and Unlock Folders or Files”: http://msdn2.microsoft.com/en-us/library/ms181420(VS.80).aspx
- For more informational about lock types, see: http://msdn2.microsoft.com/en-us/library/ms181419(VS.80).aspx

Communicate With Your Teammates When Locking Files

If you do lock a source file, let your team mates know so that they can plan their work around the unavailability of the file. Explain why you need an exclusive lock on the file and how long you are likely to retain the lock. By locking a file, you potentially introduce a bottleneck into your development cycle, if other developers need to work on the same source file.

Only lock a file while editing it if you are concerned that there will be a conflict resulting in a complicated manual merge operation. Where possible, prefer shared checkouts by selecting the lock type **None** when you check out the file.

To lock a file from Source Control Explorer, right-click the file and click **Lock**. Then choose the **Check Out** or **Check In** lock type.

More Information

For more informational on lock types, see: http://msdn2.microsoft.com/en-us/library/ms181419(VS.80).aspx

Distributed / Remote Development

- *Make sure you get an appropriate sized disk drive for your proxy*
- *Create a scheduled task to pull the latest files on a periodic basis.*
- *Monitor proxy performance counters and event log periodically.*
- *Configure executionTimeout based on file sizes and bandwidth*
- *Don't add an AT to Proxy.Config, if that AT is going to be used only for a short period of time*
- *Disable the proxy if the proxy is going to be down for an extended period of time*
- *Consider workspace cloaking to reduce unnecessary file transfers*

Make Sure You Get an Appropriate Sized Disk Drive for Your Proxy

Follow hardware recommendations in TFS install guide; specifically make sure you get a disk drive with sufficient capacity. A maximum limit is set on the amount of storage space the proxy can use for caching files. When this limit is reached, old files in the cache are deleted to free up some storage space so that the space can be used for caching newly requested files. Cleanup removes the files based on when they were last accessed. Files that have not been served for the longest time are deleted first.

More Information

- To learn more about the Team Foundation Server Proxy, see MSDN: Team Foundation Server Proxy and Source Control, http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx

Create a Scheduled Task to Pull the Latest Files on a Periodic Basis

Run a scheduled task to retrieve the latest files on a periodic basis to the proxy server. This helps to ensure that the latest version of the files are available in the proxy cache and to ensure that subsequent client requests for these files result in a cache hit.

More Information

- To learn more about the Team Foundation Server Proxy, see MSDN: Team Foundation Server Proxy and Source Control, http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx

Monitor Proxy Performance Counters and the Event Log Periodically

Monitor Proxy Performance counters and the Windows event log for errors and warnings on a periodic basis, to get a good picture of how your proxy is functioning. Make sure proxy caching is enabled and monitor the performance of your cache.

You should observe the following performance counters:

- Current Cache Size
- Total Cache Hits - count and percentage
- Total Download Requests
- Total Files in Cache
- Total Cache Miss - count and percentage

These performance counters are registered when you install the proxy. The proxy performance counters are multi-instanced, which means there is a set of counters for each application-tier that you have configured in the Proxy.config file. By collecting this data, you get an understanding of the performance when the proxy server is running.

Note that the TFS proxy saves cache performance statistics to an XML file called ProxyStatistics.xml and you can change the interval for saving these statistics. The ProxyStatistics.xml file is located in the App_Data folder in the proxy installation directory.

More Information

- To learn more about the Team Foundation Server Proxy, see MSDN: Team Foundation Server Proxy and Source Control, http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx

Configure executionTimeout Based on File Sizes and Bandwidth

If you know that large files are going to be downloaded over a low bandwidth (< 3Mbps) network, then set the **executionTimeout** configuration to an appropriate value in Web.config. Do this to reduce the likelihood of timeouts. The default value is one hour as shown here:

```
<httpRuntime executionTimeout="3600"/>.
```

More Information

- To learn more about the Team Foundation Server Proxy, see MSDN: Team Foundation Server Proxy and Source Control, http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx

Disable the Proxy if it is Going to be Down for an Extended Period of Time

Disable the proxy on the clients if the proxy is going to be down for an extended period of time. Do this to prevent fruitless reconnections. By default, these are attempted every 5 minutes.

More Information

- To learn more about the Team Foundation Server Proxy, see MSDN: Team Foundation Server Proxy and Source Control, http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx

Consider Workspace Cloaking to Reduce Unnecessary File Transfers

Consider using workspace cloaking to hide specific workspaces from being viewed and to prevent unnecessary file transfers. Cloaking hides specified workspace folders from view and also increases performance bandwidth and conserves local disk space by preventing folders and files that you do not currently require from being copied to your local workspace. Although you can cloak an existing folder mapping in your workspace, a better approach is to create a new folder mapping specifically intended to be cloaked.

To cloak folders in a workspace:

1. In Visual Studio, from the **File** menu, point to Source Control, and then click **Workspaces**.
2. In the **Manage Workspaces** dialog box, select the workspace you want to apply cloaking to, and then click **Edit**.
3. In the **Edit Workspace** dialog box, in the **Working folders** list, either highlight the folder mapping located under **Source Control Folder** and **Local Folder** that you want to cloak, or create a new one.
4. Click in the **Status** column and change the setting from **Active** to **Cloaked**. Note that you can only cloak folders in an actively mapped Team Foundation source control folder.
5. Click **OK** to close **Edit Workspaces** and **Close** to close **Manage Workspaces**.

Note that your files do not disappear locally until you perform another get for the whole workspace.

More Information

- To learn more about the Team Foundation Server Proxy, see MSDN: Team Foundation Server Proxy and Source Control, http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx

Migration

- *Use the VSS converter to migrate to Team Foundation Source Control*
- *Migrate from other source control systems to Team Foundation Source Control*

Use the VSS Converter to Migrate to Team Foundation Source Control

Team Foundation Server ships with a VSS converter which allows you to migrate files, folders, version history, labels, and user information from a Visual SourceSafe database to Team Foundation Server Version Control.

The converter does have some limitations that are important to understand, for example:

1. Historical information about file sharing is not preserved. Team Foundation Server does not support sharing. The migration of a shared file results in copying the file to the destination folder.
2. Historical information about branching is not preserved.
3. Team Foundation Server does not support pinning. Pinned files are migrated by creating two labels.
4. Time stamps associated with actions are not preserved during migration.

More Information

- For more information about migrating files, see “How To: Migrate Source Code From VSS to TFS.”

Migrate from Other Source Control Systems to Team Foundation Source Control

You can manually export files from the version control system you are migrating from and then import them into Team Foundation Server Version Control. If you need to preserve file history or other attributes from the version control system you are migrating from, you can use the TFS Migration and Synchronization Toolkit to write your own migration tool.

Microsoft is currently working on a ClearCase converter, when it is ready it will be announced on the TFS Migration blog: http://blogs.msdn.com/tfs_migration

Component software has created a converter that is compatible with GNU RCS, CS-RCS, GNU CVS, Subversion (SVN) and Visual SourceSafe (VSS)

More Information

- For more information on TFS migration, see the TFS Migration blog on MSDN: http://blogs.msdn.com/tfs_migration

- You can download the TFS Migration and Synchronization Toolkit from CodePlex: http://www.codeplex.com/MigrationSyncToolkit
- For more information on the component software converter, see: http://www.componentsoftware.com/Products/Converter/

Project / Workspace Management

- *To isolate a single developer use workspaces not branches*
- *Delete and rename files by using source control, not Windows Explorer.*
- *Only delete and rename with your solution open*
- *Create one team project per application if you want to move your assets between application versions*
- *Create one team project per version if you want to start fresh with each application version*
- *Use branching to share code and binaries that require integration testing*
- *Avoid workspace mapping to support cross-project dependencies*
- *Create workspace mappings at the team project root level*
- *On shared computers, use a unique local folder path*
- *Consider mapping only part of the source tree*
- *Structure your source tree to support branching*

To Isolate a Single Developer Use Workspaces Not Branches

Use an additional workspace and do not create a new branch to isolate your work from the rest of your team. Use your primary workspace to contain references to the files and folders being worked on by the rest of the team (i.e. containing the shared source) and create a second workspace to maintain the files and folders that you want to isolate. You might want to isolate these files in order to evolve specific files in parallel with work that is happening elsewhere. For instance, this can be used to work on risky pending changes, or experimental changes. By using a second workspace, you reduce additional branch and merge complexity.

To create a secondary workspace:

1. Click the **Workspace** drop down within Source Control Explorer and then click **Workspaces**.
2. In the **Manage Workspaces** dialog box, click **Add**.
3. In the **Add Workspace** dialog box, enter a new workspace name such as **MyIsolatedWork** and provide a comment to serve as a future reminder about the purpose of the workspace.
4. In the **Working folders** list, set the workplace status to **Active**, identify the source control folder to be included in the workspace (this can be the team

project root folder or any sub folder) and specify a local folder path on your own computer to contain the files from the workspace.
5. Click **OK** and then **Close** to create the isolated workspace.

To retrieve the latest set of source to begin work in your isolated workspace:

1. In source control explorer, make sure that your isolated workspace name is selected from the Workspace drop down list.
2. Select your team project root folder, or a sub folder if you only need part of the source tree, right-click and then click **Get Latest Version**. This copies the folder structure and latest file set from the source control server to the local directory on your computer which you mapped to the new workspace.

More Information

- For more information on creating a workspace, see "How to: Create a Workspace": http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx
- For more information on editing a workspace, see "How to: Edit a Workspace": http://msdn2.microsoft.com/en-us/library/ms245466(VS.80).aspx

Delete and Rename Files by Using Source Control, not Windows Explorer

If you need to delete or rename files that have been added to source control, delete or rename them by using the Source Control Explorer or by using Tf.exe from the command line. Do not delete or rename files by using Windows Explorer because doing so forces your local workspace files to become out of synchronization with the source control files maintained on the server.

To delete a file or folder by using Source Control Explorer:
1. Select the file or folder in Source Control Explorer.
2. Right-click and click **Delete**. An icon that indicates the item is deleted appears to the left and the status delete appears under the **Pending Change** column. The item is deleted the next time you check the file in.

Note: You cannot delete an item for which another pending change exists. For example, a checked out file cannot be deleted.

The **Tf move**, **delete** and **rename** commands are particularly useful if you need to perform the operation on multiple files or folders. With Source Control Explorer you can only move, rename or delete a single file or folder at a time.

More Information

- For more information about the Tf delete command, see "Delete Command" on MSDN at http://msdn2.microsoft.com/en-us/library/k45zb450(VS.80).aspx

- For more information about the Tf rename command, see "Rename Command" on MSDN at http://msdn2.microsoft.com/en-us/library/a79bz90w(VS.80).aspx

Only Delete and Rename with Your Solution Open

Delete and rename files inside Solution Explorer with your solution open. Do not do this directly on disk. This approach ensures that when you next check-in your pending changes, Team Foundation Server Source control is kept in synchronization.

More Information

- For more information about the Tf delete command, see "Delete Command" on MSDN at http://msdn2.microsoft.com/en-us/library/k45zb450(VS.80).aspx
- For more information about the Tf rename command, see "Rename Command" on MSDN at http://msdn2.microsoft.com/en-us/library/a79bz90w(VS.80).aspx

Create One Team Project Per Application if You Want to Move Your Assets Between Application Versions

If you want to carry forward not only source code but work items and other Team Foundation Server assets between releases, consider using one team project per application. When you use a single team project for multiple versions of the application, all of the Team Foundation Server assets are carried forward automatically for you for the next release. When you are ready to release a new version of your application you can create a branch within the project to represent the release and isolate that code.

When using one project per application, keep the following in mind:

- Releases in parallel are forced to share work items schema, check in policies, and process guidance.
- Reporting is harder since reports default to the entire project, you must add filtering by release.
- If you have hundreds of applications, each in its own project, you'll hit Team Foundation Server performance and scale limits.
- You will accumulate 'baggage' over multiple releases. The easiest way to address this issue is to create a new project and branch code you want to carry forward into that project.

More Information

- For more information on using team projects, see “When to use Team Projects”: http://blogs.msdn.com/ericlee/archive/2006/08/09/when-to-use-team-projects.aspx

Create one Team Project per Version if You Want to Start Fresh With Each Application Version

If you want each release to start fresh without carrying forward work items and other Team Foundation Server assets, consider using one project per release. When you use a new project for each release you can modify work item schema, workflow, check-in policies, and other assets without impacting the old release. This can be especially useful if the old release will be maintained by a separate team such as a sustained engineering team who may have a different process and workflow than your main development team.

When using one project per release keep the following in mind:

- While it is very easy to move the source code from one project to another, it is difficult to move work items and other TFS assets from one project to another. Work items can only be copied one at a time to another project, if you want to copy sets of work items you'll need to write your own utility.
- If you have hundreds of applications and releases, each in its own project, you'll hit Team Foundation Server performance and scale limits.
- Pick a structure you can live with, restructuring existing team projects is difficult.
- Source can be easily shared amongst team projects:
    - Branch source from one project to another
    - Map source from another project into your workspace
- Team Foundation Serveer can scale to ~500 projects using MSF Agile process or 250 projects using MSF CMMI process. If you create your own or customize an existing process keep in mind that work item schema has the largest impact on server scalability. A complex schema will result in a server capable of supporting fewer projects.

More Information

- For more information on using team projects, see "When to use Team Projects": http://blogs.msdn.com/ericlee/archive/2006/08/09/when-to-use-team-projects.aspx

Use Branching to Share Code and Binaries that Require Integration Testing

Managing shared code or binaries involves two steps:

1. Determine a location to store the dependency
2. Branch the dependency into your project

1. **Determine a location to store the dependency**
   Options for storing:
    - If the shared dependency is clearly owned by a particular team, store it in their team project.

- If the shared dependency has no clear ownership, then create a team project specifically for the shared code.

2. **Branch the dependency into your project**
   Once you have stored the dependency branch the shared source or binaries into your project. Every time you perform a merge from shared to consuming project you will pick up latest source. This allows you to pick up changes on a regular schedule and perform integration testing before impacting your main source tree.

More Information

- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

Avoid Workspace Mapping to Support Cross-Project Dependencies

In general you should avoid dependencies that cross team projects and try to have all the related/dependent solutions/projects under same team project. This reduces the need for build script customization. If you have a dependency, use project references to define it or branch the dependency from a shared project into your project. You should avoid file references as they are more difficult to manage. The exception is when the dependent project is developed in parallel and you need real-time changes. In this case, you can consider the using workspace mapping. You may still consider branching to create a buffer of isolation, if the dependent code is causing too many breaking changes

More Information

- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx
- For more information on editing a workspace, see “How to: Edit a Workspace”: http://msdn2.microsoft.com/en-us/library/ms245466(VS.80).aspx

Create Workspace Mappings at the Team Project Root Level

For new team projects, map your team project root ($/MyTeamProject) to a folder with the same name on your local drive for example C:\TeamProjects. Because mappings are recursive, your entire local folder structure is created automatically and will be exactly the same as the structure in source control.

More Information

- For more information on creating a workspace, see "How to: Create a Workspace": http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx
- For more information on editing a workspace, see "How to: Edit a Workspace": http://msdn2.microsoft.com/en-us/library/ms245466(VS.80).aspx

On Shared Computers, Use a Unique Local Folder Path

Two users of a single computer cannot share the same workspace mapping. For example, you and a colleague cannot map the same team project ($/MyTeamProject) to the same folder on the local computer. Instead, create mappings beneath My Documents (although this leads to long location paths) or establish a team folder naming convention on the local computer (e.g. C:\TeamProjects\User1, C:\TeampProjects\User2 etc).

More Information

- For more information on creating a workspace, see "How to: Create a Workspace": http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx
- For more information on editing a workspace, see "How to: Edit a Workspace": http://msdn2.microsoft.com/en-us/library/ms245466(VS.80).aspx

Consider Mapping Only Part of the Source Tree

To improve performance and reduce disk size requirements only map the files you need for your development project. In general you will only need the files and projects associated with the solution you will be working on.

More Information

- For more information on creating a workspace, see "How to: Create a Workspace": http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx
- For more information on editing a workspace, see "How to: Edit a Workspace": http://msdn2.microsoft.com/en-us/library/ms245466(VS.80).aspx

Structure Your Source Tree to Support Branching

Your source tree structure consists of a combination of folder structure, file structure, and branch structure. Within your main branch the following folder and file structure has been proven to work for teams of various sizes:

- **Main** - container for all assets you need to ship the product
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code** – container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests
        - **Lib** - container for binary dependencies

    - **Docs** - container for documentation that will ship with the product
    - **Installer** - container for installer source code and binaries
    - **Builds** - container for team build scripts
    - **Tests** - container for test team test cases

Any branches that you create off of main will copy this folder and file structure into the new branch. For example:

- **Development** – Development branch
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code** – container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests
        - **Lib** - container for binary dependencies
- **Main** – Integration branch
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code** – container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests
        - **Lib** - container for binary dependencies
    - **Docs** - container for documentation that will ship with the product
    - **Installer** - container for installer source code and binaries
    - **Builds** - container for team build scripts
    - **Tests** - container for test team test cases

More Information

- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx
- For more information on editing a workspace, see “How to: Edit a Workspace”: http://msdn2.microsoft.com/en-us/library/ms245466(VS.80).aspx

Shelving

- *Use shelving to share pending changes for review or hand off*
- *Use shelving to backup pending changes to the server*
- *Use shelving if interrupted by higher priority work*

Use Shelving to Share Pending Changes for Review or Hand Off

Use shelving if you want to discuss or review your work-in-progress code with a remote team member. Rather than emailing the code to them, you can shelve the code and then have your remote colleague retrieve the files from the shelf.

Likewise if you want to pass partially finished work to another developer for completion you can use shelving to hand the code off.

More Information

- For more information on shelving pending changes, see “How to: Shelve and Unshelve Pending Changes”: http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

Use Shelving to Backup Pending Changes to the Server

Use shelving if you have not completed work at the end of the day but want to ensure that your current work is backed up on the server. By shelving your current changes, the changes are applied to the TFS and can be retrieved by you (or others) the day.

More Information

- For more information on shelving pending changes, see “How to: Shelve and Unshelve Pending Changes”: http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

Use Shelving if Interrupted by Higher Priority Work

Use shelving if you are mid-way through making changes to a set of source code, when new higher priority work is allocated (for example an emergency bug fix is required). At this point you need to go back to a stable version of the code but you do not want to lose your changes. Before doing so you can shelve your code and easily retrieve it later on.

More Information

- For more information on shelving pending changes, see “How to: Shelve and Unshelve Pending Changes”: http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

Additional Resources

- Visual Studio Team System Guidance Project: http://www.codeplex.com/VSTSGuidance

Practices at a Glance: Source Control

Index

Accessing Version Control

- *How to use version control from non-Visual Studio clients*
- *How to automate common version control tasks*
- *How to work off-line*

Administration

- *How to add a new developer to your project*
- *How to remove a developer who has left your project*
- *How to grant permissions within your source tree*

Branch/Label/Merge

- *How to use labels*
- *How to branch*
- *How to plan your branching structure*
- *How to use branching to support a release*
- *How to use branching to maintain a previous release*
- *How to use branching to stabilize your development and build process*
- *How to use branching to stabilize feature development*
- *How to use branching to stabilize development across teams*
- *How to use branching to isolate external dependencies*
- *How to retire an old release*
- *How to perform a merge*
- *How to perform a baseless merge*
- *How to resolve merge conflicts*

Builds

- *How to use TFS to perform continuous integration builds*
- *How to build and run tests after a merge*

Check-In and Check-In Policies

- *How to work with source control change sets*
- *How to enforce coding standards prior to check-in*
- *How to override a check-in policy*
- *How to undo a check-in*
- *How to resolve a conflict*
- *How to avoid conflicts*

- *How to create a custom check-in policy*

Checkout, Get and Lock

- *How to synchronize your computer with TFS*
- *How to prepare a file for editing*

Code Sharing

- *How to share code*
- *How manage shared binaries*

Distributed/Remote Development

- *How to access TFS over the Internet*
- *How to support a remote office*

Migration

- *How to migrate your source from Visual SourceSafe*
- *How to migrate your source from other version control systems*
- *How to move Team Foundation Version Control to another server*

Project/Workspace Management

- *How to choose one team project vs multiple team projects*
- *How to organize your source tree*
- *How to define workspace mappings*
- *How to use workspaces to isolate code changes on your computer*

Security

- *How to secure the channel between developer workstation and TFS*

Shelving

- *How to use shelving to backup pending work*
- *How to use shelving to share code with a team member*

Accessing Version Control

- *How to use version control from non-Visual Studio clients*
- *How to automate common version control tasks*
- *How to work off-line*

How to Use Version Control from Non-Visual Studio Clients

You can access Team Foundation Version Control from other clients by using one of the following approaches:

- MSSCCI Integration
- Third-party Integration
- Custom Integration

MSSCCI Integration

The following clients can work directly with Team Foundation Version Control by using the MSSCCI provider:

- Visual Studio NET 2003
- Visual C++ 6 SP6
- Visual Visual Basic 6 SP6
- Visual FoxPro 9 SP1
- Microsoft Access 2003 SP2
- SQL Server Management Studio
- Sparx Systems Enterprise Architect 61
- Sybase PowerBuilder 105
- Toad for SQL Server 20

The MSSCCI provider behaves differently than Team Foundation Version Control in Visual Studio 2005 in the following ways:

- Checkout also performs a GetLatest
- Exclusive check-in lock applied at checkout
- Open-from source control and save-to source control behave like Visual SourceSafe

You can download the MSSCCI provider from MSDN:
http://www.microsoft.com/downloads/detailsaspx?FamilyId=87E1FFBD-A484-4C3A-8776-D560AB1E6198&displaylang=en

The MSSCCI provider is not supported by Microsoft. Use the MSDN forums if you have questions:
http://forums.microsoft.com/MSDN/ShowForumaspx?ForumID=22&SiteID=1

Third-party Integration

The following clients have integration solutions provided by other vendors:

- Eclipse
- Linux client

- Mac client
- HTML Web client

If you want access to Team Foundation Version Control from the Eclipse IDE, Linux, or Mac, consider installing Teamprise: http://www.teamprise.com/

If you would like read-only access to Team Foundation Version Control from the Web, consider installing TeamPlain Web Access: http://www.devbiz.com/teamplain/

Custom Integration

Other clients have no integration solution currently available. You can either access Team Foundation Version Control from the command line or build your own integration solution

To learn more about working with Team Foundation Version Control, see MSDN: Walkthrough: Working with Team Foundation Source Control from Command Line: http://msdn2.microsoft.com/en-us/library/zthc5x3faspx

You can use control scripts and command files to automate the use of the command line To learn more about working with control scripts and command files, see MSDN: Team Foundation Source Control Scripts and Command Files: http://msdn2.microsoft.com/en-us/library/1az5ay5c(VS80)aspx

Additional Resources

- You can download the MSSCCI provider from MSDN: http://www.microsoft.com/downloads/detailsaspx?FamilyId=87E1FFBD-A484-4C3A-8776-D560AB1E6198&displaylang=en
- If you want to access Team Foundation Version Control from the Eclipse IDE, Linux, or Mac, consider installing Teamprise: http://www.teamprise.com/
- If you would like to access Team Foundation Version Control from the Web, consider installing TeamPlain Web Access: http://www.devbiz.com/teamplain/
- To learn more about working with Team Foundation Version Control, see “Walkthrough: Working with Team Foundation Source Control from Command Line”: http://msdn2.microsoft.com/en-us/library/zthc5x3faspx
- To learn more about working with control scripts and command files, see “Team Foundation Source Control Scripts and Command Files”: http://msdn2.microsoft.com/en-us/library/1az5ay5c(VS80)aspx
- You can download the Visual Studio Team Foundation Server SDK from http://go.microsoft.com/fwlink/?linkid=68586
- To learn more about Team Foundation Version Control extensibility, see “Walkthru:The Version Control Object Model”: http://msdn2.microsoft.com/en-us/library/bb187335(VS.80).aspx

How to Automate Common Version Control Tasks

To automate common version control tasks, use the Team Foundation command line tool (TF.exe). With this tool you can do everything that you can do with Source Control Explorer including source control operations (add, checkin, checkout, get, lock, label and more), branching, shelving, workspace manipulation and general administration tasks.

The main reasons for using the command line tool include automating repetitive operations and scheduling operations to run at specific times or on specific events by using the Windows Task Scheduler. The following commands are also only available from the command line: deleting another user's workspace, undoing another user's check-out, unlocking another user's lock, defining label scope and performing a baseless merge.

To ensure the appropriate path and other environment variables are setup, run the command line tool from the Visual Studio 2005 Command Prompt window or run the Vsvars32 batch file, which is normally located in DriveLetter:\Program Files\Microsoft Visual Studio 8\Common7\Tools.

Tf.exe is installed as part of the TFS client and is located by default in the following folder:
C:\Program Files\Microsoft Visual Studio 8\Common 7\IDE.

To run the command line tool, you must specify a server name with the /s switch. The following command shows how to view the files in source control on the server named YourTFSServer.

```
tf.exe dir /s:YourTFSServer
```

Additional Resources

- MSDN: Walkthrough: Working with Team Foundation Source Control from Command Line: http://msdn2.microsoft.com/en-us/library/zthc5x3f.aspx
- MSDN Team Foundation Source Control Command-Line Reference: http://msdn2.microsoft.com/en-us/library/cc31bk2e(VS.80).aspx

How to Work Off-line

Offline working is not supported natively by Team Foundation Version Control.

To work off-line, you need to use the following strict workflow:

1. **Manually remove read-only flag**
   By default, all files in the workspace that have not been checked out are marked as read-only. When you work without a server connection, you must manually remove the read-only flag from files before editing or deleting them. To do this, right-click the file in Windows Explorer, click **Properties**, uncheck the **Read-only** check box and click OK. Alternatively, you can use the DOS command “attrib -r”
2. **Edit files**
   You can now edit any files for which you have removed the read-only flag.

3. **Add or delete files**
   You can add or delete files for which you have removed the read-only flag. Do not rename files, because the TFPT **online** tool cannot distinguish a rename from a deletion paired with an add. Note that you have to specify an option to the Tfpt **online** command to get it to look for deletions, as this is a more time consuming operation.
4. **Run the TFPT online command**
   When you are back online, run the TFPT online command by typing "TFPT online" at the command line. This command scans your workspace for writable files and determines what changes should be pended on the server. If you have deleted any files, use the /**delete** switch. This tells the tool to scan for deleted files in your local workspace as well. The tool then displays the online window from which you can choose which changes to pend into the workspace.

Important: You must not rename any files while you're off-line

Additional Resources

- You can download the TFPT tool from MSDN: http://www.microsoft.com/downloads/details.aspx?FamilyID=7324C3DB-658D-441B-8522-689C557D0A79&displaylang=en
- To learn more about Team Foundation Power Tool, see “Power Toy: tfptexe”: http://blogs.msdn.com/buckh/archive/2005/11/16/493401.aspx

Administration

- *How to add a new developer to your project*
- *How to remove a developer who has left your project*
- *How to grant permissions within your source tree*

How to Add a New Developer to Your Project

To add a new developer to your project, grant them appropriate access to the team project and associated SharePoint project site. To enable the developer to view reports, grant the developer's account access to SQL Server Reporting Services.

To grant access to the team project:

1. Logon to Visual Studio with an account that is a member of the Team Foundation Administrators application group.
2. Add the required project to Team Explorer (if it is not already listed)
3. Right-click the team project, point to **Team Project Settings** and click **Group Membership**.
4. Select **Project\Contributors** and click **Properties** and then add the new developer's account to this group.

Note that membership of the contributors group provides the typical set of permissions that a developer requires including the ability to add, modify and delete items within the team project and perform builds.

To grant access to the Sharepoint project site:

1. Access the team project site with an account that is a member of the SharePoint Administrator site group. The project site for a project called YourProject is located by default at http://server/sites/YourProject/default.aspx
2. Click **Site Settings**.
3. Click **Manage Users** beneath the **Administration** title.
4. Click **Add Users**.
5. Enter the account name in the form domain\useraccount of the new developer's account, select **Contributor** and then click **Next**.
6. Enter the developer's email address, and optionally type a message to welcome them to the site.
7. Click **Finish**.

Note that membership of the contributors group provides the typical set of permissions that a developer requires including the ability to add, modify and delete items within the team project and perform builds. If you need to restrict access only to specific Visual Studio solutions or to specific folders in your team project, you can also assign permissions at the folder or file level.

To grant access to SQL Server Reporting Services

1. Logon to SQL Server Reporting Services administration Web site using an administrator account. The site is located at http://server/reports.
2. Click your team project name.
3. Click the **Properties** tab.
4. Click the **Security** tab.
5. Click **New Role Assignment**.
6. Enter your developer's Windows account name, select **Browser** and then click **OK**.

Note that membership of the Browser group enables the developer to view and subscribe to reports.

Additional Resources

- For more information, see "Team Foundation Server Default Groups, Permissions, and Roles" on MSDN at http://msdn2.microsoft.com/en-us/library/ms253077.aspx
- For more information, see "MSDN: Source Control Security Rights and Permissions" at http://msdn2.microsoft.com/en-us/library/ms181761.aspx

How to Remove a Developer Who Has Left Your Project

If a developer has left your project, make sure that you delete the developer's workspace. This operation also removes any of the developer’s pending changes and undoes any locks held by the developer. Note, you cannot undo the locks without undoing the changes if exclusive lock is turned on for the team project.

To find out which files the developer has locked out run the following command:

```
tf workspaces /owner:domain\devuser /computer:*
/server:servername
```

To delete the workspace, and remove locks, run the following command:

```
Tf workspace /delete workspacename;domain\devuser
/s:servername
```

Next, remove the developer's account from the security groups. To do so, make changes in the following three places:

- **TFS Team Project –** Logon to Visual Studio with an account that is a member of the Team Foundation Administrators application group. Using Team Explorer right-click the project, point to **Team Project Settings** and click **Group Membership**. Remove the developer's account from the relevant groups (normally **Contributors**)
- **SharePoint Team Project Site –** Logon to the team site located at http://server/sites/yourprojectname/default.aspx with an administrator account. Click **Site Settings**, **Manage Users** and then remove the developer account.
- **SQL Server Reporting Services** - Logon to SQL Server Reporting Services administration Web site using an administrator account. The site is located at

http://server/reports. Click your team project name, click the **Properties** tab, and then click the **Security** tab. Delete the developer's account.

Additional Resources

- For more information on cleaning up after a developer who has left, see “How to: Clean Up Files When Users Leave”: http://msdn2.microsoft.com/en-us/library/ms194958(VS.80).aspx
- For more information on the Workspace command, see “Workspace Command”: http://msdn2.microsoft.com/en-us/library/y901w7se(VS.80).aspx
- For more information on finding pending changes, see “How do I tell who has files checked out or locked?”: http://blogs.vertigosoftware.com/teamsystem/archive/2006/07/24/3125.aspx

How to Grant Permissions within Your Source Tree

You can set permissions on a file or folder within your source tree by right-clicking it in **Source Control Explorer**, clicking **Properties**, clicking the **Security** tab, selecting the user or group for which you want to change permissions, and editing the permissions. You can also set permissions by using the Permission switch with the Tf.exe, command-line utility for source control.

You can apply source control permissions to specific folders and files but by default permissions within your source tree are inherited from the permissions applied to the project folder. If your developers are members of the Project\Contributors application group, they will be able to read, check out, check in, label and lock source files. If you need to provide restricted access to a subset of folders or source files within your team project, for example to enable a developer only to work on certain source files within your team project, you can set permissions at the folder or file level.

Additional Resources

- For more information, see "Team Foundation Server Default Groups, Permissions, and Roles" on MSDN at http://msdn2.microsoft.com/en-us/library/ms253077.aspx
- For more information, see "MSDN: Source Control Security Rights and Permissions" at http://msdn2.microsoft.com/en-us/library/ms181761.aspx
- For more information about the Tf permission command, see “Permission Command”: http://msdn2.microsoft.com/en-us/library/0dsd05ft(VS.80).aspx

Branch/Label/Merge

- *How to use labels*
- *How to branch*
- *How to plan your branching structure*
- *How to use branching to support a release*
- *How to use branching to maintain a previous release*
- *How to use branching to stabilize your development and build process*
- *How to use branching to stabilize feature development*
- *How to use branching to stabilize development across teams*
- *How to use branching to isolate external dependencies*
- *How to retire an old release*
- *How to perform a merge*
- *How to perform a baseless merge*
- *How to resolve merge conflicts*

How to Use Labels

Use labels to group a set of files and folders together for future operations. You can use labels for branching, merging, diffing, or getting files. A label provides a marker to which you can return when performing one of the above operations at a later date.

To apply a label to a file or folder, right-click it in Source Control Explorer and click **Apply Label**. From the **Choose Item Version** dialog you can modify your choice of file or folder, choose the version of the file or folder that you would like to label, and then click **OK** to apply the label.

When applying labels, consider the following:

- Labels are markers that you can apply to only one version of a file or folder at a time.
- You can apply multiple labels to a version of a file or folder.
- Labels that you apply by using Source Control Explorer are automatically scoped to the root folder of the team project within which they are created. You cannot create two labels with the same name within the same scope.
- Labels are un-versioned objects and there is no history associated with them.
- Labels are applied instantly, and they do not require a check-in
- Team Build automatically assigns a label to the set of files associated with each build that it creates.
- Labels are not applied to deleted items, this means that merge-by-label will not pick up deleted files.

To locate an existing label, from the **File** menu, point to **Source Control**, **Label** and click **Find Label**. Having found the label you can modify it or delete it from within the **Find Label** dialog box.

Additional Resources

- For more information, see “Working with labels”: http://msdn2.microsoft.com/en-us/library/ms181439(VS.80).aspx
- For more information on labels, see “How to: Apply Labels”: http://msdn2.microsoft.com/en-us/library/ms181440(VS.80).aspx

How to Branch

To create a branch, either use the **Source Control Explorer**, or use the Tf **branch** command from the command line.

To branch from **Source Control Explorer**, right-click the top level folder containing your project source and click **Branch**. Then specify a target folder location with a folder name to indicate the purpose of the branch, for example **MyProject_Release1.0_Branch**.

To branch from the command line use the **Tf branch** command from a Visual Studio 2005 Command Prompt. For example:

```
Tf branch C:\MyProject $/MyProject_Release1.0_Branch
```

Use branches only when you need to provide isolation while you do parallel development. Merging has overhead and requires you to manage conflicts, don’t branch unless you need the file isolation it provides. You can label a build and branch later if needed.

Additional Resources

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How to Plan Your Branching Structure

Branches are a mechanism to enable isolation; they are used to allow multiple developers to work on the same files in isolation. To plan your branching approach, evaluate common branching patterns and pick a strategy based upon team size, composition, and build stability requirements.

Common reasons to branch include:

- **Release** – Branch to stabilize code you want to release. You can branch before release, avoiding the need to lockdown the main branch.
- **Maintenance –** Branch to maintain a previously released build.
- **Feature –** Branch to isolate feature development that might cause instability to the rest of the project.
- **Team –** Branch to isolate sub teams so they can work without being subject to breaking changes or can work towards unique milestones. These are very similar to feature branches.

Physical Structure

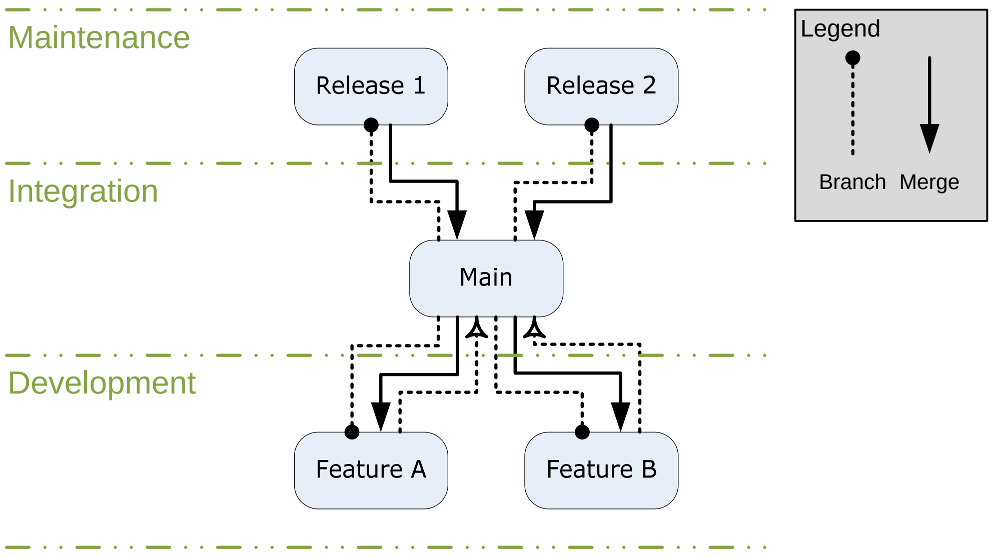
The physical structure is the view you will see in the Source Control Explorer in Visual Studio. The following example shows the use of features branches to isolate development, a main branch to integrate changes and produce a scheduled integration build, a release branch that is being used to stabilize for an upcoming release, and older maintenance branches that can be used to support previous releases.

