TEAM FOUNDATION SERVER 2008 IN ACTION

Jamil Azher

MANNING

Team Foundation Server 2008
in Action

by Jamil Azher

Chapter 2

brief contents

2 Exploring the changes in TFS 2008

This chapter covers

- Architecture of Team Build
- Creating and managing builds
- New functionalities in version control

Visual Studio 2005 Team Foundation Server (TFS 2005) was a version one product. As such, it had certain deficiencies that are typically inherent in the first generation of such an ambitious system. Visual Studio 2008 Team Foundation Server (TFS 2008) delivers improvements in several areas, such as enhanced features in Team Build and Team Foundation version control (TFVC), better performance and scalability, more flexible system configuration, simplified installation, and support for SharePoint Server 2007 and Microsoft SQL Server 2008.

Although the changes are generally incremental in nature, they add up to make TFS 2008 a more reliable, usable, and extendable product.

This chapter discusses some of the new features available in TFS 2008. We look at some of the important changes that are likely to impact your software development process. Specifically, we focus on the improvements made in Team Build

and Team Foundation version control. The material in this chapter is introductory in nature.

In this chapter, you'll learn about the following:

- *Topology and configuration of build agents and Team Build service*—A build agent represents an instance of the Team Build service that can generate builds. Learn how to create build agents and configure various properties (such as security, build directory, and communication port) to suit your build environment.
- *How to create new build definitions and queue builds*—A build definition contains metadata (such as solutions to build, tests to run, drop management policies, and so on) regarding a build. At runtime, a build definition is handed over to a build agent for execution. A build agent can execute a single build definition per team project at a time. Additional build requests are queued. You can prioritize these builds as well as remove them from the queue. You can also stop or pause builds that are in progress. Learn how to create continuous integration (CI builds), rolling builds, and scheduled builds.
- *How to execute parallel builds*—Executing builds in parallel (building configurations and solutions simultaneously) on multi-processor or multi-core machines can speed up the build process significantly for large projects. Learn how to configure Team Build to support parallel builds and what takes place behind the scenes.
- *How to use the Build Explorer*—The Build Explorer serves as the build management dashboard. You can view the status of queued and completed builds, set priorities, start and stop builds, manage build agents, and filter the displayed list using various criteria.
- *How to manage build qualities*—Build qualities indicate the status of a build after evaluation by the Software Quality Assurance (SQA) group, or by the Central Build group, depending upon how the roles in your organization are structured. You can now customize the available build qualities in Team Build. Learn how to define build qualities and assign them to individual builds.
- *How to generate incremental builds*—Downloading only the modified source files and building only the corresponding binaries can significantly speed up the build process for large projects. Learn how to use the new properties offered by Team Build and perform incremental builds.
- *How to use the Team Build object model*—The new object model for Team Build exposes many more ways to interact with the build system. Learn about the main interfaces and classes. Create sample programs to discover real-time status of build agents, queue builds, and to find a build given a work item number.
- *How to get the latest version on check-out*—Team Foundation version control now offers the capability to automatically download the latest version when you check out a file for editing. Learn about the pros and cons of this approach and how to turn it on or off.

- *How to work with files in offline mode*—Visual Studio 2008 offers built-in support for manipulating source files in offline mode when Team Foundation Server 2008 isn't available. Learn how to use the offline feature and how to synchronize the modified files with TFVC once TFS 2008 is available again.
- *How to compare folders*—You can now compare folders (in addition to files) in TFVC. Both local and server folders can be selected for comparison. Learn how to use this feature and configure various options foroptimum results.
- *How to use annotations*—Annotations enable you to see the change history of a file on a line-by-line basis. This information is helpful for understanding the evolution of a file at a fine-grained level. Learn how to determine who changed what and when.

2.1 Team Build

Team Build has gone through significant changes in TFS 2008. The changes involve how builds are set up as well as the internal structure of the Team Build targets file. In this section, we look at the major changes and how they affect the way you use Team Build.

2.1.1 Topology and security

The most important topology change in Team Build involves using Windows Communication Foundation (WCF) web services to talk to the build agent (the build machine) from the application tier machine (see figure 2.1). The application tier machine in TFS 2005 used .NET remoting to talk to the build machine, whereas the application tier machine in TFS 2008 uses WCF. The WCF service is hosted in a managed Windows service in the build machine. Consequently, Internet Information Server (IIS) is not required to be installed on the build computer.

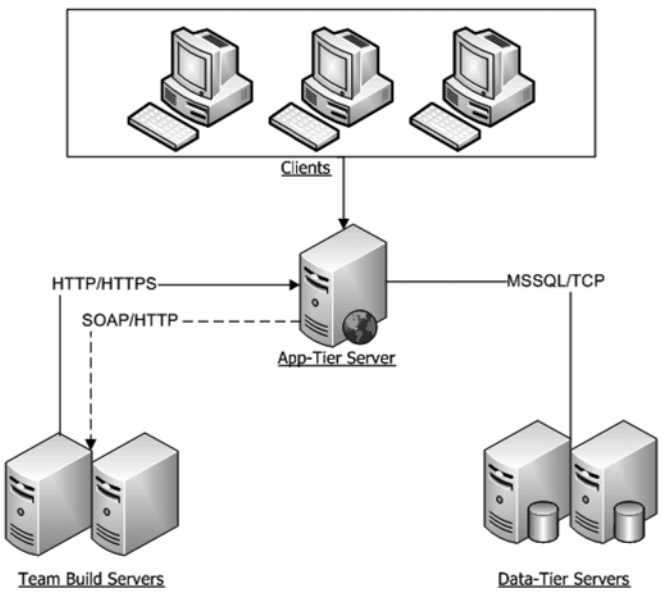

Figure 2.1 TFS 2008 uses WCF web services to communicate with build agents.

The Team Build service is implemented in an assembly named TFSBuildService.exe located in the %ProgramFiles%\Microsoft Visual Studio 9.0\Common7\IDE\PrivateAssemblies folder. Within this assembly, a class named `Microsoft.TeamFoundation.Build.Agent.BuildServiceManager` (inherited from `System.ServiceProcess.ServiceBase`) implements the Windows service. In the `OnStart` method (inherited from the `ServiceBase` class), an instance of the `System.ServiceModel.ServiceHost` class is created for hosting the WCF service. The main class for implementing the WCF service is called `Microsoft.TeamFoundation.Build.Agent.AgentService`. This class implements the WCF service contract and exposes key methods for starting and stopping builds, getting information about in-progress builds, and so on (see figure 2.2).

You can configure the build agent to use a secure HTTPS communication channel as well as require a client certificate. You can also select the authentication protocol to be used, such as NTLM (the default), Kerberos, and so forth. To further limit access, you can specify a single user account that's allowed to communicate with the build agent.

The security settings are specified in the TFSBuildService.exe.config file, located in the %ProgramFiles%\Microsoft Visual Studio 9.0\Common7\IDE\PrivateAssemblies folder. When using HTTPS, also make sure that the Require Secure Channel (HTTPS) check box is selected in the Build Agent Properties dialog box (see figure 2.3). To learn about how to set up build agents to require HTTPS, visit http://msdn2.microsoft.com/en-us/library/bb778431(VS.90).aspx.

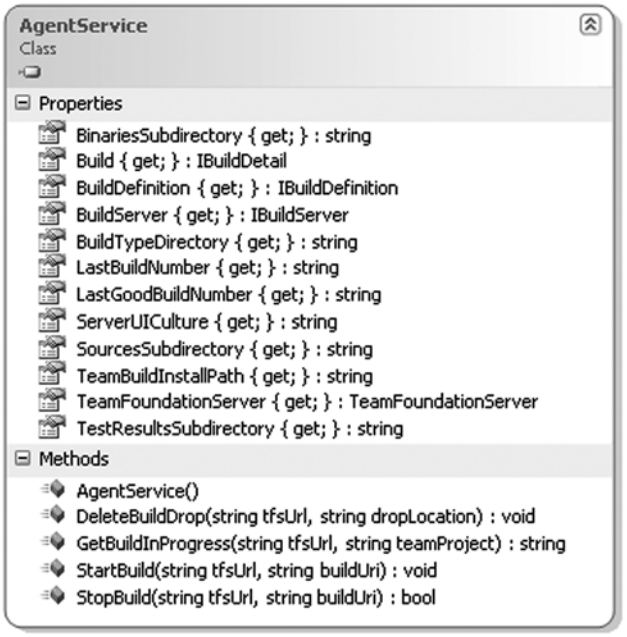

Figure 2.2 The `AgentService` class contains high-level methods and properties for managing the build process.

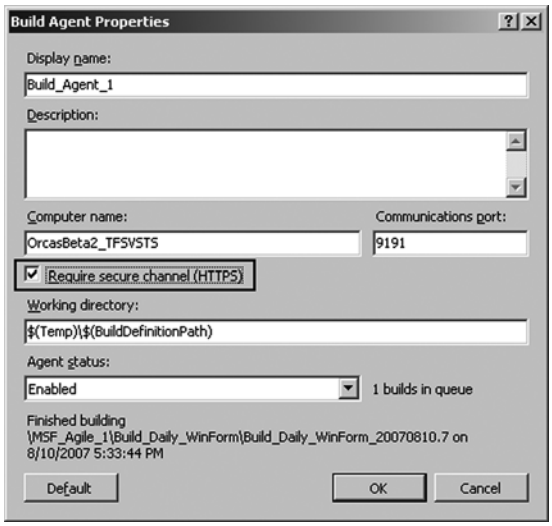

Figure 2.3 You can use HTTPS to secure communications with build agents.

2.1.2 Build agents and team projects

In TFS 2008, the notion of a build machine has been abstracted to some degree. Conceptually, a build agent represents a service that can create builds. The long-term goal is to eventually have a pool of build agents, each publishing information about supported team projects as well as other operational metadata. The published information will enable you to select a build agent to generate a particular build, based on current conditions and performance characteristics.

In TFS 2008, build agents are scoped to team projects. In the Build menu, if you click Manage Build Agents, the Manage Build Agents dialog box appears, displaying only the build agents that belong to the currently selected team project (see figure 2.4). This means that if you want to create builds for different team projects on the same build machine, you need to define a separate build agent for each team project. This behavior is inconvenient and counterintuitive, and is expected to be changed in a future version.

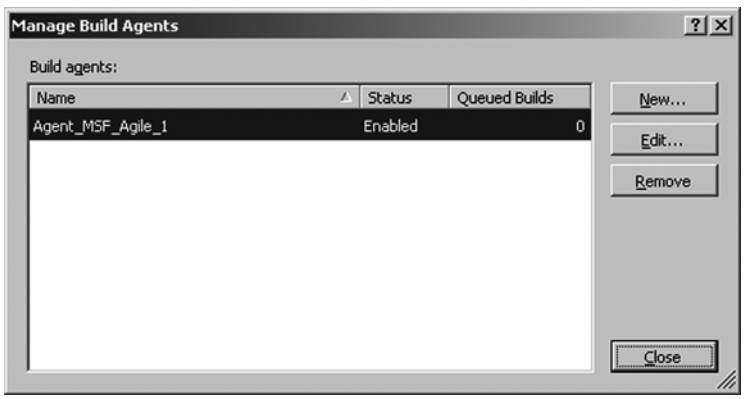

Figure 2.4 The Manage Build Agents dialog box enables you to create, update, and delete build agents.

Although typically a single build agent runs on a given build machine, you can manually configure multiple build agents to run on the same machine (listening on different ports). A typical scenario might be to use the same physical build machine to accept build requests from multiple TFS servers.

2.1.3 Understanding the structure of the build directory

In TFS 2005, you could specify the build directory when launching a build. In TFS 2008, the build directory path is a property associated with a build agent (see figure 2.3 displayed earlier), and can't be specified at runtime.

In TFS 2008, the default directory path associated with a build agent is `$(Temp)\$(BuildDefinitionPath)`. The `$(Temp)` variable represents the temp directory associated with the account under which the Team Build service is running (for example, C:\Documents and Settings\TFSBuild\Local Settings\Temp on my machine). The `$(BuildDefinitionPath)` variable expands to <TeamProject_Name>\<BuildDefinition_Name>. For example, if the team project name is MSF_Agile_1 and the build definition name is Nightly_Build, then `$(BuildDefinitionPath)` will expand to MSF_Agile_1\Nightly_Build.

If you're concerned about the length of the path, you can use the `$(BuildDefinitionId)` variable instead of `$(BuildDefinitionPath)`. `$(BuildDefinitionId)` expands to an integer associated with the build definition. You can also use environment variables to represent the path, or simply hard-code it.

Under the path defined by `$(Temp)\$(BuildDefinitionPath)`, Team Build creates the following three subdirectories (see figure 2.5):

- *Sources*—Contains the source files downloaded from TFVC
- *Binaries*—Contains the generated binaries
- *BuildTypes*—Contains the TFSBuild.proj file, the TFSBuild.rsp file, and build log

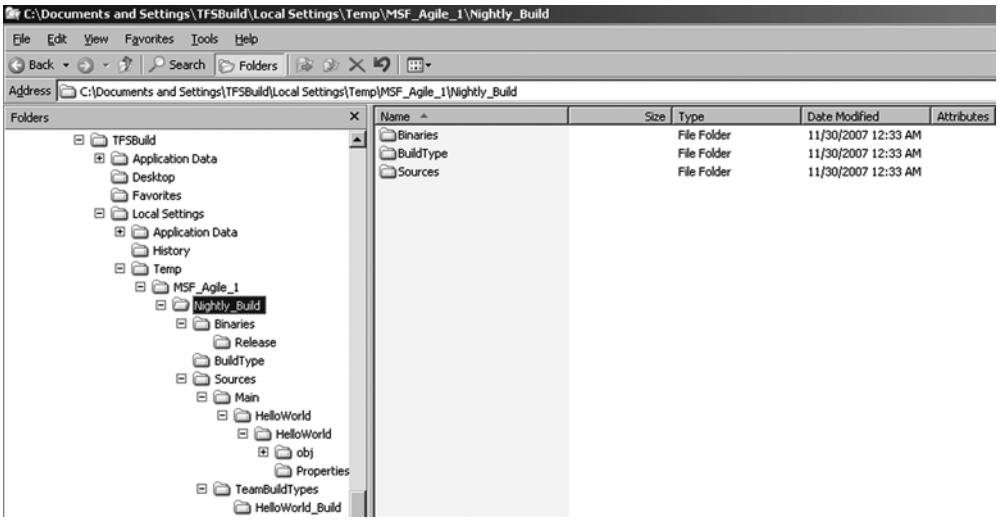

Figure 2.5 The Build directory contains various subdirectories for downloaded source files and generated build binaries.

If you want to change the names of the Sources and Binaries subdirectories, you can modify the `SourcesSubdirectory` and `BinariesSubdirectory` keys, respectively, in the TFSBuildService.exe.config file (located in the %ProgramFiles%\Microsoft Visual Studio 9.0\Common7\IDE\PrivateAssemblies folder). You can also specify the subdirectory for storing test results (not displayed in figure 2.5) using the `TestResultsSubdirectory` key.

Understanding the structure of the build directory is important in various scenarios. For example, if you encounter the 260-character limit in Windows for file paths, you can use the techniques described earlier to shorten the path of the build directory. (For more information regarding the 260-character limit, visit http://msdn2.microsoft.com/en-us/library/aa365247.aspx.) Furthermore, if you create multiple build agents (for processing builds from the same team project or for processing builds from multiple TFS servers) on the same machine, you need to specify different build directories to prevent potential conflicts.