- **My Project**
    - **Development –** Container to isolate active development branches
        - **Feature A –** Branch for isolated development
            - **Source**
        - **Feature B –** Branch for isolated development
            - **Source**
    - **Main** – Main integration and build branch. All changes come together here.
        - **Source**
        - **Other Asset Folders**
    - **Release 3 –** Branch containing the code currently being locked down to release
        - **Source**
    - **Maintenance –** Container for maintenance branches
        - **Release 1** – Active maintenance branch
            - **Source**
            - **Other Asset Folders**
        - **Release 2** – Active maintenance branch
            - **Source**
            - **Other Asset Folders**

Note: Each branch will have source files and folders under it. In the diagram above, this is labeled Source – it is the set of source files that were branched out of Main.

Logical Structure

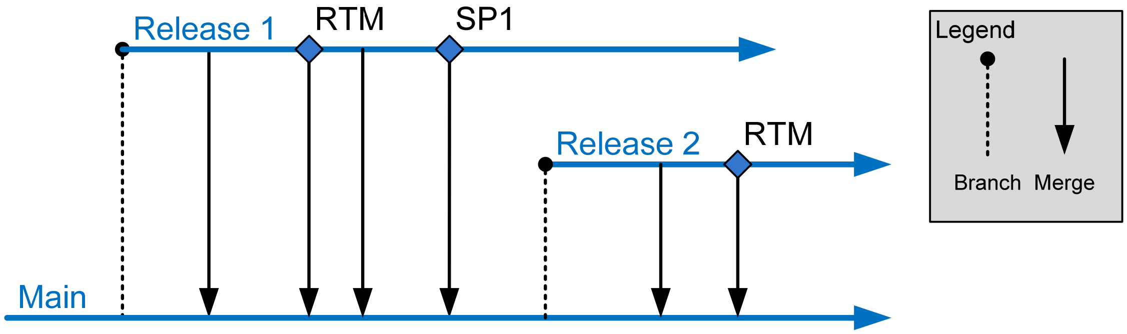
The logical structure consists of the parent/child relationships for each branch. This may be different from the physical structure you can see in the Source Control Explorer. For instance, in the physical structure above Release 2 and Main appear to be peers, when Release 2 is actually a child of Main. See the diagram below for an example of this logical relationship and how branches and merges flow through the structure.



Release Process

The following diagram shows a typical timeline when branching for a release:

1. Release 1 branch is created off of Main once the release is ready to be locked down.
2. Periodic merges into Main ensure bug fixes from the release are moved into the main integration branch
3. The RTM build is labeled in the RTM branch and merged back into Main.
4. A service pack, SP1, is released. The build is labeled and changes are merged into Main.
5. Release 1 branch lives on in support of SP1 and to enable future service packs.
6. The process repeats for future releases.

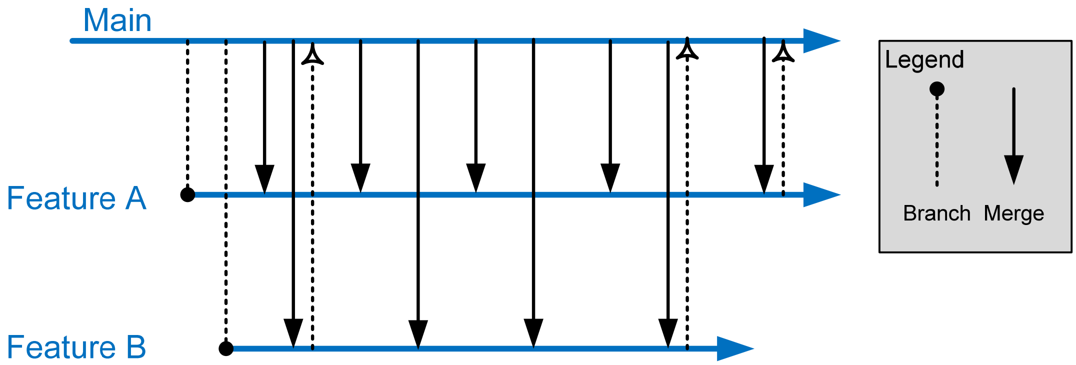


Successful use of release branches requires identifying which branch the fix should be applied to before implementing it. If a fix will be released as a hotfix or service pack, it best to make the change first on the appropriate RELEASE branch, then merge it into MAIN to ensure it gets applied to future releases.

Isolated Development Process

The following diagram shows a typical timeline when branching for development isolation:

1. A feature branch is created to isolate development for Feature A.
2. A feature branch is created to isolate development for Feature B.
3. Each team merges their changes to Main as feature milestones are complete so that they can be integrated into the main build and picked up by other teams.
4. On a periodic schedule each team merges the latest changes from Main in order to stay in sync with the work of other teams and reduce the number of conflicts per merge.



Considerations

When branching, consider the following:

- Structure your branch trees so that you only need to merge along the hierarchy (up and down the branch tree) rather than across the hierarchy. Branching across the hierarchy requires that you use a baseless merge which requires more manual conflict resolution.
- The branch hierarchy is based upon the branch parent and branch child, which may be different than the physical structure of the source code on disk. When planning your merges, keep the logical branch structure in mind rather than the physical structure on disk.
- Do not branch too deeply. Because there is latency in merging branches, a deep branching structure can mean that changes in a child branch may take a very long time to propagate to the main branch. This can negatively impact project schedules and increase the time to fix bugs.
- Branch at a high-level and include configuration and source files.
- Allow your branching structure to evolve over time.
- Do not branch unless your development team needs to work on the same set of files concurrently. If you are unsure, you can label a build and create a branch from that at a later point. Merging branches, especially if there are significant changes between them, can be costly.
- Merging requires one or more developers to execute the merge and sort out conflicts. The merged source must be thoroughly tested as it is not uncommon to make bad merge decisions that can destabilize the build.

- Merging across the branch hierarchy is especially difficult and requires you to manually handle many conflicts that could otherwise be handled automatically.

The decision whether to create a branch can be boiled down to the following:
- Will the cost of merging conflicts in real time be higher or will the overhead cost of merging conflicts between branches be higher?

Large Project Considerations
Large development teams are likely to distinguish themselves from smaller teams in the following ways:
- Require a more complex branching and merging structure
- More likely to manage dependencies across solutions and team projects
- More likely to maintain multiple builds for components and teams

Large teams are likely to branch both by team and by feature and are also more likely to have one or more branches designed to integrate external dependencies. Since the branching structure is deeper there will be more lag between a change in a development branch and getting that change into the main integration branch. For this reason it pays to think through your merging strategy when creating branches.

For example, will your merges be scheduled or event driven?
1. Forward Integration is generally scheduled, for example from the main integration branch into a development branch.
2. Reverse Integration, such as from a development branch to the main integration branch, is generally based on event (eg. feature milestone complete). This event is instigated when the team behind the development branch feels they are ready to merge changes into their parent branch.

Rationalize your branch/merge strategy with the frequency with which you want to produce builds. A deeper branching structure will result in more time to merge from child branches to the main integration branch. Symptoms of this causing a problem for your development team include:
- Build is broken and the fix takes too long to propagate up to the main integration branch.
- Milestones are missed because features take too long to propagate up to the main branch.
- The team is spending large amounts of time merging changes between various branches.

If this becomes a problem for your team consider reducing the depth of your branching structure.

The following is an example of a branching structure for a large team:
- **My Project**
    - **Development** – Container to isolate active development branches
        - **Team 1**

      - **Feature A** –Branch for isolated development
        - **Source**
      - **Feature B** –Branch for isolated development
        - **Source**
    - **Team 2**
      - **Feature A** – Branch for isolated development
        - **Source**
      - **Feature B** – Branch for isolated development
        - **Source**
  - **Main** – Main integration and build branch. All changes come together here.
    - **Source**
    - **Other Asset Folders**
  - **Release 4 –** Branch containing the code currently being locked down to release
    - **Source**
  - **Maintenance** – Container for maintenance branches
    - **Release 2**– Active maintenance branch
      - **Source**
      - **Other Asset Folders**
    - **Release 3** – Active maintenance branch
      - **Source**
      - **Other Asset Folders**
  - **Safe Keeping**
    - **Release 1** – Old release in safe keeping
      - **Source**
      - **Other Asset Folders**

This structure includes:

- Feature branches for multiple teams to work in isolation.
- A main branch to gather changes by all teams as well as bug fixes from the release branches
- A release branch used to lockdown the current release
- A maintenance branch for an old release that is being actively maintained and supported
- Safekeeping branches for older releases that are no longer being maintained

Additional Resources

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx

- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How to Use Branching to Support a Release

Use a Release branch when you are ready to stabilize your build prior to release.

The following is an example of what your branch structure may look like after you have created a Release branch:

- **Main** – Integration Branch
  - **Source**
  - **Other Asset Folders**
- **Release 1** – Release branch
  - **Source**

Keep the following recommendations in mind when working with a release branch:

- **When to branch**: When you are ready to release, integrate everything into the Main branch and then create a release branch that can be used to stabilize the application prior to release.
- **When not to branch**: If you are using one TFS project per release, then you can use the new project to continue development instead of a branch in the current project.
- **Permissions on branch**:
  - Prior to release – Read/write for all developers.
  - After release – Read/write for developers working on hot fixes, read-only for everyone else.
- **Build frequency in branch**: On-demand builds.
- **Testing focus in branch**: Sign off on release.

You should use the Release branch for the targeted fixes and changes necessary in order to stabilize a build prior to release. Development of the next version of your application can occur in parallel in your main development or integration branches so they can get the benefit of the stabilization changes you made prior to release. Once a final release build has been created, you can merge the changes from the release branch back into your main development and integration branches

Additional Resources

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx

- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How to Use Branching to Maintain a Previous Release

Use maintenance branches to support previously released builds.

The following is an example of what your branch structure may look like after you have moved a release to the maintenance folder:

- **Main –** Integration Branch
  - **Source**
  - **Other Asset Folders**
- **Maintenance –** Container for maintenance branches
  - **Release 1** – Maintenance Branch
    - **Source**
    - **Other Asset Folders**

Keep the following recommendations in mind when working with a maintenance branch:

- **When to branch**: After you have released, move your release branch into the maintenance folder. Ongoing maintenance should occur in this branch unless you are creating a new supportable version. For instance, create another maintenance branch when working on a service pack that will be released as its own SKU.
- **When not to branch**: If you will not ever need to maintain the release, you can use a label to mark the old released build and continue work in the main branch.
- **Permissions on branch**: Read/write for developers working on hot fixes, read-only for everyone else.
- **Build frequency in branch**: On-Demand builds.
- **Testing focus in branch**: Sign off on release.

You should use maintenance branches to support an older version of your application. You might choose to merge these changes into your main integration branch, or leave them specific to the maintenance branch.

Additional Resources

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx

- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How to use Branching to Stabilize Your Development and Build Process

Use a **Development** branch for active development and a **Main** branch for your integration build, thus avoiding build breaks.

The following is an example of what your branch structure might look like after you have created a Development branch:

- **Development –** Development Branch
  - **Source**
- **Main –** Integration Branch
  - **Source**
  - **Other Assets folders**

Keep the following recommendations in mind when working with a release branch:

- **When to branch**: If you are creating daily builds and are having problems with build stabilization and integration, then create both a main and a development branch to provide more predictability to your daily build. You may also want to consider more stringent check-in policies to improve check-in quality.
- **When not to branch**: If you are only creating continuous integration builds, or your daily builds are already predictably stable you might not need the extra overhead of an integration branch
- **Permissions on branch**:
  - The Main branch should be read/write for developers responsible for merging and integration, but read-only for everyone else.
  - The Dev branch should be read/write for everyone.
- **Build frequency in branch**:
  - Daily builds on the Main branch.
  - Continuous integration builds on the Dev branch.
- **Testing focus on branch:**
  - Perform integration, performance and security testing on the Main branch.
  - Perform feature and quick feedback testing on the Dev branch.

Use the Main branch as a staging ground for integrating changes that have been checked into the development branch. Perform all active development in the Dev branch, and integrate non-breaking changes into the Main branch.

Additional Resources

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How to Use Branching to Stabilize Feature Development

Use feature branches to improve the integration and stabilization of breaking changes across features.

The following is an example of what your branch structure might look like after you have created feature branches:

- **Development –** Container for feature branches
  - **Feature A –** Feature branch
    - **Source**
  - **Feature B –** Feature branch
    - **Source**
  - **Feature C –** Feature branch
    - **Source**
- **Main –** Integration branch
  - **Source**
  - **Other Asset Folders**

Keep the following recommendations in mind when working with a release branch:

- **When to branch**: Feature teams often have source file overlap, increasing the potential for build breaks and check-in conflicts. If you are experiencing these problems, consider feature branches for each feature to provide feature isolation. You can create these off a main branch for small teams or off a team branches for larger teams.
- **When not to branch**: If you are only creating continuous integration builds, or your daily builds are already predictably stable you might not need the extra overhead of feature branches.
- **Permissions in branch**: Read/write permissions for developers on the feature branch, read-only for everyone else.
- **Build frequency on branch**: Continuous integration builds.
- **Testing focus in branch**: Feature and quick feedback testing.

Use the feature branch to allow each feature to be developed in parallel. Perform all active development in the feature branches and then integrate your code into the Main branch.

Additional Resources

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How to Use Branching to Stabilize Development Across Teams

Use team branches to improve the integration and stabilization of breaking changes across teams.

The following is an example of what your branch structure may look like after you have created team branches:

- **Development -** Container for team branches
  - **Team 1 -** Team branch
    - **Source**
  - **Team 2 -** Team branch
    - **Source**
- **Main –** Integration branch
  - **Source**
  - **Other Asset Folders**

Keep the following recommendations in mind when you work with a release branch:

- **When to branch**: If your development teams overlap on files and components, use team branches to isolate each team's work. You might also choose to create feature branches beneath the team branches.
- **When not to branch**: If your source tree is organized by component, and you are confident that there will not be breaking interface changes or a large number of check-in conflicts across your teams, then you probably do not need team branches.
- **Permissions in branch**: Read/write for developers on the team, read-only for everyone else
- **Build frequency on branch**: Continuous integration builds
- **Testing focus in branch**: Feature and quick feedback testing

The team branches should be used to allow teams to complete development tasks in parallel. This can be useful to isolate teams from breaking changes originating on other teams, or to allow teams to work towards unique milestones. All active development should occur in the team branches and then integrated into the Main branch. You may also want to create feature branches beneath each team branch.

Additional Resources

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How to Use Branching to Isolate External Dependencies

Use an **External** branch to improve the integration and stabilization of breaking changes caused by external dependencies.

The following is an example of what your branch structure may look like after you have created an **External** branch:

- **External –** External branch
    - **Source**
- **Main –** Integration branch
    - **Source**
    - **Other Asset Folders**

Keep the following recommendations in mind when working with a release branch:

- **When to branch**: When you have external dependencies that may cause breaking changes in your project. If your dependencies are making interface changes or substantial logic changes that will impact your code, create an external branch to isolate these changes.
- **When not to branch**: When your external dependencies are stable or you are confident they will not introduce breaking changes.
- **Permissions in branch**: Read/write for developers responsible for external dependency integration, read-only for everyone else.
- **Build frequency on branch**: On-demand builds.
- **Testing focus in branch**: Integration testing.

You should use the external branch as an isolated location to test changes in external dependencies before you integrate them into the Main branch. This can be useful to isolate the development team from breaking changes caused by external dependencies such as headers and libraries.

Additional Resources

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How to Retire an Old Release

When a release branch in your maintenance folder is old enough that you no longer have to support it, create a **Safe Keeping** folder in which to store the old release. The following is an example of what your branch structure may look like after you have created a **Safe Keeping** folder:

- **Main** – Integration Branch
    - **Source**
    - **Other Asset Folders**
- **Maintenance –** Container for maintenance branches
    - **Release 2** – Maintenance Branch
        - **Source**
        - **Other Asset Folders**
    - **Release 2 Service Pack 1** – Maintenance Branch
        - **Source**
        - **Other Asset Folders**
- **Safe Keeping -** Container for safe keeping branches
    - **Release 1 -** Safe keeping branch
        - **Source**
        - **Other Asset Folders**

The goal of moving the branch from **Production** to **Safe Keeping** is to reduce clutter in the maintenance folder without having to delete old releases. This is not a new branch, but rather moving an old branch to a new folder.

Additional Resources

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How to Perform a Merge

Use the **merge** command to integrate changes from one branch into another. To perform a merge you can use the merge functionality in **Source Control Explorer** or you can use the **tf merge** command line. You can merge by changeset, label, date, or version. To start a merge, right click the branch in the **Source Control Explorer** and click **Merge**. The **Source Control Merge Wizard** allows you to choose the target branch to merge with.

Depending upon your branch structure you might need to merge changes up the branch hierarchy, down the hierarchy, or across the hierarchy. If you are merging across the hierarchy you are performing a baseless merge and you must use the **Tf merge** command line, because you cannot do this with Visual Studio. A baseless merge allows you to merge files and folders that do not have a branch/merge relationship. After a baseless merge, a merge relationship exists and future merges are not baseless. You still need to execute them from the command line, but the number of merge conflicts will be reduced.

Keep the following in mind when performing merges:

- Merging along the hierarchy, from parent to child or child to parent, results in fewer conflicts than merging across the hierarchy.
- The branch hierarchy is based upon the branch parent and branch child, which may be different than the physical structure of the source code on disk. For example:
  - **Physical Structure**:
    - **Development –** Container for development branches
      - **Team 1 -** Team branch
      - **Team 2 -** Team branch
        - **Feature A –** Feature branch
        - **Feature B –** Feature branch
    - **Main –** Integration branch
    - **Production** – Release branch
      - **Source**
    - **Maintenance –** Container for maintenance branches

    - **Release 2 –** Maintenance Branch
      - **Source**
      - **Other Asset Folders**
    - **Release 2 Service Pack 1 –** Maintenance Branch
      - **Source**
      - **Other Asset Folders**
  - **Safe Keeping -** Container for safe keeping branches
    - **Release 1 -** Safe keeping branch
    - **Other Asset Folders**
- **Logical Structure**:
  - **Main**
    - **Team 1**
    - **Team 2**
      - **Feature A**
      - **Feature B**
    - **Production**
      - **Release 1**
      - **Release 2**
        - **Release 2 Service Pack 1**

Additional Resources

- To get more information on merging, see "Understanding Merging": http://msdn2.microsoft.com/en-us/library/ms181427(VS.80).aspx
- For a walkthrough describing how to perform a merge, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx

How to Perform a Baseless Merge

To perform a baseless merge use the **tf merge /baseless** command from the Visual Studio 2005 Command Prompt.

For example, the following command line will perform a baseless merge from the source branch to the target branch, the **/recursive** switch is used to do recursive merge of all the files and folders within the specified branch path

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

Example

```
Tf merge /baseless c:\data\proj1 c:\data proj2 /recursive
```

The process of merging items that are not directly branched from each other is called a *baseless merge*. For example, you may want to merge a change between two release branches, which are siblings of each other, without merging up to the parent branch. Baseless merge only works from the command line; you cannot perform a baseless merge from Visual Studio.

When you perform a baseless merge Team Foundation Server does not have any information about the relationship of the files in the branches. For instance, a file rename will be viewed as a deleted file and a new file in the branch. For this reason you will have to perform more manual conflict resolution than when you perform a normal merge. However, you only have to perform this conflict resolution one time. After you have performed the baseless merge, Team Foundation Server records history and establishes a relationship between the folders/files.

Baseless merges can only be created from the command line. Even after the first baseless merge, when a relationship has been created between the branches, future merges still need to be created from the command line.

Additional Resources

- For merge command syntax, see “Merge Command”: http://msdn2.microsoft.com/en-us/library/bd6dxhfy(VS.80).aspx
- To get more information on merging, see “Understanding Merging”: http://msdn2.microsoft.com/en-us/library/ms181427(VS.80).aspx
- For a walkthrough describing how to perform a merge, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx

How to Resolve Merge Conflicts

To resolve merge conflicts use the Visual Studio merge tool. If a conflict is detected during a merge you can resolve conflicts automatically or resolve them manually. If you choose to resolve the conflict manually you can keep the source changes, keep the target changes, or resolve the conflict in the merge tool.

You may need to resolve conflicts when merging changes between branches, getting files into your workspace, or checking in new versions of files. There are three conflict types:

- **Version**. The file has evolved along divergent paths. This could be the result of a file edit, rename, delete, or undelete.
- **File Name Collision**. Two or more items are trying to occupy the same path.
- **Local Overwrite**. Only occurs during a get operation, if you are trying to overwrite a local, editable file.

Most conflicts can be resolved automatically, version conflicts are the only conflict type that may result in a manual merge operation. Manual merge is most common in the following scenarios:

- A file has been edited in both branches with changes to the same lines of code.
- A baseless merge is being conducted in which the branch file relationships are not known to Team Foundation Server.

The merge tool shows you the details for each conflict and allows you to choose which change you want to keep in the final merged file. You can choose to keep the source change, the destination change, integrate both changes, or you can manually modify the final version by typing directly into the file.

When you have resolved every conflict in the file, save the final version as a pending-change to be checked into the target branch.

Be careful when merging, it is easy to make mistakes that may result in build instability. Once you are done merging, compile the resulting source and run unit tests to test for major breaks.

Additional Resources

- For more information on how to resolve conflicts, see “How to: Resolve Conflicts”: http://msdn2.microsoft.com/en-us/library/ms181433(VS.80).aspx

Builds

- *How to use TFS to perform continuous integration builds*

How to Use TFS to Perform Continuous Integration Builds

To perform continuous integration and rebuild your project every time a file is checked in, you need to use a Web service to trigger the build process. You subscribe the Web service to check-in events so that the build is triggered each time someone checks in a file. To trigger the build you also need to use an appropriate build type that determines what configuration is built, what post build steps and test are executed, what drop location is used and so on.

Additional Resources

- A web service to trigger the build process is available from Microsoft at the following download location: http://download.microsoft.com/download/6/5/e/65e300ce-22fc-4988-97de-0e81d3de2482/ci.msi.

Check-In and Check-in Policies

- *How to work with source control change sets*
- *How to enforce coding standards prior to check-in*
- *How to override a check-in policy*
- *How to undo a check-in*
- *How to resolve a conflict*
- *How to avoid conflicts*
- *How to create a custom check-in policy*

How to Work with Source Control Change Sets

A changeset groups together a set of changes associated with a given check-in. The common operations you need to perform on changesets are:

- Checking in a changeset associated with a work item
- Labeling a changeset
- Viewing details about a changeset.
- Changing details about a changset.
- Rolling back a changeset

To check in a changeset associated with a work item:

1. From the Visual Studio **View** menu, point to **Other Windows** and click **Pending Changes**.
2. Enter a suitable comment to reflect the nature of the changes.
3. Click the **Work Items** icon to display the list of associated work items.
4. Update the associated work items by selecting them and then specifying the check in action **Associate** or **Resolve** (if the check-in means that the work item is resolved).
5. Click **Check In** to check in the changeset to the source control server.

To label a change set:

1. Right-click your Team Project folder in the **Source Control** explorer in Visual Studio and click **Apply Label**.
2. Select **Changeset** from the **Version** drop down list, enter a Changeset number and click **OK**.
3. In the **Apply Label** dialog box enter a label name and comment and then click **OK**.

To view changeset details:

- Use the **Tf changeset** command. The following command displays details about changeset number 1234 by displaying the **Details for Changeset** dialog box.

  ```
  Tf changeset 1234
  ```

  From the dialog box you can view the source files contained in the changeset, view the check in comments and notes, view associated work items and view any policy warnings that were generated when the change set was checked in to the server.

To change details about a changset:

- Use the **Tf changeset** command. You can alter the comments and/or notes associated with the changeset.
- The following command displays the **Details for Changeset** dialog box for changset 1234 and updates the displayed comment field allowing you to change the comment.

```
Tf changeset /comment:"This is a much better comment than the last one." 1234
```

- The following command updates the code reviewer and security reviewer notes associated with changeset 1234.

```
Tf changeset /notes:"Code Reviewer"="C Davis";"Security Reviewer"="F Smith" 1234
```

To rollback a changeset and remove the changes from the source control server use the Team Foundation Power Tool:

- Use the **TFPT rollback** command. The following command rolls back change set number 1234.
  ```
  Tfpt rollback /changeset:1234
  ```
  This command causes the **Roll Back Changeset** window to display. You can select the files contained within the changeset to roll back.

Additional Resources

- For more information about the Tf changeset command, see “Changeset Command”: http://msdn2.microsoft.com/en-us/library/w51xa47k(VS.80).aspx
- For more information on retrieving changesets, see “How to: Retrieve Old Versions of Files from Changesets”: http://msdn2.microsoft.com/en-us/library/ms181416(VS.80).aspx
- You can download the Team Foundation Power Tool (TFPT) from MSDN: http://www.microsoft.com/downloads/details.aspx?FamilyID=7324C3DB-658D-441B-8522-689C557D0A79&displaylang=en
- To learn more about Team Foundation Power Tool, see “Power Toy: tfptexe”: http://blogs.msdn.com/buckh/archive/2005/11/16/493401.aspx

How to Enforce Coding Standards Prior to Check-in

Use Team Foundation Server check-in policies to ensure that all code checked into the server meets your required coding standards.

The following check-in policies are available by default:

- **Code Analysis.** Requires that code analysis be run before check-in.
- **Test Policy.** Requires that check-in tests be completed before check-in.
- **Work Items.** Requires that one or more work items be associated with the check-in.

The default code analysis check-in policy provides checks for both managed and unmanaged code. Managed code analysis uses the FxCop engine to statically analyze your code to ensure that it conforms to standard .NET design rules, globalization rules, interoperability rules, naming rules, performance rules, security rules and more. If you need to extend the code analysis you must develop your own FxCop rules.

To configure the code analysis check-in policy:

1. From Team Explorer, right-click your team project, point to **Team Project Settings**, and then click **Source Control**.
2. Click **Check-in Policy** and then click **Add**.
3. In the **Check-in policy** list, select **Code Analysis** and then click **OK**.

4. Specify the type of code analysis that you want performed by selecting the appropriate check boxes. If you select **Enforce Code Analysis For Managed Code**, select the required rules in the **Rule settings for Managed Code Analysis** list.
5. Click **OK** and then **OK** again.

Important: While the above procedure ensures that the configured policy is executed whenever a developer checks in a source file, developers can always override policy at check-in time. If you want to monitor this in order to manage this scenario, you can hook the policy override event.

You can create your own custom check-in policies which will enforce your project specific quality gates for example you want to enforce that user adds comments, runs additional acceptance tests, certain pragma directives in C# are not used to suppress XML documentation warnings or the projects are configured to generate XML documentation during compilation.

To create custom policy plug-ins that appear in the **Add Checkin Policy** dialog box use the extensibility features provided in the Visual Studio Team Foundation Server SDK.

Additional Resources

- To learn how to customize a check-in policy, see “Walkthrough: Customizing Check-in Policies and Notes”, http://msdn2.microsoft.com/en-us/library/ms181281(VS.80).aspx
- To see sample code that will disallow selected patterns on checkin, see Checkin Policy to Disallow Certain Patterns: http://blogs.msdn.com/jmanning/archive/2006/02/02/523125.aspx
- To see sample code that will enforce comments on checkin, see Sample Checkin Policy: Make Sure the Comment Isn't Empty: http://blogs.msdn.com/jmanning/archive/2006/01/21/515858.aspx
- To learn how to register a new checkin policy, see I've Made a New Check-In Policy! How Do I Add It?: http://blogs.msdn.com/jmanning/archive/2006/02/07/526778.aspx
- For more information about how to use code analysis tools, see “Guidelines for Using Code Analysis Tools”: http://msdn2.microsoft.com/en-us/library/ms182023(VS.80).aspx.
- You can download the Visual Studio Team Foundation Server SDK from: http://go.microsoft.com/fwlink/?linkid=68586

How to Override a Check-in Policy

To override a check-in policy, check the **Override policy failure and continue check-in** checkbox in the **Policy Failure** dialog box. Anyone with permission to check in files can override a check-in policy.

If you would like to detect when a member of your team overrides check-in policy you can use the Team Foundation Eventing Service to hook check-in events.

Additional Resources

- To learn more about overriding a check-in policy, see "How to: Override a Check-in Policy": http://msdn2.microsoft.com/en-us/library/ms245460(VS.80).aspx
- To learn more about the Team Foundation Eventing Service, see "Eventing Service": http://msdn2.microsoft.com/en-us/library/bb130154(vs.80).aspx
- To learn how to use the TFS Object Model to detect a policy override, instead of eventing, see James Manning's blog post on MSDN: http://blogs.msdn.com/jmanning/archive/2005/11/04/489193.aspx

How to Undo a Check-in

To undo the check-in of a file, use the Team Foundation Power Tools (TFPT) **rollback** command to revert the file to the previous version. The **rollback** command enables you to rollback an entire changeset at a time, but you can also select individual files from within the changeset to rollback. This is useful if you need to undo a change to a file that was mistakenly checked in or where you want to undo the check-in because the changes are causing serious integration build issues.

To undo a check-in of a file and revert to the previous version:

1. Run the following command from a command window that includes the "\program files\microsoft team foundation server power tools" folder in the path.
   Tfpt rollback filename.cs
   Note: If you know the changeset number of the changeset containing the check-in you want to undo, you can supply the changeset number on the command line as follows:
   Tfpt rollback filename.cs /changeset:54
2. The **rollback** command needs to ensure your local workspace is up to date. Click **Yes** in response to the **Roll Back Changeset** message box. Your workspace is updated at this point with files from the server.
3. If you did not specify a changeset number on the command line, the **Find Changset** dialog box is displayed. Enter search criteria or just click **Find** and then locate and select the changeset that contains the check-in you want to undo and then click **Roll Back**.
4. The **Roll Back Changeset** dialog box is displayed .The changeset might contain multiple check-ins so select the file for which you want to undo the check-in and then click **Roll Back**. Note that if you specified the filename on the command line then only this file in the changeset will be selected.

When undoing check-ins with the TFPT **rollback** command, you should be aware of the following:

- **Workspace location.** TFPT finds locates workspaces as follows. If you specify a file path as an argument to filter the rollback, the file paths are used to find the workspace. If you do not specify file paths, the local directory is used to identify the workspace, if the local directory is mapped. The simplest approach to ensure that the tool locates the right workspace is to run the command from a locally mapped directory.

- **Pending changes**. You cannot rollback a changeset that contains pending changes and if you attempt to do so the tool reports an error. In this instance you must move pending changes to a shelveset, undo or check-in all pending changes before running the **rollback** command.
- **Merge scenarios**. If you are undoing the check-in of a file that has very recently been checked in you are unlikely to need to merge changes because it is unlikely that anyone else will have updated the item in the intervening period. However, if you want to undo a check-in and the check-in to undo is not the latest change to a file, then a three way merge is required. The current version on the server, the version you have in your workspace and the version you want to roll back to must be merged together. If there are no conflicts, then the changes from the version to rollback are extracted from the file, and any changes that came after that version are preserved.
- **Resolving conflicts**. If you do need to perform a merge, the merge window is displayed. To resolve the merge, select the item and click **Merge**. An automatic merge is initially attempted, and if it fails, the merge tool is invoked to resolve the merge. The **Auto-Merge All** button attempts an automatically merge each of the items in the merge list, but does not attempt to invoke the merge tool.

Additional Resources

- You can download the TFPT tool from MSDN: http://www.microsoft.com/downloads/details.aspx?FamilyID=7324C3DB-658D-441B-8522-689C557D0A79&displaylang=en
- To learn more about Team Foundation Power Tool, see “Power Toy: tfptexe”: http://blogs.msdn.com/buckh/archive/2005/11/16/493401.aspx

How to Resolve a Conflict

To resolve merge conflicts use the Visual Studio merge tool. If a conflict is detected during a merge you can resolve conflicts automatically or resolve them manually. If you choose to resolve the conflict manually you can keep the source changes, keep the target changes, or resolve the conflict in the merge tool.

You may need to resolve conflicts when merging changes between branches, getting files into your workspace, or checking in new versions of files. There are three conflict types:

- **Version**. The file has evolved along divergent paths. This could be the result of a file edit, rename, delete, or undelete.
- **File Name Collision**. Two or more items are trying to occupy the same path.
- **Local Overwrite**. Only occurs during a get operation, if you are trying to overwrite a local, editable file.

Most conflicts can be resolved automatically, version conflicts are the only conflict type that may result in a manual merge operation. Manual merge is most common in the following scenarios:

- A file has been edited in both branches with changes to the same lines of code.
- A baseless merge is being conducted in which the branch file relationships are not known to Team Foundation Server.

The merge tool shows you the details for each conflict and allows you to choose which change you want to keep in the final merged file. You can choose to keep the source change, the destination change, integrate both changes, or you can manually modify the final version by typing directly into the file.

When you have resolved every conflict in the file save the final version as a pending change to be checked into the target branch.

Be careful when merging, it is easy to make mistakes that may result in build instability. Once you are done merging, compile the resulting source and run unit tests to test for major breaks.

Additional Resources

- For Additional Resources on resolving conflicts, see How to: Resolve Conflicts: http://msdn2.microsoft.com/en-us/library/ms181433(VS.80).aspx

How to Avoid Conflicts

To help avoid conflicts:

- **Ensure effective team communication**. When you work on a source file, let other team members know you are working on the file and which aspects you will be updating. While automatic conflict resolution can resolve many conflicts, you should avoid situations where two or more developers are working on the same functional areas in the same source file where there is a strong likelihood of you both making changes to the same lines of code. Conflicts on the same lines of code require manual conflict resolution, which complicates the merge operation.
- **Check who else has a file checked out**. Before updating a file, check to see if another team member already has the file checked out. If so, ask your colleague what he or she is working on and then decide if you can wait until they complete their changes or if it is safe to continue to work in parallel on the same file because you are working on separate functionality in separate source code regions within the file.

To see pending changes use the following steps:

- From within the Source Control Explorer right click on the solution, project, folder, or file for which you want to see pending changes
- Select View Pending Changes

This method shows you all of the pending changes within the scope you've selected. You may also use the command line checkout tool to learn about pending changes:

```
Tf status /format:detailed /user:*
```

This command displayed detailed status information about any pending changes made by all users. From the resulting list, you can see who has which files checked out with pending changes. To check the pending changes for a specific file, specify the filename as the first argument to Tf.exe.

Additional Resources

- For more information on viewing pending changes in your workspace, see “How to: View and Manage All Pending Changes in Your Workspace”: http://msdn2.microsoft.com/en-us/library/ms181400(VS.80).aspx
- For more information on viewing pending changes in other workspaces, see “How to: View Pending Changes in Other Workspaces”: http://msdn2.microsoft.com/en-us/library/ms181401(VS.80).aspx
- For more information about the Tf status command see “Status Command”: http://msdn2.microsoft.com/en-us/library/9s5ae285(VS.80).aspx

How to Create a Custom Check-in Policy

To create a custom check-in policy use the plug-in model exposed by the policy framework.

You can create a custom check-in policy to implement your own policy logic. For example if you want to check if users have been adding comments or using regular expressions correctly at the time of check-in.

Plug-ins are used both within the policy definition process as well as during the policy evaluation process. Policy plug-ins are installed either by themselves or as part of a separate application and are registered with the policy framework so that they can be loaded as needed.

A policy plug-in must expose the following interfaces:

- **IPolicyDefinition** The **IPolicyDefinition** interface exposes methods that are used in the process of defining policy requirements for a team project. It includes methods for invoking user interface specific to the policy plug-in to allow users to easily define a check-in policy.
- **IPolicyEvaluation** The **IPolicyEvaluation** interface exposes methods that are used in the process of evaluating policy compliance during the check-in process. It includes methods that accept the contents of a check-in operation and analyze them to establish whether the defined policy is satisfied.

You can package multiple policy plug-ins n the same assembly. The only requirement is that you implement the plug-ins as separate classes.

Note: These interfaces are exposed in **PolicyBase**. As an alternative to implementing **IPolicyDefinition** and **IPolicyEvaluation**, you can derive a class from **PolicyBase**.