2.1.4 Creating a new build definition

Build definitions in TFS 2008 replace the old build types of TFS 2005. A build type in TFS 2005 was represented by three files—TFSBuild.proj, TFSBuild.rsp, and WorkspaceMappings.xml. These files were located in the $/<Team Project>/TeamBuildTypes/<Build_Type> folder in TFVC. A build definition in TFS 2008 encapsulates more information (such as retention policies and triggers) and can be queued for execution. The MSBuild project-related data for a build definition is still stored in the TFVC repository (in TFSBuild.proj and TFSBuild.rsp files located in your chosen TFVC folder). But other information associated with a build definition is stored in the TFSBuild database (for example, workspace mapping information is stored in tbl_WorkSpaceMapping table, drop location is stored in tbl_BuildDefinition, build agent information is stored in tbl_BuildAgent, and so on). Unlike build types in TFS 2005, you get a user interface in TFS 2008 to edit some of the information associated with a build definition, which you access by right-clicking the build definition and selecting Edit Build Definition from the context menu. However, the MSBuild project-related information still has to be edited manually by checking out the TFSBuild.proj and/or TFSBuild.rsp files, making changes, and checking them back in.

To create a new build definition, right-click the Builds node in Team Explorer and click New Build Definition from the context menu. The Build Definition dialog box has six tabs oriented vertically. Let us review the functionality of each tab.

THE GENERAL TAB

The General tab (see figure 2.6) contains the name of the build definition as well as an optional description. You can deactivate a build definition by selecting the Disable This Build Definition check box. The ability to disable a build definition is useful, for example, when you want to temporarily disable a scheduled or continuous integration build definition while the codebase is being fixed. (You wouldn't want broken builds to be piling up based on a faulty codebase.) But if there are builds that are already

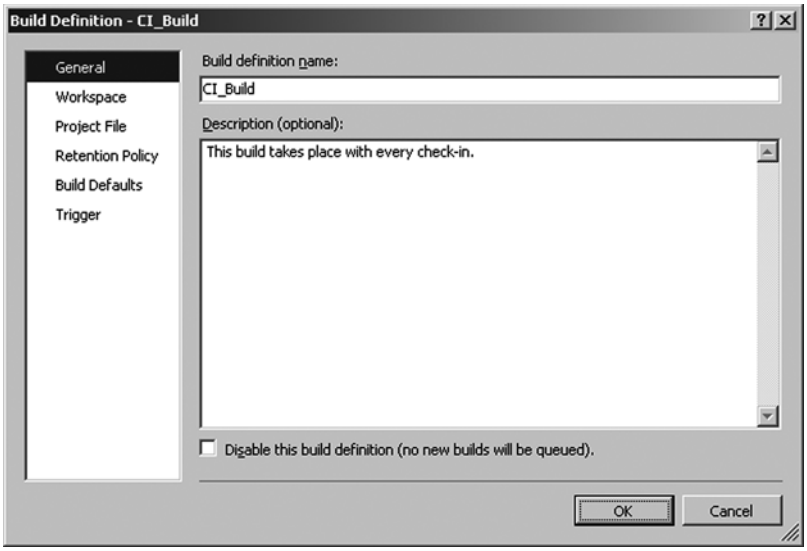

Figure 2.6 Use the General tab to specify the name, description, and current availability of the build definition.

queued or running, those won't be affected when the Disable This Build Definition check box is selected.

THE WORKSPACE TAB

The Workspace tab (see figure 2.7) allows you to construct a *build workspace*. A build workspace maps TFVC folders to local folders. The local folders reside under the root

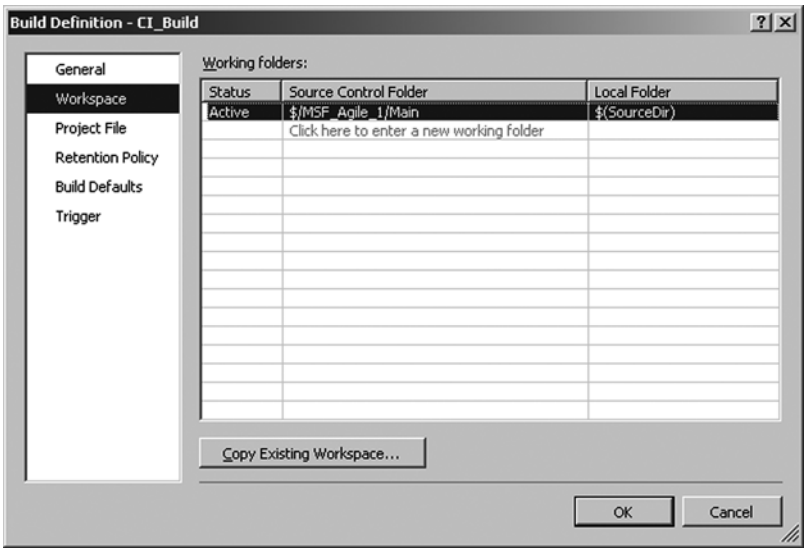

Figure 2.7 Use the Workspace tab to map TFVC folders to local folders.

build directory specified for the build agent. Unlike TFS 2005, you can't specify the build directory at runtime. If you've used TFS 2005, you'll find that defining build workspaces is more intuitive in TFS 2008. As in TFS 2005, a build workspace filters the solutions available for generating builds. Build workspaces also filter the files that are downloaded from TFVC during a build (as in TFS 2005, only the files located in the specified server folders are downloaded). Furthermore, as in TFS 2005, build workspaces limit the changesets associated with a build by labeling only the files that belong to the specified server folders. However, unlike TFS 2005, build workspaces in TFS 2008 allow you to select code from multiple team projects. Instead of creating a workspace from scratch, you can copy an existing workspace from TFVC by clicking Copy Existing Workspace.

THE PROJECT FILE TAB

The Project File tab (see figure 2.8) enables you to create a new TFSBuild.proj file or select an existing one. Note that unlike TFS 2005, where the TFSBuild.proj file was required to be located in the $/<Team Project>/TeamBuildTypes/<Build_Type> folder, in TFS 2008, this file can be in any folder in TFVC. This allows you to branch and merge the TFSBuild.proj file along with the corresponding source files. If a TFSBuild.proj file hasn't been selected, a Create button appears in the Project File tab. Click Create to construct a new TFSBuild.proj file. This action displays the MSBuild Project File Creation Wizard (see figure 2.9). This wizard is similar to the one available in TFS 2005 for creating build types. You can select the solutions to build (order the solutions based upon dependencies), specify the build verification tests to run, collect code analysis data, and so on.

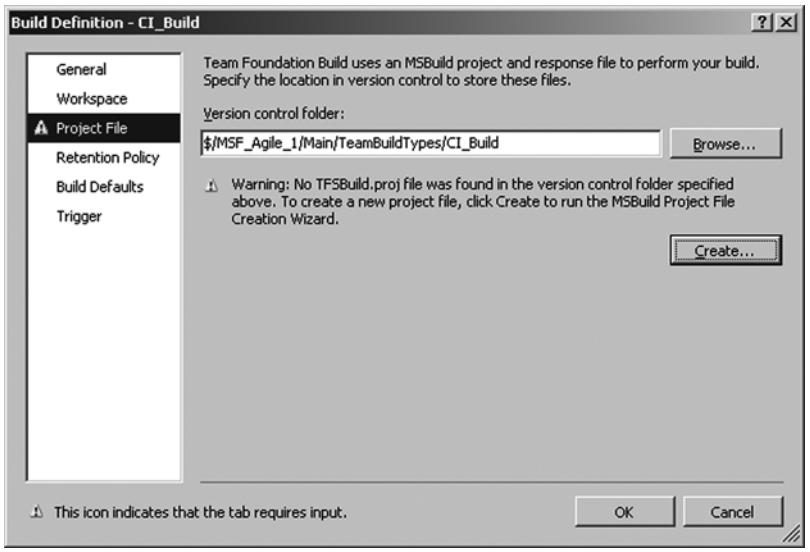

Figure 2.8 Use the Project File tab to select an existing MSBuild project file or to create a new one.