Additional Resources

- To learn how to customize a check-in policy, see “Walkthrough: Customizing Check-in Policies and Notes”, http://msdn2.microsoft.com/en-us/library/ms181281(VS.80).aspx

- To see sample code that will disallow selected patterns on checkin, see Checkin Policy to Disallow Certain Patterns: http://blogs.msdn.com/jmanning/archive/2006/02/02/523125.aspx
- To see sample code that will enforce comments on checkin, see Sample Checkin Policy: Make Sure the Comment Isn't Empty: http://blogs.msdn.com/jmanning/archive/2006/01/21/515858.aspx
- To learn how to register a new checkin policy, see I've Made a New Check-In Policy! How Do I Add It?: http://blogs.msdn.com/jmanning/archive/2006/02/07/526778.aspx
- For more information about how to use code analysis tools, see “Guidelines for Using Code Analysis Tools”: http://msdn2.microsoft.com/en-us/library/ms182023(VS.80).aspx.
- You can download the Visual Studio Team Foundation Server SDK from: http://go.microsoft.com/fwlink/?linkid=68586
- For more information on creating a new policy, see “Policy Plug-ins”: http://msdn2.microsoft.com/en-us/library/bb130343(VS.80).aspx

Checkout, Get and Lock

- *How to synchronize your computer with TFS*
- *How to prepare a file for editing*

How to Synchronize Your Computer with TFS

To synchronize your computer with the version control server, use the Tf **get** command. This is a quick way of ensuring that your computer is synchronized with the rest of your team to ensure you have the latest copy of shared work. To download all files and not just those that are out of date, run the following command from a Visual Studio 2005 command prompt window.

```
Tf get /all
```

With this command any local writable files that you might have on your computer are not overwritten. If you want to overwrite local writable files to completely synchronize your computer with the server use the /**force** switch as shown here.

```
Tf get /force
```

While this command overwrites local writable files, it does not overwrite writable files for which you also have pending edits. If you have pending edits on a file that you need to keep, check in or shelve your edits prior to re-synchronizing with the server. To perform the same operation from Visual Studio, double-click the **Source Control** folder in the Team Explorer window, right-click your server or team project and click **Get Specific Version**. Select **Overwrite writable files that are not checked out** and **Force get of file versions already in workspace**, ensure that **Latest Version** is selected in the **Type** drop-down list and then click **Get**.

If you want to completely synchronize your computer with the version control server, do not use the **Get Latest Version** option available in Visual Studio, because this command only downloads those files not already in your workspace and does not overwrite writable files that you have checked out in your local directory, so your computer remains out of synchronization with the server.

Additional Resources

- For more information on the get command, see “Get Command”: http://msdn2.microsoft.com/pt-br/library/fx7sdeyf(VS.80).aspx

How to Prepare a File for Editing

To prepare a file for editing:

1. From **Source Control Explorer**, select the file, right-click and click **Get Latest Version**. This downloads a read-only copy of the latest version of the file into the workspace on your computer.
2. Right-click the file and click **Check Out for Edit**.

3. Choose your required lock type. Select **None** to allow other users to check the file in and out during the period of time you are working on the file. This type of shared check out is generally recommended because most conflicts should they arise can be resolved automatically.

Note that the get latest version operation does not check out the file and the check out for edit operation does not do a get. You need to perform both steps individually. This behavior is different from VSS behavior.

When selecting your lock type, consider the following:

- A shared checkout (**None**) avoids introducing potential bottlenecks into your development process by preventing someone else working in the same file.
- You should only lock a file while editing it if you are concerned that there will be a conflict resulting in a complicated manual merge operation.
- Select the **Check Out** lock type to prevent other users from checking out and checking in the file. This option prevents other people from editing the file which could represent a potential bottleneck to your development process. This option ensures that you can apply your changes back to the source control database without the possibility of other changes having been made to the file by other people.
- Select the **Check In** lock type to allow other users to check out the file but prevent them from checking it in. Again, this option ensures that you will be able to check-in your edits without conflicts.

Additional Resources

- For more information on the checkout command, see “Checkout and Edit Commands”: http://msdn2.microsoft.com/pt-br/library/1yft8zkw(VS.80).aspx

Code Sharing

- *How to share code*
- *How to manage shared binaries*

How to Share Code

If you have source code that is used by multiple team projects, you can either manage the source code within the team project of the owning team or create a team project specifically for the shared source code.

For the teams consuming the shared source code, there are following two options:

1. Reference the code from shared location
2. Branch the Shared code

1. Reference the code from shared location
In this case the projects consuming the shared code can merely map the source from the shared location into their workspaces on the client machine. This creates a configuration that unifies the source from the shared location and their project on the client-side.

The advantage of this approach is that shared source changes are picked up every time the latest source is retrieved into the workspace.

For example you have 2 team projects Client and Shared Code, and Shared Code is the shared source, for referencing the code from shared location, these projects will share a common path on the client's hard drive

- c:\TestProject\Client
- c:\TestProject\Shared Code

They will have workspace mappings to those local hard drive paths:

Source Control Folder	**Local Folder**
$/Client	c:\TestProject\Client
$/Shared Code	c:\TestProject\Shared Code

For more information see "Working with multiple team projects in Team Build" at http://blogs.msdn.com/manishagarwal/archive/2005/12/22/506635.aspx

2. Branch the shared code
In this case the projects consuming the shared code can branch the source from the shared location into their team project. This creates a configuration that unifies the source from the shared location and their project on the server-side.

The difference is that the shared source changes are picked up as part of a merge process between the branches. This makes the decision to pick up changes in the shared source much more explicit.

For example you have 2 team projects Client and Shared Code, and Shared Code is the shared source, for branching the code from shared location.

- In the **Source Control Explorer**, right click on the root folder of Shared Code
- Select the **Branch…** option.
- In the **Branch** dialog box, set the **Target:** to the root folder of the Client team project and click **Ok**.
- Once the branch operation is completed don't forget to check-in the branched source code.

Additional Resources

- For more information see "Working with multiple team projects in Team Build" at http://blogs.msdn.com/manishagarwal/archive/2005/12/22/506635.aspx
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on project references, see "Project References": http://msdn2.microsoft.com/en-us/library/ez524kew(VS.80).aspx

How to Manage Shared Binaries

Managing shared binaries is similar to managing shared source, you must decide where you want to store the binaries and how you want your team to access the binaries.

Options for storing the binaries:

  - If the shared binaries are clearly owned by a particular team, store it in their team project.
  - If the shared binaries have no clear ownership, then create a team project specifically for the shared binaries.

Options for using the binaries in another project

  - Usually shared binaries are updated only periodically. If this is the case, branch from the shared location into the consuming team project. When the binaries change, you can execute a merge to get the latest version.
  - If you need to stay in sync with the shared binaries at all times, map the source from the shared location into a local workspace on client machines.

To branch shared binaries into your project:

- In the **Source Control Explorer**, right click on the root folder of Shared Binaries
- Select the **Branch…** option.
- In the **Branch** dialog box, set the **Target:** to the root folder of the Client team project and click **Ok**.
- Once the branch operation is completed don't forget to check-in the branched source code.

Whether you use workspace mapping or branching, use naming conventions that make it clear where the shared binary location is in your project, for instance:

- **Main**
    - **Source –** Contains code for this project
    - **Lib –** Contains shared binaries

Additional Resources

- For more information see “Working with multiple team projects in Team Build” at http://blogs.msdn.com/manishagarwal/archive/2005/12/22/506635.aspx
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on project references, see “Project References”: http://msdn2.microsoft.com/en-us/library/ez524kew(VS.80).aspx

Distributed/Remote Development

- *How to access TFS over the Internet*
- *How to support a remote office*

How to Access TFS over the Internet

If you are supporting an office with multiple clients connecting to a remote Team Foundation Server, install and configure Team Foundation Server Proxy in the Remote Office. This improves performance by caching source control files on the remote office's proxy server.

If you are supporting a single client connecting to a remote Team Foundation Server over a local area network, configure the client to connect directly to the Team Foundation Server. However, if as client computers are added, server performance begins to degrade consider installing Team Foundation Server Proxy to help reduce the workload on the server.

Additional Resources

- To learn more about the Team Foundation Server Proxy, see "Team Foundation Server Proxy and Source Control": http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx
- For more information about examining TFS proxy performance, see: http://msdn2.microsoft.com/en-us/library/ms252455(VS.80).aspx

How to Support a Remote Office

Client and server communication in TFS uses HTTP and HTTPS. To support Internet access, publish your TFS application tier in your extranet and install your TFS database on a separate server also in the extranet (dual server deployment). Configure your TFS server for HTTPS access by using a server-side certificate.

At your remote location, install and configure Team Foundation Server Proxy. This improves performance by caching source control files in the remote office's proxy. To configure and optimizing proxy performance:

- Make sure caching is enabled and monitor the performance of your cache. Monitor the performance counters (installed by default) and event logs (for errors/warnings) on your proxy server on a periodic basis, to see how your proxy is performing. Note that TFS proxy saves cache performance statistics to an XML file called ProxyStatistics.xml and you can change the interval for saving these statistics. The ProxyStatistics.xml file is located in the App_Data folder in the proxy installation directory.
- Run a scheduled task to retrieve the latest files on a periodic basis to the proxy server. This helps to ensure that the latest version of the files are available in the proxy cache and to ensure that subsequent client requests for these files result in a cache hit.

- If you know that large files are going to be downloaded over a low bandwidth (< 3Mbps) network, then set the **executionTimeout** configuration to an appropriate value in Web.config. The default value is one hour <httpRuntime executionTimeout="3600"/>.

Additional Resources

- To learn more about the Team Foundation Server Proxy, see "Team Foundation Server Proxy and Source Control": http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx
- For more information about examining TFS proxy performance, see: http://msdn2.microsoft.com/en-us/library/ms252455(VS.80).aspx
- To learn more about the TFS Proxy File Cache Server, see the MSDN Webcast: http://msevents.microsoft.com/CUI/WebCastEventDetails.aspx?culture=en-US&EventID=1032291120&CountryCode=US

Migration

- *How to migrate your source from Visual SourceSafe*
- *How to migrate your source from other version control systems*
- *How to move Team Foundation Version Control to another server*

How to Migrate Your Source from Visual SourceSafe

To migrate your source from VSS, perform the following process. Note that to perform these steps, you must be a member of the Team Foundation Administrators group.

Step 1: Prepare VSS

Prepare to migrate by backing up your VSS database, ensuring that files are checked in and running the Visual SourceSafe Analyze tool to identify and resolve data integrity issues in your existing database.

Step 2: Analyze your projects

Use the converter command-line tool (VSSConverter.exe) passing the analyze command switch together with the name of a settings XML file.

```
VSSConverter analyze conversionsettings.xml
```

A sample XML settings file is shown here:

```
  <?xml version="1.0" encoding="utf-8"?>
  <SourceControlConverter>
    <ConverterSpecificSetting>
        <Source name="VSS">
            <VSSDatabase name="c:\VSSDatabase"></VSSDatabase>
        </Source>
        <ProjectMap>
        <Project Source="$/MyFirstProject"></Project>
        <Project Source="$/MySecondProject"></Project>
        </ProjectMap>
    </ConverterSpecificSetting>
  </SourceControlConverter>
```

The settings file contains the name of the VSSDatabase. You set the name attribute to point to the folder that contains your source safe ini file. The Project elements define the path to the projects you want to convert within your source safe database. To migrate an entire VSS database, use <Project Source="$/"></Project>.

The VssConverter.exe analyze command also generates a usermap.xml file. By adding mappings to this file you can change the names associated with version history etc from VSS login names to your TSF Windows login names.

Step 3: Migrate your projects
Choose the folders you want to migrate and then use VSSConverter.exe with the migrate argument.

```
VSSConverter migrate conversionsettings.xml
```

You again pass the configuration settings XML file but with two important additions as shown here:

```
  <?xml version="1.0" encoding="utf-8"?>
  <SourceControlConverter>
    <ConverterSpecificSetting>
        <Source name="VSS">
            <VSSDatabase name="c:\VSSDatabase"></VSSDatabase>
        </Source>
        <ProjectMap>
          <Project Source="$/MyFirstProject"
Destination="$/MyTeam_ProjectOne"></Project>
          <Project Source="$/MySecondProject"
Destination="$/MyTeam_ProjectTwo"></Project>
        </ProjectMap>
    </ConverterSpecificSetting>
    <Settings>
     <TeamFoundationServer name="YourTFSServerName" port="PortNumber"
protocol="http"></TeamFoundationServer>
    </Settings>
  </SourceControlConverter>
```

Note the addition of the Destination attribute on the <Project> elements. This points to your team project in TFS (which you must have previously created). Also note the addition of the <Settings> element that contains connection details for your TFS application tier.

Additional Resources

- For more information on how to prepare for migration, see: http://msdn2.microsoft.com/en-us/library/ms181246(en-us,vs.80).aspx
- For more information on how to perform the migration, see: http://msdn2.microsoft.com/en-us/library/ms181247(VS.80).aspx
- For more information on the limitations of the Visual SourceSafe converter, see: http://msdn2.microsoft.com/en-us/library/ms252491(VS.80).aspx

How to Migrate Your Source from other Version Control Systems

You can manually export files from the version control system you're migrating from and then import them into Team Foundation Server Version Control. If you would like to preserve file history or other attributes from the version control system you're migrating from, you can use the Team Foundation Server Version Control object model to write your own migration tool.

Microsoft is currently working on a ClearCase converter, when it is ready it will be announced on the TFS Migration blog: http://blogs.msdn.com/tfs_migration

Component software has created a converter that is compatible with GNU RCS, CS-RCS, GNU CVS, Subversion (SVN) and Visual SourceSafe (VSS)

Additional Resources

- You can download the Visual Studio Team Foundation Server SDK from http://go.microsoft.com/fwlink/?linkid=68586
- To learn more about Team Foundation Version Control extensibility, see "Walkthru:The Version Control Object Model": http://msdn2.microsoft.com/en-us/library/bb187335(VS.80).aspx
- For more information on the component software converter, see: http://www.componentsoftware.com/
Products/Converter/

How to move Team Foundation Version Control to another server

Team Foundation Server does not support copying a server from one location to another, nor does it support mirroring. You can backup and restore a complete server, you can move your existing server hardware to a new domain, or you can upgrade to a dual-server deployment. You cannot perform partial moves, eg. moving some projects from your server but keeping others.

There are three supported move types for Team Foundation Server:

- **Backup Restore**. Use this move type when you need to move your Team Foundation Server to a new machine. For specific steps, see How to: Move Your Team Foundation Server from One Hardware Configuration to Another.

- **Environment**. Use this type of move when you need to move your Team Foundation Server to a new domain or workgroup. For specific steps, see How to: Move Your Team Foundation Server from One Environment to Another.
- **Single Server to Dual Server**. Use this type of move when you need to move from a single server deployment to a dual server deployment. For specific steps, see How to: Move from a Single-Server to a Dual-Server Deployment.

Keep the following considerations in mind when moving a Team Foundation Server:

- Changing the Team Foundation application-tier server name requires that all Team Foundation clients must connect to a new server name.
- All query-bound Microsoft Office documents will no longer work if the server name is changed. The documents are bound to the server for which they were created. This includes all the query bound Microsoft Office documents that are created automatically at project creation time in the project **Documents** node.
- Any embedded links to documents will point to an unknown server name if the server name is changed.
- Local accounts existed on the original Team Foundation Server. You must decide whether these accounts will be recreated as local accounts on the moved Team Foundation Server, or as domain accounts in the moved Team Foundation Server's new domain.
- Domain accounts existed on the original Team Foundation Server, but you are moving Team Foundation Server to a domain that does not trust the original domain. You must decide whether these accounts will be re-created as local accounts on the moved Team Foundation Server, or as domain accounts in the moved Team Foundation Server's new domain.

Test your server after the move to ensure nothing major was broken in the transition. Testing should check the following areas:

- Were all the assets move correctly? Look at your source tree, work items, and team pages to be sure they are all in place.
- Are the user accounts still operative? Try logging in with a few different account types to make sure they still work.

Additional Resources

- For more information on moving Team Foundation Server, see: http://msdn2.microsoft.com/en-us/library/ms404879(VS.80).aspx

Project/Workspace Management

- *How to choose one team project vs multiple team projects*
- *How to organize your source tree*
- *How to define workspace mappings*
- *How to use workspaces to isolate code changes on your computer*

How to Choose one Team Project vs Multiple Team Projects

To plan your project structure, evaluate common project organization strategies and choose the one that best fits your organization's size, server constraints, and process workflow. Projects are intended to represent the largest unit of work possible in your organization. You should preferably choose single project versus creating multiple projects. Use multiple projects when there is strong reason for doing so, for example the team changes, or there is significant change from one release to another and you don't wish to carry the unwanted work items bugs etc going forward, or there is change in the process template, etc.

The main advantages of creating single project versus multiple projects are that it simplifies moving requirements, features, scenarios, and bugs between releases

The most important decision points are:

1. Do you want to maintain work items, and other assets from release to release? If so, choose to keep all releases in the same project.
2. Do you want to create a new process and work item structure from scratch when you move to a new release? If so, choose to create a new project for each release.

Common project structures include:

- **One project per application.** Use a single project to contain all versions of your application. Create a branch per release within that project.
    - **Reasons to use this structure:**
        - Makes it easy to carry forward code, work items and other assets.
    - **Reasons not to use this structure:**
        - Releases in parallel are forced to share work items schema, check-in policies, and process guidance.
        - Reporting is harder since reports default to entire project, you must add filtering by release.
        - If you have hundreds of applications, each in its own project, you may hit Team Foundation Server scalability limits.
        - You will accumulate 'baggage' over multiple releases. Easiest way to clean this up is to create a new project and branch code you want to carry forward into that project.
- **One project per release**. Create a new project for each version of your application.
    - **Reasons to use this structure:**

        - Works well if you want to start with a clean project after every release.
        - Old project can be used for maintenance and handed off to a separate sustained engineering team.
        - It is easy to move source from one project to another.
    - **Reasons not to use this structure:**
        - It is difficult to move work items and Team Foundation Server assets from one project to another. Work items can only be copied one at a time to another project. If you want to move sets of items you'll need to do so manually or write your own utility.
        - If you are managing hundreds of projects you may hit Team Foundation Server scalability limits.

Keep the following considerations in mind when choosing your strategy:

- Pick a structure you can live with, restructuring existing team projects is difficult.
- Source can be easily shared amongst team projects:
    - Branch source from one project to another
    - Map source from another project into your workspace
- Team Foundation Serveer can scale to ~500 projects using MSF Agile process or 250 projects using MSF CMMI process. If you create your own or customize an existing process keep in mind that work item schema has the largest impact on server scalability. A complex schema will result in a server capable of supporting fewer projects.

Additional Resources

- For more information about Project strategy, see Eric Lee's blog post "When to use Team Projects": http://blogs.msdn.com/ericlee/archive/2006/08/09/when-to-use-team-projects.aspx

How to Organize Your Source Tree

Your source tree structure consists of a combination of folder structure, file structure, and branch structure. Within your main branch the following folder and file structure has been proven to work for teams of various sizes:

- **Main** - container for all assets you need to ship the product
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code** – container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests
        - **Lib** - container for binary dependencies
    - **Docs** - container for documentation that ships with your product
    - **Installer** - container for installer source code and binaries
    - **Builds** - container for team build scripts
    - **Tests** - container for test team tests

Any branches that you create off of main will copy this folder and file structure into the new branch. For example:

- **Dev** – Development branch
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code** – container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests
        - **Lib** - container for binary dependencies
- **Main** – Integration branch
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code** – container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests
        - **Lib** - container for binary dependencies
    - **Docs** - container for documentation that ships with your product
    - **Installer** - container for installer source code and binaries
    - **Builds** - container for team build scripts
    - **Tests** - container for test team tests

Additional Resources

- For more information on creating a workspace, see "How to: Create a Workspace": http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx

How to Define Workspace Mappings

To create a workspace mapping for a project that is not yet on your hard drive:

1. Select the root source folder in Source Control Explorer.
2. Right click and select **Get Latest Version**.
3. Choose the local folder to which you want to map the workspace.

To modify a workspace mapping for a project that is already on your hard drive:

1. Select File->Source Control->Workspaces…
2. Use the **Manage Workspaces** dialog box to add, remove or edit existing workspaces.

To view an existing workspace mapping:

1. Select a source folder in Source Control Explorer
2. Right-click and then click **Properties**.
3. The workspace mapping to your local drive appears in the **Local Name** field.

Consider the following best practices for creating workspace mappings:

- **Create mappings at the team project root level**. For new team projects, map your team project root ($/MyTeamProject) to a folder with the same name on your local drive for example C:\TeamProjects. Because mappings are recursive, your entire local folder structure is created automatically and will be exactly the same as the structure in source control.

- **On shared computers, use a unique local folder path**. Two users of a single computer cannot share the same workspace mapping. For example, you and a colleague cannot map the same team project ($/MyTeamProject) to the same folder on the local computer. Instead, create mappings beneath My Documents (although this leads to long location paths) or establish a team folder naming convention on the local computer (e.g. C:\TeamProjects\User1, C:\TeampProjects\User2 etc).
- **You may not need the entire tree**. To improve performance and reduce disk size requirements only map the files you need for your development project. In general you will only need the files and projects associated with the solution you will be working on.
- **Avoid using workspace mapping to support cross project dependencies**. In general you should avoid dependencies that cross team projects and try to have all the related/dependent solutions/projects under same team project. This reduces the need for build script customization. If you have a dependency, use project references to define it or branch the dependency from a shared project into your project. You should avoid file references as they are more difficult to manage. The exception is when the dependent project is developed in parallel and you need real-time changes. In this case, you can consider the using workspace mapping. You may still consider branching to create a buffer of isolation, if the dependent code is causing too many breaking changes

Additional Resources

- For more information on creating a workspace, see "How to: Create a Workspace": http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx
- For more information on editing a workspace, see "How to: Edit a Workspace": http://msdn2.microsoft.com/en-us/library/ms245466(VS.80).aspx

How to use Workspaces to Isolate Code Changes on Your Computer

As a developer you can create two workspaces, one containing references to files and folders being worked on by the rest of the team, another containing files and folders that you want to isolate. You might want to isolate these files in order to evolve specific files in parallel with work that is happening elsewhere. For instance, this can be used to work on risky pending changes, or to conduct a code review.

To create a secondary workspace:

1. Click the **Workspace** drop down within Source Control Explorer and then click **Workspaces**.
2. In the **Manage Workspaces** dialog box, click **Add**.
3. In the **Add Workspace** dialog box, enter a new workspace name such as **MyIsolatedWork** and provide a comment to serve as a future reminder about the purpose of the workspace.
4. In the **Working folders** list, set the workplace status to **Active**, identify the source control folder to be included in the workspace (this can be the team project

root folder or any sub folder) and specify a local folder path on your own computer to contain the files from the workspace.

5. Click **OK** and then **Close** to create the isolated workspace.

To retrieve the latest set of source to begin work in your isolated workspace:

1. In source control explorer, make sure that your isolated workspace name is selected from the Workspace drop down list.
2. Select your team project root folder, or a sub folder if you only need part of the source tree, right-click and then click **Get Latest Version**. This copies the folder structure and latest file set from the source control server to the local directory on your computer which you mapped to the new workspace.

Additional Resources

- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx
- For more information on editing a workspace, see “How to: Edit a Workspace”: http://msdn2.microsoft.com/en-us/library/ms245466(VS.80).aspx

Security

- *How to secure the channel between developer workstation and TFS*

How to Secure the Channel Between Developer Workstation and TFS

To secure the channel between developer workstation and TFS, you can configure your Team Foundation Server to use HTTPS and Secure Sockets Layer (SSL). You can configure your Team Foundation Server to use only HTTPS and SSL, and disallow HTTP connections. In order to do this, you must first configure Team Foundation Server to allow HTTPS and SSL, and then perform the additional steps to require HTTPS and SSL.

Using HTTPS and SSL encrypts the network traffic between Team Foundation Server and the Team Foundation clients that request access to Team Foundation Server's Web resources, including team project portals, reports, and work items.

- **Set up Team Foundation Server with Secure Sockets Layer (SSL)**. "Walkthrough: Setting up Team Foundation Server with Secure Sockets Layer (SSL) " at http://msdn2.microsoft.com/en-us/library/ms242875(VS.80).aspx
- **Configure Team Foundation Server for HTTPS and SSL Only.** "How to: Configure Team Foundation Server for HTTPS and SSL Only " at http://msdn2.microsoft.com/en-us/library/aa395285(VS.80).aspx

Additional Resources

- For more information, see "Securing Team Foundation Server with HTTPS and Secure Sockets Layer (SSL)": http://msdn2.microsoft.com/en-us/library/aa395265(VS.80).aspx
- For more information, see "Walkthrough: Setting up Team Foundation Server with Secure Sockets Layer (SSL)": http://msdn2.microsoft.com/en-us/library/ms242875(VS.80).aspx
- For more information see "How to: Configure Team Foundation Server for HTTPS and SSL Only": http://msdn2.microsoft.com/en-us/library/aa395285(VS.80).aspx

Shelving

- *How to use shelving to backup pending work*
- *How to use shelving to share code with a team member*

How to Use Shelving to Backup Pending Work

To backup pending changes to the server, shelve the files you are not yet ready to check in. This ensures that the source is uploaded to the server without checking in partially completed work that could potentially lead to build instability.

To shelve a set of pending changes, first view them, by right-clicking your solution in Solution Explorer and clicking **View Pending Changes**. Select the files you want to shelve and then click **Shelve**. Type a shelfveset name, a comment that identifies the purpose of the shelveset and then click **Shelve**.

To restore your work the following day, from the **File** menu, point to **Source Control** and then click **Unshelve**. Select the required shelveset and click **Unshelve**. Team Foundation restores each shelved revision into the destination workspace as a pending change as long as the revision does not conflict with a change that is already pending in your workspace.

Additional Resources

- For more information on shelving pending changes, see "How to: Shelve and Unshelve Pending Changes": http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

How to use Shelving to Share Code with a Team Member

To shelve a source code for sharing code with a team member, perform a **Get Latest** operation to synchronize your workspace with the latest server version and then build your application to ensure that it compiles. Shelve the source using the **Source Control Explorer**, the team member who has been handed off the code then needs to unshelve the code.

Shelving is useful when you have work in progress that is to be completed by another team member, you can shelve your changes to make a handoff easier. By synchronizing the latest code, you get an opportunity to incorporate changes to source files that have been made outside of your workspace.

To shelve folders and files from Source Control Explorer

1. In Source Control Explorer, right-click, and choose **Shelve Pending Changes**.
2. In the **Shelve - Source Files** dialog box, type the shelveset name, for example **shelvetest** in the **Shelveset name** box.
3. In the **Comment** box, type **Testing my shelveset**, and then click **Shelve**.

The files and folders are copied to the source control server and are available for other team members to unshelve.

When the other team member unshelve a shelveset, Team Foundation restores each shelved revision into the destination workspace as a pending change as long as the revision does not conflict with a change that is already pending in the workspace.

To unshelve a set of pending changes

1. In Visual Studio 2005 Team System, click **File**, point to **Source Control**, and then click **Unshelve**.
2. In the **Owner name** box, type the shelveset creator's name (for example, ADVENTUREWORKS\JuanGo or simply juango) and then click **Find**.
3. In the **Results** pane, select the shelveset you want to unshelve into your workspace, and then click **Details**.
4. If you want to delete the shelveset from the Team Foundation source control server, deselect the **Preserve shelveset on server** option.
5. Optionally deselect the **Restore work items and check-in notes** option if you do not want to have the work items and check-in notes associated with the shelveset restored.
6. When the **Details** dialog box appears, select the shelveset or shelveset items you want to unshelve into your workspace, and then click **Unshelve**.

Additional Resources

- For more information see "How to: Shelve and Unshelve Pending Changes" at http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

Additional Resources

- Visual Studio Team System Guidance Project: http://www.codeplex.com/VSTSGuidance

Questions and Answers: Team Foundation Server Source Control and Versioning

Index

Accessing Version Control

- *What is the MSSCCI Provider and when is it used?*
- *What other IDEs support TFS?*
- *When should I use the team foundation power tool?*
- *What are the most common version control extensibility scenarios?*
- *How do I work with version control from the command line?*

Administration

- *How do I grant permissions on a file within a folder that has inherited permissions?*
- *What do I do, if a developer has left the project?*
- *How do I manage interns or other developers that I do not trust to check-in?*
- *How should I modify permissions once my application has shipped?*

Branch/Label/Merge

- *When should I use labels?*
- *How do TFS labels differ from VSS labels?*
- *What is branching?*
- *When should I consider branching?*
- *What are the reasons to not branch?*
- *How do I use branching to release my application?*
- *How do I use branching to maintain my application?*
- *How do I use branching to reduce conflicts between teams*
- *How do I use branching to reduce conflicts between features*
- *What are the proven practices for branching and merging?*
- *What is the difference between branching and labeling?*
- *What is "path space" branching model?*
- *How does the TFS promotion model work?*
- *How do I merge two branches together?*
- *Can I merge across Team Projects?*
- *What is a baseless merge?*
- *What is the code promotion model?*
- *What is the difference between the logical and physical view of branches?*
- *If I use the code promotion model, how often should I merge?*

Checkin and Checkin Policies

- *What is a changeset?*
- *What's a check-in policy?*
- *When and how can I override check-in policy?*
- *How do I enforce a policy?*
- *How do I use a check-in verification system?*
- *If I modify file names or delete files on the disk does version control get out of sync?*
- *How does automatic conflict resolution work?*
- *How do I resolve conflicts manually?*
- *How do I avoid conflicts?*

Checkout, Get, and Lock

- *How do I find out who was the last developer to modify a file?*
- *How does get work?*
- *What's the difference between shared and exclusive checkout?*
- *When should I use the lock command?*
- *What are the different lock types that TFS supports?*

Distributed/Remote Development

- *How do I work off-line?*
- *How do I optimize for distributed team development?*
- *What is the version control proxy?*
- *How do I optimize version control proxy performance?*

Migration

- *How is TFS version control different than Visual Source Safe?*
- *How is the checkout model different from Visual SourceSafe?*
- *How should I migrate my source from VSS to TFS?*
- *How should I migrate my source from other version control systems?*

Project/Workspace Management

- *How should I organize my team projects?*
- *How should I manage dependencies between projects?*
- *What's a workspace?*
- *How do I use workspaces to isolate a developer?*
- *What are the proven practices for workspace mapping?*
- *What are the proven practices for managing shared components and code?*
- *When should I create a new Team Project versus create a new Branch?*
- *How should I manage source code that is shared across multiple projects?*
- *How should I manage binaries that are shared across multiple projects?*

- *How should I organize my source tree?*

Reporting

- *How do I create a custom report?*
- *How do I find out who was the last developer to modify a file?*
- *How do I find all of the changes associated with a work item?*
- *What reports are available out of the box?*

Shelving

- *What is shelving?*
- *What is a shelveset?*
- *When would I typically use shelving?*
- *How do I use shelving to back up my work?*
- *Why would I want to unshelve a shelveset?*

Accessing Version Control

- *What is the MSSCCI Provider and when is it used?*
- *What other IDEs support TFS?*
- *When should I use the team foundation power tool?*
- *What are the most common version control extensibility scenarios?*
- *How do I work with version control from the command line?*

What is the MSSCCI Provider and when is it used?

The Microsoft Source Code Control Interface (MSSCCI) provider is used to provide an integrated version control user experience with products that don't support the Visual Studio Team Explorer. For example if you use Visual Studio 6.0 you can use the MSSCCI client or command line to interact with Team Foundation Server Version Control.

The following clients can work directly with Team Foundation Version Control by using the MSSCCI provider:

- Visual Studio NET 2003
- Visual C++ 6 SP6
- Visual Visual Basic 6 SP6
- Visual FoxPro 9 SP1
- Microsoft Access 2003 SP2
- SQL Server Management Studio
- Sparx Systems Enterprise Architect 61
- Sybase PowerBuilder 105
- Toad for SQL Server 20

The MSSCCI provider behaves differently than Team Foundation Version Control in Visual Studio 2005 in the following ways:

- Checkout also performs a GetLatest.
- An exclusive check-in lock is applied at checkout.
- Open-from source control and save-to source control behave like Visual SourceSafe.

More Information

- To read more about MSSCCI on MSDN, see "The Microsoft Source-Code Control Interface**"**: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcug98/html/_asug_the_microsoft_source_code_control_interface.asp
- To learn more about the MSSCCI provider, see Update on the TFS MSSCCI Provider: http://blogs.msdn.com/bharry/archive/2006/03/24/559876.aspx
- The MSSCCI add-in is a TFS power tool developed but not officially supported by Microsoft. You can download the tool from MSDN: http://www.microsoft.com/downloads/details.aspx?FamilyId=87E1FFBD-A484-4C3A-8776-D560AB1E6198&displaylang=en

What other IDEs support TFS?

Team Foundation Server can be used from any SKU of Visual Studio 2005 with Team Explorer installed. You can also run Team Explorer side-by-side with any non-Visual Studio 2005 IDE in order to work with team projects and manage work items.

The following clients have integration solutions provided by other vendors:

- Eclipse
- Linux client
- Mac client
- HTML Web client

If you want access to Team Foundation Version Control from the Eclipse IDE, Linux, or Mac, consider installing Teamprise: http://www.teamprise.com/

If you would like read-only access to Team Foundation Version Control from the Web, consider installing TeamPlain Web Access: http://www.devbiz.com/teamplain/

More Information

- For more information on using Team Explorer, see: http://msdn2.microsoft.com/en-us/library/ms242912(vs.80).aspx
- For more information on Teamprise, see http://www.teamprise.com
- For more information on TeamPlain, see http://www.devbiz.com/teamplain/

When should I use Team Foundation Power Tools?

The Team Foundation Power Tools (TFPT) provides version control functionality that is not available from the UI. For example, you can use it to help support offline working or to perform rollback operations to undo check-ins of a changeset. Consider using TFPT if you need to perform any of the following operations:

- **Unshelve**: The unshelve operation supported by TFS does not allow shelved changes and local changes to be merged together. By using TFPT to unshelve a change from a change set, if an item in your local workspace has a pending change that is an edit, and the shelved change is also an edit, then TFPT can merge the changes with a three-way merge.
- **Rollback**: The ability to undo a checkin of a changeset is not directly supported by TFS. By using the TFPT rollback command, you can attempt to undo any changes made in a specified changeset. Not all changes can be rolled back but the rollback works for most scenarios.
- **Work Offline**. The TFPT online tool allows you to work without a server connection for a period of time by providing functionality that informs the server about changes you make to your local workspace.
- **Get a Changeset**. The TFPT GetCS tool enables you to get all the items listed in a changeset at that changeset version. This is useful if a colleague has checked in a change which you need to have in your workspace, but you cannot update your entire workspace to the latest version.

- **Remove Pending Edits**. The TFPT Undo Unchanged (UU) tool removes pending edits from files which have not actually been edited. This is useful in a scenario where you check out a large number of files for edit, but only actually make changes to a small number of them. You can back out your edits on the unchanged files by running the TFPT UU tool, which compares hashes of the files in the local workspace to hashes the server has to determine whether or not the file has actually been edited

You run each of these commands from the command line by using Tfpt.exe.

More Information

- You can download Team Foundation PowerTools from here: http://www.microsoft.com/downloads/details.aspx?FamilyID=7324c3db-658d-441b-8522-689c557d0a79&DisplayLang=en
- A forum to discuss Team Foundation Power Tools: http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=930&SiteID=1

What are the most common version control extensibility scenarios?

The most common version control extensible scenario is customizing a check-in policy in order to enforce standards at check-in time. To create custom policy plug-ins that appear in the **Add Checkin Policy** dialog box use the extensibility features provided in the Visual Studio Team Foundation Server SDK. You can download the Visual Studio Team Foundation Server SDK from http://go.microsoft.com/fwlink/?linkid=68586

Though not as common, it is also possible to write an integration layer to allow a non-Visual Studio 2005 client to work with Team Foundation Version Control.

More Information

- To learn how to customize a check-in policy, see “Walkthrough: Customizing Check-in Policies and Notes”, http://msdn2.microsoft.com/en-us/library/ms181281(VS.80).aspx
- To see sample code that will disallow selected patterns on checkin, see Checkin Policy to Disallow Certain Patterns: http://blogs.msdn.com/jmanning/archive/2006/02/02/523125.aspx
- To see sample code that will enforce comments on checkin, see Sample Checkin Policy: Make Sure the Comment Isn't Empty: http://blogs.msdn.com/jmanning/archive/2006/01/21/515858.aspx
- To learn how to register a new checkin policy, see I've Made a New Check-In Policy! How Do I Add It?: http://blogs.msdn.com/jmanning/archive/2006/02/07/526778.aspx
- You can download the Visual Studio Team Foundation Server SDK from http://go.microsoft.com/fwlink/?linkid=68586

- To learn more about Team Foundation Version Control extensibility, see “Walkthru:The Version Control Object Model”: http://msdn2.microsoft.com/en-us/library/bb187335(VS.80).aspx

How do I work with version control from the command line?

Team Foundation Server provides the TF command line tool (Tf.exe) to enable you to perform source control operations. For example, you can use the command line to schedule operations by using the Windows Task Scheduler.

To ensure the appropriate path and other environment variables are setup, run the tool from the Visual Studio 2005 Command Prompt window or run the Vsvars32 batch file, which is normally located in *DriveLetter*:\Program Files\Microsoft Visual Studio 8\Common7\Tools. The command line supports most source control commands including Commands, Checkin, Checkout, Get, History, Shelve, Branch, Merge, Label, Status, Undelete, and Undo.

The following are common operations you may want to execute from the commandline:

- Synchronize files from the server to your local machine – tf get
- Add a file to the server – tf add
- Check out a file for editing – tf checkout
- Check in pending changes – tf checkin
- Retrieve a particular changeset from the server – tf get /version

There are certain operations that you can only do from the commandline:

- Delete another user's workspace – tf workspace /delete
- Undo another user's check-in – tf undo
- Unlock another user's lock – tf lock
- Define label scope – tf label
- Perform a baseless merge – tf merge

More Information

- MSDN: Walkthrough: Working with Team Foundation Source Control from Command Line: http://msdn2.microsoft.com/en-us/library/zthc5x3f.aspx
- For more information on commands only available from the command line, see “Operations Available Only From the Command-Line (Team Foundation Source Control)”: http://msdn2.microsoft.com/en-us/library/ms194957(VS.80).aspx

Administration

- *How do I grant permissions on a file within a folder that has inherited permissions?*
- *What do I do, if a developer has left the project?*
- *How do I manage interns or other developers that I do not trust to check-in?*
- *How should I modify permissions once my application has shipped?*

How do I grant permissions on a file within a folder that has inherited permissions?

You set permissions by right-clicking the folder or file in Source Control Explorer, clicking Properties, and on the Security tab, selecting the user or group for which you want to change permissions, and editing the permissions listed in Permissions. You can also set these permissions on the command-line by using the Permission command of the TF command-line utility.

Team Foundation source control allows you to grant access control permissions to Windows Groups, Windows Users, and Team Foundation Groups. Permissions can be inherited from the containing folder, or you can declare permissions explicitly. Permission settings are in the form of either Allow or Deny. Deny always overrides Grant, even if Deny is inherited and Grant is explicitly defined. The inherited permissions are combined with the explicit permissions to determine a user’s or group’s effective permissions on an item. Deny always overrides allow. So, you can keep inherit turned on and deny Checkin, for example.

You should be careful when turning inheritance off. You should first set the explicit permissions required, make sure to assign your own account specific permissions, and then turn inheritance off. If you turn inheritance off without setting the desired permissions, you can lock yourself out of a file or folder and require a TFS administrator who is also an administrator on the AT computer to fix the permissions. AT local administrators can never completely lock themselves out by design.

More Information

- For more information, see "Team Foundation Server Default Groups, Permissions, and Roles" on MSDN at http://msdn2.microsoft.com/en-us/library/ms253077.aspx
- For more information, see "MSDN: Source Control Security Rights and Permissions" at http://msdn2.microsoft.com/en-us/library/ms181761.aspx

What do I do, if a developer has left the project?

If a developer has left your project, make sure that you delete the developer's workspace. This operation also removes any of the developer’s pending changes and undoes any locks held by the developer. Note, you cannot undo the locks without undoing the changes if exclusive lock is turned on for the team project.

To find out which files the developer has locked out run the following command:

```
tf workspaces /owner:domain\devuser /computer:*
/server:servername
```

To delete the workspace, and remove locks, run the following command:

```
Tf workspace /delete workspacename;domain\devuser
/s:servername
```

More Information

- For more information on cleaning up after a developer who has left, see “How to: Clean Up Files When Users Leave”: http://msdn2.microsoft.com/en-us/library/ms194958(VS.80).aspx
- For more information on the Workspace command, see “Workspace Command”: http://msdn2.microsoft.com/en-us/library/y901w7se(VS.80).aspx

How do I manage interns or other developers that I do not trust to check-in?

Un-trusted developers, such as new hires or interns, may be denied check-in permission on the source tree. Make sure that you set your desired permissions (including permissions for your own account) before turning off inheritance. Rather than check-in directly they can make pending changes and then shelve these changes. A more experienced developer can unshelve the changes, review them, and then check them in.

More Information

- For more information on removing permissions, see “How to: Remove Access to Source Control Files”: http://msdn2.microsoft.com/en-us/library/ms400718(VS.80).aspx

How should I modify permissions once my application has shipped?

Once a branch is in maintenance, for instance after it has been shipped, you can turn off inheriting permissions to lock down the tree. Once you’ve done this, you may allow individual users the pend-change and check-in permissions as needed for hot fixes.

More Information

- For more information on removing permissions, see “How to: Remove Access to Source Control Files”: http://msdn2.microsoft.com/en-us/library/ms400718(VS.80).aspx

Branch/Label/Merge

- *When should I use labels?*
- *How do TFS labels differ from VSS labels?*
- *What is branching?*
- *When should I consider branching?*
- *What are the reasons to not branch?*
- *How do I use branching to release my application?*
- *How do I use branching to maintain my application?*
- *How do I use branching to reduce conflicts between teams*
- *How do I use branching to reduce conflicts between features*
- *What are the proven practices for branching and merging?*
- *What is the difference between branching and labeling?*
- *What is "path space" branching model?*
- *How does the TFS promotion model work?*
- *How do I merge two branches together?*
- *Can I merge across Team Projects?*
- *What is a baseless merge?*
- *What is the code promotion model?*
- *What is the difference between the logical and physical view of branches?*
- *If I use the code promotion model, how often should I merge?*

When should I use labels?

Use labels to group a set of files and folders together for future operations. You can use labels for branching, merging, diffing, or getting files. A label provides a marker to which you can return when performing one of the above operations at a later date.

The Team Foundation Build automatically labels the file versions associated with each build that it creates.

Note: If you are unsure whether you need a branch or not, you can label a set of files and later create a branch based on this label.

More Information

- For more information, see “Working with labels”: http://msdn2.microsoft.com/en-us/library/ms181439(VS.80).aspx
- For more information on labels, see “How to: Apply Labels”: http://msdn2.microsoft.com/en-us/library/ms181440(VS.80).aspx

How do TFS labels differ from VSS labels?

TFS labels are quite different from VSS labels. VSS labels are “point in time” labels that you normally assign to all or part of a tree in VSS. VSS therefore displays labels in time sequence along with the file history. Everything before the labeled file in the list is included in the label and everything after it is not.

In TFS, instead of being a single point in time, labels tie together specific versions of a set of source files. A common scenario for labels is to use them to mark daily builds. This allows you to easily retrieve the source file set corresponding to a particular build, for example if you need to apply a fix.

More Information

- For more information, see “Comparing SourceSafe Labels to Team Foundation Server Labels” at http://blogs.vertigosoftware.com/teamsystem/archive/2006/05/03/Comparing_SourceSafe_Labels_to_Team_Foundation_Server_Labels.aspx

What is branching?

Branching (also known as forking) is the act of splitting a collection of files into separate development paths. TFS supports branching and sophisticated merging which allows you to join files from separate branches back together. For instance, you can use branching to isolate major releases of your application. Once you have branched the released version of your application you can more easily maintain it in the future. Merging allows you to selectively make fixes to both branches.

The purpose of branching and merging is to allow you to isolate concurrent streams of development. For instance you create a branch when your team has produced a build that you want to maintain going forward. You could also choose to branch in order to isolate feature teams in a very large development organization, or to support development on multiple versions of your application at once.

Team Foundation Server Version Control does not make separate copies of the file content when you branch it, so it does not consume a lot of additional space in the source control database. A branch consists of a pointer in the database to the source content in the list of deltas that modify it from the base version.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

When should I consider branching?

Branches enable multiple developers to work on the same files in isolation. Merging has overhead and requires you to manage conflicts, don't branch unless you need the file isolation it provides. You can label a build and branch later if needed.

The decision whether to create a branch can be boiled down to the following:

- Will the cost of merging conflicts in real time be higher or will the overhead cost of merging conflicts between branches be higher?

Common reasons to branch include:

- **Release** – Branch for builds you want to maintain or branch for multiple releases in parallel.
- **Maintenance –** Branch to maintain a previously released build.
- **Feature –** Isolate work on experimental or risky features that might cause instability to the rest of the project. Isolate work on interface change that will cause instability for the rest of the project.
- **Team –** Isolate sub teams so they can work without being subject to breaking changes from other teams. Isolate sub teams so they can work towards unique milestones. This are very similar to feature branches.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

What are the reasons to not branch?

Do not branch unless your development team needs to work on the same set of files concurrently. If you are unsure, you can label a build and create a branch from that at a later point. Merging branches, especially if there are significant changes between them, can be costly.

Merging requires one or more developers to execute the merge and sort out conflicts. The merged source must be thoroughly tested as it is not uncommon to make bad merge decisions that can destabilize the build.

Merging across the branch hierarchy is especially difficult and requires you to manually handle many conflicts that could otherwise be handled automatically.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How do I use branching to release my application?

Use a Release branch when you are ready to stabilize your build prior to release.

The following is an example of what your branch structure may look like after you have created a Release branch:

- **Main** – Integration Branch
    - **Source**
    - **Other Asset Folders**
- **Release 1** – Release branch
    - **Source**

Keep the following recommendations in mind when working with a release branch:

- **When to branch**: When you are ready to release, integrate everything into the Main branch and then create a release branch that can be used to stabilize the application prior to release.
- **When not to branch**: If you are using one TFS project per release, then you can use the new project to continue development instead of a branch in the current project.
- **Permissions on branch**:
    - Prior to release – Read/write for all developers.
    - After release – Read/write for developers working on hot fixes, read-only for everyone else.
- **Build frequency in branch**: On-demand builds.

- **Testing focus in branch**: Sign off on release.

You should use the Release branch for the targeted fixes and changes necessary in order to stabilize a build prior to release. Development of the next version of your application can occur in parallel in your main development or integration branches so they can get the benefit of the stabilization changes you made prior to release. Once a final release build has been created, you can merge the changes from the release branch back into your main development and integration branches

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How do I use branching to maintain my application?

Use maintenance branches to support previously released builds.

The following is an example of what your branch structure may look like after you have moved a release to the maintenance folder:

- **Main –** Integration Branch
    - **Source**
    - **Other Asset Folders**
- **Maintenance –** Container for maintenance branches
    - **Release 1 –** Maintenance Branch
        - **Source**

Keep the following recommendations in mind when working with a release branch:
Keep the following recommendations in mind when working with a maintenance branch:

- **When to branch**: After you have released, move your release branch into the maintenance folder. Ongoing maintenance should occur in this branch unless you are creating a new supportable version. For instance, create another maintenance branch when working on a service pack that will be released as its own SKU.
- **When not to branch**: If you will not ever need to maintain the release, you can use a label to mark the old released build and continue work in the main branch.
- **Permissions on branch**: Read/write for developers working on hot fixes, read-only for everyone else.

- **Build frequency in branch**: On-Demand builds.
- **Testing focus in branch**: Sign off on release.

You should use maintenance branches to support an older version of your application. You might choose to merge these changes into your main integration branch, or leave them specific to the maintenance branch.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How do I use branching to reduce conflicts between teams?

Use team branches to improve the integration and stabilization of breaking changes across teams.

The following is an example of what your branch structure may look like after you have created team branches:

- **Development -** Container for team branches
    - **Team 1 -** Team branch
        - **Source**
    - **Team 2 -** Team branch
        - **Source**
- **Main –** Integration branch
    - **Source**
    - **Other Asset Folders**

Keep the following recommendations in mind when you work with a release branch:

- **When to branch**: If your development teams overlap on files and components, use team branches to isolate each team's work. You might also choose to create feature branches beneath the team branches.
- **When not to branch**: If your source tree is organized by component, and you are confident that there will not be breaking interface changes or a large number of check-in conflicts across your teams, then you probably do not need team branches.

- **Permissions in branch**: Read/write for developers on the team, read-only for everyone else
- **Build frequency on branch**: Continuous integration builds
- **Testing focus in branch**: Feature and quick feedback testing

The team branches should be used to allow teams to complete development tasks in parallel. This can be useful to isolate teams from breaking changes originating on other teams, or to allow teams to work towards unique milestones. All active development should occur in the team branches and then integrated into the Main branch. You may also want to create feature branches beneath each team branch.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

How do I use branching to reduce conflicts between features?

Use feature branches to improve the integration and stabilization of breaking changes across features.

The following is an example of what your branch structure might look like after you have created feature branches:

- **Development –** Container for feature branches
    - **Feature A –** Feature branch
        - **Source**
    - **Feature B –** Feature branch
        - **Source**
    - **Feature C –** Feature branch
        - **Source**
- **Main –** Integration branch
    - **Source**
    - **Other Asset Folders**

Keep the following recommendations in mind when working with a release branch:

- **When to branch**: Feature teams often have source file overlap, increasing the potential for build breaks and check-in conflicts. If you are experiencing these problems, consider feature branches for each feature to provide feature isolation. You can create these off a main branch for small teams or off a team branches for larger teams.

- **When not to branch**: If you are only creating continuous integration builds, or your daily builds are already predictably stable you might not need the extra overhead of feature branches.
- **Permissions in branch**: Read/write permissions for developers on the feature branch, read-only for everyone else.
- **Build frequency on branch**: Continuous integration builds.
- **Testing focus in branch**: Feature and quick feedback testing.

Use the feature branch to allow each feature to be developed in parallel. Perform all active development in the feature branches and then integrate your code into the Main branch.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see “Branching and Merging Team Foundation Source Control”: http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

What are proven practices for branch structure?

When branching, consider the following guidelines:

- Structure your branch trees so that you only need to merge along the hierarchy (up and down the branch tree) rather than across the hierarchy. Branching across the hierarchy requires that you use a baseless merge which requires more manual conflict resolution.
- The branch hierarchy is based upon the branch parent and branch child, which may be different than the physical structure of the source code on disk. When planning your merges, keep the logical branch structure in mind rather than the physical structure on disk.
- Do not branch too deeply. Because there is latency in merging branches, a deep branching structure can mean that changes in a child branch may take a very long time to propagate to the main branch. This can negatively impact project schedules and increase the time to fix bugs.
- Branch at a high-level and include configuration and source files.
- Allow your branching structure to evolve over time.

- Do not branch unless your development team needs to work on the same set of files concurrently. If you are unsure, you can label a build and create a branch from that at a later point. Merging branches, especially if there are significant changes between them, can be costly.
- Merging requires one or more developers to execute the merge and sort out conflicts. The merged source must be thoroughly tested as it is not uncommon to make bad merge decisions that can destabilize the build.
- Merging across the branch hierarchy is especially difficult and requires you to manually handle many conflicts that could otherwise be handled automatically.

More Information

- For additional Branching and Merging guidance, see the Team Foundation Server Branching Guidance project on CodePlex: http://www.codeplex.com/BranchingGuidance
- For an introduction to branching and merging, see "Branching and Merging Primer": http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- For additional descriptions of how to branch and merge in Visual Studio 2005, see "Branching and Merging Team Foundation Source Control": http://msdn2.microsoft.com/en-us/library/ms181423(VS.80).aspx

What is the difference between branching and labeling?

Branches are designed to split a collection of files into separate development paths. You use branching to isolate two sets of file versions so that you can evolve the two paths separately. For example, you can use branches to manage and maintain older released versions of your application. You can also use branches to isolate risky code changes that span multiple developers so that team impact is minimized. However, if you need to isolate only a single developer, use workspaces to reduce branch and merge complexity.

Labels are used to tag a file or collection of files with a distinct notation so that it may be retrieved later. For instance, you can use labels to mark daily builds to ease future retrieval for example if you want to address an issue related to that release.

More Information

- For more information on the Branch command, see "Branch Command": http://msdn2.microsoft.com/en-us/library/d73s8b27(VS.80).aspx
- For more information on the Label command, see "Label Command": http://msdn2.microsoft.com/en-us/library/9ew32kd1(VS.80).aspx

What is "path space" branching model?

Branches are created within the path structure of the repository, similar to file copy operations. This is different than the pin-and-share mechanism used by Visual SourceSafe. This makes it easier to maintain old versions of your software, because it's simpler to merge changes from one branch to another or make changes in two branches at once.

Team Foundation Server Version Control does not make separate copies of the file content when you branch it, so it does not consume a lot of additional space in the source control database. A branch consists of a pointer in the database to the source content in the list of deltas that modify it from the base version.

More Information

- For an introduction to branching and merging, see “Branching and Merging Primer”: http://msdn2.microsoft.com/en-us/library/aa730834(VS.80).aspx

How does the TFS promotion model work?

A promotion model is the means by which a change is propagated from one branch to another. Visual SourceSafe also had branching and merging but in practice customers used pinning and sharing as a crude promotion model. Team Foundation Server Version Control uses branching and merging to support promotion modeling. You can use multiple branches to isolate specific versions of the application you're working on. You can propagate a change by merging from files in one branch to the files in another branch.

More Information

- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on merging, see “How to: Merge Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- To learn more about the differences between Visual SourceSafe and Team Foundation Server Version Control, see Visual Source Safe vs. Team Foundation Server: http://msmvps.com/blogs/vstsblog/archive/2005/07/02/56401.aspx

How do I merge two branches together?

To perform a merge you can use the merge functionality in Source Control Explorer or you can use the merge command line. You can merge by changeset, label, date, or version.

You may find that your branching structure becomes deeper than one level. If this happens, keep in mind that it is only possible to merge one level at a time. For instance, imagine the following source tree structure:

- **Development –** Container for isolation development branches.

  - **Feature A**
    - **Source**
  - **Feature B**
    - **Source**
- **Main** – Main integration build branch
  - **Source**
  - **Other Asset Folders**
- **Production –** Branch containing the code you are currently locking down to release
- **Maintenance** – Container for production release branches, which needs isolated parallel development, may be for patches or hot fixes.
  - **Release 1**
    - **Source**
  - **Release 1 Service Pack 1**
    - **Source**

When the source code contained in the **Production** branch changes, you may want to eventually merge that change into a feature branch, for instance to incorporate a bug fix into the latest feature code. It is not easy to merge directly from **Production** into **Feature B**, instead you merge from **Production** into its logical parent, **Main**, and then from **Main** into its logical child **Feature B**. If you need to merge between branches that do not have a direct parent child relationship you need to perform a baseless merge.

More Information

- To learn how to merge files and folders, see How to: Merge Files and Folders: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- To learn more about baseless merge, see the /baseless option in "Merge Command": http://msdn2.microsoft.com/en-us/library/bd6dxhfy(VS.80).aspx

Can I merge across Team projects?

Yes, There is nothing special about merging across team projects. Regular merges work in this scenario. The Team Project Creation Wizard enables you to create a team project as a branch of a different one.

More Information

- For more information on merging, see "How to: Merge Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx

What is a baseless merge?

The branch command will create a relationship between two folders which you can leverage to perform merges of code between the branches. Branches that are not a direct child or a direct parent of the merge target do not have this relationship. A baseless merge "performs a merge without a basis version. That is, allows the user to merge files and folders that do not have a branch/merge relationship. After a baseless merge, a merge relationship exists and future merges do not have to be baseless".

A baseless merge will result in many more conflicts than a normal merge. After the first baseless merge, future merges between these branches will no longer be baseless and conflicts should be at a minimum.

Baseless merges can only be created from the command line. Even after the first baseless merge, when a relationship has been created between the branches, future merges still need to be created from the command line.

More Information

- To learn how to merge files and folders, see How to: Merge Files and Folders: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- To learn more about baseless merge, see the /baseless option in “Merge Command”: http://msdn2.microsoft.com/en-us/library/bd6dxhfy(VS.80).aspx
- For more information about other merging options please see "Merge Command" at http://msdn2.microsoft.com/en-us/library/bd6dxhfy(VS.80).aspx
- Team Foundation Server Branching Guidance: http://www.CodePlex.com/BranchingGuidance

What is the code promotion model?

Code promotion is a strategy that uses branches to promote code through some stabilization phases. For instance, you may have a Dev branch in which your team does active development, a Main branch in which your team is integration and testing, and a Production branch that your team uses for final release stabilization.

More Information

- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx

What is the difference between the logical and physical view of branches?

The physical view is the hierarchy that appears within your source tree. For example:

- $/MyTeamProject
  - Development
    - Source
  - Main
    - Source
  - Maintenance
    - Release1
      - Source
    - Release2
      - Source

Development, Main, Release1, and Release2 are branches that directly contain source code. Maintenance is a folder that contains multiple branches.

The logical view is the hierarchy of parent and child branches as they were created. This hierarchy is not shown in the source tree but can be visualized based upon how each branch was created. For example:

- Main
  - Development
  - Release1
  - Release2

The logical view is important because merging is easiest when you do it up and down the logical hierarchy. If you execute a merge across the hierarchy you have to do a baseless merge which requires use of the commandline and significantly more conflict resolution.

More Information

- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- To learn how to merge files and folders, see How to: Merge Files and Folders: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- To learn more about baseless merge, see the /baseless option in "Merge Command": http://msdn2.microsoft.com/en-us/library/bd6dxhfy(VS.80).aspx

If I use the code promotion model, how often should I merge?

Merge frequency depends upon your branch structure. For instance, a common branch structure is as follows:

- **Development –** Container for isolation development branches.
  - **Feature A**
    - **Source**
  - **Feature B**
    - **Source**
- **Main –** Main integration build branch
  - **Source**
  - **Other Asset Folders**
- **Production –** Branch containing the code you are currently locking down to release
- **Maintenance –** Container for production release branches, which needs isolated parallel development, may be for patches or hot fixes.
  - **Release 1**
    - **Source**
  - **Release 1 Service Pack 1**
    - **Source**

In this case active development is occurring in the feature branches, and you have two merge frequency decisions to make:

1. How often should feature teams merge changes from main into their feature branch?

2. How often should feature teams merge their changes into the main branch?

The following merge frequency is a proven strategy to reduce breaking changes and reduce migration times:

1. Feature teams should merge their changes into main as soon as the feature is complete.
2. Feature teams should merge changes from main on a regular basis to ensure integration compatibility of their features when they are complete. Ideally you should merge once per day, and not go more than a couple of days between merges.

More Information

- To learn how to merge files and folders, see How to: Merge Files and Folders: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx

Checkin and Checkin Policies

- *What is a changeset?*
- *What's a check-in policy?*
- *When and how can I override check-in policy?*
- *How do I enforce a policy?*
- *How do I use a check-in verification system?*
- *If I modify file names or delete files on the disk does version control get out of sync?*
- *How does automatic conflict resolution work?*
- *How do I resolve conflicts manually?*
- *How do I avoid conflicts?*

What is a changeset?

A change-set represents the set of changes associated with a check-in. All changes in the change-set are applied to TFS in an atomic fashion when you check-in the change-set. Change-sets are identified by unique, sequentially increasing, numbers.

Later in time you can retrieve a change-set to view details of what files changed, check-in comments, and associated work items. You can also retrieve the file versions associated with the change-set if you need to review, test, or build, with the old change.

More Information

- For more information on retrieving changesets, see “How to: Retrieve Old Versions of Files from Changesets”: http://msdn2.microsoft.com/en-us/library/ms181416(VS.80).aspx

What's a check-in policy?

You can use check-in policies to enforce coding standards at check-in time. For instance, you can automatically run FxCop static code analysis on code as it is checked in to ensure that the relevant rules are satisfied.

The following check-in policies are available by default:

- **Code Analysis.** Requires that code analysis be run before check-in.
- **Test Policy.** Requires that check-in tests be completed before check-in.
- **Work Items.** Requires that one or more work items be associated with the check-in.

You can create a custom check-in policy to perform checks that are not available by default. For instance, you can disallow code patterns such as banned API calls, you can force the use of check-in comments, or you can check against your team's style guidelines.

More Information

- To learn how to customize a check-in policy, see “Walkthrough: Customizing Check-in Policies and Notes”, http://msdn2.microsoft.com/en-us/library/ms181281(VS.80).aspx
- To see sample code that will disallow selected patterns on checkin, see Checkin Policy to Disallow Certain Patterns: http://blogs.msdn.com/jmanning/archive/2006/02/02/523125.aspx
- To see sample code that will enforce comments on checkin, see Sample Checkin Policy: Make Sure the Comment Isn't Empty: http://blogs.msdn.com/jmanning/archive/2006/01/21/515858.aspx
- To learn how to register a new checkin policy, see I've Made a New Check-In Policy! How Do I Add It?: http://blogs.msdn.com/jmanning/archive/2006/02/07/526778.aspx

When and how can I override check-in policy?

You can override a check-in policy by clicking the, "override policy failure and continue check-in" checkbox. Anyone with permission to check in files can override a check-in policy.

If you would like to detect when a member of your team overrides check-in policy you can use the Team Foundation Eventing Service to hook check-in events.

More Information

- To learn more about overriding a check-in policy, see “How to: Override a Check-in Policy”: http://msdn2.microsoft.com/en-us/library/ms245460(VS.80).aspx
- To learn more about the Team Foundation Eventing Service, see “Eventing Service”: http://msdn2.microsoft.com/en-us/library/bb130154(vs.80).aspx

How do I enforce a policy?

Team Foundation Version Control will not prevent someone from overriding a policy. However, you can use the following steps to detect if a policy has been overridden:

- Use the Team Foundation Eventing Service (from the Team Foundation Core Services API) for hooking in to check-in events.
- Write a Notify method that parses the details of the changeset and then react to it if an override has occurred

Alternatively, you can manually scan changeset history to discover policy overrides.

More Information

- To learn how to receive an e-mail when a policy has been violated, see: http://blogs.infosupport.com/marcelv/archive/2005/10/18/1635.aspx

How do I use a check-in verification system?

You can use Team Foundation Version Control check-in policies as a check-in verification system. Team Foundation ships with check-in policies to ensure the following is true before a check-in can be committed:

- A work item is associated with the change
- Unit tests have all passed
- Static analysis has run cleanly

You can create your own check-in requirements by creating new policy plug-ins. Your plug-in can ensure that code matches your team's coding standards, has run build verification tests, or any other requirement that is critical to your team's needs.

More Information

- To learn more about creating and customizing check-in policies, see Walkthrough: Customizing Check-in Policies and Notes: http://msdn2.microsoft.com/en-us/library/ms181281(VS.80).aspx

If I modify file names or delete files on the disk does version control get out of sync?

Yes. If you accidentally delete files or folders the Team Foundation Server still thinks you have the latest versions locally. This means that the get latest version command will not add the deleted files back onto your disk. Use the force get command in order to restore the files or folders.

If you need to rename files or delete files or folders do so through the source explorer so the server stays synchronized with your local changes.

More Information

-

How does automatic conflict resolution work?

A conflict can occur due to a pending change, merge, or get operation. When you select to resolve the conflict you can choose to have Visual Studio resolve the conflict automatically. Automatic conflict resolution only works on non-binary files which have no overlapping changes (for example changes applied to the same line of code). In this case changes for both file versions are merged into a new file version.

If automatic conflict resolution does not work you can choose to only accept changes from one of the files, or you can perform a manual merge by using the graphical compare and differencing tools provided by VSTS.

More Information

- For more information on resolving conflicts, see How to: Resolve Conflicts: http://msdn2.microsoft.com/en-us/library/ms181433(VS.80).aspx

How do I resolve conflicts manually?

To resolve merge conflicts use the Visual Studio merge tool. If a conflict is detected during a merge you can resolve conflicts automatically or resolve them manually. If you choose to resolve the conflict manually you can keep the source changes, keep the target changes, or resolve the conflict in the merge tool.

You may need to resolve conflicts when merging changes between branches, getting files into your workspace, or checking in new versions of files. There are three conflict types:

- **Version**. The file has evolved along divergent paths. This could be the result of a file edit, rename, delete, or undelete.
- **File Name Collision**. Two or more items are trying to occupy the same path.
- **Local Overwrite**. Only occurs during a get operation, if you are trying to overwrite a local, editable file.

Most conflicts can be resolved automatically, version conflicts are the only conflict type that may result in a manual merge operation. Manual merge is most common in the following scenarios:

- A file has been edited in both branches with changes to the same lines of code.
- A baseless merge is being conducted in which the branch file relationships are not known to Team Foundation Server.

The merge tool shows you the details for each conflict and allows you to choose which change you want to keep in the final merged file. You can choose to keep the source change, the destination change, integrate both changes, or you can manually modify the final version by typing directly into the file.

When you have resolved every conflict in the file save the final version as a pending change to be checked into the target branch.

Be careful when merging, it is easy to make mistakes that may result in build instability. Once you are done merging, compile the resulting source and run unit tests to test for major breaks.

More Information

- For more information on resolving conflicts, see How to: Resolve Conflicts: http://msdn2.microsoft.com/en-us/library/ms181433(VS.80).aspx

How do I avoid conflicts?

Most conflict can be automatically resolved by Team Foundation Version Control. The scenarios in which you're most likely to need a manual merge, and therefore are worth avoiding, are scenarios in which there are overlapping changes (to the same lines of code) or more than two developers working on a file at the same time.

To see pending changes use the following steps:

- From within the Source Control Explorer right click on the solution, project, folder, or file for which you want to see pending changes
- Select View Pending Changes

This method will show you all of the pending changes within the scope you've selected.

If you use the command line checkout tool it informs you which other users have checked out this file including information about the nature of their pending changes. In the following example, a file called Math.cs is checked out and Team Foundation provides a notification that two other users are working on the file (James and Sally). The notification indicates exactly what type of pending change each user has in their local workspace (Edit and Rename) and confirms that the local version of the file, the current workspace version, is not based on the latest repository version.

```
c:\dev\projects\calc\src>tf edit math.cs
ExplorerScc.cs

$/MathProject/dev/calc/src/math.cs:
  opened for edit in Workspace21;contoso\james
  opened for rename in WS24;contoso\sally
  newer version exists in the repository
```

You should communicate regularly with other developers to indicate the nature of your changes on a particular file to help avoid overlapping changes that need to be manually merged. Coordination is a good way to make sure you do not work on the same part of the code, but otherwise there is nothing wrong with two or more people editing different parts of the same file.

More Information

- For more information on viewing pending changes in your workspace, see "How to: View and Manage All Pending Changes in Your Workspace": http://msdn2.microsoft.com/en-us/library/ms181400(VS.80).aspx
- For more information on viewing pending changes in other workspaces, see "How to: View Pending Changes in Other Workspaces": http://msdn2.microsoft.com/en-us/library/ms181401(VS.80).aspx

Checkout, Get, and Lock

- *How do I find out who was the last developer to modify a file?*
- *How does get work?*
- *What's the difference between shared and exclusive checkout?*
- *When should I use the lock command?*
- *What are the different lock types that TFS supports?*

How do I find out who was the last developer to modify a file?

You can do this by examining the file's history. To do so, right-click the file in **Solution Explorer** and click **View History**. This displays the **History** window which shows the list of modifications and users who made those modifications. The last developer to modify the file is shown at the top of the list. You can also view version history for a file from the command line by using the History command from the tf.exe command line tool.

More Information

- For more information on file history, see "How to: View Historical Data": http://msdn2.microsoft.com/en-us/library/ms181415(VS.80).aspx

How does get work?

Get is used to synchronize the files on your machine with the files on the Team Foundation Server. When you perform a "get" operation, Visual Studio conducts the following steps. Note that the following sequence assumes that you have installed TFS proxy in order to boost performance:

- The server determines which files are out of date based on the fact that it knows which versions are on your disk.
- The server then tells the client what it needs to do to update its workspace.
- The client downloads, moves, and deletes files as necessary and notifies the server of which operations it accomplished.

Get does not mark any files for editing and it will not, by default, overwrite any files you have checked out for editing.

More Information

- For more information on the get command, see "Get Command": http://msdn2.microsoft.com/pt-br/library/fx7sdeyf(VS.80).aspx

What's the difference between shared and exclusive checkout?

Team Foundation Source Control supports shared and exclusive checkouts. With an exclusive checkout, nobody else can check out a file until you check it back in. This can lead to bottlenecks in the development process.

By default, Team Foundation also enables multiple users to check out the same source-controlled item concurrently. This is referred to as shared checkout. With this model, multiple developers can be working on copies of the same source file in their own workspaces. TFS knows which version is in your workspace and you must resolve conflicts prior to checkin.

In most collaborative development environments, the probability that you will make a change in your workspace that conflicts with a pending change in another workspace or vice versa, is unlikely. A great majority of workspace conflicts that do occur are resolved automatically by Team Foundation. For conflicts that cannot be resolved automatically, you can use the Resolve command to safely decide which change, yours or someone else's, you want to keep.

More Information

- For more information on locks, see “How to: Lock and Unlock Folders or Files”: http://msdn2.microsoft.com/en-us/library/ms181420(VS.80).aspx

When should I use the lock command?

The lock command can be used to prevent other developers from checking out or checking in their changes until you unlock the file. You should only lock a file, if you're concerned that there will be a conflict resulting in a complicated manual merge operation. Since most conflicts can be resolved automatically, you should use the lock command sparingly, if at all.

More Information

- For more informational on lock types, see: http://msdn2.microsoft.com/en-us/library/ms181419(VS.80).aspx

What are the different lock types that TFS supports?

Team Foundation provides two types of locks: check-in and checkout. A check-in lock is less restrictive than a check-out lock. When you apply a check-in lock, users can continue to make local changes to the item in other workspaces. The changes cannot be checked in until you explicitly remove the check-in lock from the workspace or you check the file in.

A check-out lock is more restrictive than a check-in lock. When you apply a check-out lock to a source-controlled file or folder, users can neither check out the file for edit nor check in pre-existing pending changes. You cannot acquire a check-out lock if there are currently any pending changes to an item.

More Information

- For more informational on lock types, see: http://msdn2.microsoft.com/en-us/library/ms181419(VS.80).aspx

Distributed/Remote Development

- *How do I work off-line?*
- *How do I optimize for distributed team development?*
- *What is the version control proxy?*
- *How do I optimize version control proxy performance?*

How do I work off-line?

Off-Line is not supported natively by Team Foundation Version Control. If you want to work off-line you must adhere to strict workflow:

1. Manually remove read-only flag
2. Edit files
3. Add or delete files
4. Run the Team Foundation Power Tool (TFPT) online command

You must not rename any files while you're off-line.

1. Manually remove read-only flag

By default, all files in the workspace that have not been checked out are marked as read-only. When you're working without a server connection, you must manually remove the read-only flag from files before editing or deleting them. You can do this by right clicking the file in the Windows Explorer, click properties, uncheck the read-only box and click okay. Or you can use the DOS command “attrib -r”

2. Edit files

You may now freely edit any files for which you have removed the read-only flag.

3. Add or delete files

You may add files or delete files for which you have removed the read-only flag. Do not rename files, as the TFPT online tool cannot distinguish a rename from a deletion paired with an add. Note that you have to specify an option to the Tfpt online command to get it to look for deletes, as this is a more time consuming operation.

4. Run the TFPT online command

When you are back online, run the TFPT online command by typing "TFPT online" at the commandline. This command scans your workspace for writable files and determines what changes should be pended on the server. If you have deleted any files, use the /deletes commandline option. This tells the tool to scan for deleted files in your local workspace as well. The tool then displays the online window from which you can choose which changes to pend into the workspace.

More Information

- You can download the TFPT online tool from MSDN: http://www.microsoft.com/downloads/details.aspx?FamilyId=3F21144A-4E98-4CE0-830F-D1F3E8AC9D67&displaylang=en
- To learn more about Team Foundation Power Tools, see Power Tools: tfpt.exe: http://blogs.msdn.com/buckh/archive/2005/11/16/493401.aspx

How do I optimize for distributed team development?

If you are supporting an office with multiple clients connecting to a remote Team Foundation Server, install and configure Team Foundation Server Proxy in the Remote Office. This improves performance by caching source control files in the remote office's proxy.

If you are supporting a single client connecting to a remote Team Foundation Server, configure the client to connect directly to the Team Foundation Server.

More Information

- To learn more about the Team Foundation Server Proxy, see MSDN: Team Foundation Server Proxy and Source Control, http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx
- To learn more about the TFS Proxy File Cache Server, see the MSDN Webcast: http://msevents.microsoft.com/CUI/WebCastEventDetails.aspx?culture=en-US&EventID=1032291120&CountryCode=US

What is the version control proxy?

Client and server communication in TFS uses HTTP. The TFS Version Control proxy is installed on a server at remote locations (team locations separated by a wide-area connection from the TFS database) to boost performance and to avoid unnecessary roundtrips to the server. The proxy caches copies of source controlled files in the remote location away from the central TFS server. When a file is not in the local cache, the file is downloaded by the proxy to the local cache from Team Foundation Server, before returning the files to the client.