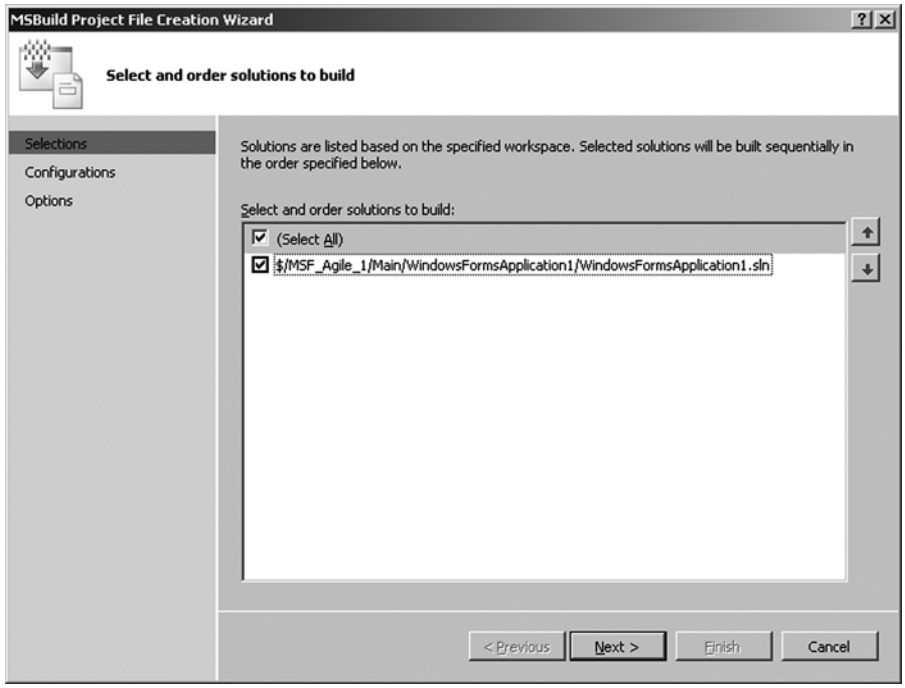

Figure 2.9 Use the MSBuild Project File Creation Wizard to generate an MSBuild project file based on your selections.

THE RETENTION POLICY TAB

The Retention Policy tab (see figure 2.10) helps you manage the binaries in the drop location. For each build outcome, you can specify whether to keep old build binaries,

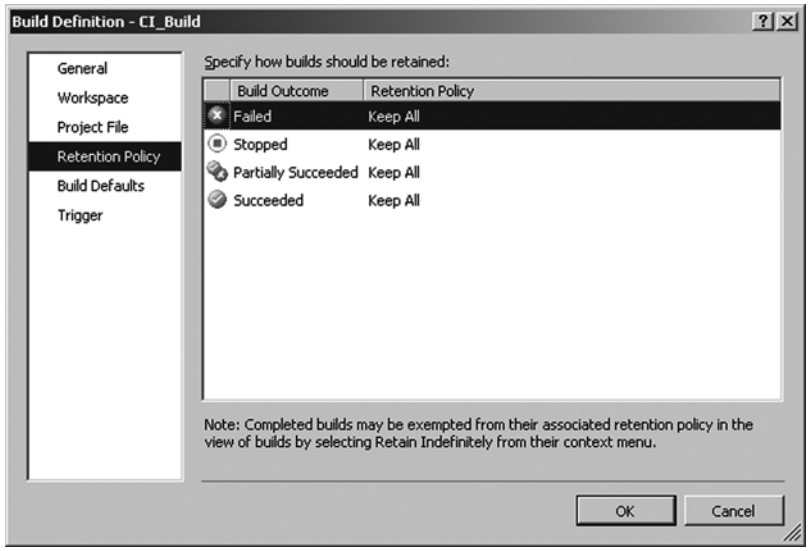

Figure 2.10 Use the Retention Policy tab to indicate automated drop management choices.

and how many versions should be kept. Although you can't specify custom retention policies in TFS 2008, the out-of-the-box retention policies help automate routine housekeeping tasks related to optimization of build storage space. Retention policies are especially useful in the context of continuous integration builds, where a great number of build binaries may be generated in an active codeline, to avoid proliferation of unneeded builds in the filesystem.

If you don't want a particular build to be governed by the specified retention policies, you can make exceptions on a case-by-case basis. Select the build using Build Explorer, right-click a build to display the context menu, and select Retain Indefinitely (see figure 2.11). Later on, if you want the build to be maintained as per the retention policies, you can turn the option off again.

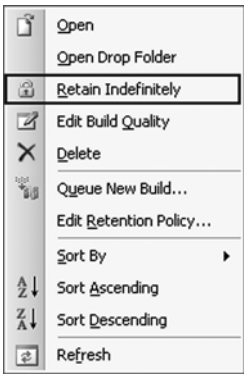

Figure 2.11 Choose Retain Indefinitely to exempt a build from the retention policies.

THE BUILD DEFAULTS TAB

The Build Defaults tab (see figure 2.12) lets you specify the default build agent and the drop location. To create a new build agent, click New. This displays the Build Agent Properties dialog box (see figure 2.3, displayed previously). A build agent represents a single build machine and accepts build requests from a single TFS server. Many of the properties related to a build agent can be customized by modifying the TFSBuildService.exe.config file located in the %ProgramFiles%\Microsoft Visual Studio 9.0\Common7\IDE\PrivateAssemblies folder.

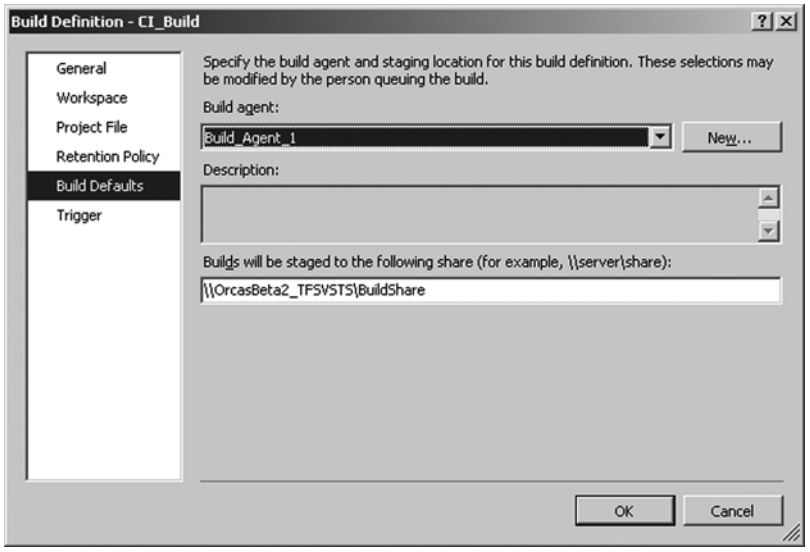

Figure 2.12 Use the Build Defaults tab to specify the default build agent and the drop location.

THE TRIGGER TAB

The Trigger tab (see figure 2.13) allows you to select the timing of builds. TFS 2008 supports continuous integration (CI builds) by launching automatic builds whenever a check-in takes place in TFVC (TFS 2005 doesn't have built-in support for CI builds). You can generate a build upon every check-in or accumulate check-ins until the previous build is completed (this is called a *rolling build*). When selecting rolling builds, you can also choose to initiate builds at certain intervals, limiting the number of builds that are created each day.

If you select Accumulate Check-ins Until the Prior Build Finishes and specify a delay time, Team Build will only launch builds at specified intervals (assuming there are outstanding check-ins). However, the first check-in will trigger a build immediately; the delay will kick in from the second check-in onward.