Although remote access is the most common scenario for using the proxy, you can also use it anytime you want to take load off of the main server. For instance, if your server is bogged down by many local simultaneous requests to get the latest source, you can offload this work to the proxy. Every AT includes a proxy in it. So, generally there is nothing to be gained by putting a proxy in front of an AT.

More Information

- For more information, see "Team Foundation Server Proxy and Source Control" at http://msdn2.microsoft.com/en-us/library/ms252490(VS.80).aspx

How do I optimize version control proxy performance?

Consider the following approaches for optimizing proxy performance:

- Make sure caching is enabled and monitor the performance of your cache. Monitor the performance counters (installed by default) and event logs (for errors/warnings) on your proxy server on a periodic basis, to see how your proxy is performing. Note that TFS proxy saves cache performance statistics to an XML file called ProxyStatistics.xml and you can change the interval for saving these statistics. The ProxyStatistics.xml file is located in the App_Data folder in the proxy installation directory.
- Run a scheduled task to retrieve the latest files on a periodic basis to the proxy server. This helps to ensure that the latest version of the files are available in the proxy cache and to ensure that subsequent client requests for these files result in a cache hit.
- If you know that large files are going to be downloaded over a low bandwidth (< 3Mbps) network, then set the **executionTimeout** configuration to an appropriate value in Web.config. The default value is one hour <httpRuntime executionTimeout="3600"/>.

More Information

- For more information about examining TFS proxy performance, see http://msdn2.microsoft.com/en-us/library/ms252455(VS.80).aspx

Migration

- *How is TFS version control different than Visual Source Safe?*
- *How is the checkout model different from Visual SourceSafe?*
- *How should I migrate my source from VSS to TFS?*
- *How should I migrate my source from other version control systems?*

How is Team Foundation Server Version Control different than Visual SourceSafe?

Visual Source Safe is targeted at individual developers and small teams. Team Foundation Server Version Control is designed for professional developers and very large teams, with up to 2,500 users per server. Other major improvements with Team Foundation Server Version control include deep integration with Visual Studio 2005, and distributed team support supporting efficient file access over wide area networks. The major differences between the two systems include the following:

- **Code Churn.** Visual SourceSafe has no features that support code churn metrics. Team Foundation Server Version Control gives you code churn metrics for every file that you check into the data warehouse, and you can slice the data in any way that you'd like.
- **Check Out.** Visual SourceSafe always get the latest file or the pinned file from the database and overwrites your local version. Team Foundation Server Version Control makes your workspace version of the file writable, but does not overwrite your version with a version from the database.
- **File Locking.** Visual SourceSafe automatically applies an exclusive lock to a file when you check it out. By default, Team Foundation Server Version Control allows multiple developers to check out a file concurrently. Conflicts when they do occur can usually be automatically resolved.
- **Branching**. Visual SourceSafe uses a pin-and-share branching method. Pinning is not supported in Team Foundation Server Version Control. Team Foundation Server Version Control uses path-space branching. This makes it easier to maintain old versions of your software, because it's easy to merge changes from one branch to another or make changes in two branches at once.
- **Sharing.** Team Foundation Server Version Control does not support sharing. The migration of a shared file results in copying the file to the destination folder.
- **Workflow Integration.** Visual SourceSafe is a standalone version control system. Team Foundation Server Version Control is integrated with work item tracking and Team Foundation Server reporting.

More Information

- To learn more about the differences between Visual SourceSafe and Team Foundation Server Version Control, see Visual Source Safe vs. Team Foundation Server: http://msmvps.com/blogs/vstsblog/archive/2005/07/02/56401.aspx

How is the checkout model different from Visual SourceSafe?

When you checkout from Team Foundation Server Version Control the file you check out is marked writable in your workspace, but is not synchronized with the database server. If you want to get the latest version from the database server you need to first use the get command. In contrast, a Visual Source Safe check-out operation gets the file from the database server and marks it writable in your workspace.

Visual SourceSafe uses exclusive checkout by default, Team Foundation Server uses shared checkout by default. Shared checkout enables multiple developers to make changes to a file at the same time. Unlike in Visual SourceSafe, most conflicts in Team Foundation Server can be resolved automatically.

More Information

- For differences between Visual SourceSafe checkout and Team Foundation Server Version Control checkout, see Team Foundation vs. SourceSafe | Checking Out: http://blogs.msdn.com/korbyp/archive/2004/07/28/199720.aspx

How should I migrate my source from VSS to TFS?

Team Foundation Server ships with a VSS converter which allows you to migrate files, folders, version history, labels, and user information from a Visual SourceSafe database to Team Foundation Server Version Control.

The converter does have some limitations that are important to understand, for instance:

- Historical information about file sharing is not preserved. Team Foundation Server does not support sharing. The migration of a shared file results in copying the file to the destination folder.
- Historical information about branching is not preserved.
- Team Foundation Server does not support pinning. Pinned files are migrated by creating two labels.
- Time stamps associated with actions are not preserved during migration.

More Information

- For more information on how to prepare for migration, see: http://msdn2.microsoft.com/en-us/library/ms181246(en-us,vs.80).aspx
- For more information on how to perform the migration, see: http://msdn2.microsoft.com/en-us/library/ms181247(VS.80).aspx
- For more information on the limitations of the Visual SourceSafe converter, see: http://msdn2.microsoft.com/en-us/library/ms252491(VS.80).aspx
- For more information about the VSS Analyze tool, see http://msdn2.microsoft.com/en-us/library/ysxsfw4x.aspx.
- For more information about the VSSConverter migrate command, see http://msdn2.microsoft.com/en-us/library/ms400685(VS.80).aspx

How should I migrate my source from other version control systems?

You can manually export files from the version control system you're migrating from and then import them into Team Foundation Server Version Control. If you would like to preserve file history or other attributes from the version control system you're migrating from, you can use the Team Foundation Server Version Control object model to write your own migration tool.

Microsoft is currently working on a ClearCase converter, when it is ready it will be announced on the TFS Migration blog: http://blogs.msdn.com/tfs_migration

Component software has created a converter that is compatible with GNU RCS, CS-RCS, GNU CVS, Subversion (SVN) and Visual SourceSafe (VSS)

More Information

- You can download the Visual Studio Team Foundation Server SDK from http://go.microsoft.com/fwlink/?linkid=68586
- To learn more about Team Foundation Version Control extensibility, see "Walkthru:The Version Control Object Model": http://msdn2.microsoft.com/en-us/library/bb187335(VS.80).aspx
- For more information on the component software converter, see: http://www.componentsoftware.com/Products/Converter/

Project/Workspace Management

- *How should I organize my team projects?*
- *How should I manage dependencies between projects?*
- *What's a workspace?*
- *How do I use workspaces to isolate a developer?*
- *What are the proven practices for workspace mapping?*
- *What are the proven practices for managing shared components and code?*
- *How can I merge two team projects together?*
- *When should I create a new Team Project versus create a new Branch?*
- *How should I manage source code that is shared across multiple projects?*
- *How should I manage binaries that are shared across multiple projects?*
- *How should I organize my source tree?*

How should I manage my team projects?

Team projects are intended to represent the largest unit of work possible in your organization. Most often this will be a product.

Consider one of the following strategies for managing team projects:

- **Create one project per application**
- **Create one project per release**

One project per application
If you want to carry forward not only source code but work items and other Team Foundation Server assets between releases, consider using one project per application. When you use a single project for multiple versions of the application, all of the Team Foundation Server assets are carried forward automatically for you for the next release. When you're ready to release a new version of your application you can create a branch within the project to represent the release and isolate that code.

The following are reasons not to use one project per application:

- Releases in parallel are forced to share work items schema, check in policies, and process guidance.
- Reporting is harder since reports default to the entire project, you must add filtering by release.
- If you have hundreds of applications, each in its own project, you'll hit Team Foundation Server performance and scale limits.
- You will accumulate 'baggage' over multiple releases. Easiest way to clean house is to create a new project and branch code you want to carry forward into that project

One project per release
If you want each release to start fresh without carrying forward work items and other Team Foundation Server assets, consider using one project per release. When you use a

new project for each release you can modify work item schema, workflow, check-in policies, and other assets without impacting the old release. This can be especially useful if the old release will be maintained by a separate team such as a sustained engineering team who may have a different process and workflow than your main development team.

The following are reasons not to use one project per release:

- While it is very easy to move the source code from one project to another, it is difficult to move work items and other TFS assets from one project to another. Work items can only be copied one at a time to another project, if you want to copy sets of work items you'll need to write your own utility.
- If you have hundreds of applications and releases, each in its own project, you'll hit Team Foundation Server performance and scale limits.

Keep the following considerations in mind when choosing your strategy:

- Pick a structure you can live with, restructuring existing team projects is difficult.
- Source can be easily shared amongst team projects:
  - Branch source from one project to another
  - Map source from another project into your workspace
- Team Foundation Serveer can scale to ~500 projects using MSF Agile process or 250 projects using MSF CMMI process. If you create your own or customize an existing process keep in mind that work item schema has the largest impact on server scalability. A complex schema will result in a server capable of supporting fewer projects.

More Information

- For more information on using team projects, see "When to use Team Projects": http://blogs.msdn.com/ericlee/archive/2006/08/09/when-to-use-team-projects.aspx

How should I manage dependencies between projects?

If you need to reference an external assembly that is not built as part of your Visual Studio solution such as a third party assembly for which you only have the binary DLL, you have three choices:

- You can store the external assembly as a binary resource as part of your project. This works well if you plan to maintain the dependency for other projects who may want to reference it, or you are the only project who relies upon this dependency.
- You can reference an assembly on your build server by using either a virtual drive letter or Universal Naming Convention (UNC) path.
- You can store the set of external assemblies in a folder in a shared team project. This approach works well if there are many projects that wish to reference this shared dependency and your project is not the clear owner for it.
  - If you wish to receive updates to the binary instantly upon it being changed you can create a workspace mapping for this team project to your local

    - computer and you can set a file reference to the assembly within your local file structure.
    - If you wish to control the integration schedule for this dependency you can create a branch from the shared project into your project and then merge whenever you are ready to pick up changes.

In general you should avoid dependencies that cross team projects and try to have all the related/dependent solutions/projects under same team project. This reduces the need for build script customization. If you have a dependency, use project references to define it. You should avoid file references as they are more difficult to manage. Note that project to project references are supported within a solution. Also a project file (csproj, vjsproj, vcproj, vbproj, etc) can be included by multiple solutions (sln files). Therefore, a project can be shared by multiple separate solutions.

More Information

- For more information on project references, see "Project References": http://msdn2.microsoft.com/en-us/library/ez524kew(VS.80).aspx
- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on workspaces, see: http://msdn2.microsoft.com/en-us/library/ms181383(VS.80).aspx

What's a workspace?

A workspace is a client-side copy of the files and folders in the TFS version control server. Local pending changes are isolated within your workspace until you check them into the server as an atomic unit (referred to as a changeset). A single workspace can contain references to multiple team projects. You can use multiple workspaces to isolate files or versions for your use only. Note that workspaces are per machine, per user account. Therefore, you can have different workspace definitions for each computer you use. Also, you can have multiple workspaces on a single computer by using multiple user accounts to log onto the computer.

More Information

- For more information on workspaces, see: http://msdn2.microsoft.com/en-us/library/ms181383(VS.80).aspx

How do I use workspaces to isolate the work of a developer?

As a developer you can create two workspaces, one containing references to files and folders being worked on by the rest of the team, another containing files and folders that you want to isolate. You might want to isolate these files in order to evolve specific files in parallel with work that is happening elsewhere. For instance, this can be used to work on risky pending changes, or to conduct a code review.

More Information

- For more information about workspaces, see: http://msdn2.microsoft.com/en-us/library/ms181383(VS.80).aspx

What are proven practices for workspace mapping?

Workspace mapping is the means by which you map files and folders on the server to a location on your local hard drive.

Consider the following practices for creating workspace mappings:

- **Create mappings at the team project root level**. For new team projects, map your team project root ($/MyTeamProject) to a folder with the same name on your local drive for example C:\TeamProjects. Because mappings are recursive, your entire local folder structure is created automatically and will be exactly the same as the structure in source control.
- **On shared computers, use a unique local folder path**. Two users of a single computer cannot share the same workspace mapping. For example, you and a colleague cannot map the same team project ($/MyTeamProject) to the same folder on the local computer. Instead, create mappings beneath My Documents (although this leads to long location paths) or establish a team folder naming convention on the local computer (e.g. C:\TeamProjects\User1, C:\TeampProjects\User2 etc).
- **You may not need the entire tree**. To improve performance and reduce disk size requirements only map the files you need for your development project. In general you will only need the files and projects associated with the solution you will be working on.
- **Avoid using workspace mapping to support cross project dependencies**. In general you should avoid dependencies that cross team projects and try to have all the related/dependent solutions/projects under same team project. This reduces the need for build script customization. If you have a dependency, use project references to define it or branch the dependency from a shared project into your project. You should avoid file references as they are more difficult to manage. The exception is when the dependent project is developed in parallel and you need real-time changes. In this case, you can consider the using workspace mapping. You may still consider branching to create a buffer of isolation, if the dependent code is causing too many breaking changes

More Information

- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx
- For more information on editing a workspace, see “How to: Edit a Workspace”: http://msdn2.microsoft.com/en-us/library/ms245466(VS.80).aspx

When should I create a new Team Project versus create a new Branch?

Create a branch if you want to isolate the code that developers are working on within a project but want to shares all other TFS assets such as work items, process guidance, and reports.

Create a new Team Project if you want to use different TFS assets such as work items, process guidance, and reports. Create a new Team Project if you want to carry forward your code but don't want to carry forward any other TFS assets.

More Information

- For more information on branching, see "How to: Branch Files and Folders": http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on adding a new project, see "How to: Add a Project or Solution to Source Control": http://msdn2.microsoft.com/en-us/library/ms181373(VS.80).aspx

How should I manage source code that is shared across multiple projects?

Managing shared source involves two decision points.

1. Where do you want to store the source?
2. How do you want your team to access the shared source?

Options for storing the source:

- If the shared source is clearly owned by a particular team, store it in their team project.
- If the shared source has no clear ownership, then create a team project specifically for the shared code.

Options for using the source in another project

- If you need to stay in sync with shared code at all times, map the source from shared location into local workspace on client machines.
- If you only need to stay in sync periodically, branch from the shared location into the consuming team project. Every time you perform a merge from shared to consuming project you will pick up latest source.

Whether you use workspace mapping or branching, use naming conventions that make it clear where the shared source location is in your project, for instance:

- **Main** - container for all assets you need to ship the product
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code –** container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests

    - **Lib** - container for binary dependencies
  - **Docs** - container for documentation that will ship with the product
  - **Installer** - container for installer source code and binaries
  - **Builds** - container for team build scripts
  - **Tests** - container for test team test cases

More Information

- For more information on project references, see “Project References”: http://msdn2.microsoft.com/en-us/library/ez524kew(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on workspaces, see: http://msdn2.microsoft.com/en-us/library/ms181383(VS.80).aspx

How should I manage binaries that are shared across multiple projects?

Managing shared binaries is similar to managing shared source, you must decide where you want to store the binaries and how you want your team to access the binaries.

Options for storing the binaries:

- If the shared binaries are clearly owned by a particular team, store it in their team project.
- If the shared binaries have no clear ownership, then create a team project specifically for the shared binaries.

Options for using the binaries in another project

- Usually shared binaries are updated only periodically. If this is the case, branch from the shared location into the consuming team project. When the binaries change, you can execute a merge to get the latest version.
- If you need to stay in sync with the shared binaries at all times, map the source from the shared location into a local workspace on client machines.

Whether you use workspace mapping or branching, use naming conventions that make it clear where the shared binary location is in your project, for instance:

- **Main** - container for all assets you need to ship the product
  - **Source** - container for everything you need to build
    - **Code** - container for source code
    - **Shared Code –** container for source code that is shared from other projects
    - **Unit Tests** - container for unit tests
    - **Lib** - container for binary dependencies
  - **Docs** - container for documentation that will ship with the product
  - **Installer** - container for installer source code and binaries
  - **Builds** - container for team build scripts
  - **Tests** - container for test team test cases

More Information

- For more information on project references, see “Project References”: http://msdn2.microsoft.com/en-us/library/ez524kew(VS.80).aspx
- For more information on branching, see “How to: Branch Files and Folders”: http://msdn2.microsoft.com/en-us/library/ms181425(VS.80).aspx
- For more information on workspaces, see: http://msdn2.microsoft.com/en-us/library/ms181383(VS.80).aspx

How should I organize my source tree?

Your source tree structure consists of a combination of folder structure, file structure, and branch structure. Within your main branch the following folder and file structure has been proven to work for teams of various sizes:

- **Main** - container for all assets you need to ship the product
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code –** container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests
        - **Lib** - container for binary dependencies
    - **Docs** - container for documentation that will ship with the product
    - **Installer** - container for installer source code and binaries
    - **Builds** - container for team build scripts
    - **Tests** - container for test team test cases

Any branches that you create off of main will copy this folder and file structure into the new branch. For example:

- **Development** – Development branch
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code –** container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests
        - **Lib** - container for binary dependencies
- **Main** – Integration branch
    - **Source** - container for everything you need to build
        - **Code** - container for source code
        - **Shared Code –** container for source code that is shared from other projects
        - **Unit Tests** - container for unit tests
        - **Lib** - container for binary dependencies
    - **Docs** - container for documentation that will ship with the product
    - **Installer** - container for installer source code and binaries
    - **Builds** - container for team build scripts
    - **Tests** - container for test team test cases

More Information

- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx

Shelving

- *What is shelving?*
- *What is a shelveset?*
- *When would I typically use shelving?*
- *How do I use shelving to back up my work?*
- *Why would I want to unshelve a shelveset?*

What is shelving?

The process of shelving a change or set of changes (also known as creating a shelveset), allows you to store pending changes under source control, without having to check them in. This means that you benefit from having the files on the (backed up) server but ensure that the potentially incomplete work on those files does not impact and break your regular builds.

More Information

- For more information on shelving pending changes, see “How to: Shelve and Unshelve Pending Changes”: http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

What is a shelveset?

A shelveset is a set of files that are saved, but are not yet ready to be committed to source control. Files can be shelved in order to save pending changes to the workspace, or to share with other team members for feedback. They can also be used to pass off partially completed work.

More Information

- For more information on shelving pending changes, see “How to: Shelve and Unshelve Pending Changes”: http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

When would I typically use shelving?

There are a number of common scenarios for using shelving:

- You are mid-way through making changes to a set of source code, when new higher priority work is allocated (for example an emergency bug fix is required). At this point you need to go back to a stable version of the code but you don't want to lose your changes. Before doing so you can shelve your code and easily retrieve it later on.
- You have not completed work at the end of the day but want to ensure that your current work is backed up on the server. By shelving your current changes, the changes are applied to the TFS and can be retrieved by you (or others) the day.

- You want to discuss or review your work-in-progress code with a remote team member. Rather than emailing the code to them, you can shelve the code and then have your remote colleague retrieve the files from the shelf.
- You want to pass partially finished work to another developer for completion.

More Information

- For more information on shelving pending changes, see "How to: Shelve and Unshelve Pending Changes": http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

How do I use shelving to back up my work?

At the end of the day, if your work on one or more source files is not complete you can shelve your code. This ensures that the source is uploaded to the server without checking in partially completed work.

More Information

- For more information on shelving pending changes, see "How to: Shelve and Unshelve Pending Changes": http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

Why would I want to unshelve a shelveset?

You might unshelve a shelveset to:

- Resume work on a set of files that you backed up previously by using a shelveset
- Integrate shelved pending changes into your work going forward.
- To review someone else's code
- Pick up another developer's partially completed work.

More Information

- For more information on unshelving pending changes, see "How to: Shelve and Unshelve Pending Changes": http://msdn2.microsoft.com/en-us/library/ms181404(VS.80).aspx

Additional Resources

- Visual Studio Team System Guidance Project: http://www.codeplex.com/VSTSGuidance

How To: Add a New Developer To Your Project in Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

How to add a new developer to your project

When you add a new developer to a project you need to grant the developer appropriate permissions to access your team foundation project and the associated Sharepoint project site. The new developer's account must also have sufficient permissions with SQL Server Reporting Services to be able to view reports such as those presented by the team site portal.

To grant access to the team project:

1. Logon to Visual Studio with an account that is a member of the Team Foundation Administrators application group.
2. Add the required project to Team Explorer (if it is not already listed)
3. Right-click the team project, point to Team Project Settings and click Group Membership.
4. Select Project\Contributors and click Properties and then add the new developer's account to this group.

Note that membership of the contributors group provides the typical set of permissions that a developer requires including the ability to add, modify and delete items within the team project and perform builds.

To grant access to the Sharepoint project site:

1. Access the team project site with an account that is a member of the SharePoint Administrator site group. The project site for a project called YourProject is located by default at http://server/sites/YourProject/default.aspx
2. Click Site Settings.
3. Click Manage Users beneath the Administration title.
4. Click Add Users.
5. Enter the account name in the form domain\useraccount of the new developer's account, select Contributor and then click Next.
6. Enter the developer's email address, and optionally type a message to welcome them to the site.
7. Click Finish.

Note that membership of the Contributors group allows the developer to view and add content to existing document libraries and lists. Membership of the Reader group, providing read only access to the site may be sufficient depending on the needs of the new developer.

To grant access to SQL Server Reporting Services
1. Logon to SQL Server Reporting Services administration Web site using an administrator account. The site is located at http://server/reports.
2. Click your team project name.
3. Click the Properties tab.
4. Click the Security tab.
5. Click New Role Assignment.
6. Enter your developer's Windows account name, select Browser and then click OK.

Note that membership of the Browser group enables the developer to view and subscribe to reports.

More Information

Security Groups

When adding a new dev to the team you need to add the dev's account to members of the approprate security groups to provide access to 3 distinct areas:

- The TFS Team Project
- The SHarePoint portal project site
- SQL Server Reporting Services

You can do this manually - by using the admin features of each tool OR you can use the TFS Administration Tool (currently at beta 1) available for download here:

TFS Server Level

At the TFS Server level, TFS defines three TFS application groups (not AD groups):

SERVER\Service Accounts - Contains TFS service accounts. By default, members can:
  - Administer shelved changes
  - Administer warehouse
  - Administer workspaces
  - Create workspaces
  - Trigger events

- View system synchronization information

Default Members

- SERVER\TFSSERVICE

SERVER\Team Foundation Administrators - Members can perform admin operations on the server. By default, members can:

- Administer shelved changes
- Administer warehouse
- Administer workspaces
- Alter trace settings
- Create a workspace
- Create new projects
- Edit server level information
- Manage process template
- Trigger events
- View server level information
- View system synchronization information

Default Members

- SERVER\Service Accounts
- Builtin\Administrator
- Builtin\Administrators

SERVER\Team Foundation Valid Users - Contains all users and groups that can access the server. By default, members can:

- Create a workspace
- View server level information

Default Members

- Project\Build Services
- Project\Contributors
- Project\Project Administrators
- Project\Readers

SERVER\Team Foundation Licensed Users - Users licensed to use the workgroup edition of TFS

- No permissions by default (??)

Default Members

- Builtin\Administrator

Team Project Level

At the team project level, TFS defines the following application groups (not AD groups)

TeamProject\BuildServices - build service permissions (for service accounts)

- Edit build quality
- Publish test results
- Start a build
- View project level information
- Write to build operational store

Default Members
- None

TeamProject\Contributors - can add modify and delete items in a team project

- Publish test results
- Start a build
- View project level information

Default Members - None

TeamProject\Project Administrators - can perform all operations in a team project

- Administer a build
- Delete this project
- Edit build quality
- Edit project level information
- Publish test results
- Start a build
- View project level information
- Write to build operational store

Default Members
- Builtin\Administrator

TeamProject\Readers - read access to the team project

- View project level information

Default Members
- None

How To: Automatically Run Code Analysis with Team Build in Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To shows you how to turn code analysis on for your team build. This will automatically run code analysis as part of your team build and report the results of the analysis in the build results.

Contents

- Objective
- Overview
- Summary of Steps
- Before You Begin
- Step 1 – Test the Build
- Step 2 – Turn Code Analysis on for the Build
- Step 3 – Test Code Analysis
- Additional Resources

Objective

- Learn how to run code analysis as part of the build, to validate code quality.

Overview

Team Build allows you to define build types for your project that enable a build server to compile your application and make it available on a network share. You can turn code analysis on for the team build so that each build automatically performs code analysis and reports the results to the build results report page. This How To walks you through the process of configuring Team Build to include code analysis as a build step.

Summary of Steps

- Step 1 – Test the Build
- Step 2 – Turn Code Analysis on for the Build
- Step 3 – Test Code Analysis

Before You Begin

Before you can turn code analysis on for team build, you must ensure you have the following pre-requisites in place:

- Your Team Foundation user ID must have permission to administer a build. Ask you administrator if you are unsure of your user's allowed permissions.

- A team build must already exist for your project. You can check this by looking at Team Build in the Visual Studio Team Explorer.

Step 1 – Test the Build

Start by testing the team build to make sure there are no problems before you turn code analysis on. You can do this by taking the following steps:

1. Open **Team Explorer** in Visual Studio
2. Expand the node for your team project
3. Expand the **Team Builds** node
4. Right Click an existing team build and select **Build Team Project**

Ensure the build has completed successfully. If there are build breaks or the build is not able to complete, fix these errors before moving to the next step.

Step 2 – Turn Code Analysis on for the Build

Once you have verified that the build is working properly, you can turn code analysis on. Use the following steps to turn code analysis on:

1. Open **Source Control Explorer.**
2. Expand your team project folder in **Source Control Explorer.**
3. Expand the **TeamBuildTypes** folder.
4. Select the team build folder for which you want to turn on code analysis.
5. Check out **TFSBuild.proj** file from source control. You may need to perform a **Get Latest Version** on the folder first.
6. Double click **TFSBuild.Proj** from **Source Control Explorer** to open it.
7. If you want all projects to run code analysis, regardless of project settings, change the **<RunCodeAnalysis>** tag to **Always**.
8. If you want to run code analysis on each project based on project settings, change the **<RunCodeAnalysis>** tag to **Default.**
9. If you are using per-project settings and want to turn code analysis on for a project:
    a. Open the solution in Visual Studio.
    b. Right click on the project in **Solution Explorer.**
    c. Select P**roperties.**
    d. Click on C**ode Analysis.**
    e. Check, **Enable Code Analysis.**
    f. Check out the .csproj file for the project from source control.
    g. Save the file by clicking the **save** button in toolbar while properties window is displayed.
    h. Check the .csproj file for the project back into source control.

Step 3 – Test Code Analysis

Once you have turn code analysis on for a team build, you can test to make sure it is working properly. Use the following steps to test code analysis for your build:

1. Right click the build type in **Team Explorer** and click **Build Team Project.**
2. When the build has completed, click on the link to the build log

3. At the end of the build log you should see some code analysis warnings. The warning ids will start with 'CA', such as:
    - MSBUILD : warning : CA2209 : Microsoft.Usage : No valid permission requests were found for assembly 'HelloWorldTest'. You should always specify the minimum security permissions using SecurityAction.RequestMinimum.
    - MSBUILD : warning : CA2210 : Microsoft.Design : Sign 'HelloWorldTest' with a strong name key.
    - MSBUILD : warning : CA1014 : Microsoft.Design : 'HelloWorldTest' should be marked with CLSCompliantAttribute and its value should be true.

Additional Resources

- For more information on code analysis tools, see “Guidelines for Using Code Analysis Tools”: http://msdn2.microsoft.com/en-us/library/ms182023(VS.80).aspx
- For more information on team build, see “Overview of Team Foundation Build”: http://msdn2.microsoft.com/en-us/library/ms181710(VS.80).aspx

How To: Create a Custom Report for Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To describes how to create a new custom report and then publish it to the team reporting portal in Team Foundation Server.

Contents

- Objectives
- Overview
- Summary of Steps
- Before You Begin
- Step 1 – Create a new Reporting Project
- Step 2 – Create the data sources
- Step 3 – Create a new report in your project
- Step 4 – Modify the report
- Step 5 – Deploy the report to your Team Foundation Server
- Step 6 – Test the report
- Additional Resources

Objectives

- Create a reporting project in Visual Studio
- Create a new custom report in the reporting project
- Publish the new report to the report server

Overview

The reports that ship with VSTS are SQL Server Reporting Services reports. You can amend these reports or create your own custom reports by using the SQL Server 2005 Reporting Services Designer inside Visual Studio (Business Intelligence Development Studio), which ships with the SQL Server 2005 client tools. To create a custom report, you create a Report Project in Visual Studio, and create data sources to connect to the TFS relational database and OLAP database.

Summary of Steps

- Step 1 – Create a new Reporting Project
- Step 2 – Create the data sources
- Step 3 – Create a new report in your project
- Step 4 – Modify the report

- Step 5 – Deploy the report to your Team Foundation Server
- Step 6 – Test the report

Before You Begin

Before you can customize a report for Team Foundation Server, you must ensure you have the following pre-requisites in place:

- You must have Business Intelligence Development Studio installed on the machine you will be using to customize the report. To test if it is installed, check Visual Studio to see if you have Business Intelligence Project type when you create a new project.
- Your user account must be a member of the Microsoft Analysis Server TfsWarehouseDataReaders security role on the data-tier server.
- Your user account must have administrator rights to the TFSWarehouse database on the data tier.
- Your user account must be a member of the SQL Server Reporting Services Publisher role on the application-tier server.

Step 1 - Create a new reporting project

Create a new reporting project so that you can add a new report to the project and customize it. Use the following steps to create a new reporting project in Visual Studio:

1. Click **File**, then click **New**, then click **Project.**
2. Select the **Business Intelligence Project** type.
3. Select the **Report Server Project** template.
4. Set your project's **Name** and **Location** and then click ok.

Step 2 - Create the data sources

In order to edit and publish the customized report you need to add data sources for the Team Foundation Server data warehouse and OLAP cube. Once these data sources are added to the Visual Studio project the report can pull data from the server.

Create the warehouse data source:

1. In the Visual Studio **Solution Explorer**, right click **Shared Data Sources** and click **Add New Data Source**.
2. Enter **TfsReportDS** into the **Name** text box on the **General** tab.
3. Select **Microsoft SQL Server** from the **Type** combo box.
4. Click on the **Edit…** button.
5. Fill in your data tier server name.
6. Select the the database **TFSWarehouse**.
7. Click the **OK** button twice to add the data source.

Create the OLAP data source:

1. In **Solution Explorer**, right click **Shared Data Sources** and click **Add New Data Source**.
2. Enter **TfsOlapReportDS** into the **Name** text box on the **General** tab.
3. Select **Microsoft SQL Server Analysis Services** from the **Type** combo box.
4. Click on the **Edit…** button.

5. Fill in your data tier server name.
6. Select the database **TFSWarehouse**.
7. Click the **OK** button twice to add the data source.

Step 3 - Create a new report in your project

Now that the data sources have been added to your project you can add a new report. Use the following steps to add a new report to your project and customize it:

1. In **Solution Explorer**, right click **Reports** and select **Add->New Item...**
2. Select **Report** template.
3. Name the report and click **ok**

Step 4 – Modify the report

Once you've added a report to the project you can modify it:

1. If the **Report Designer** doesn't open automatically, open the report for modification by double clicking it in the **Solution Explorer**.
2. Select the **Dataset** drop down and choose **<New Dataset...>**
3. Name the dataset, example TestDataSet.
4. Choose **TFSOlapReportDS (shared).**
5. Click **OK.**
6. Click the **...** button next to **Build**, just below the **Dataset** drop down and select Team System.

You are now set up to modify report by dragging measures and dimensions from the **Dataset** tree into the **Query Pane** and **Filter Pane.** You can modify report layout by clicking on the **Layout** tab. You can view a preview of your report by clicking on the **Preview** tab.

Step 5 - Deploy the report to your Team Foundation Server

After you've made modifications to the report, you can deploy it to your team project's reporting portal by using the following steps:

1. Right click on the report project in **Solution Explorer** and click **Properties**.
2. Ensure **OverwriteDataSources** is set to **false**.
3. Modify **TargetDataSourceFolder** to your team project name. Example: **TargetDataSourceFolder** = TestProject.
4. Modify **TargetReportFolder** to your team project name. Example: **TargetDataSourceFolder** = TestProject.
5. Modify **TargetDataSourceFolder** to http://<data-tier servername>/reportserver. Example: **TargetDataSourceFolder** = http://tfsrtm/reportserver.
6. Click **OK.**
7. Right click on the rdl file in **Solution Explorer** and click **Deploy.**
8. Watch the **Output Pane** for successful completion.

Step 6 – Test the report

After you've published the report to your team project's report server you can test it to make sure it was successfully deployed:

1. Right click on your team project and select **Show Project Portal.**
2. Click **Reports** in the Quick Launch bar on the left hand side of the portal web site.
3. Select the report you created.
4. Verify the report looks as you expected.

Additional Resources

- For more tutorials explaining how to work with reporting projects, see "Reporting Services Tutorials": http://msdn2.microsoft.com/en-us/library/ms170246.aspx
- For the MSDN article on editing reports, see "How to: Edit Reports in Report Designer": http://msdn2.microsoft.com/en-us/library/ms244655(VS.80).aspx
- For more information on security roles in the data-tier, see "Securing Access Through Analysis Services": http://msdn2.microsoft.com/en-us/library/ms174839.aspx
- For more information on security roles in the application-tier, see "Securing Reporting Services": http://msdn2.microsoft.com/en-us/library/ms157198.aspx

How To: Create a Risk Over Time Report for Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To describes how to create a new report that shows a trend-line of **Risk** work items over time and then publish it to the team reporting portal in Team Foundation Server.

Contents

- Objectives
- Overview
- Summary of Steps
- Before You Begin
- Step 1 – Create a new Reporting Project
- Step 2 – Create the data sources
- Step 3 – Create a new report in your project
- Step 4 – Modify the report
- Step 5 – Deploy the report to your Team Foundation Server
- Step 6 – Test the report
- Additional Resources

Objectives

- Create a reporting project in Visual Studio
- Create a new Risk Over Time report in the reporting project
- Publish the Risk Over Time report to the report server

Overview

The reports that ship with VSTS are SQL Server Reporting Services reports. You can amend these reports or create your own custom reports by using the SQL Server 2005 Reporting Services Designer inside Visual Studio (Business Intelligence Development Studio), which ships with the SQL Server 2005 client tools. To create a custom report, you create a Report Project in Visual Studio, and create data sources to connect to the TFS relational database and OLAP database. In this How To we create a simple report from scratch - the Risk Over Time report. This report tells us how many Risk work items there are over time.

Summary of Steps

- Step 1 – Create a new Reporting Project
- Step 2 – Create the data sources

- Step 3 – Create a new report in your project
- Step 4 – Modify the report
- Step 5 – Deploy the report to your Team Foundation Server
- Step 6 – Test the report

Before You Begin

Before you can customize a report for Team Foundation Server, you must ensure you have the following pre-requisites in place:

- You must have Business Intelligence Development Studio installed on the machine you will be using to customize the report. To test if it is installed, check Visual Studio to see if you have Business Intelligence Project type when you create a new project.
- Your user account must be a member of the Microsoft Analysis Server TfsWarehouseDataReaders security role on the data-tier server.
- Your user account must have administrator rights to the TFSWarehouse database on the data tier.
- Your user account must be a member of the SQL Server Reporting Services Publisher role on the application-tier server.

Step 1 - Create a new reporting project

Create a new reporting project so that you can add a new report to the project and customize it. Use the following steps to create a new reporting project in Visual Studio:

1. Click **File**, then click **New**, then click **Project.**
2. Select the **Business Intelligence Project** type.
3. Select the **Report Server Project** template.
4. Set your project's **Name** and **Location** and then click ok.

Step 2 - Create the data sources

In order to edit and publish the customized report you need to add data sources for the Team Foundation Server data warehouse and OLAP cube. Once these data sources are added to the Visual Studio project the report can pull data from the server.

Create the warehouse data source:

1. In the Visual Studio **Solution Explorer**, right click **Shared Data Sources** and click **Add New Data Source**.
2. Enter **TfsReportDS** into the **Name** text box on the **General** tab.
3. Select **Microsoft SQL Server** from the **Type** combo box.
4. Click on the **Edit…** button.
5. Fill in your data tier server name.
6. Select the the database **TFSWarehouse**.
7. Click the **OK** button twice to add the data source.

Create the OLAP data source:

1. In **Solution Explorer**, right click **Shared Data Sources** and click **Add New Data Source**.
2. Enter **TfsOlapReportDS** into the **Name** text box on the **General** tab.

3. Select **Microsoft SQL Server Analysis Services** from the **Type** combo box.
4. Click on the **Edit…** button.
5. Fill in your data tier server name.
6. Select the database **TFSWarehouse**.
7. Click the **OK** button twice to add the data source.

Step 3 - Create a new report in your project

Now that the data sources have been added to your project you can add a new report. Use the following steps to add a new report to your project and customize it:

1. In **Solution Explorer**, right click **Reports** and select **Add->New Item...**
2. Select **Report** template.
3. Name the report and click **ok**

Step 4 – Modify the report

Once you've added a report to the project you can modify it:

1. If the **Report Designer** doesn't open automatically, open the report for modification by double clicking it in the **Solution Explorer**.
2. Select the **Dataset** drop down and choose **<New Dataset...>.**
3. Name the dataset, example TestDataSet.
4. Choose **TFSOlapReportDS (shared).**
5. Click **OK.**
6. Click the **...** button next to **Build**, just below the **Dataset** drop down and select Team System.
7. Expand **Measures** in the **Dataset Tree.**
8. Expand **Current Work Item** in the **Dataset Tree.**
9. Drag **Current Work Item Count** into the main query window.
10. Collapse **Measures** in the **Dataset Tree.**
11. Scroll down to **Team Project** and drag it into the **Dimensions Grid.**
12. In the **Dimensions Grid** click on the **Filter Expression** cell and select your team project name. This filters the results to just your team project.
13. Expand the **Work Item** dimension in the **Dataset Tree.**
14. Drag **System_WorkItemType** from the **Dataset Tree** into the **Dimensions Grid.**
15. Drag **System_WorkItemType** from the **Dataset Tree** into the main query window, drop it in front of the work item count column.
16. Click on the **Filter Expression** cell in the **Dimensions Grid** and select **Risk** type. This filters the results to only risk work item types.
17. Expand the **Date** dimension in the **Dataset Tree.**
18. Drag the **Date** dimension value into the main query window, drop it in front of the work item type column.
19. Click the **Layout** tab.
20. Open the **Toolbox** window.
21. Drag the **Chart** item from the **Toolbox** to the layout grid.
22. Adjust the size of the chart to fit.
23. Right click on the chart, select **Chart Type->Line->Smooth Line.**

24. Open the **Datasets Pane**.
25. Expand your data set, for example, TestDataSet.
26. Highlight the graph so that the data, series and category drop targets appear.
27. Drop **Current_Work_Item_Count** into the **Drop Data Fields Here** drop target box.
28. Drop **Work_Item_Type** into the **Drop Series Fields Here** drop target box.
29. Drop **Date** into the **Drop Category Fields Here** drop target box.
30. Right click the graph and select **Properties.**
31. Enter a title for your graph.
32. Click the **Preview** tab to view what the report will look like.

Step 5 - Deploy the report to your Team Foundation Server

After you've created the Risk Over Time report, you can deploy it to your team project's reporting portal by using the following steps:

1. Right click on the report project in **Solution Explorer** and click **Properties**.
2. Ensure **OverwriteDataSources** is set to **false**.
3. Modify **TargetDataSourceFolder** to your team project name. Example: **TargetDataSourceFolder** = TestProject.
4. Modify **TargetReportFolder** to your team project name. Example: **TargetDataSourceFolder** = TestProject.
5. Modify **TargetDataSourceFolder** to http://<data-tier servername>/reportserver. Example: **TargetDataSourceFolder** = http://tfsrtm/reportserver.
6. Click **OK.**
7. Right click on the rdl file in **Solution Explorer** and click **Deploy.**
8. Watch the **Output Pane** for successful completion.

Step 6 – Test the report

After you've published the report to your team project's report server you can test it to make sure it was successfully deployed:

1. Right click on your team project and select **Show Project Portal.**
2. Click **Reports** in the Quick Launch bar on the left hand side of the portal web site.
3. Select the report you created.
4. Verify the report looks as you expected.

Additional Resources

- For more tutorials explaining how to work with reporting projects, see "Reporting Services Tutorials": http://msdn2.microsoft.com/en-us/library/ms170246.aspx
- For the MSDN article on editing reports, see "How to: Edit Reports in Report Designer": http://msdn2.microsoft.com/en-us/library/ms244655(VS.80).aspx
- For more information on security roles in the data-tier, see "Securing Access Through Analysis Services": http://msdn2.microsoft.com/en-us/library/ms174839.aspx
- For more information on security roles in the application-tier, see "Securing Reporting Services": http://msdn2.microsoft.com/en-us/library/ms157198.aspx

How To: Step Through Creating a Custom Check-in Policy for Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To steps you through creating, registering and applying a custom check-in policy for Team Foundation Server. Check-in polices allow you to run rules whenever a developer attempts to check-in a source file in order to ensure that the source file and check-in meet a specified set of criteria. As an example, this How To uses a comment check-in custom policy to enforce that check-in comments are supplied with all check-ins. To implement a custom check-in policy you create a class that derives from **PolicyBase** and implements **IPolicyDefinition** and **IPolicyEvaluation**. You register the policy assembly in the Windows registry and you apply the policy to your team project.

Contents

- Objectives
- Overview
- Before You Begin
- Summary of Steps
- Step 1. Create and Build a Custom Policy Class.
- Step 2. Register the Custom Policy Class in the Windows Registry
- Step 3. Apply the Custom Policy.
- Step 4. Validate the Custom Policy.
- Additional Considerations

Objectives

- Learn how to create, register and apply custom check-in policies.

Overview

Check-in policies enforce constraints whenever files are checked into source control. Team Foundation Server provides a number of check-in policies including policies to check that unit tests have run and been passed, to perform static code analysis to ensure that code meets coding standards and .NET guidelines, and to check that work items are associated with check ins. The Team Foundation Power Tool also provides a number of additional check-in policies. In this How To, you will see how to create, register and apply a custom policy. The example policy ensures that developers supply check-in comments whenever they check in a file.

Before You Begin

Note that to create a check in policy you must have the **Manipulate** security settings permission set to **Allow**.

Summary of Steps

- Step 1. Create and build a custom policy class
- Step 2. Register the custom policy in the Windows registry
- Step 3. Apply the custom policy
- Step 4. Validate the custom policy

Step 1. Create and Build a Custom Policy Class

In this step you create a custom policy class by deriving from **PolicyBase** in the **Microsoft.TeamFoundation.VersionControl.Client** namespace. By deriving from this base class your class implements the **IPolicyDefinition** and **IPolicyEvaluation** interfaces. The example policy code shown below, forces a developer to supply check-in comments whenever he or she checks in a source file.

1. Use Visual Studio to create a new C# class library project.
2. Add an assembly reference to System.Windows.Forms.dll. You use this assembly to display message boxes.
3. Add an assembly reference to Microsoft.TeamFoundation.VersionControl.Client.dll. By default, this is installed in the following folder:
   \Program Files\Visual Studio 2005 Team Foundation Server\Tools
4. Replace your skeleton class code implementation with the following source. Note that the class derives from **PolicyBase** and is marked as serializable.

```
using System;
using System.Windows.Forms;
using Microsoft.TeamFoundation.VersionControl.Client;

[Serializable]
public class CheckForCommentsPolicy : PolicyBase
{
    public override string Description
    {
        get { return "Remind users to add meaningful comments to their 
checkins"; }
    }

    // This is a string that is stored with the policy definition on the 
source
    // control server. If a user does not have the policy plugin installed, 
this string
    // is displayed.  You can use this to explain to the user how they 
should
    // install our the policy plug-in.
    public override string InstallationInstructions
    {
        get { return "To install this policy, read 
InstallInstructions.txt."; }
    }
    // This string identifies the type of policy. It is displayed in the
     // policy list when you add a new policy to a Team Project.
    public override string Type
    {
        get { return "Check for Comments Policy"; }
    }

    // This string is a description of the type of policy. It is displayed
    // when you select the policy in the Add Check-in Policy dialog box.
    public override string TypeDescription
    {
        get { return "This policy will prompt the user to decide whether or
not they should be allowed to check in."; }
    }

    // This method is called by the policy framework when you create
    // a new checkin policy or edit an existing checkin policy.
    // You can use this to display a UI specific to this policy type
    // allowing the user to change the parameters of the policy.
    public override bool Edit(IPolicyEditArgs args)
    {
        // Do not need any custom configuration
        return true;
    }

    // This method performs the actual policy evaluation.
    // It is called by the policy framework at various points in time
    // when policy should be evaluated. In this example, the method
    // is invoked when various asyc events occur that may have
    // invalidated the current list of failures.
    public override PolicyFailure[] Evaluate()
    {
        string proposedComment = PendingCheckin.PendingChanges.Comment;
        if (String.IsNullOrEmpty(proposedComment))
        {
            return new PolicyFailure[] {
                new PolicyFailure("Please provide some comments about your
checkin", this) };
        }
        else
        {
            return new PolicyFailure[0];
        }
    }

    // This method is called if the user double-clicks on
    // a policy failure in the UI. In this case a message telling the user
    // to supply some comments is displayed.
    public override void Activate(PolicyFailure failure)
    {
        MessageBox.Show("Please provide comments for your checkin.", "How to
fix your policy failure");
    }

    // This method is called if the user presses F1 when a policy failure
    // is active in the UI. In this example, a message box is displayed.
    public override void DisplayHelp(PolicyFailure failure)
    {
        MessageBox.Show("This policy helps you to remember to add comments
to your checkins.", "Prompt Policy Help");
    }
}
```

Step 2. Register the Custom Policy in the Windows Registry

In this step, you add an entry to the Windows registry so that your policy appears in the **Add Check-in Policy** dialog box. Note that you must install the policy assembly on any computer that needs to reference the assembly. This includes the computer of the team project's administrator who needs to associate the policy with the team project and on all of your the team members' computers where the policy is actually evaluated.

Important: The policy is evaluated on the client when a developer checks in a file.

To register your custom policy class:

1. Start Regedit.exe and locate the following key HKEY_LOCAL_MACHINE\Software\Microsoft\VisualStudio\8.0\TeamFoundation\SourceControl\Checkin Policies
   The registered policies are listed in the right hand pane.
2. Right-click in the right-hand pane, point to **New** and click **String Value**.
3. Type the name of your custom policy DLL (without the DLL extension); **CheckForCommentsPolicy** in the above example.
   Important: The new string name must match your DLL filename exactly, without the DLL extension.
4. Double-click the new string value and set its value to the fully qualified path and filename to the .dll containing your custom policy.

Step 3. Apply the Custom Policy

In this step you add the custom policy to your team project. This ensures that the policy is evaluated each time a developer checks in a file to this team project.

To apply the custom policy:

1. In **Team Explorer**, right-click your team project, point to **Team Project Settings** and click **Source Control**.
2. Click the **Check-in Policy** tab and then click **Add**.
3. Select your custom policy **Check for Comments Policy** and click **OK** and then **OK** again.

The policy is now applied each time a developer checks a file into this team project.

Step 4. Validate the Custom Policy

In this step, you check-in a source file to ensure that the custom policy works.

1. Make a change to a source file and check in the file without supplying a check in comment.
2. Verify that the check-in is prevented because the policy rule is not satisfied.

3. Add some comments and complete the checkin. With a supplied comment the checkin should work successfully and you will not see a policy failure notification.

Additional Resources

- To learn how to customize a check-in policy, see “Walkthrough: Customizing Check-in Policies and Notes”, http://msdn2.microsoft.com/en-us/library/ms181281(VS.80).aspx
- To see sample code that will disallow selected patterns on checkin, see “Checkin Policy to Disallow Certain Patterns”: http://blogs.msdn.com/jmanning/archive/2006/02/02/523125.aspx
- To see sample code that will enforce comments on checkin, see “Sample Checkin Policy: Make Sure the Comment Isn't Empty”: http://blogs.msdn.com/jmanning/archive/2006/01/21/515858.aspx
- To learn how to register a new checkin policy, see “I've Made a New Check-In Policy! How Do I Add It?”: http://blogs.msdn.com/jmanning/archive/2006/02/07/526778.aspx

How To: Step Through Creating Your Source Tree in Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To steps you through the process of creating a new source code tree structure in Team Foundation Server. The purpose of this how to is to familiarize you with the end-to-end steps required to create your source tree.

Contents

- Objectives
- Overview
- Summary of Steps
- Step 1. Create a new team project.
- Step 2. Create a workspace mapping
- Step 3. Create your folder structure in Source Control
- Step 4. Add your source to your source tree
- Additional Resources

Objectives

- Learn how to create your source tree in Team Foundation Server.

Overview

While you can quickly add a solution to source control by right-clicking your solution in Solution Explorer and clicking **Add Solution To Source Control**, this option does not enable you to explicitly set up your source tree structure in source control. By explicitly defining your source tree structure in source control, you can arrange your source beneath top level folders and use separate top level folders to contain your main source base and your branched source base such as the branches you might use during development or to maintain production releases.

In this how to you will see the steps required to explicitly create a source control tree structure.

Summary of Steps

- Step 1. Create a new team project
- Step 2. Create a workspace mapping
- Step 3. Create your folder structure in Source Control

- Step 4. Add your source code to your source tree

Step 1. Create a New Team Project

In this step you create a new team project with default settings.

To create a new team project:

1. In **Team Explorer**, right-click your Team Foundation Server and click **New Team Project…**
2. In the **New Team Project** dialog box, type a project name such as **MyTeamProject1** and click **Next**.
3. On the **Select a Process Template** page, leave the default **MSF for Agile Software Development - v4.0** and click **Next**.
4. On the **Specify the Settings for the Project Portal** page, leave the team project portal name as **MyTeamProject1** enter a description for the team project portal and then click **Next**.
5. On the **Specify Source Control Settings** page, leave the default option **Create an empty source control folder** selected and then click **Next**.
6. Click **Finish** to create the project.

A new team project is created on your team foundation server, using the selected process template with an empty source control node.

Step 2. Create a Workspace Mapping

In this step you create a workspace mapping to define the mapping between folder structure on the TFS server and client. You need to do this so that you can create a source tree structure. First the source tree structure is created in your workspace and then you need to perform a check-in to your Team Foundation Server.

You can create a workspace mapping in one of two ways:

- **Set the workspace mapping explicitly**
- **Perform a get operation on your team project.**

To set a workspace mapping explicitly

1. In Visual Studio from the **File** menu, point to **Source Control** and then click **Workspaces**.
2. In the **Manage Workspaces** dialog box, select your computer name and click **Edit**.
3. In the **Edit Workspace** dialog box, in the **Working folders** list, click **Click here to enter a new working folder**.
4. Click the ellipsis button, select your team project for example **MyTeamProject1**, and click **OK**.
5. Click the local folder cell to display another ellipsis button.

6. Click the ellipsis button beneath **Local Folder** and browse to and select the local folder on your development computer where you want to locate your team project, for example C:\DevProjects\MyTeamProject1.
7. Click **OK** and then **OK** again to close the **Edit Workspace** dialog box**.**
8. Click **OK** in response to the Microsoft Visual Studio message box that informs you than one or more working folders have changed.
9. Click **Close** to close the **Manage Workspaces** dialog box.

To perform a Get operation on your team project

1. In **Team Explorer**, expand your team project node **MyTeamProject1**.
2. Double click **Source Control** beneath your team project.
3. In **Source Control Explorer**, right-click the root folder **MyTeamProject1** and click **Get Latest Version**.
4. In the **Browse For Folder** dialog box select your local path, for example C:\DevProjects\MyTeamProject1 and then click **OK**. This maps the team project root folder within Team Foundation Server to a local path on your computer.

Step 3. Create Your Folder Structure in Source Control

In this step depending upon your strategy and project requirements you create a source control folder structure on your server. This should generally start with a /Main/Source folder structure. This enables you to subsequently create **Development** and **Maintenance** branches at the same level as **Main**. For example, the **Maintenance** folder would be used to contain branched code corresponding to your software releases. The **Development** folder contains your working development branch.

```
/Main
    /Source
        /MyApp1                         → Contains MyApp1.sln
            /Source                     → Container folder
                /ClassLibrary1          → Contains ClassLibrary1.csproj
                /MyApp1Web              → Contains Default.aspx
            /UnitTests                  → Contains unit test projects
                /ClassLibrary1Tests     → Test project for ClassLibrary1
                /MyApp1WebTests         → Test project for MyApp1Web
    /Build                              → Contains build output (binaries)
    /Docs                               → Contains design docs etc
    /Scripts                            → Contains build scripts
    /TestCases                          → Contains test case documentation

/Development
    /FeatureBranch1
        /Source
            /MyApp1
                /Source
                    /MyApp1Web
                    /ClassLibrary1
```

                        /UnitTests
                                /ClassLibrary1Tests
                                /MyApp1WebTests
        /FeatureBranch2

/Maintenance
        /Release1
                /MyApp1
                        /Source
                                /ClassLibrary1
                                /MyApp1Web
                        /UnitTests
                                /ClassLibrary1Tests
                                /MyApp1WebTests
        /Release 1.1
        /Release 1.2

To create a folder structure on the server:

1. In **Team Explorer**, expand your team project node **MyTeamProject1**.
2. Double click **Source Control** beneath your team project.
3. In **Source Control Explorer**, select the root node, right-click in the **Local Path:** pane and click **New Folder**.
4. Type the name **Main** and then press ENTER.
5. Create a **Source** folder beneath Main.
6. Repeat the previous steps to create any other root folders you might need including **Development** and **Maintenance** folders.
7. Once you have created your tree structure, right click the **MyTeamProject1** root node in **Source Control Explorer** and click **Check-in Pending Changes.**
8. In the **Check In - Source Files – Workspace** dialog box, select the folders you need to check-in, add a comment and click **Check In**. This builds your local folder structure and adds the structure to TFS source control.

Step 4. Add Your Source Code to Your Source Tree

In this step, you add source code from your local drive to your source control tree on the server. In this example, you create a new Web application and class library project and add these to source control.

To create a new Visual Studio solution file:

1. From the **File** menu, point to **New**, and click **Project**.
2. Expand **Other Project Types** and click **Visual Studio Solutions**.
3. Select **Blank Solution** in the **Templates** pane.
4. Type **MyApp1** into the **Name** text box and C:\DevProjects\MyTeamProject1\Main\Source into the **Location** text box.

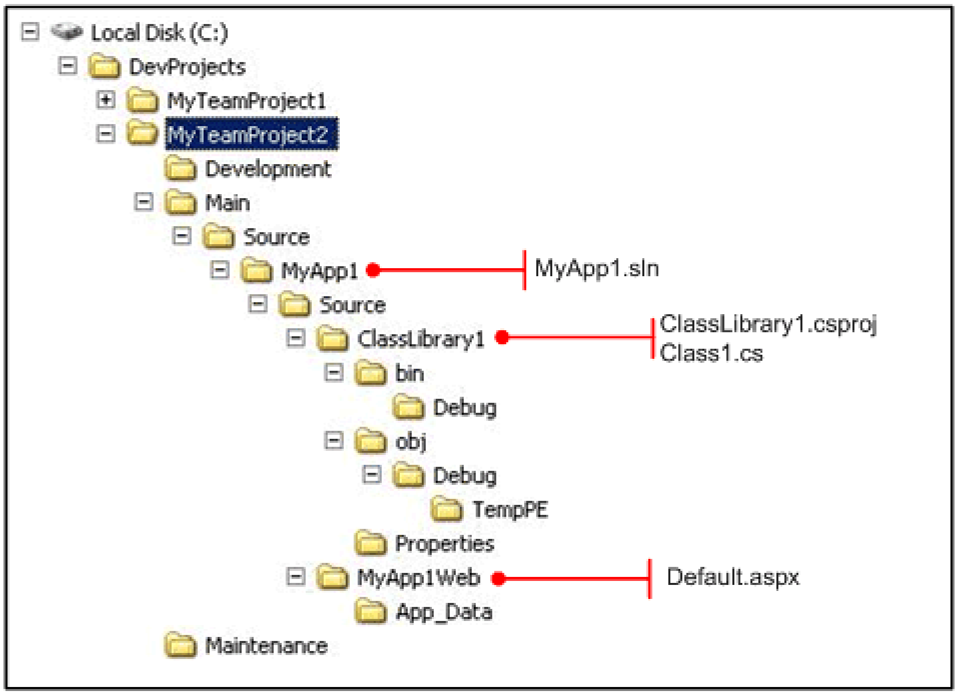
5. Click **OK**. Visual Studio creates your new solution and places the solution (.sln) file in the C:\DevProjects\ MyTeamProject1\Main\Source\MyApp1 folder.

To add a new Web site to your solution:

1. Right-click your solution in Solution Explorer, point to **Add**, and click **New Web Site**.
2. Select **ASP.NET Web Site** from the **Templates** list, **File System** as the **Location** and C:\DevProjects\MyTeamProject1\Main\Source\MyApp1\Source\MyApp1Web as the path.
3. Click **OK**. Visual Studio creates the Web site.

To add a new class library project to your solution:

1. Right-click your solution in Solution Explorer, point to **Add**, and click **New Project**.
2. Select **Visual C#** from the **Project types** list, and **Class Library** from the **Templates** list.
3. Leave the name as **ClassLibrary1** and set the **Location** to C:\DevProjects\MyTeamProject1\Main\Source\MyApp1\Source.
4. Click **OK**. Visual Studio creates the new project structure. Your local file structure should now look like this:

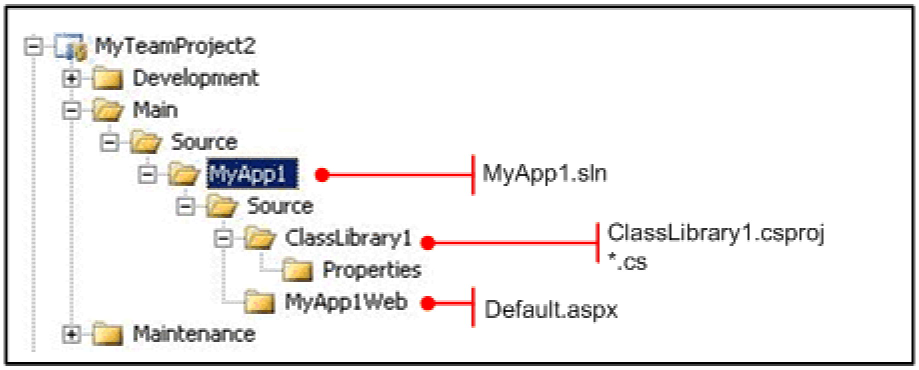


To add your solution to source control:

1. Right-click your solution in Solution Explorer and click **Add Solution To Source Control**.

2. Your solution, and two projects are added to Team Foundation Source Control.

Your source control tree structure should look like the following:



Additional Resources

- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx

How to: Customize the Process Template using Process Editor

Applies To

- Visual Studio Team Foundation Server

Summary

This How To explains how to modify a process template in order to better suit the needs of your team. Process templates define the initial process settings for team projects. By customizing a process template, you can define security for team project control, what templates are available on the project portal, source code control check-in notes, new work item types and queries, reports for monitoring and status, and which iterations and organization units are used.

Content

- Objective
- Overview
- Summary of Steps
- Step 1 – Install Process Editor
- Step 2 – Choose the Process Template
- Step 3 – Download the Process Template
- Step 4 – Open the Process Template in Process Editor
- Step 5 – Modify Process Template Plug-ins
- Step 6 – Modify Work Item Types
- Step 7 – Modify the Default Work Items
- Step 8 – Modify and Manage Queries
- Step 9 – Modify Areas and Iterations
- Step 10 – Modify Groups and Permissions
- Step 11 – Modify Source Control Settings
- Step 12 – Modify Project Portal
- Step 13 – Modify Reports
- Step 14 – Upload the Modified Process Template
- Additional Resources

Objective

- Learn how to customize process template using the Process Editor tool.
- Learn about various editable parts of process template

Overview

Visual Studio Team System and Team Foundation Server provide an integrated environment that supports most of the process activities involved in a software-development project. TFS implements its life-cycle methodologies by using process templates. A process template is a set of XML files that provides specifications for different artifacts and processes of a methodology. By modifying a process template you can change the default work item types, security settings, source control settings, and reports that are set up when a new team project is created.

To modify a process template you must first download the template you would like to modify and then use the Process Editor to make necessary changes.

Summary of Steps

- Step 1 – Install Process Editor
- Step 2 – Choose the Process Template
- Step 3 – Download the Process Template
- Step 4 – Open the Process Template in Process Editor
- Step 5 – Modify Process Template Plug-ins
- Step 6 – Modify Work Item Types
- Step 7 – Modify the Default Work Items
- Step 8 – Modify and Manage Queries
- Step 9 – Modify Areas and Iterations
- Step 10 – Modify Groups and Permissions
- Step 11 – Modify Source Control Settings
- Step 12 – Modify Project Portal
- Step 13 – Modify Reports
- Step 14 – Upload the Modified Process Template

Step 1 – Install Process Editor

In this step you will install **Process Editor**. It’s a tool which provides a convenient UI based method of viewing and customizing process templates. The **Process Editor** is part of the **Power tools**.

1. Install **DSL Tools for Visual Studio 2005** before you run the Power Tools installer. It is a pre-requisite for the Team System Process Editor. If you don't install this, the Power tools installer will install everything except the Process Editor. DSL tools download is available at http://go.microsoft.com/fwlink/?LinkId=82410
2. Download the **Power tools** from the following URL http://www.microsoft.com/downloads/details.aspx?familyid=7324c3db-658d-441b-8522-689c557d0a79&displaylang=en
3. Run through the default installation, and verify that the **Power tools** has been installed correctly with the **Process Editor** as follows
   a. Click on **Start** and then **Programs**
   b. There should be an option **Microsoft Team Foundation Server Power Tools**, click on it.
   c. It should display **Microsoft Visual Studio Team System Process Editor**.
   d. This means the Process Editor has been installed correctly, if not then uninstall the Power Tools and reinstall with the right sequence as specified in above steps.

Step 2 – Choose the Process Template

In this step you choose an out-of-box template which is closest to your adopted process, so that you have to make very few changes to confirm to your customized process. Here are the details about out-of-box process templates to help you in making the decision.

MSF for Agile Software Development

The MSF for Agile Software Development is optimally designed for small projects on a rapid delivery schedule. You might want to choose the MSF for Agile Software Development template if your business:

- Does not have a great deal of documented process and does not have interest in developing formal processes.
- Has small software development teams.
- Can accommodate many small development teams working in multiple ways.
- Has short software development cycles (measured in weeks or months).
- Supports multiple software releases in a calendar year.

For more information, see Visual Studio Team System: MSF for Agile Software Development (http://go.microsoft.com/fwlink/?LinkId=51460).

MSF for CMMI Process Improvement

The MSF for CMMI Process Improvement process model was created in the business world largely outside the software development culture by professionals who believe that formalizing and improving best practices through lessons learned is critical to long-term success. You might want to choose the MSF for CMMI Process Improvement template if your business:

- Wants to evaluate current business practices.
- Has large software development teams.
- Needs to integrate traditionally separate groups or organizations.
- Can provide guidance for quality processes.
- Has longer software development cycles.

For more information, see Visual Studio Team System: MSF for CMMI Process Improvement (http://go.microsoft.com/fwlink/?LinkId=51462).

Step 3 – Download the Process Template

In this step you will download the chosen process template to be modified for adapting to your customized process.

1. In Visual Studio instance click **Team** and select **Team Foundation Server Settings**
2. Click **Process Template Manager…**
3. In the **Process Template Manager** dialog, select the process template you want to modify and click **Download**.
4. In the **Download Process Template** dialog select the folder location on your local drive and click **Save**.

Step 4 – Open the Process Template in Process Editor

In this step you load the downloaded process template in the process editor for modifying various settings.

1. In a Visual Studio instance, select the **Team** menu.
2. Click **Process Editor**, and then click **Open Process Template**.
3. In the **Open Process Template fileset** dialog, navigate to the downloaded process template and click **Open**
4. It will open the **ProcessTemplate.xml** file in the Visual Studio instance.
5. Set the **Name** of the methodology you are customizing for.

Step 5 – Modify Process Template Plug-ins Selection

In this step you will modify the process template plug-ins selection that runs when a new team project is created. A plug-ins sets up required files or configures data for its area. Microsoft provides following plug-ins with Microsoft Visual Studio 2005 Team System:

- **Classification –** Structure definition for the project
- **Reporting –** Project Reports uploading
- **Portal –** Creating project site.
- **Work item tracking** – Work items definition uploading.
- **Version control** - Creating Version control
- **Groups –** Create groups and assign permissions.

In the **Methodology Detail** page select the wizard's you want to display at the time of project creation. So that users can modify them as required

Step 6 – Modify Work Item Types

In this step you can add new work items types specific to your process or modify the default Work Item Types.

1. In the Process Template Explorer click **Work Item Tracking.**
2. Then select the **Type Definitions** tab in the right pane.
3. To create a new work item, click **Add** in the toolbar in the right pane.
4. In the **New Work Item Type** dialog, enter a name for the work item type and select an existing work item type from the **Copy From** drop-down list.
5. The new work item type will be created and available in the **Item List** in the **Type Definitions** tab in the right pane.
6. To save the changes click **Save** option under **File** menu.
7. If you want to add / remove attributes / fields to the new work item type or an existing work item type.
8. In the **Type Definitions** tab, right-click the work item type you want to edit
9. Select **Open**.
10. The selected work item type will open in a new Visual Studio window.
11. Here you can add / remove the attributes you want.

Step 7 – Modify the Default Work Items

In this step you can add or remove the default work items which are created at the time of team project creation.

1. In the in the Process Template Explorer click **Work Item Tracking**.
2. Then select the **Default Work Items** tab in the right pane.
3. To create a new default work item type, click **Add** in the toolbar in the right pane.
4. In the **Choose Type** dialog box, select a work item type.
5. In the dialog box which opens, fill in the appropriate fields.
6. Click **OK**.

Step 8 – Modify and Manage Queries

In this step you can modify, add or remove the default queries created at the time of team project creation.

1. In the Process Template Explorer click **Work Item Tracking**.
2. Then select the **Queries** tab in the right pane.
3. To create a new query Click **Add** in the toolbar in the right pane.
4. In the **Query Reference** dialog box, enter the name of the new query.
5. Then click the **Edit Query Definition** button.
6. Use the Fields, Sorting, and Criteria tabs in the **Query Edit** dialog to define your query.
7. Click **OK**.

Important - The Process Editor does not provide comprehensive features for editing queries. It is better to edit and test a query in a running Team project in Visual Studio than in a process template. You can then save them to files that can be imported to other projects or copied into process templates.

Step 9 – Modify Areas and Iterations

In this step you can set up the default iterations and areas that are available at the time of team project creation.

1. In the Process Template Explorer click **Areas & Iterations.**
2. Then select the **Areas** or **Iterations** tab in the right pane.
3. Use the toolbar in the right pane to add, remove, or move areas or iterations

Step 10 – Modify Groups and Permissions

In this step you modify, remove or add groups and their permissions that are available at the time of team project creation.

1. In the Process Template Explorer click **Groups & Permissions.**
2. To create a new group, click **Add** in the toolbar in the right pane.
3. In the **Group** dialog box, enter the name of the new group and a brief description of what the members of the group can do and then click **OK**

4. Select a group in the top section of the right pane, and assign permissions to that group in the section below. You can select **Allow**, **Deny** or leave it **Unset**. An **Unset** permission is an implicit deny.

Step 11 – Modify Source Control Settings

In this step you can modify the source control settings for parallel edits and check-in edits.

1. In the Process Template Explorer, click **Source Control**
2. Then select the **Checkout Settings** tab in the right pane.
3. If you want to allow more than one person to edit a file at the same time, select the **Enable Multiple Checkout** check box.
4. Select the **Checkin Notes** tab in the right pane.
5. To create a new Checkin Note Field click **Add** in the toolbar in the right pane
6. To edit an existing Checkin Note Field select it and click **Open.**
7. Make necessary changes in the **Checkin Note** dialog box.
8. Then click the **OK** button.
9. You cannot make any changes on the **Permissions** tab in this version of the Process Editor.

Step 12 – Modify Project Portal

In this step modify the project portal to display specific reports, documents and process guidance.

1. In the Process Template Explorer click **Portal**
2. To create a new library right-click the **Portal** node in the middle pane and click **New Document Library**.
3. Enter the name and description of the library in the **Document Library** dialog box.
4. Then click **OK**.
5. Right-click a document library and click **New Folder** to create a new folder.
6. In the **Folder Properties** dialog box, enter the name of the new folder.
7. Select a folder and click **Add** on the toolbar in the right pane to upload a new document.
8. In the **File Import** dialog box, browse to the file you want to upload.
9. The **Destination Folder** and **Share Point Folder** fields will be automatically populated. Leave the **Query Id** field unchanged
10. Then click **Import**.
11. To edit the properties of an existing document in the document library, select the document in the right pane, and click **Open** on the toolbar.
12. In the **File Edit** dialog box, change the name of the file in the **Name** field. Do not change the other fields.
13. Then click **OK**.

Step 13 – Modify Reports

In this step you add, remove and modify the default reports created at the time of team project creation.

1. In the Process Template Explorer click **Reports**
2. Click **Add** on the toolbar.
3. In the **Report** dialog, on the **Report Detail** tab, enter a name for the report.
4. Then browse to the .rdl file you want to add in the **File Name** field. Leave the other fields unchanged.
5. Do not make any changes to the data in the **Properties** and **Parameters** tabs.
6. On the **DataSources** tab, enter the appropriate data sources. The default data sources for process templates shipped in Team Foundation Server are /TfsOlapReportDS and /TfsReportDS
7. Then click **Ok**.

Step 14 – Upload the Modified Process Template

In this step you upload the modified process template to TFS server, so that its available when the new team project is created.

1. In Visual Studio instance click **Team** and select **Team Foundation Server Settings**
2. Click **Process Template Manager…**
3. In the **Process Template Manager** dialog, click Upload button.
4. Browse to the local folder where the process template was downloaded and modified.
5. Click the **Upload** button on the **Upload Process Template** dialog.
6. The new process template should be listed in the **Process Templates:** list.
7. Click the close button.

Now when you create a new team project you will have the newly created process as one of the options when choosing the process templates.

Additional Resources

- For more information on process template customization, see “Process Template Customization Overview”: http://msdn2.microsoft.com/en-us/library/ms194945(VS.80).aspx

How To: Customize a Report in Visual StudioTeam Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To describes how to customize an existing report and then publish it to the team reporting portal in Team Foundation Server.

Contents

- Objectives
- Overview
- Summary of Steps
- Before You Begin
- Step 1 – Create a new reporting project
- Step 2 – Export the report you want to customize
- Step 3 – Create the data sources
- Step 4 – Add the report to your project
- Step 5 – Modify the report
- Step 6 – Deploy the report to your Team Foundation Server
- Step 7 – Test the report
- Additional Resources

Objective

- Create a reporting project in Visual Studio
- Customize an existing report within the reporting project
- Publish the new report to the report server

Overview

The reports that ship with VSTS are SQL Server Reporting Services reports. You can amend these reports or create your own custom reports by using the SQL Server 2005 Reporting Services Designer inside Visual Studio (Business Intelligence Development Studio), which ships with the SQL Server 2005 client tools.

Customizing a report allows you to add functionality to an existing report without having to build a new report from scratch. If a report you need is similar to one that already exists, customize the existing report in order to save time. To customize an existing report it must be exported from the report server, added to an existing report project in Visual Studio, and then redeployed to the reporting portal after changes are made.

Summary of Steps

- Step 1 – Create a new reporting project
- Step 2 – Export the report you want to customize
- Step 3 – Create the data sources
- Step 4 – Add the report to your project
- Step 5 – Modify the report
- Step 6 – Deploy the report to your Team Foundation Server
- Step 7 – Test the report

Before You Begin

Before you can customize a report for Team Foundation Server, you must ensure you have the following pre-requisites in place:

- You must have Business Intelligence Development Studio installed on the machine you will be using to customize the report. To test if it is installed, check Visual Studio to see if you have Business Intelligence Project type when you create a new project.
- Your user account must be a member of the Microsoft Analysis Server TfsWarehouseDataReaders security role on the data-tier server.
- Your user account must have administrator rights to the TFSWarehouse database on the data tier.
- Your user account must be a member of the SQL Server Reporting Services Publisher role on the application-tier server.

Step 1 - Create a new reporting project

Create a new reporting project so that you can add an existing report to the project and customize it. Use the following steps to create a new reporting project in Visual Studio:

1. Click **File**, then click **New**, then click **Project.**
2. Select the **Business Intelligence Project** type.
3. Select the **Report Server Project** template.
4. Set your project's **Name** and **Location** and then click ok.

Step 2 - Export the report you want to customize

Export the report you would like to customize from the project portal so you can import it into the new reporting project. Use the following steps to export a report:

1. Right click on your team project and select **Show Project Portal**.
2. Click **Reports** in the Quick Launch bar on the left hand side of the portal web site.
3. Click the report you are interested in customizing.
4. Click **Properties.**
5. Select **Edit.**
6. Save the report rdl file into the reporting project folder you created in step 1.