You can also generate builds based on a schedule (select the Build Every Week on the Following Days option and make appropriate selections). The scheduled build feature enables you to create daily or nightly builds.

When creating builds, Team Build is smart enough to consider only those check-ins that take place in one of the server paths specified in the build workspace. If you don't want a check-in (that's in a server path specified in the build workspace) to trigger automatic builds, specify `***NO_CI***` in the Comment field when checking in the changeset. A changeset containing `***NO_CI***` in the Comment field doesn't trigger a CI build. You can obtain this special string from the `NoCICheckInComment` property of the `IBuildServer` interface (discussed later in section 2.1.11).

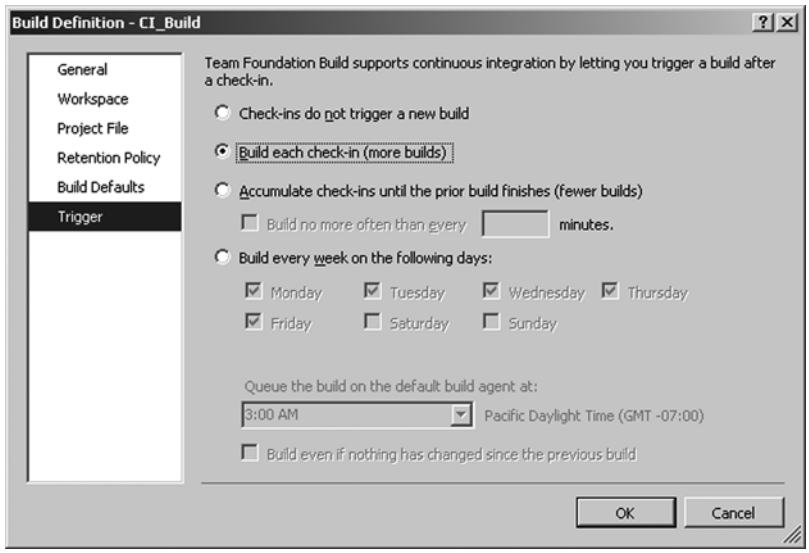

Figure 2.13 Use the Trigger tab to generate CI, rolling, or scheduled builds.

2.1.5 Launching and queuing builds

Team Build queues build requests if another build belonging to the same team project is in progress. In such an event, the build requests are put in a queue and executed one at a time. This is a major difference between the build engines of TFS 2005 and TFS 2008. TFS 2005 displayed an error message if you tried to concurrently execute a build type belonging to the same team project in the same build machine.

Keep in mind that similar to TFS 2005, TFS 2008 can still execute only a single build definition per team project using a given build agent. As discussed, the difference is that if you try to concurrently execute another build definition that belongs to the same team project (using the same build agent), TFS 2008 will queue the build request instead of producing an error message. Of course, using multiple build agents, you can always run build definitions from the same team project in parallel. (Also, as discussed, typically a single build agent runs on a single build machine, although it's possible to configure multiple build agents to run on a single build machine.)

Support for queued builds in TFS 2008 allows you to send build requests to build agents without worrying about the current status of the build machine. Build queues are also helpful for CI builds, since Team Build can queue up multiple build requests when a build is in progress (instead of rejecting pending requests).

To launch a new build, right-click a build definition and click Queue New Build from the context menu. In the Queue Build dialog box (see figure 2.14), you can specify a build agent for executing the build. You can also indicate the priority of the build request. Based upon a build request's priority relative to other build requests in the queue, the build agent displays the position of the build request in the queue. The queue position gives you an idea of when the build might get executed. You may not be able to predict the exact time, since you may not know how long the preceding build requests in the queue might take; furthermore, the priority of the preceding build requests could change or they could be postponed or cancelled.

In the Queue Build dialog box, you can also specify the drop location and the command-line parameters to pass to MSBuild. The ability to pass parameters to MSBuild at runtime is an enhancement over TFS 2005, where you had to edit the TFSBuild.rsp file to send parameters to MSBuild (you can still do this in TFS 2008). Modify the TFSBuild.rsp file for durable parameters and use the Queue Build dialog box for ad hoc choices.

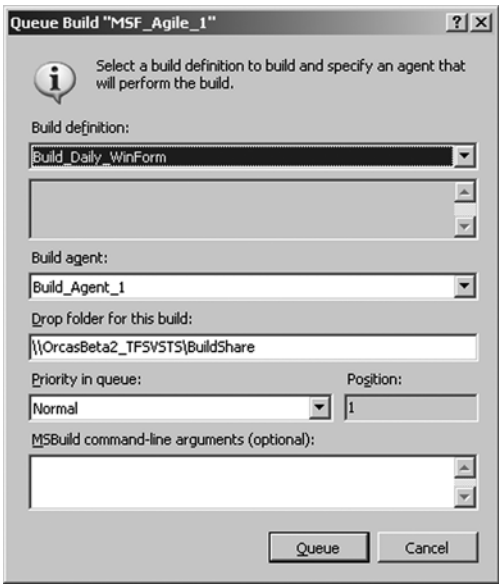

Figure 2.14 The Queue Build dialog box provides various runtime options for build execution.

2.1.6 Build definitions and security

After you create a build definition, you might want to limit who can execute it and generate builds. If you've created multiple build definitions—for example, for development, SQA, and release builds (possibly based on different source control folders as well as other settings)—you'd probably want to restrict the groups of people who can run each build definition.

Unfortunately, in TFS 2008, you can't apply security on build definitions. Any user who has access to a team project can execute all the build definitions contained in it. This deficiency is expected to be corrected in a future version of TFS.

For now, the best you can do is prevent people from editing the build definitions. To prevent a group of users from editing a build definition, select the folder in TFVC that contains the corresponding TFSBuild.proj file and deny Read permission to the group. If a user from the prohibited group tries to edit the build definition, TFS 2008 will display an error message (see figure 2.15).

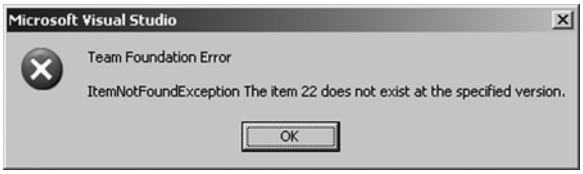

Figure 2.15 TFS 2008 displays an error message if a user attempts to edit a build definition but doesn't have Read access to the folder containing the corresponding TFSBuild.proj file.

2.1.7 Running parallel builds

Running builds in parallel can speed up the build process for large projects. MSBuild 3.5 introduces support for launching parallel builds on multi-processor or multi-core machines. Team Build uses the MSBuild engine to generate builds. To generate parallel builds using MSBuild, you need to do two things:

- Set the `BuildInParallel` task parameter to `true`.
- Specify a value (greater than 1) for the `/maxcpucount` (or `/m` for short) command-line parameter.

You should set up project-to-project references in your solutions so that MSBuild can resolve the dependencies and create parallel builds in the correct order. For more information regarding how MSBuild creates parallel builds, visit http://msdn2.microsoft.com/en-us/library/bb383805(VS.90).aspx and http://msdn2.microsoft.com/en-us/library/bb651793(VS.90).aspx.

Since Team Build invokes MSBuild behind the scenes to create builds, Team Build needs to pass the correct parameter values to MSBuild in order to launch parallel builds. This is achieved as follows.

Team Build attempts to build each configuration and each solution in parallel, if possible. In the Microsoft.TeamFoundation.Build.targets file (located in the %ProgramFiles%\MSBuild\Microsoft\VisualStudio\TeamBuild directory), the `MSBuild` task is called with the `BuildInParallel` task parameter set to the values of the `BuildConfigurationsInParallel` or `BuildSolutionsInParallel` variables (see listings 2.1 and 2.2). The

values of `BuildConfigurationsInParallel` and `BuildSolutionsInParallel` are initialized to `true` in the Microsoft.TeamFoundation. Build.targets file.