Step 3 - Create the data sources

In order to edit and publish the customized report you need to add data sources for the Team Foundation Server data warehouse and OLAP cube. Once these data sources are added to the Visual Studio project the report can pull data from the server.

Create the warehouse data source:

1. In the Visual Studio **Solution Explorer**, right click **Shared Data Sources** and click **Add New Data Source**.
2. Enter **TfsReportDS** into the **Name** text box on the **General** tab.
3. Select **Microsoft SQL Server** from the **Type** combo box.
4. Click on the **Edit…** button.
5. Fill in your data tier server name.
6. Select the the database **TFSWarehouse**.
7. Click the **OK** button twice to add the data source.

Create the OLAP data source:

1. In **Solution Explorer**, right click **Shared Data Sources** and click **Add New Data Source**.
2. Enter **TfsOlapReportDS** into the **Name** text box on the **General** tab.
3. Select **Microsoft SQL Server Analysis Services** from the **Type** combo box.
4. Click on the **Edit…** button.
5. Fill in your data tier server name.
6. Select the database **TFSWarehouse**.
7. Click the **OK** button twice to add the data source.

Step 4 - Add the report to your project

Now that the data sources have been added to your project you can add the report that you exported in step 2:

1. In **Solution Explorer**, right click **Reports** and select **Add->Existing Item...**
2. Browse to the rdl file you exported in step 2

Step 5 - Modify the report

Now that you've added the report to your project you can make modifications in order to customize it for your needs. To open the report for modification, double click it in the **Solution Explorer**. Once you've opened it you can make modifications such as:

- Modify query statements in the **Data Pane**.
- Drag new measures or members into the **Data Pane**.
- Modify the report layout in the **Layout Pane**.

Step 6 - Deploy the report to your Team Foundation Server

After you've made modifications to the report, you can deploy it to your team project's reporting portal by using the following steps:

1. Right click on the report project in **Solution Explorer** and click **Properties**.
2. Ensure **OverwriteDataSources** is set to **false**.
3. Modify **TargetDataSourceFolder** to your team project name. Example: **TargetDataSourceFolder** = TestProject.
4. Modify **TargetReportFolder** to your team project name. Example: **TargetDataSourceFolder** = TestProject.

5. Modify **TargetDataSourceFolder** to http://<data-tier servername>/reportserver.
   Example: **TargetDataSourceFolder** = http://tfsrtm/reportserver.
6. Click **OK.**
7. Right click on the rdl file in **Solution Explorer** and click **Deploy.**
8. Watch the **Output Pane** for successful completion.

Step 7 - Test the report

After you've published the report to your team project's report server you can test it to make sure it was successfully deployed:

1. Right click on your team project and select **Show Project Portal.**
2. Click **Reports** in the Quick Launch bar on the left hand side of the portal web site.
3. Select the report you customized.
4. Verify the customization you created is in place.

Additional Resources

- For more tutorials explaining how to work with reporting projects, see "Reporting Services Tutorials": http://msdn2.microsoft.com/en-us/library/ms170246.aspx
- For the MSDN article on editing reports, see "How to: Edit Reports in Report Designer": http://msdn2.microsoft.com/en-us/library/ms244655(VS.80).aspx
- For more information on security roles in the data-tier, see "Securing Access Through Analysis Services": http://msdn2.microsoft.com/en-us/library/ms174839.aspx
- For more information on security roles in the application-tier, see "Securing Reporting Services": http://msdn2.microsoft.com/en-us/library/ms157198.aspx

How To: Manage Projects in Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To shows you how to initiate new software development projects by creating and configuring new team projects based on a selected template. It also describes how to use Team Foundation Server tools to manage, control and monitor the ongoing software development processes throughout the software development lifecycle.

Contents

- Objective
- Overview
- Summary of Steps
- Before You Begin
- Step 1 – Choose a Process Template
- Step 2 – Create a Team Project
- Step 3 – Create Security Groups (Optional)
- Step 4 – Add Team Members in Team Foundation Server
- Step 5 – Determine Your Iteration Cycle
- Step 6 – Capture Your Project Scenarios in TFS
- Step 7 – Identify the Scenarios for the Iteration
- Step 8 – Define Your Quality of Service Requirements
- Step 9 – Plan Your Iteration
- Step 10 – Monitor Progress
- Areas and Iteration Consideration
- Additional Resources

Objective

- Learn how to manage projects in TFS

Overview

Visual Studio Team System provides tools and reports to help project managers to control and support the entire software development lifecycle in a unified and centralized manner. By controlling the software process directly through Visual Studio Team System and a centralized database shared by all team members, team communication is improved, and handoffs required between team members are automated. By using reports driven from the team foundation server data warehouse it is also much easier to monitor and track project progress.

This How To walks you through the process or creating and configuring a new team project and then how to monitor the progress of an ongoing project.

Summary of Steps

- Step 1 – Choose a Process Template
- Step 2 – Create a Team Project
- Step 3 – Create Security Groups (Optional)
- Step 4– Add Team Members in Team Foundation Server
- Step 5 – Determine Your Iteration Cycle
- Step 6 – Capture Your Project Scenarios in TFS
- Step 7 – Identify the Scenarios for the Iteration
- Step 8 – Define Your Quality of Service Requirements
- Step 9 – Plan Your Iteration
- Step 10 – Monitor Progress.

Before You Begin

Before you start creating a new team project, you should gather the following information:

- **The style of project you want to manage**. This determines the initial template you should select and base your project on. It also helps you evaluate the degree of customization you need to make to the supplied templates. If you favor an agile style of project, start by selecting the MSF for Agile Software Development template and then consider any customizations you night need.
- **The project name**. Give careful consideration to the naming convention that you use for the team projects you create. Make sure that your project names are sufficiently unique and use names that enable other people to easily find the project. Some organizations use project codes as part of their names, others might use a combination of department and project title.
- **Source control structure**. Consider whether you need to start with an empty source control tree of whether your project should be based on existing source code. Evaluate an appropriate source control structure based on your code and branching needs. For more information, see “How To: Structure Your Source Control Folders.”

Step 1 – Choose a Process Template

Start by selecting your process template. Evaluate the style of project you want to manage and examine the different features provided by each of the two supplied templates. Features to evaluate include work item definitions (do they provide sufficient fields for your needs?), document templates, workflows, check in policies and reports.

The Team Foundation Server provides following process templates

- **MSF for Agile Software Development** – This is a lightweight process for small and informal software projects. It is based on scenarios and context driven actions and is project and team member (people) centric.
- **MSF for CMMI® Process Improvement –** This is designed for more mature software projects. It extends MSF for Agile Software Development by providing support for auditing, verification and formal processes. It relies on process and conformance to process and is organization centric.

If required you can customize the supplied process templates to align them more closely with your processes. For more information about customizing the process templates see "How To – Customize Process Template".

The remainder of this How To, assumes that you have selected the **MSF for Agile Software Development** Template.

Step 2 – Create Team Project

To create a new team project:

1. Ensure that Visual Studio is connected to Team Foundation Server instance.
2. In the **Team Explorer** right click the server node and click **New Team Project…**
3. On the first page of the project creation wizard, enter the name of your Team Project and then click **Next**.
4. On the **Select a Process Template** page, from the drop down select the process template you have chosen in Step 1. For this example choose the **MSF for Agile Software Development –v4.0** and then click **Next**.
5. On the **Specify the Settings for the Project Portal** page, enter the name and description for team project portal and then click **Next**. The name you supply here is used to build the Windows SharePoint Services Web site for your project portal.
6. On the **Specify Source Control Settings**, select **Create an empty source control folder** and then click **Next**.
7. On the **Confirm Team Project Settings**, review the settings and click **Finish**.

This creates a Team Project on the TFS server based on the MSF for Agile Software Development process template.

Step 3 – Create Security Groups (Optional)

When you create a project in Team Foundation Server, four default groups are created for that project regardless of your choice of process template. By default, each of these groups has a set of permissions defined for them that govern what members of those groups are authorized to do. The four groups are:

- Project Administrator
- Contributor
- Reader
- Build Services.

You can create security groups for your team project to better meet the security requirements of your organization. Creating a security group is an efficient way to grant a specific set of permissions to a group of users on your team project. Make sure that you allow only the minimum permissions necessary for the group, and add only those users or groups who must belong to this new team project group.

To perform this procedure, you must be a member of the **Project Administrators** group although you do not need to be a Windows administrator.

1. In Team Explorer, select the team project for which you want to create a group.
2. On the **Team** menu, point to **Team Project Settings**, and then click **Group Membership**.
3. In the **Project Groups** dialog box, click **New**.
4. In the **Create New Team Foundation Server Group** dialog box, in the **Group name** box, type the name for the team project group.
5. In the **Description** box, type a description for the group.
6. Click **OK** and then **Close**.

After you have created a team project group, give the group the appropriate permissions, and add members to the group. By default, a team project group is created without any permission granted.

Step 4 – Add Team Members in Team Foundation Server

In this step identify the resources that will be working on the project, their roles and assign these team members to Team Foundation Server. You can add users to existing team project groups or server-level groups. You must also add members to groups that you create. To perform this procedure, you must be a member of the **Team Foundation Administrators** group.

1. In Team Explorer, select your team project.
2. On the **Team** menu, point to **Team Project Settings**, and then click **Group Membership**, or, to add users to a server-level group, point to **Team Foundation Server Settings**, and then click **Group Membership**.
3. In the **Project Groups** dialog box, select the group to which you want to add users, and then click **Properties**.
4. In the **Team Foundation Server Group Properties** dialog box, on the **Members** tab, under **Add member**, select **Windows User or Group**.
5. Click **Add**.
6. In the **Select Users or Groups** dialog box, under **Enter the object names to select**, enter the domain name and alias of the users you want to add in the following format:

domain\username

To add more than one user at a time, separate the entries with a semicolon (;).

7. Click **OK, OK** again and then **Close**.

Step 5 – Determine Your Iteration Cycle

Iterations are fixed-length chunks of time in which you plan, schedule, and perform work. All components of the software development lifecycle, from requirements definition through analysis, design, development, coding, and testing are grouped into these iterations, which typically last anywhere from 2 to 6 weeks.

In this step you determine the iteration cycle for your project. The important points to keep in mind while determining the iteration cycle are:

- The iteration cycle should be large enough to get substantial work done, and cover at least a few scenarios
- The iteration cycle should be small enough to be flexible to accommodate changes and priorities.

This will depend upon the size and complexity of your project. In practice 2 weeks iteration cycle works for most projects.

Step 6 – Capture Your Project Scenarios in TFS

In this step you capture your project scenarios in TFS so that you can plan better execution of your project and track its progress.

1. Create a project back log (PBL) document from inputs from the various stake holders including customers, business analysts, end users and product managers. The PBL is generally a Word document designed mainly to capture requirements
2. Use the PBL as input and scope out the scenarios for your project.
3. You can create all the scenarios at the start or you can create scenarios as you move through iterations. To gain a better picture of your whole project and to help track progress you are recommended to capture all your scenarios upfront.
4. Create an iteration called “Iteration 999” for your project and assign your scenarios to it for ease of retrieval and tracking.

To create a scenario in a project that uses the MSF Agile Software Development template.

1. In **Team Explorer** expand the project node and right click on the **Work Items** folder
2. Point to **Add Work Item** and then click **Scenario**
3. In the **New Scenario** page, enter the details for the Scenario. Make sure to set the Iteration to “Iteration 999”
4. Save your new Scenario.
5. Repeat the above steps for all the scenarios that you have identified for the project.

Step 7 – Identify the Scenarios for the Iteration

In this step you identify the scenarios to be worked on during a specific iteration. You repeat this step for each iteration cycle.

1. Identify the work items to be worked on during a specific iteration, based on inputs from the project stakeholders and your feature priorities.
2. Assign these work items to the iteration

To retrieve work items and assign them to a specific iterations.

1. Right click on the **Work Items** folder in your project and click **Add Query**
2. In the **New Query**, add a new And clause. Select **Iteration Path** from the **Field** drop down and set its value to **Iteration 999**
3. Add a new And clause to specify **Work Item Type** = **Scenario**
4. Press F5 to execute the query and display all the pending scenarios.
5. Double click the scenario you want to work on in the iteration.
6. Change the **Iteration** to the current iteration path and click the save button.
7. Repeat the above steps for all the identified scenarios.

Important – You can save your query for future use.

Step 8 – Define Your Quality of Service Requirements

In this step you define your Quality of Service Requirements (QOS) for each of the scenarios to be worked on during the iteration cycle. You repeat this step for each iteration cycle. This helps define the acceptance criteria for the scenario. The inputs for the QOS come from project goals, requirements and specification documentation if available.

1. Right click on the **Work Items** folder of your project and point to **Add Work Item** and then click **Quality of Service Requirements**.
2. In the **New Quality of Service Requirements** page, add the following details
    a. Set the **Type** to an appropriate value such as performance, scalability, stress or securit.
    b. Set the Iteration to the current iteration cycle.
    c. From the **Links** tab, link the QOS to a specific scenario, for easier traceability.
3. Save the new QOS requirement.
4. Create one QOS for each discipline or kind of quality requirements, bearing in mind that each scenario can have multiple QOS.
5. Make sure you create QOS for all of the scenarios being worked in a specific iteration cycle.

Important – You can break down the QOS into test tasks later on.

Step 9 – Plan Your Iteration

In this step you plan for the iteration by breaking down the scenarios into tasks, estimating for them and assigning the tasks to the relevant team member. You repeat this step for each iteration cycle.

Developer Tasks

1. Break down the chosen scenarios into developer stories
2. Subdivide the developer stories into developer tasks.
3. Capture developer tasks in TFS as task work items.
    a. In Team Explorer, under your project node, right click the **Work Items** folder.
    b. Point to **Add Work Item** and then click **Task**.
    c. In the **New Task** page, add the following details
        i. Set **Discipline** to **Development**.
        ii. Set **Iteration** to the current iteration cycle.
        iii. On the **Links** tab link the task to the specific scenario for easier traceability.
        iv. Here along with the description you can capture the acceptance criteria for the task, which can determine if the task is completed successfully.
        v. Set the **Assigned to** field to the developer who will be working on the task.
    d. Save the new task.
    e. Repeat the above steps for all the identified tasks.
4. Repeat the above steps for all the identified scenarios for the iteration.

Test Tasks

1. Break down the QOS for the identified scenario into test cases.
2. Divide the test cases into test tasks.
3. This test tasks are captured in TFS as task work items
    a. In Team Explorer, under your project node, right click the **Work Items** folder.
    b. Point to **Add Work Item** and then click **Task**.
    c. In the **New Task** page, add the following details
        i. Set **Discipline** to **Test**.
        ii. Set **Iteration** to the current iteration cycle.
        iii. On the **Links** tab link the task to the specific QOS for easier traceability.
        iv. Here along with the description you can capture the acceptance criteria for the task, which can determine if the task is completed successfully.
        v. Set the **Assigned to** field to the tester who will be working on the task.
    d. Save the new task.
    e. Repeat the above steps for all the identified tasks.
4. Repeat the above steps for all the QOS for the iteration

Others

1. Capture any relevant tasks for the iteration for other disciplines like Architecture, Release Management, Project Management and Requirements which needs to be tracked.
2. Link each of these tasks to the appropriate scenario for easier traceability.

For large projects that contain a huge number of work items, you can use the Excel integration feature to create work items in the Excel spread sheet and then upload it into TFS. For more information see “Working with Work Item Lists in Microsoft Excel” at http://msdn2.microsoft.com/en-us/library/ms181694(VS.80).aspx

For large projects that contain large number of resources, you can use Microsoft Project integration feature to track the work items and balance the task assignment. For more information see “Working with Work Items in Microsoft Project” at http://msdn2.microsoft.com/en-us/library/ms244368(VS.80).aspx

Step 10 – Monitor Progress

In this step you identify the key performance indicators (KPIs) and link to the reporting feature, for easy tracking of the project.

Areas and Iteration Consideration

- You can use root level as area to start with
- Later on as and when you have logical differentiation of various areas within your project for example UI, Biz, Data etc, you can assign your scenarios / workitems to the specific areas.

Additional Resources

- For more information see “Working with Work Item Lists in Microsoft Excel” at http://msdn2.microsoft.com/en-us/library/ms181694(VS.80).aspx
- For more information see “Working with Work Items in Microsoft Project” at http://msdn2.microsoft.com/en-us/library/ms244368(VS.80).aspx

How To: Migrate Source Code to Visual Studio Team Foundation Server from Visual Source Safe

Applies To

- Visual Studio Team Foundation Server

Summary

To migrate version controlled source code from VSS to TFS, you use the VSS converter tool. This tool allows you to migrate files, folders, version history, labels, and user information from a VSS database to a TFS source control database. Prior to migrating your source you need to backup and prepare your VSS source database.

Contents

- Objectives
- Overview
- Before You Begin
- Summary of Steps
- Step 1. Backup your VSS database
- Step 2. Analyze your VSS database to resolve data integrity issues
- Step 3. Analyze your projects in VSS
- Step 4. Prepare to migrate your projects
- Step 5. Migrate your projects
- Additional Considerations
- Additional Resources

Objectives

- Migrate Visual Source Safe projects to TFS

Overview

Team Foundation Server provides tools to help you migrate existing source maintained in a Visual Source Safe (VSS) database to Team Foundation Server source control. TFS provides a VSS converter tool which allows you to migrate files, folders, version history, labels, and user information.

Before You Begin

To successfully perform the steps outlined in this How To:

- You must be logged in with an account that is a member of the Team Foundation Administrators group.
- You must have the VSS 2005 client installed on the computer the converter is running on. The converter stops with a warning if you are running an earlier version of VSS.

You should also make sure that you use the VSS 2005 client Analyze command in the event that the 2005 Analyze command is able to detect issues that previous versions were not. The VSS database does not need to be a native 2005 database.

- You must have SQL Express installed and enabled on the computer the converter is running on. The converter tool uses the local SQL instance as a temporary database during the conversion. SQL Express is installed by default with Visual Studio 2005.
- Make sure you gather your TFS domain name and the list of TFS user names as defined in your Active Directory.
- Make sure you have your VSS administrator user name and password and your TFS project administrator username and password.

Summary of Steps

- Step 1. Backup your VSS database
- Step 2. Analyze your VSS database to resolve data integrity issues
- Step 3. Analyze your projects in VSS
- Step 4. Prepare to migrate your projects
- Step 5. Migrate your projects

Step 1. Backup Your VSS Database

Prior to performing a migration start by creating a backup copy of the Visual SourceSafe Database you want to migrate.

To backup your VSS database:

1. Ask all users to check in their files and then log off the VSS database. Ask them to close both the Visual Studio IDE and VSS explorer.
   Important: Checked out files are not be migrated to TFS.
2. Check that no one is currently logged into the database.
3. Make sure that no analyze jobs are scheduled to run against the database.
4. Copy the following folders to your backup location:
   \DATA
   \Temp
   \USERS
5. Copy the following files to your backup location:
   User.txt
   Srcsafe.ini

Step 2. Analyze your VSS Database to Resolve Data Integrity Issues

In this step, you use the Visual SourceSafe Analyze utility to locate and fix data integrity issues in the database.

To use the VSS Analyze Utility:

1. Run Analyze.exe to search for database corruption or database errors. Use the following command-line:

   analyze "<sourcesafe data directory>"

   Use the **-I** option for unattended execution. This command reports potential problems.

2. If there are permissions problems, "unable to checkout files" errors, "losing checkout status" errors, or any other errors that refer to the Status.dat file or to the Rights.dat file, run the Ddconv.exe program or the Ddconvw.exe program. These programs update a SourceSafe database from an older format to the current format. By default, these programs are installed in the \Admin subdirectory.

Step 3. Analyze Your Projects in VSS

In this step, you determine which projects you want to migrate and then run the TFS command line tool VSSConverter.exe to analyze the VSS database for any potential issues that might cause migration problems.

To analyze your projects by using VSSConverter.exe

1. Create an XML settings file (called for example ConversionSettings.xml) as shown here:

```
<?xml version="1.0" encoding="utf-8"?>
 <SourceControlConverter>
    <ConverterSpecificSetting>
      <Source name="VSS">
            <VSSDatabase name="c:\VSSDatabase">
            </VSSDatabase>
      </Source>
      <ProjectMap>
        <Project Source="$/MyFirstProject"></Project>
        <Project Source="$/MySecondProject"></Project>
      </ProjectMap>
    </ConverterSpecificSetting>
 </SourceControlConverter>
```

   In the above sample, **MyFirstProject** and **MySecondProject** represent the names of the project folders in VSS that you want to migrate. To migrate an entire VSS database, use **<Project Source="$/"></Project>**.

2. Run VSSConverter.exe passing in the **Analyze** switch, from a Visual Studio command prompt to analyze your projects:

   VSSConverter Analyze ConversionSettings.xml

When prompted, enter the Visual SourceSafe administrator password.

3. Examine the output report. The conversion tool creates an output report called VSSAnalysisReport.xml and a user mapping file called Usermap.xml, in the current directory. You can use the user mapping file to map Visual SourceSafe users to Team Foundation users as described below.
4. Map VSS users to TFS users. The VSSConverter tool produces a UserMapping.xml file that contains a list of all VSS users that performed at least one operation on the VSS database. Edit the UserMapping.xml file and add your associated TFS user names. You must specify user names including the domain (MyDomain\MyUserName).

Step 4. Prepare to Migrate Your Projects

In this step, you use the VSSConverter.exe tool to migrate your VSS projects.

1. Modify the ConversionSettings.xml file created in Step 3 and add a new <**Settings**> element containing a <**TeamFoundationServer**> element as shown here:

```
<Settings>
  <TeamFoundationServer name="YourTFSServerName"
                        port="PortNumber"
                        protocol="http">
  </TeamFoundationServer>
 </Settings>
```

   Add the <**Settings**> element directly after </**ConverterSpecificSettings**> as a child element of <**SourceControlConverter**>.

2. Modify the ConversionSettings.xml file created and add **Destination** attributes to the <**Project**> elements as shown below. Set the value of the **Destination** attributes to the folder location in your TFS Team project where you want to migrate your files.

```
<?xml version="1.0" encoding="utf-8"?>
 <SourceControlConverter>
   <ConverterSpecificSetting>
      <Source name="VSS">
          <VSSDatabase name="c:\VSSDatabase">
          </VSSDatabase>
      </Source>
      <ProjectMap>
        <Project Source="$/MyFirstProject"
                 Destination="$/MyTeam_ProjectOne">
        </Project>
        <Project Source="$/MySecondProject"
```

```
                Destination="$/MyTeam_ProjectTwo">
       </Project>
     </ProjectMap>
  </ConverterSpecificSetting>
  <Settings>
    <TeamFoundationServer name="YourTFSServerName"
                          port="PortNumber"
                          protocol="http">
    </TeamFoundationServer>
  </Settings>
</SourceControlConverter>
```

In the **<ProjectMap>** section, for each VSS folder that you are migrating, replace **MyFirstProject** with the source VSS folders and replace **MyTeam_ProjectOne** with the destination folders in Team Foundation source control. Include a **<Project>** entry for all the folders that you want to migrate.

Step 5: Migrate Your Projects

Copy your Visual SourceSafe database to a local folder on the computer on which you want to run analysis and migration. Although you can migrate a Visual SourceSafe database in a shared folder on a remote computer, the migration takes much longer to finish.

To perform the migration:

1. At the command prompt, type the following.

   VSSConverter Migrate Conversionsettings.xml

2. Enter **Y** to confirm the migration. When prompted, enter the password for the Visual SourceSafe admin user.

Additional Considerations

If you used, sharing, branching or pinning in VSS you should be aware of the following issues.

- **Sharing**. Team Foundation Server does not support sharing. Shared files are migrated by copying the version of the file at the time sharing began to a destination folder. Any subsequent changes made to the shared file are replicated to both the shared and the original copies.
- **Branching**. Because branching in VSS uses sharing, the migration of a branched file results in the file being copied to the destination folder in TFS source control. After the branch event, the changes to any branch are migrated to the respective copy in Team Foundation source control.

- **Pinning**. Team Foundation Server does not support pinning. To help you locate items in Team Foundation source control that were once pinned in the VSS database, the VSSConverter tool labels any file that was pinned with the “PINNED” label.

For optimum performance (especially important for large VSS databases), run the conversion on your TFS server computer.

Additional Resources

- For more information on how to prepare for migration, see: http://msdn2.microsoft.com/en-us/library/ms181246(en-us,vs.80).aspx
- For more information on how to perform the migration, see: http://msdn2.microsoft.com/en-us/library/ms181247(VS.80).aspx
- For more information on the limitations of the Visual SourceSafe converter, see: http://msdn2.microsoft.com/en-us/library/ms252491(VS.80).aspx
- For more information about the VSS Analyze tool, see http://msdn2.microsoft.com/en-us/library/ysxsfw4x.aspx.
- For more information about the VSSConverter migrate command, see http://msdn2.microsoft.com/en-us/library/ms400685(VS.80).aspx

How To: Perform a Baseless Merge in Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To shows you how to merge files from two branches that are not related to one another. The process of merging items that are not directly branched from each other is called a *baseless merge*. To perform a baseless merge, you must use the Tf merge command. You cannot perform a baseless merge operation from Visual Studio.

Contents

- Objectives
- Overview
- Summary of Steps
- Step 1. Evaluate if a baseless merge is needed
- Step 2. Perform a baseless merge by using Tf.exe
- Step 3. Resolve merge conflicts
- Step 4. Check-in the merged changes
- Additional Resources

Objectives

- Learn how to perform a baseless merge

Overview

In this how to you learn to do a baseless merge. The process of merging items that are not directly branched from each other is called a *baseless merge*. For example, you might want to merge a change between two release branches, which are siblings of each other, without merging up to the parent branch. You can only perform a baseless merge by using the Tf.exe command line utility. You cannot perform a baseless merge from Visual Studio.

When you perform a baseless merge Team Foundation Server does not have any information about the relationship of the files in the branches. For example, if you have renamed a file, this will be viewed as a deleted file and a new file in the branch. For this reason you have to perform more manual conflict resolutions than when you perform a normal merge. However, you only have to perform this conflict resolution once. After you have performed the baseless merge, Team Foundation Server records history and establishes a relationship between the folders and files.

Summary of Steps

- Step 1. Evaluate if a baseless merge is needed
- Step 2. Perform a baseless merge by using Trif.exe
- Step 3. Resolve merge conflicts
- Step 4. Check-in the merged changes

Step 1. Evaluate if a Baseless Merge Is Needed

In this step you evaluate the branches and items to be merged to determine if you need to do a baseless merge or whether a regular merge will work.

If you are owner / administrator of your team project you will know the relationships of the branches or items. You can only merge from Visual Studio the branches or items which have a parent-child relationship. If your project contains branches or items which do not have this relationship, you require a baseless merge.

If you are not aware of all the relationships of the branches or items, you can evaluate if you need to use a baseless merge as follows.

- Open **Source Code Explorer**.
- Right-click the branched folder and click **Merge**.
- In the **Source Control Merge Wizard** dialog box, look under the drop down for **Target branch**.
- If you do not see an entry for the branch you want to merge it indicates that no merge relationship exists between these branches. In this case you must perform a baseless merge.

Step 2. Perform a Baseless Merge By Using Tf.exe

In this step you use the Team Foundation Server command line tool Tf.exe to perform a baseless merge.

To perform a baseless merge:

1. Set the Workspace for both branches by performing a get latest operation on the branches to be merged.
   1.1. Open **Source Code Explorer**.
   1.2. Right-click the folder for the first branch to be merged and click **Get Latest Version**.
   1.3. Repeat the previous step for the second branch to be merged.

   Note that if you do not have a workspace mapped Visual Studio prompts you to select a folder on your local drive.

2. Open a Visual Studio Command Window.
3. Run the following Tf.exe command from the command line.

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

Example

```
Tf merge /baseless c:\data\proj1 c:\data proj2 /recursive
```

If you need to merge specific versions of the code changes you can use the version switch with Tf.exe as follows

```
tf merge /baseless <<source path>> <<target path>> / recursive /version:<<from Changeset>>~<<to Changeset>>
```

Example

```
tf merge /baseless c:\data\proj1 c:\data\proj2 /recursive /version:C123~C125
```

Step 3. Resolve Merge Conflicts

In this step you resolve conflicts that might occur when performing a baseless merge. Once you have run the Tf.exe command it display a **Resolve Conflicts** dialog box with a list of the files with conflicts.

To resolve merge conflicts
1. Select each file and click **Resolve**.
2. In the **Resolve version conflict** dialog box, if there are no content changes choose the **Keep changes in the target branch** resolution option and click **OK**.
3. If there are content changes, select **Merge changes in merge tool** resolution option and click **OK**.
4. In the merge tool click the conflict regions in the upper panes or type in the lower pane to apply change for each conflicting line.
5. Once you have applied changes for all the conflicts, click **OK** in the merge tool.
6. Once all the conflicts are resolved click **Close**.

Step 4. Check-in the Merged Changes

In this step you check-in the baseless merged changes

To check-in merged changes:

1. Open **Source Code Explorer**.
2. Right-click the target folder in which you have merged the changes and click **Check-in pending changes**.
3. In the **Check-In Source Files** dialog box, ensure that all the files to be checked in are selected.
4. Click **Check In**.

Additional Resources

- To learn how to merge files and folders, see How to: Merge Files and Folders: http://msdn2.microsoft.com/en-us/library/ms181428(VS.80).aspx
- To learn more about baseless merge, see the /baseless option in “Merge Command”: http://msdn2.microsoft.com/en-us/library/bd6dxhfy(VS.80).aspx
- For more information about other merging options please see "Merge Command" at http://msdn2.microsoft.com/en-us/library/bd6dxhfy(VS.80).aspx
- Team Foundation Server Branching Guidance: http://www.CodePlex.com/BranchingGuidance

How To: Setup a Continuous Integration Build with Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

The Team Foundation Server 2005 does not provide CI solution out of box, but does provide the framework for you to implement your own CI build solution. This How To shows you, how to setup a continuous integration build with Team Foundation Server 2005, using the solution provided by the Visual Studio Team System team. The solution installs a Web service running under the account which has access to the TFS server; you have to register the web service for **CheckinEevnts**. So whenever a checkin happens, the web service initiates a Team Build.

Contents

- Objectives
- Overview
- Summary of Steps
- Before You Begin
- Step 1 - Create and test your build
- Step 2 - Install the Continuous Integration Add On
- Step 3 - Configure the Continuous Integration Add On
- Step 4 - Register for CheckinEvent
- Step 5 - Test the continuous integration build
- Step 6 - Set up email alerts
- Additional Resources

Objectives

- Learn what Continuous Integration build is.
- Learn to setup Continuous Integration build

Overview

Developers always need continuous feedback about the quality of the checkins they make, especially whether their changes are breaking with respect to changes made by other developers. The earlier the feedback the better, it gives developers quick chance of fixing the code, thus unblocking other developers / testers and improving productivity of everyone on the team.

This is where Continuous Integration build type comes into the picture. It is the process of creating builds whenever a developer checks-in the code into Source Control. The CI builds quick feedback to the developers; additionally you can configure your CI builds for quality gates thus ensuring the build quality all the time. The CI build can cause bottlenecks and put developers code at risk of losing as they cannot check-in the partially completed code.

Summary of Steps

- Step 1 - Create and test your build
- Step 2 - Install the Continuous Integration Add On
- Step 3 - Configure the Continuous Integration Add On
- Step 4 - Register for CeckinEvent
- Step 5 - Test the continuous integration build
- Step 6 - Set up email alerts

Before You Begin

Make sure the account that runs your build service has the following permission in Team Foundation Server:

- Permissions - Start a build = Allow

Step 1 - Create and test your build

In this step you create a test CI build and make sure the team build works from within Visual Studio IDE. If it doesn't, then fix it before moving on.

1. Create a Windows Form project which you can use for testing the build.
2. Build the project and make sure that it is working correctly.
3. Check-in your project into source control
4. Create team build
    - In Team Explorer, right-click **Team Builds**, click **New Team Build Type**
    - Complete the self explanatory **Team Build Type Creation Wizard**
5. Verify that the team build is working
    - In Team Explorer, double click the Team Build Project you created
    - From the **Build** menu of Visual Studio instance, click **Build Team Project <<Your Team Build Name>>**
    - Confirm the correct build type is selected and click the **Build** button
6. Review the build output and make sure that there are no errors in the build process

Important – Make sure your build services account (TFSService), has Full Control permission to the shared folder for the drops, specified in the **Team Build Type Wizard**.

Step 2 - Install the Continuous Integration add on

In this step we are going to install the solution provided by Visual Studio Team System team for setting up the CI build. The detailed information about the solution is available at http://msdn2.microsoft.com/en-us/library/ms364045(VS.80).aspx

1. Install the solution on your TFS server from following location - http://download.microsoft.com/download/6/5/e/65e300ce-22fc-4988-97de-0e81d3de2482/ci.msi
2. Make sure you choose “Team Foundation Server” site on the second page of the installer wizard.
3. The setup installation creates a virtual directory underneath the root of the application-tier server for Team Foundation Server. This should be an IIS Web site called "Team Foundation Server" on port 8080.

Step 3 - Configure the Continuous Integration Add On

In this step you will configure the Continuous Integration settings to specify the project to be built, build machine to be used, and the team build type to be used.

1. Make sure that the virtual root runs in the "TfsAppPool" application pool
    - Open IIS manager by clicking **Start** and then **Administrative Tools**, finally select the **IIS Manager**
    - Under the **Web Sites** expand the **Team Foundation Server** web site.
    - Right click on the **CI Web Application** and make sure **Application Pool** is set to **TfsAppPool**.
2. Configuring Continuous Integration Settings
    - Open the **CI Web Application** web.config file located at C:\Program Files\Microsoft Visual Studio 2005 Team Foundation Server\Web Services\CI\
    - Set the following properties by adding a new key under **<appsettings>** node:
        - **TeamFoundationServer** - —URL for the application tier in the format http://machine:8080.
        - **TeamProject**—team project for which you want to enable continuous integration
        - **BuildType**—build type to be used while creating continuous integration builds. Typically this is configured just for builds. You can also add a basic level of testing and static analysis.
        - **Build Machine**—an optional parameter, use this to override the default build machine specified in the build type

Here is the example of the configuration settings

```
….
<add key="1"
value="TeamServer=http://TFSRTM:8080;TeamProjectName=AdventureWorks;Bui
ldType=Test Build"/>
….
```

Step 4 - Register for CheckinEvent

In this step you will register for the **CheckinEvent** event using the **bisubscribe** tool available with Team Foundation Server.

1. Open a command prompt and change to the following location C:\Program Files\Microsoft Visual Studio 2005 Team Foundation Server\TF Setup\
2. Run the following command

```
Bissubscribe /eventType CheckinEvent /address
http://TFSRTM:8080/ci/notify.asmx /deliveryType Soap /domain
http://TFSRTM:8080
```

3. If you get any errors or you want top confirm that you have registered correctly, do the following.
    - Open **SQL Server Management Studio**
    - Open the **tfsIntegration** database
    - Open **tbl_subscription** table

The table will consists all the events already subscribed and you can decide to unsubscribe event if necessary, or you may find the CI solution is already subscribed to the event.

Step 5 - Test the Continuous Integration Build

In this step you verify that the Continious Integration Build is working correctly.

1. Open the Windows Forms Application, created in Step 1
2. Make some nominal changes, ensuring that the build won't break.
3. Check-in the code changes you made.
4. If you have access to the build machine open taskmanager look at CPU usage - it should elevate soon after the checkin to indicate a build is occurring. If you look at the process list you should see msbuild, csc, and/or aspnet_compiler using up cpu cycles - this indicates the build is in progress.
5. Give the build time to run and then dbl-click on **All Build Types** in the **Team Explorer** and look for the build after it completes.

Trouble shooting

In case the build does not show up in the build list, you can trouble shoot at follows

1. Make sure that the event is subscribed correctly, for more information see sub step 3 of Step 4.
2. Make sure that the CI Web Application's web.config is configured correctly, for more information see Step 3.
3. Make sure that the CI Web Application is running in the right context and has access to the TFS server.
4. If the above is correct and still the build fails you can debug the CI Web Application as follows.
    - Create a new web site project
    - Add an existing web site, select the CI Web Application from the local IIS
    - Open notify.cs and set a breakpoint on **Notify** method
    - Click the f5 to start debugging
    - Make changes to Windows Forms test application and checkin the code in source control.
        - If breakpoint is not hit, the event is not being captured - try subscribing again
        - If breakpoint is hit, debug further.

    - Be sure script debugging is enabled in IE (advanced options)

Step 6. Setup email alerts

In this step you will setup and email alert for build completion, so that the concerned parties are informed.

1. Right click the relevant team project in team explorer
2. Select **Project Alerts**
3. Check '**A build completes**' option and enter email address for notification

Additional Resources

- Continuous Integration Using Team Foundation Build
  http://msdn2.microsoft.com/en-us/library/ms364045(VS.80).aspx
- The Visual Studio Team System CI solution MSI -
  http://download.microsoft.com/download/6/5/e/65e300ce-22fc-4988-97de-0e81d3de2482/ci.msi
- Agile Dev Article -
  http://msdn.microsoft.com/msdnmag/issues/06/03/TeamSystem/default.aspx

How To: Setup a Scheduled Build with Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

The Team Foundation Server 2005 does not provide Scheduled Build feature out of box, however you can use the **TFSBuild** command to implement your own Scheduled builds. This How To shows you, how to setup a scheduled build with Team Foundation Server 2005, using **TFSBuild** command and the Windows Task scheduler.

Contents

- Objectives
- Overview
- Summary of Steps
- Before You Begin
- Step 1 - Create and Test the build
- Step 2 - Create the TFSBuild command line
- Step 3 - Test the TFSBuild command line
- Step 4 - Create a batch file
- Step 5 - Test the batch file
- Step 6 - Add a scheduled task
- Step 7 - Test the scheduled task
- Additional Resources

Objectives

- Create a scheduled build using TFSBuild and Windows Scheduler

Overview

For development team it’s important to share the progress they have been making on the project and get timely feedback. This can be achieved with scheduled builds. The source code is complied and released on regular basis to other teams for example testers etc for getting the feedback. The scheduled builds can be nightly, weekly, fortnightly or monthly depending upon the project size and requirements.

Team Build feature in Team Foundation Server does not have support scheduled builds out of box. However, it does provide TFSBuild command utility start team-builds from command line. Any scheduler like the Windows Task Scheduler can be used to run the TFSBuild command utility to start builds at predetermined time.

Summary of Steps:

- Step 1 - Create and Test the build
- Step 2 - Create the TFSBuild command line
- Step 3 - Test the TFSBuild command line
- Step 4 - Create a batch file
- Step 5 - Test the batch file
- Step 6 - Add a scheduled task
- Step 7 - Test the scheduled task

Before You Begin

Make sure the account that runs your build service has the following permission in Team Foundation Server:

- Permissions - Start a build = Allow

Step 1 - Create and Test the build

In this step you create a test scheduled build and make sure the team-build works from within Visual Studio IDE. If it doesn't, then fix it before moving on.

1. Create a Windows Form project which you can use for testing the build.
2. Build the project and make sure that it is working correctly.
3. Check-in your project into source control
4. Create team build
    - In Team Explorer, right-click **Team Builds**, click **New Team Build Type**
    - Complete the self explanatory **Team Build Type Creation Wizard**
5. Verify that the team build is working
    - In Team Explorer, double click the Team Build Project you created
    - From the **Build** menu of Visual Studio instance, click **Build Team Project <<Your Team Build Name>>**
    - Confirm the correct build type is selected and click the **Build** button
6. Review the build output and make sure that there are no errors in the build process

Important – Make sure your build services account (TFSService), has Full Control permission to the shared folder for the drops, specified in the **Team Build Type Wizard**.

Step 2 - Create the TFSBuild command line

In this step you create a command for the **TFSBuild** utility for starting a build.

1. The TFSBuild command utility needs number of parameters to start a build, first you need to determine the parameters to be used.
    - **Team Foundation Server –** The Team Foundation server URL where the solutions being built are checked in
    - **Team Project –** It’s the name of the team project which you want to build.
    - **Build Type –** It’s the Build Type which you have created using the build type wizard.

- **Build Machine –** It's the name of the build server which you want to use for building your project. Its an optional parameter, by default it uses the build server specified in the team build type.
- **Build Directory –** It's the path to the directory where the build occurs. Its an optional parameter, by default it uses the path specified in the tem build type.

2. The build command created is as follows

```
TfsBuild start <<TeamFoundationServer>> <<TeamProject>>
<<BuidlTypeName>>
```

If you plan to override the build machine name and build directory path specified while creating the build type, here is the command created

```
TfsBuild start <<TeamFoundationServer>> <<TeamProject>>
<<BuidlTypeName>> /m:<<MachineName>> /d:<<BuildDirectory>>
```

Step 3 - Test the TFSBuild command line

In this step you test that the TFS command is working correctly.

1. Open the visual studio command prompt
2. Type the **TfsBuild** command as created in Step 2
3. Review the output and make sure that the build works correctly.

Step 4 - Create a batch file

In this step you will create a batch file which will be sued for scheduling the builds

1. Open notepad and type following command, please note we specify the full path to the TFSBuild.exe file so that it can run from windows command prompt. Here is an example of the command

```
"C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\TFSBuild" start
<<TeamFoundationServer>> <<TeamProject>> <<BuildTypeName>>
```

If you are overriding the build machine and build directory path then the command in the batch file would look as follows

```
"C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\TFSBuild" start
<<TeamFoundationServer>> <<TeamProject>> <<BuildTypeName>>
/m:<<BuildMachineName>> /d:<<BuildDirectoryPath>>
```

2. Save the file as a batch file for example batchbuild.bat
3. Place the file in a build scripts location in TFS server Source Control, for example **Main\Scripts**

Step 5 - Test the batch file

In this step you will verify that the batch file you have created works correctly.

1. Open a windows command prompt, not VS command prompt, because by default the windows scheduler will run the batch file in windows command prompt.
2. Start the batch file on the command line.

3. Ensure that the build works correctly without any errors.

Step 6 - Add a scheduled task

In this step you will add a schedule task for running the build command at regular intervals as suitable and required by your project.

1. Go to **Start** and then click **Control Panel**
2. Then select the **Scheduled Tasks** option and click **Add Scheduled Tasks**
3. On the **Scheduled Task Wizard** click **Next.**
4. Click **Browse** button and select the batch file you have created in Step 4
5. Type the name of the task.
6. Select the frequency with which you want to run the tasks, let's say it's **Daily** and then click **Next**
7. Configure the **Start time**, exceptions and **Start date** and click **Next.**
8. Enter user name and password of an account with Start a build permissions and click **Next**
9. Then click **Finish.**

Step 7. Test the scheduled task

In this step you verify that the scheduled task is running at the right time and the builds work properly.

1. Wait for the time at which the scheduled task should execute. Or, Go to **Start** and then click **Control Panel**
2. Then select **Scheduled Tasks** option, from the list right click on your scheduled task and select **Run**.
3. You should see a command prompt appear and your build start.
4. If you do not have access to the build machine at the time the task should execute you can check back later and see if a build has completed.
   - In **Team Explorer** double click **All Build Types**.
   - Look at the list of builds and see if a build completed at the time you scheduled

Additional Resources

- For more information see, "How to: Configure a Scheduled Build (Command Line) " at http://msdn2.microsoft.com/en-us/library/ms181727(VS.80).aspx

How To: Structure Your ASP.NET Applications for Team Foundation Server

Applies To

- Team Foundation Server
- Visual Studio Team System
- ASP.NET Applications

Summary

This How To shows you how to organize and structure your ASP.NET Web applications for Team Foundation Server. It explains an appropriate source tree structure to use within Team Foundation Server source control.

Contents

- Objectives
- Overview
- Summary of Steps
- Step 1. Create Local Folders for Your Web Project
- Step 2. Create a Blank Solution
- Step 3. Add a Web Site to Your Solution
- Step 4. Add a Class Library (Optional)
- Step 5. Check Your Solution Structure
- Step 6. Check Your Local Folder Structure
- Step 7. Add Your Solution to Source Control
- Shared Code Considerations
- Additional Resources

Objectives

- Learn how to structure an ASP.NET application in Team Foundation Server source control.

Overview

This how to shows you how to build a source control folder structure that is appropriate for ASP.NET Web applications. ASP.NET Web projects often include additional class libraries so a structure is required to accommodate those also. Folders to maintain your ASP.NET Web projects are located beneath a **/Main/Source** top level structure in source control. This enables you to easily use additional **Development** and **Maintenance** folders if you need to create branches for isolated development and for release maintenance. For more information about creating this top level folder structure, see “How To: Structure Your Source Control Folders.”

Summary of Steps

- Step 1. Create Local Folders for Your Web Project
- Step 2. Create a Blank Solution
- Step 3. Add a Web Site to Your Solution
- Step 4. Add a Class Library (Optional)
- Step 5. Check Your Solution Structure
- Step 6. Check Your Local Folder Structure
- Step 7. Add Your Solution to Source Control

Step 1. Create Local Folders for Your Web Project

In this step, you create an appropriate local folder structure on your development computer for your Web project. To ensure a consistent approach for your team development and to keep team projects well organized on your development computer, you should keep your entire development source from all of the team projects you are working on grouped together beneath a single root folder such as C:\DevProjects.

Create a top level folder such as C:\DevProjects.

Step 2. Create a Blank Solution

To create an ASP.NET Web Application, start by explicitly creating a Visual Studio Solution (.sln) file and then add your Web site and any supplementary projects you might need, such as class libraries. In the following steps, you create your solutions beneath a top level C:\DevProjects folder

1. From the **File** menu, point to **New** and click **Project**.
2. Expand **Other Project Types** and select **Visual Studio Solutions**.
3. Select **Blank Solution**.
4. Name your solution **MyWebAppSln**.
5. Set the **Location** to C:\DevProjects and click **OK**.

This creates a C:\DevProjects\MyWebAppSln folder. Visual Studio adds the solution (.sln) file and solution user options (.suo) file to this folder. Note that only the .sln file is subsequently added to source control in Step 7.

Step 3. Add a Web Site to Your Solution

In this step, you add an ASP.NET Web site to your solution. The way you do this varies slightly depending on whether you are creating a new file system-based Web site that uses the Visual Studio development Web server or an HTTP Web site that uses IIS.

File System

To add a file system-based Web project to your solution:

1. In **Solution Explorer**, right click **Solution MyWebAppSln**, point to **Add** and click **New Web Site**.

2. In the **Add New Web Site** dialog box, leave the **Location** as **File System**, and the **Language** as **Visual C#**.
3. Set the **Location** directory to C:\DevProjects\MyWebAppSln\Source\MyWebAppWeb.
4. Click **OK** to close the **Add New Web Site** dialog box.

Note that the “Web” suffix is used in this example to clearly denote the folder as a Web site root folder.

HTTP

To create an HTTP-based ASP.NET Web site that uses IIS, start by explicitly creating your virtual directory so that your Web site directory ends up in your defined location rather than beneath \inetpub\wwwroot.

To make your Web site’s virtual directory:

1. Use Windows Explorer to browse to C:\DevProjects\MyWebAppSln\Source
2. Make a new folder beneath this location called **MyWebAppWeb**.
3. Right-click **MyWebAppWeb** and click **Sharing and Security**.
4. Click the **Web Sharing** tab.
5. Click **Share this folder**.
6. Leave the **Alias** as **MyWebAppWeb**, leave the default **Access permissions** and **Application permissions** and click **OK**.
7. Click **OK** and **OK** again.

To add your Web site to your solution:

1. In **Solution Explorer**, right click **Solution MyWebAppSln**, point to **Add** and click **New Web Site**.
2. In the **Add New Web Site** dialog box, set the **Location** to **HTTP** and leave the **Language** as **Visual C#**.
3. Set the **Location** URL to http://localhost/MyWebAppWeb.
4. Click **OK** to close the **Add New Web Site** dialog box.

Visual Studio adds your Default.aspx, Default.aspx.cs files to the C:\DevProjects\MyWebAppSln\Source\MyWebAppWeb folder and creates **Bin** and **App_Data** child folders.

Step 4. Add a Class Library (Optional)

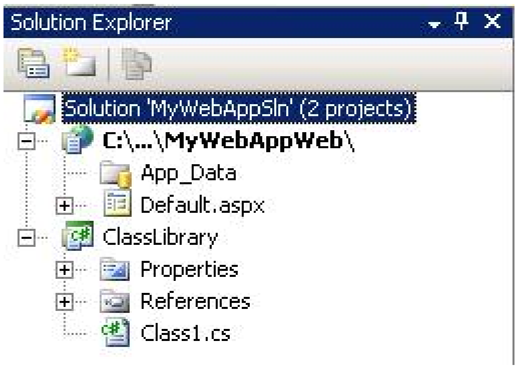
If your Web application requires additional class libraries, add them as follows:

1. In **Solution Explorer**, right-click your **MyWebAppSln** solution, point to **Add** and click **New Project**.
2. Select **Visual C#** as the project type and **Class Library** as the template.
3. Type the name **ClassLibrary**, set the **Location** to C:\DevProjects\MyWebAppSln\Source and click **OK**.

All new projects are added beneath the **Source** folder.

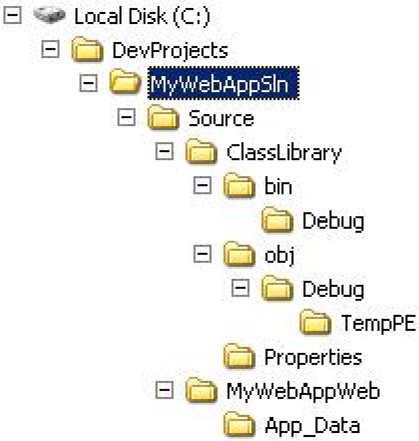
Step 5. Check Your Solution Structure

Use Solution Explorer to verify your solution structure. It should resemble the following:



Step 6. Check your Local Folder Structure

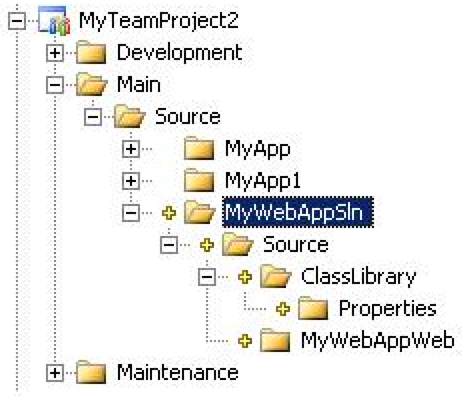
Use Windows Explorer to verify your local folder structure. It should resemble the following:



Step 7. Add Your Solution to Source Control

In this step you add your solution to Team Foundation Server source control.

1. Right-click your solution and click **Add Solution to Source Control**.
2. In the **Add Solution MyWebAppSln to Source Control** dialog box, choose your team project
3. In the dialog box, click **Make New Folder**, name the new folder as **Main**
4. Select the newly created **Main** folder and click **Make New Folder**, name the new folder **Source**
5. Select the newly created **Source** folder and then click **OK** .
6. Check your source control folder structure by clicking **Source Control** within the **Team Explorer**. It should resemble the following:



7. At this point you can view pending changes and check-in your solution source files to the server. To do so, from the **View Menu**, point to **Other Windows** and click **Pending Changes**. Select your project and source files to check-in, supply a check-in comment and click **Check In**.

Shared Code Considerations

For ASP.NET Web applications that reference shared source code, you can consider the following two main options:

- **Reference the code from a shared location**
- **Branch the shared code**

Reference the Code from a Shared Location

With this approach, you map the source from a shared location such as another team project into the workspace on your development computer. This creates a configuration that unifies the shared source from the shared location and your project code on your development computer.

The advantage of this approach is any changes to the shared source are picked up every time you retrieve the latest version of the source into your workspace. For example consider a team project called Common that contains the shared source. To reference the code from this location, you map both team projects to a common path location on your development computer. For example:

- C:\DevProjects\MyWebAppSln\
- C:\DevProjects\SharedCommon\

The following workspace mappings are used:

Source Control Folder	**Local Folder**
$/MyTeamProject1/Main/Source/MyWebAppApp	C:\DevProjects\MyWebAppSln
$/MyTeamProject2/Main/Source/Common	C:\DevProjects\Common

For more information see “Working with multiple team projects in Team Build” at http://blogs.msdn.com/manishagarwal/archive/2005/12/22/506635.aspx

Branch the Shared Code

With this approach, you branch the shared code from the common team project into your team project. This also creates a configuration that unifies the source from the shared location and your project.

The difference with this approach is that the shared source changes are picked up as part of a merge process between the branches. This makes the decision to pick up changes in the shared source much more explicit. You decide when to perform a merge to pick up latest changes.

For example consider the team project called **Common** which contains shared source. To branch the code from this location:

1. In **Source Control Explorer**, right-click the root folder of the **Common** team project.
2. Click **Branch…**
3. In the **Branch** dialog box, set the **Target:** to the root folder of the $/MyTeamProject1/Main/Source/ team project and click **OK**.
4. Once the branch operation is completed do not forget to check-in the branched source code.

Additional Resources

- Team Foundation Server Branching Guidance: http://www.CodePlex.com/BranchingGuidance
- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx

How To: Structure Your Windows Applications for Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To shows you how to organize and structure your Windows Form applications for Team Foundation Server. It explains an appropriate source tree structure to use within Team Foundation Server source control.

Contents

- Objectives
- Overview
- Summary of Steps
- Step 1. Create Local Folders for Your Windows Forms Project
- Step 2. Create a Blank Solution
- Step 3. Add a Windows Forms Project to Your Solution
- Step 4. Add a Control Library (Optional)
- Step 5. Add a Class Library (Optional)
- Step 6. Check Your Solution Structure
- Step 7. Check Your Local Folder Structure
- Step 8. Add Your Solution to Source Control
- Shared Code Consideration
- Complete Folder Structure
- Additional Resources

Objectives

- Learn how to structure a Win Forms application in Team Foundation Server source control.

Overview

This how to shows you how to build a source control folder structure that is appropriate for Windows Forms applications. Windows Forms projects often include additional class and control libraries so a structure is required to accommodate those also. Folders to maintain your Windows Forms projects are located beneath a **/Main/Source** top level structure. This enables you to easily use additional **Development** and **Maintenance** folders if you need to create branches for isolated development and for release maintenance. For more information about creating this top level folder structure, see “How To: Structure Your Source Control Folders.”

Summary of Steps

- Step 1. Create Local Folders for Your Windows Forms Project
- Step 2. Create a Blank Solution
- Step 3. Add a Windows Forms Project to Your Solution
- Step 4. Add a Control Library (Optional)
- Step 5. Add a Class Library (Optional)
- Step 6. Check Your Solution Structure
- Step 7. Check Your Local Folder Structure
- Step 8. Add Your Solution to Source Control

Step 1. Create Local Folders for Your Windows Forms Project

In this step, you create an appropriate local folder structure on your development computer for your Windows Forms project. To ensure a consistent approach for your team development and to keep team projects well organized on your development computer, you should keep your entire development source from all of the team projects you are working on grouped together beneath a single root folder such as C:\DevProjects.

Create a top level folder such as C:\DevProjects.

Step 2. Create a Blank Solution

To create a Windows Forms Application, start by explicitly creating a Visual Studio Solution (.sln) file and then add your Windows Forms project and any supplementary projects you might need, such as class or control libraries. In the following steps, you create your solutions beneath a top level C:\DevProjects

1. From the **File** menu, point to **New** and click **Project**.
2. Expand **Other Project Types** and select **Visual Studio Solutions**.
3. Select **Blank Solution**.
4. Name your solution **MyApp**.
5. Set the **Location** to C:\DevProjects and click **OK**.

This creates a C:\DevProjects\MyApp folder. Visual Studio adds the solution (.sln) file and solution user options (.suo) file to this folder. Note that only the .sln file is subsequently added to source control in Step 8.

Step 3. Add a Windows Forms Project to Your Solution

In this step you add a new Windows Forms project to your solution.

1. In **Solution Explorer**, right click **Solution MyApp**, point to **Add** and click **New Project…**.
2. In the **Add New Project** dialog box, select **Visual C#** as the project type and **Windows Application** as the template.
3. Set the **Location** to C:\DevProjects\MyApp\Source directory.

4. Set the name of your project to **MyWinFormApp**.
5. Click **OK** to close the **Add New Project** dialog box and add your project.

Step 4. Add a Control Library (Optional)

If your Windows Forms application requires additional control libraries, add them as follows:

1. In **Solution Explorer**, right-click your solution, point to **Add** and click **New Project…**.
2. In the **Add New Project** dialog box, select **Visual C#** as the project type and **Windows Control Library** as the template.
3. Set the **Location** to C:\DevProjects\MyApp\Source directory.
4. Set the name of your Windows Control Library to **ControlLibrary**.
5. Click **OK** to close the **Add New Project** dialog box.

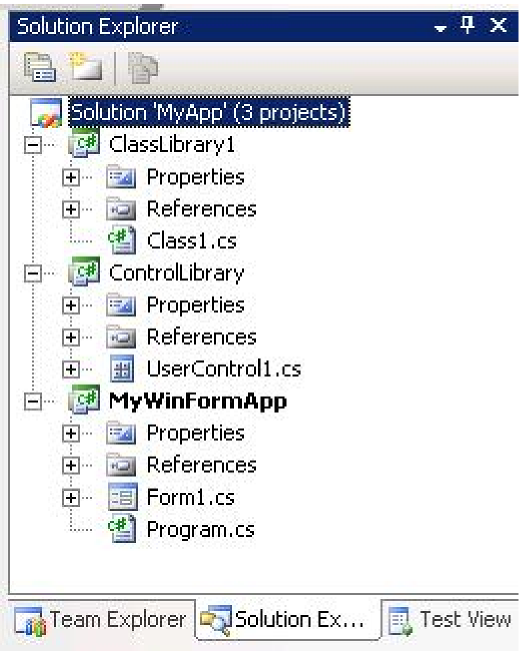
Step 5. Add a Class Library (Optional)

If your Windows Forms application requires additional class library projects, add them as follows:

1. In **Solution Explorer**, right-click your solution, point to **Add** and click **New Project…**.
2. In the **Add New Project** dialog box, select **Visual C#** as the project type and **Class Library** as the template.
3. Set the **Location** to C:\DevProjects\MyApp\Source directory.
4. Set the name of your Windows Class Library to **ClassLibrary1**
5. Click **OK** to close the **Add New Project** dialog box.

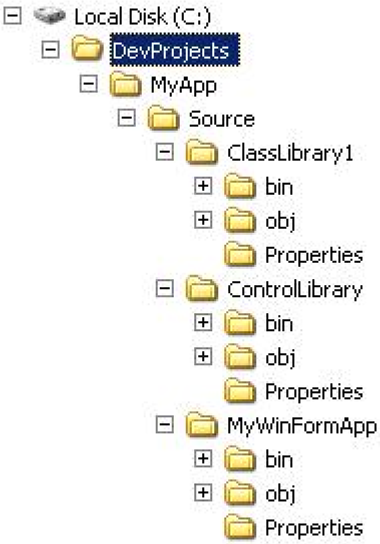
Step 6. Check Your Solution Structure

Use Solution Explorer to verify your solution structure. It should resemble the following:



Step 7. Check Your Local Folder Structure

Use Windows Explorer to verify your local folder structure. It should resemble the following:

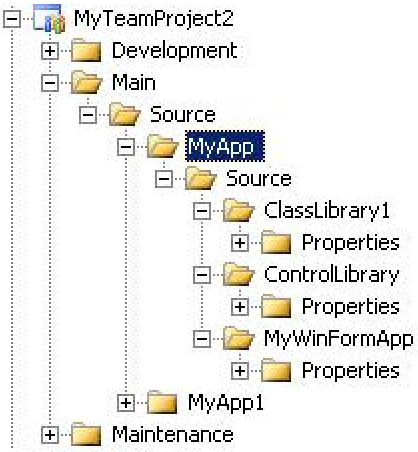


Step 8. Add Your Solution to Source Control

In this step you add your solution containing your Windows Form project and optional class libraries and control libraries to Team Foundation Server source control.

1. Right-click your solution **MyApp** and click **Add Solution to Source Control**.

2. In the **Add Solution MyApp to Source Control** dialog box, choose your team project.
3. In the dialog box, click **Make New Folder**, name the new folder as **Main.**
4. Select the newly created **Main** folder and click **Make New Folder**, name the new folder **Source**.
5. Select the newly created **Source** folder and then click **OK**.
6. Check your source control folder structure by double-clicking **Source Control** within the **Team Explorer**. This displays source control explorer. Your structure should resemble the following:



7. At this point you can view pending changes and check-in your source files to the server. To do so, from the **View Menu**, point to **Other Windows** and click **Pending Changes**. Select your project and source files to check-in, supply a check-in comment and click **Check In**.

Shared Code Considerations

For Windows Forms applications that reference shared source code, you can consider the following two main options:

- **Reference the code from a shared location**
- **Branch the shared code**

Reference the Code from a Shared Location

With this approach, you map the source from a shared location such as another team project into the workspace on your development computer. This creates a configuration that unifies the shared source from the shared location and your project code on your development computer.

The advantage of this approach is any changes to the shared source are picked up every time you retrieve the latest version of the source into your workspace. For example consider a team project called Common that contains the shared source. To reference the code from this location, you map both team projects to a common path location on your development computer. For example:

- C:\MyProjects\MyApp\
- C:\MyProjects\SharedCommon\

The following workspace mappings are used:

Source Control Folder	**Local Folder**
$/MyTeamProject1/Main/Source/MyApp	C:\DevProjects\MyApp
$/MyTeamProject2/Main/Source/Common	C:\DevProjects\Common

For more information see “Working with multiple team projects in Team Build” at http://blogs.msdn.com/manishagarwal/archive/2005/12/22/506635.aspx

Branch the Shared Code

With this approach, you branch the shared code from the common team project into your team project. This also creates a configuration that unifies the source from the shared location and your project.

The difference with this approach is that the shared source changes are picked up as part of a merge process between the branches. This makes the decision to pick up changes in the shared source much more explicit. You decide when to perform a merge to pick up latest changes.

For example consider the team project called **Common** which contains shared source. To branch the code from this location:

1. In **Source Control Explorer**, right-click the root folder of the **Common** team project.
2. Click **Branch…**
3. In the **Branch** dialog box, set the **Target:** to the root folder of the $/MyTeamProject1/Main/Source/ team project and click **OK**.
4. Once the branch operation is completed do not forget to check-in the branched source code.

Complete Folder Structure

A more complete folder structure in source control showing additional folders for unit tests, build output, documentation, build scripts and tests is shown here.

/Main	→ Contains .sln files that span projects
/Source	
/MyApp	→ Contains MyApp.sln file
/Source	
/MyWinFormApp	→ Contains MyWinFormApp.csproj and forms
/ControlLibrary	→ Contains ControlLibrary.csproj and other

control class files
/ClassLibrary1 → Contains ClassLibrary1.csproj and other class files
/UnitTests
  /MyWinFormsAppTests → Contains unit test project for MyWinFormsApp
  /ControlsLibraryTests → Contains unit tests for ControlsLibrary
  /ClassLibrary1Tests → Contains unit tests for ClassLibrary1
/Build → Contains build out put (binaries)
/Docs → Contains design docs etc
/Script → Contains build scripts, database scripts etc
/Tests → Container folder for tests
  /FunctionalTests → Contains functional test scripts and code
  /PerformanceTests → Contains performance test scripts and code
  /SecurityTests → Contains security test scripts and code

Additional Resources

- Team Foundation Server Branching Guidance: http://www.CodePlex.com/BranchingGuidance
- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx

How To: Structure Your Source Control Folders in Visual Studio Team Foundation Server

Applies To

- Visual Studio Team Foundation Server

Summary

This How To shows you how to structure your folders in Team Foundation Server by using lessons learned in practice. The folder structure presented here is a starting point. More important than the structure, is the rationale behind it. Use the rationale to help evaluate an appropriate folder structure and folder naming convention for your own scenario.

Contents

- Objectives
- Overview
- Summary of Steps
- Step 1. Create a workspace mapping
- Step 2. Create your Main folder
- Step 3. Create folders for your project artifacts
- Step 4. Add your solution to your source tree
- Step 5. Consider creating a branched Development folder for isolation
- Step 6. Consider creating a Maintenance folder for release builds you need to isolate
- Additional Resources

Objectives

- Learn how to structure and name your source control folders

Overview

This how to shows you how to build an appropriate source control folder structure that you can use with most common project types. The folder structure presented in this How To uses three top level folders:

- **Main**. This is the main root folder which acts as a container folder for your main source tree together with accompanying project artifacts such as design documentation, scripts, and test cases. The **Main** folder also contains your Visual Studio Solution (.sln) files.
- **Development**. This is an optional root level folder if you need to create isolation for features or teams. This folder is created by branching from **Main**.
- **Maintenance**. This is an optional root level folder for released builds that you need to isolate for example for ongoing maintenance to a particular release.

The folder structure presented in this How To is summarized here:

- **/Main** → Can contain solution (.sln) files for solutions that span projects
  - /Source
    - /MyApp1 → Contains MyApp1.sln file
      - /Source → Contain folder for all source
        - /ClassLibrary1 → Contains ClassLibrary1.csproj
        - /MyApp1Web → Contains Default.aspx
  - /Build → Contains build output (binaries)
  - /Docs → Contains product docs etc
  - /Scripts → Contains build scripts
  - /Tests → Container for tests

- **/Development**
  - /FeatureBranch1
    - /Source
      - /MyApp1
        - /Source
          - /MyApp1Web
          - /ClassLibrary1
  - /FeatureBranch2

- /**Maintenance**
  - /Release1
    - /Source
      - /MyApp1
        - /Source
          - /MyApp1Web
          - /ClassLibrary1
  - /Release 1.1
  - /Release 1.2

Remember that this folder structure is a starting point with a suggested set of folder names. More important than the structure and naming convention is the rationale behind it and the purpose of the different folders. For more information about the rationale for this structure, see “Ch 5 – Defining Your Branching and Merging Strategy.”

Summary of Steps

- Step 1. Create a workspace mapping
- Step 2. Create your **Main** folder
- Step 3. Create folders for your project artifacts
- Step 4. Add your solution to your source tree
- Step 5. Consider creating a branched **Development** folder for isolation
- Step 6. Consider creating a **Maintenance** folder for release builds you need to isolate

Step 1. Create a Workspace Mapping

In this step you create a workspace mapping to define the mapping between the folder structure on the TFS server and client. You need to do this so that you can create a source tree structure. First the source tree structure is created in your workspace and then you need to perform a check-in to your Team Foundation Server.

You can create a workspace mapping in one of two ways:

- **Set the workspace mapping explicitly**
- **Perform a get operation on your team project.**

To set a workspace mapping explicitly

1. In Visual Studio from the **File** menu, point to **Source Control** and then click **Workspaces**.
2. In the **Manage Workspaces** dialog box, select your computer name and click **Edit**.
3. In the **Edit Workspace** dialog box, in the **Working folders** list, click **Click here to enter a new working folder**.
4. Click the ellipsis button, select your team project for example **MyTeamProject1**, and click **OK**.
5. Click the local folder cell to display another ellipsis button.
6. Click the ellipsis button beneath **Local Folder** and browse to and select the local folder on your development computer where you want to locate your team project, for example C:\DevProjects\MyTeamProject1.
7. Click **OK** and then **OK** again to close the **Edit Workspace** dialog box**.**
8. Click **OK** in response to the Microsoft Visual Studio message box that informs you than one or more working folders have changed.
9. Click **Close** to close the **Manage Workspaces** dialog box.

To perform a Get operation on your team project

1. In **Team Explorer**, expand your team project node, for example **MyTeamProject1**.
2. Double click **Source Control** beneath your team project.
3. In **Source Control Explorer**, right-click the root folder **MyTeamProject1** and click **Get Latest Version**.
4. In the **Browse For Folder** dialog box select your local path, for example C:\DevProjects\MyTeamProject1 and then click **OK**. This maps the team project root folder within Team Foundation Server to a local path on your computer.

Step 2. Create Your Main Folder

Create a new root folder in your Team Project called **Main**. You use the **Main** folder as a root level container folder for your source and other project artifacts. You can also use the **Main** folder to store any Visual Studio solution (.sln) files that span multiple projects. Solution files are also maintained in their own application folders lower in the tree structure.

/Main

To create a Main folder:

1. From Team Explorer, expand your team project and double-click **Source Control**.
2. Select your Team Project root folder within Source Control Explorer.
3. In the right-hand pane, right-click and then click **New Folder**.
4. Type **Main** and press ENTER.

Step 3. Create Folders for Your Project Artifacts

Create folders beneath **Main** to hold your main source code tree and accompanying artifacts such as builds, documentation, script files and tests.

/Main
    /Build
    /Docs
    /Scripts
    /Source
    /Tests

To create folders for your project assets:

1. Expand your team project and Select the **Main** folder in the left hand pane of **Source Control Explorer**.
2. Right-click in the right-hand pane, click **New Folder**, type **Build** and then press ENTER.
3. Repeat step 2 to create the remaining folders beneath **Main**.

Step 4. Add Your Solution to Your Source Tree

In this step, you add your Visual Studio solution (.sln), project (.vsproj, .vbproj) and source files to your source tree in Team Foundation Source Control.

To add your solution to your source tree:

1. Open your solution by using Visual Studio.
2. Right-click your solution in Solution Explorer and click **Add Solution to Source Control**.
3. If you have already established a workspace mapping the solution is added at this point. Otherwise you are prompted to select your team project and target folder in source control. In the **Add Solution to Source Control** dialog box, expand your **Team Project** folder, expand **Main** and select **Source**.
4. Make sure that the correct **Local Workspace** is selected. This should be the same workspace that you used while creating your folder structure in Team Foundation Source control.
5. Click **OK** to add your solution to source control.

You source control folder structure should resemble the following:

```
/Main
        /Source
                /MyApp1
                        /Source
                                /MyApp1Web
                                /ClassLibrary1
        /Build
        /Docs
        /Scripts
        /Tests
```

For more information about how to setup an appropriate client-side folder structure, see:

- “How To: Structure Your Windows Applications for Team Foundation Server.”
- “How To: Structure Your ASP.NET Applications for Team Foundation Server.”

With Unit Tests

If your project also includes unit tests, consider the following folder structure to store your unit tests separate from your main source folder.

```
/Main
        /Source
                /MyApp1
                        /Source
                                /MyApp1Web
                                /ClassLibrary1
                        /UnitTests
        /Build
        /Docs
        /Scripts
        /Tests
```

Step 5. Consider Creating a Branched Development Folder for Isolation

If you need to create isolation for features or teams, consider creating a **Development** folder by branching from the **Main** folder. Note that you must check any pending changes in before you can branch.

To create a branched **Development** folder:

1. Create a root level **Development** folder (as a sibling of **Main**).
2. Create a sub folder called **FeatureBranch1**.
3. Select the $/TeamProject/Main/Source folder, right-click and click **Branch**.

Note: If **Branch** is grayed out, make sure you have checked in any pending changes.
4. In the **Branch** dialog box, click **Browse** and select MyTeamProject/Development/FeatureBranch1 and click **OK**.
5. Click **OK** to create the branch.
6. Check-in your pending changes to push them to the server.

Your new folder structure in Source Control should resemble the sample one shown here.

/Development
- **/FeatureBranch1**
  - **/Source**
    - /MyApp1
      - /Source
        - /MyApp1Web
        - /ClassLibrary1
- **/FeatureBranch2**

/Main
- /Build
- /Docs
- /Scripts
- /Source
- /Tests

Step 6. Consider Creating a Maintenance Folder for Release Builds You Need to Isolate

If you need to perform maintenance on a build, while continuing development, consider creating a **Maintenance** folder. Beneath this folder you can create multiple sub folders, one per release.

To create a branched Maintenance folder:

1. Create a root level **Maintenance** folder (as a sibling of **Main** and **Development**).
2. Create a sub folder called **Release1**.
3. Select the $/TeamProject/Development/Source folder, right-click and click **Branch**. **Note**: If **Branch** is grayed out, make sure you have checked in any pending changes.
4. In the **Branch** dialog box, click **Browse** and select $/TeamProject/Maintenance/Release1 and click **OK**.
5. Click **OK** to create the branch.

Your new folder structure in Source Control should resemble the sample one shown here.

/Maintenance
- /Release1
  - /Source
    - /MyApp1

/Source
/MyApp1Web
/ClassLibrary1
/Release 1.1
/Release 1.2

Additional Considerations

- Do not branch unless you need to. You can label releases and branch at a later time if you need isolation.
- The above folder structure is ideally suited to scenarios where your project consists of one Visual Studio solution (.sln) file containing all of your project files. In this scenario, the **Main** folder contains the .sln file and you have one sub folder for each project file (.vsproj, .vbproj). In the above example project folders are represented by MyApp1, MyApp2 etc. You can also use the above approach for a single project file, for example if you use a single project, with no solution file.
- For multi-solution scenarios, you can locate multiple .sln files in the **Main** folder.

Additional Resources

- Team Foundation Server Branching Guidance: http://www.CodePlex.com/BranchingGuidance
- For more information on creating a workspace, see “How to: Create a Workspace”: http://msdn2.microsoft.com/en-us/library/ms181384(VS.80).aspx