Team Build offers a key named `MaxProcesses` in the TfsBuildService.exe.config file, where you can specify the number of worker processes that MSBuild should generate (the default is 1). The value you specify should depend upon the number of processors or cores in your machine. The value specified in the `MaxProcesses` key is passed to MSBuild at runtime.

Listing 2.1 The `CoreCompile` target invokes the MSBuild task

```
<!-- Main compile target -->
  <Target Name="CoreCompile"
        DependsOnTargets="$(_CoreCompileDependsOn)"
        Outputs="@(CompilationOutputs)">

    <MakeDir Directories="$(BinariesRoot)"
          Condition="!Exists('$(BinariesRoot)')" />

    <MSBuild BuildInParallel="$(BuildConfigurationsInParallel)"
          Projects="@(ConfigurationList)"
          Targets="CompileConfiguration"
          StopOnFirstFailure="$(StopOnFirstFailure)">
      <Output TaskParameter="TargetOutputs" ItemName="CompilationOutputs"
      />
    </MSBuild>

  </Target>
```

Listing 2.2 The `CoreCompileConfiguration` target invokes the MSBuild task

```
  <!-- Compile an individual configuration -->
  <Target Name="CoreCompileConfiguration"
   DependsOnTargets="$(CoreCompileConfigurationDependsOn)">

    <MSBuild BuildInParallel="$(BuildSolutionsInParallel)"
          Projects="@(SolutionList)"
          Targets="CompileSolution"
          StopOnFirstFailure="$(StopOnFirstFailure)">
      <Output TaskParameter="TargetOutputs" ItemName="CompilationOutputs"
      />
    </MSBuild>

    <!-- Add Platform and Configuration metadata to CompilationOutputs. -->
    <ItemGroup>
      <CompilationOutputs>
        <Platform>$(Platform)</Platform>
        <Configuration>$(Configuration)</Configuration>
      </CompilationOutputs>
    </ItemGroup>

  </Target>
```

2.1.8 Using the Build Explorer

The Build Explorer acts as the build management dashboard, providing access to queued as well as completed builds. To view the Build Explorer, right-click the Builds

node in Team Explorer and select View Builds from the context menu (or double-click the All Build Definitions element located under the Builds node). There are two tabs in Build Explorer—Queued and Completed. The Queued tab (see figure 2.16) shows build requests that are currently in the execution queue. The Completed tab (see figure 2.17) shows builds that have been generated already. Note that the columns available in these two tabs are different. The options available in the build toolbar also change depending on which tab is currently selected.

The Queued tab in Build Explorer displays builds that are in progress as well as ones waiting in the queue. Using the build toolbar or the context menu associated with a build entry, you can stop in-progress builds and postpone or cancel queued builds. You can also change the priority of a queued build.

Note the Requested By column is available in both tabs in Build Explorer. This is a new piece of metadata associated with a build that wasn't captured in TFS 2005. Knowing who initiated a build is useful in many organizations, since it helps the audit process. For CI builds, the Requested By column displays the name of the developer whose check-in triggered the build. If there are multiple accumulated check-ins, the

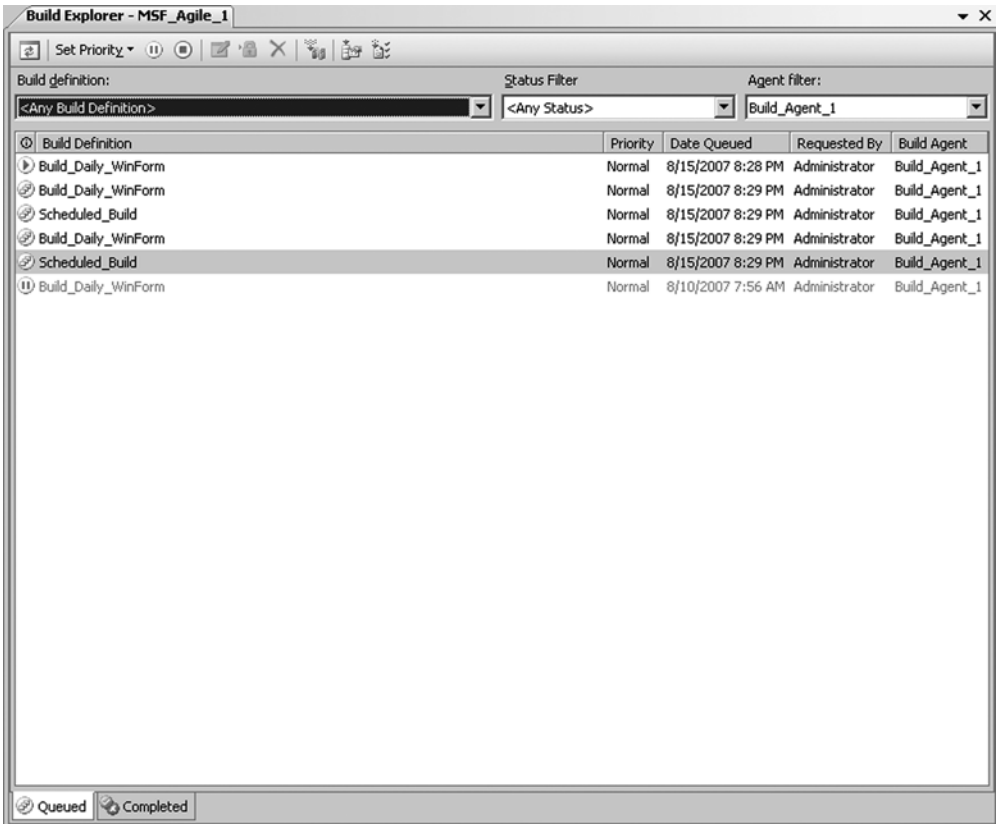

Figure 2.16 The Queued tab in Build Explorer displays builds that are waiting to be executed.

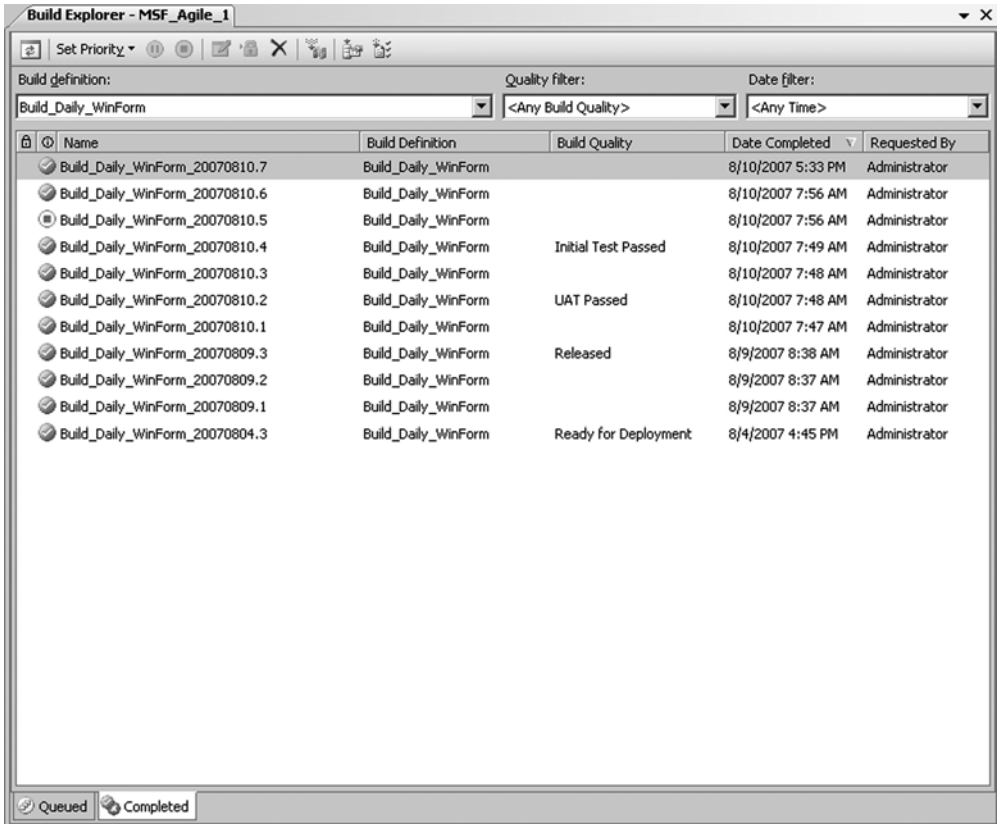

Figure 2.17 The Completed tab in Build Explorer displays builds that have been generated.

Requested By column shows the name of the user associated with the last (highest numbered) check-in. For scheduled builds, Team Build displays "Team Build System Account" in the Requested By column. For builds launched manually, the name of the TFS user who launched the build is displayed in the Requested By column.

The Completed tab in Build Explorer is similar to the one available in TFS 2005, but contains additional options. You can filter builds using build definition, build quality, and build date. When you right-click a build, the context menu offers options to delete the build, indicate the build quality, retain the build indefinitely, navigate to the drop location, and so on. As in TFS 2005, you can double-click an entry to view the detailed build report.

2.1.9 *Managing build qualities*

You can indicate the build quality using the Build Quality drop-down in Build Explorer. Note that TFS 2008 offers more Build Quality settings (see figure 2.18) than were available in TFS 2005. You can also customize the available build quality settings. On the Build menu,

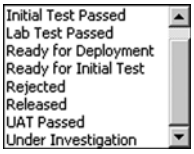

Figure 2.18 Use the Build Quality drop-down to indicate the testing and release status of a build.

click Manage Build Qualities to launch the Edit Build Qualities dialog box (see figure 2.19). Changing the build quality settings in Build Explorer triggers TFS events. You can monitor the build quality change events and take appropriate actions (such as copying files from the drop location to a testing, staging, or production server).

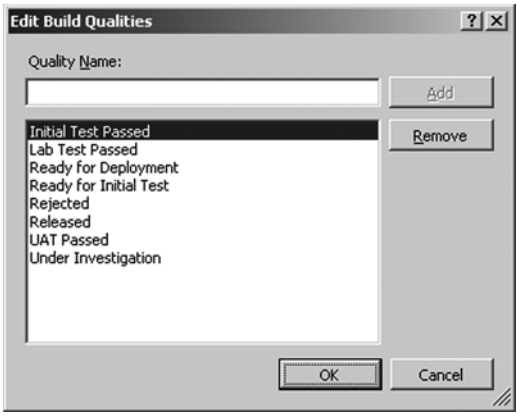

Figure 2.19 You can customize the build quality options to suit your development process.

2.1.10 Generating incremental builds

TFS 2008 introduces two properties (specified in the TFSBuild.proj file) that facilitate downloading only the changed source files as well as generating only the affected binaries. They are

- `IncrementalGet`—Set this property to `true` if you want Team Build to download only the modified source files (the files that have been changed since the last build). By default, Team Build downloads every source file located in the server paths specified in the build workspace.
- `IncrementalBuild`—Set this property to `true` if you want to download only the modified source files and generate only the corresponding binaries. By default, Team Build downloads all source files associated with the workspace and regenerates all binaries. For more information regarding how `IncrementalBuild` works, visit http://msdn2.microsoft.com/en-us/library/aa833876(vs.90).aspx.

Incremental builds are especially useful when creating CI builds for large projects. Downloading the full source code and doing a full build every time could create a potential bottleneck in the build process (depending on project size, check-in frequency, build schedule, hardware setup, connectivity speed, and so forth).

2.1.11 Exploring the Team Build object model

TFS 2008 introduces a new object model for Team Build. Conceptually, the object model acts as the access point to the build system. Behind the scenes, the object model communicates with the Team Build service to perform requested actions and return corresponding results.

The top-level interface in the new Team Build object model (in TFS 2008) is called `Microsoft.TeamFoundation.Build.Client.IBuildServer`. In TFS 2005, the top-level class was called `Microsoft.TeamFoundation.Build.Proxy.BuildStore` and it was a thin wrapper around the build web services. The build object model in TFS 2008 is a lot richer.

`IBuildServer` is implemented by an internal sealed class named `Microsoft.TeamFoundation.Build.Client.BuildServer`, located in the `Microsoft.TeamFoundation.Build.Client` assembly. Since the `BuildServer` class is internal, you can't instantiate it

directly. When you call the `GetService` method of the `Microsoft.TeamFoundation.Client.TeamFoundationServer` class and pass `IBuildServer` as the service type, the method internally creates an instance of the `BuildServer` class and returns the reference to you.

As you can see in figure 2.20, `IBuildServer` exposes methods to create and delete build agents and build definitions, queue builds, stop in-progress builds, and run

Figure 2.20 The `IBuildServer` interface offers methods to interact with the build system.

queries (in order to get information about builds, build agents, build definitions, build qualities, and so on).

Other important interfaces include `IBuildAgent` (represents a build agent), `IBuildDefinition` (represents a build definition), and `IBuildDetail` (represents an individual build). See Figure 2.21 to learn about their properties and methods.

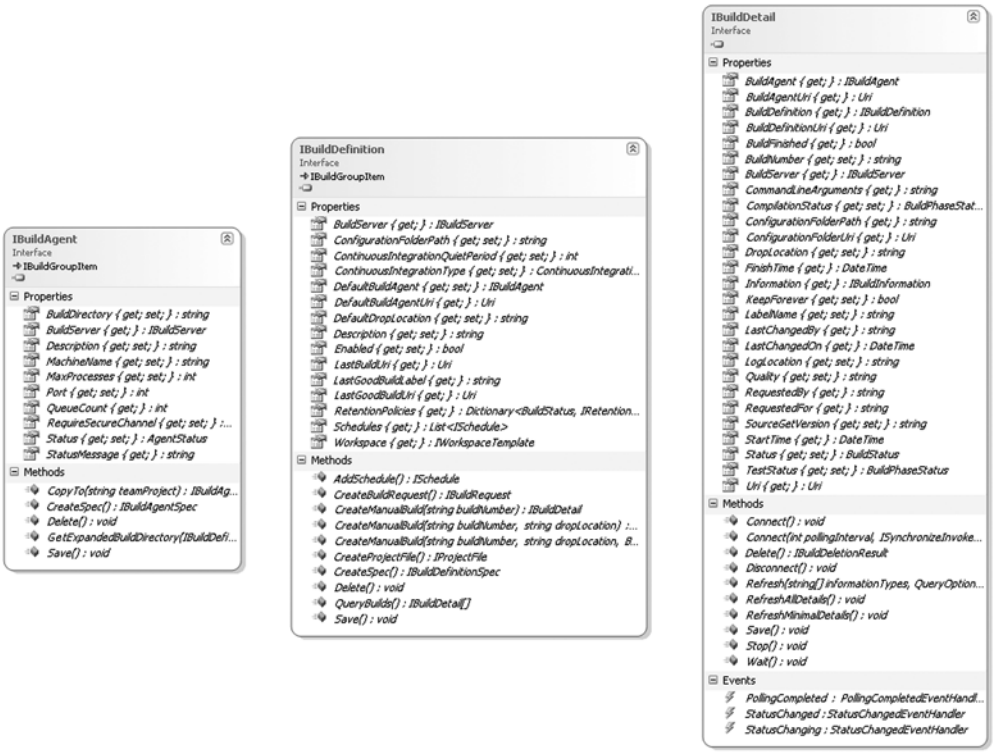

Figure 2.21 `IBuildAgent`, `IBuildDefinition`, and `IBuildDetail` provide information about build machines, build definitions, and specific builds, respectively.

Now that we have an overview of the Team Build object model, let's look at two examples of how this knowledge can be applied to solve real-life problems.

2.1.12 A sample program for queuing builds

In this section, we look at a sample program that finds the build agent that's least busy and queues a build in it (see listing 2.3). The steps are as follow:

1. Get a reference to the build server and get a list of build agents. The build server offers a method named `QueryBuildAgents` for retrieving the collection of build agents.
2. Iterate the collection of build agents and select the build agent that has the lowest number of build requests queued. The `IBuildAgent` interface contains a property named `QueueCount` to report the number of requests currently queued in the build agent.

3 Retrieve the build definition from the build server (using the `GetBuildDefinition` method) and create a build request from it. The `IBuildDefinition` interface contains a method named `CreateBuildRequest` for creating build requests. After creating a build request, we can customize its properties. In particular, we set the `BuildAgent` property to the build agent selected in the previous step.

4 Finally, queue the build in the selected build agent, using the `QueueBuild` method of the build server.

Listing 2.3 Queuing a build in the build agent with the lightest load

```
using System;
using System.Diagnostics;
using System.Collections.Generic;

using Microsoft.TeamFoundation.Client;
using Microsoft.TeamFoundation.Build.Client;

namespace OrcasBuildTests
{
   class TeamBuildHelper
   {
      string _tfsUrl = TFS_NAME;
      string _teamProject = TEAM_PROJECT_NAME;
      string _buildDefinition = BUILD_DEFINITION_NAME;

      public void QueueBuildDefinition()
      {
         try
         {
            TeamFoundationServer tfs =
             new TeamFoundationServer(_tfsUrl);

            IBuildServer buildServer =
             (IBuildServer)tfs.GetService(typeof(IBuildServer));

             IBuildAgent[] buildAgents =
              buildServer.QueryBuildAgents(_teamProject);

            //find the build agent with the lowest queue count
            IBuildAgent selectedBuildAgent = null;
            int lowestQueueCount = 0;

            foreach (IBuildAgent buildAgent in buildAgents)
            {
               if (selectedBuildAgent == null)
               {
                  selectedBuildAgent = buildAgent;
                  lowestQueueCount = buildAgent.QueueCount;
                  continue;
               }

               if (buildAgent.QueueCount < lowestQueueCount)
               {
                  selectedBuildAgent = buildAgent;
                  lowestQueueCount = buildAgent.QueueCount;
               }

            }
```

Change these values for your project

Get reference to TFS

Get reference to build server

Get list of agents that belong to project

Initialize currently selected agent if empty

Select agent if queue count is lower

```
            //now queue the build definition to the
            //selected build agent

            //retrieve the build definition and create a new build
            //request using the selected build agent
            IBuildDefinition buildDefinition =
               buildServer.GetBuildDefinition(
               _teamProject, _buildDefinition);

            IBuildRequest buildRequest =
               buildDefinition.CreateBuildRequest();

            buildRequest.BuildAgent = selectedBuildAgent;

            buildServer.QueueBuild(buildRequest);
         }
         catch (Exception ex)
         {
            Debug.WriteLine(ex);
            throw;
         }
      }
   }
}
```

Queue build request in selected agent

2.1.13 Determining which build contains a particular fix

Continuing our discussion of the new Team Build object model, let's look at another real-life problem. A common request from SQA groups involves knowing which build contains a particular work item. In order to retrieve this information, you could launch the Build Explorer, go through each build, and view the Associated Work Items section of each build report (see figure 2.22) to identify the build that contains

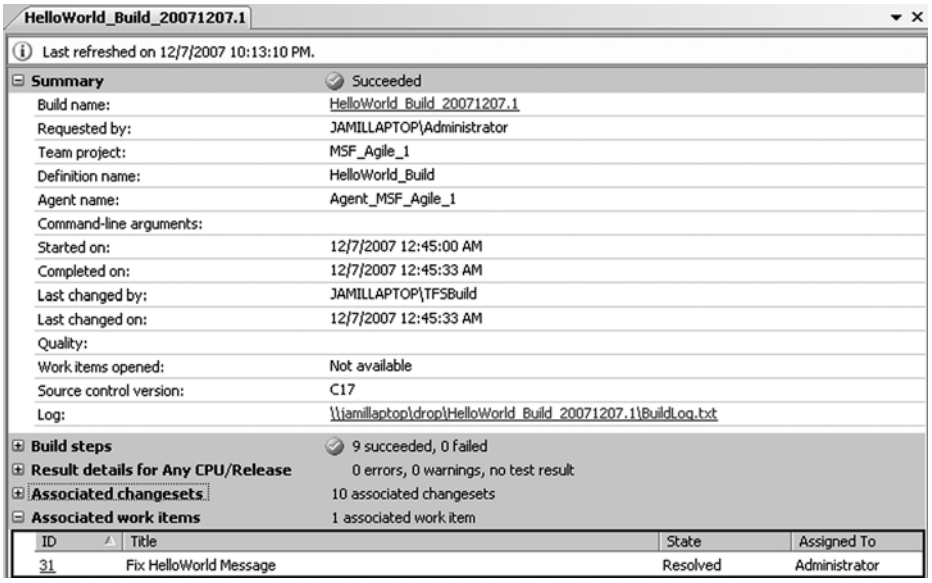

Figure 2.22 The build report includes information regarding associated work items.

the work item. Although this manual method would work, it would be time consuming. An easier approach would be to determine this information using a custom program, leveraging the object model. Listing 2.4 shows the code needed to obtain the corresponding build number, given a work item number.

Listing 2.4 Finding which build contains a specific work item

```
public void FindBuild()
{
   string found_in_build = null;

   try
   {
     TeamFoundationServer tfs =
      new TeamFoundationServer(_tfsUrl);

     IBuildServer buildServer =
      (IBuildServer)tfs.GetService(typeof(IBuildServer));

     IBuildDetail[] buildDetails =
      buildServer.QueryBuilds(_teamProject);

     foreach (IBuildDetail buildDetail in buildDetails)
     {
      List<IWorkItemSummary> workitems =
      InformationNodeConverters.GetAssociatedWorkItems(buildDetail);

     foreach (IWorkItemSummary workitem in workitems)
     {
      if (_workitem_number == workitem.WorkItemId)
      {
       found_in_build = buildDetail.BuildNumber;
       break;
      }
     }
   }

   if (found_in_build != null)
     Debug.WriteLine("Work Item Id:" + _workitem_number +
                 " found in build:" + found_in_build);
   else
     Debug.WriteLine("Work Item Id:" + _workitem_number +
                 " not found");
   }
   catch (Exception ex)
   {
     Debug.WriteLine(ex);
     throw;
   }
```

Get reference to TFS

Get reference to build server

In listing 2.4, we invoke the `QueryBuilds` method of the `IBuildServer` interface to obtain information related to all builds for a given team project. The information is returned as a collection of `IBuildDetail` objects. We iterate over the collection and extract the information exposed by the `IBuildDetail` interface. In particular, we're interested in the `Information` property of the `IBuildDetail` interface.

The `Information` property is of type `IBuildInformation` and contains a `Nodes` collection (see figure 2.23). Each member of the `Nodes` collection is of type `IBuildInformationNode`. The members of the `Nodes` collection represent build steps, associated changesets, associated work items, and so forth. Instead of parsing the information contained in `IBuildInformationNode`, we take a simpler approach and use the `InformationNodeConverters` class.

The `InformationNodeConverters` class is a utility class that offers various static methods for adding and retrieving build steps, associated changesets, associated work items, and so on (see figure 2.24). We use the `GetAssociatedWorkItems` method of the `InformationNodeConverters` class to obtain the list of work items associated with a particular build. This method returns a collection of `IWorkItemSummary` objects. The `WorkItemId` property belonging to the `IWorkItemSummary` interface contains the

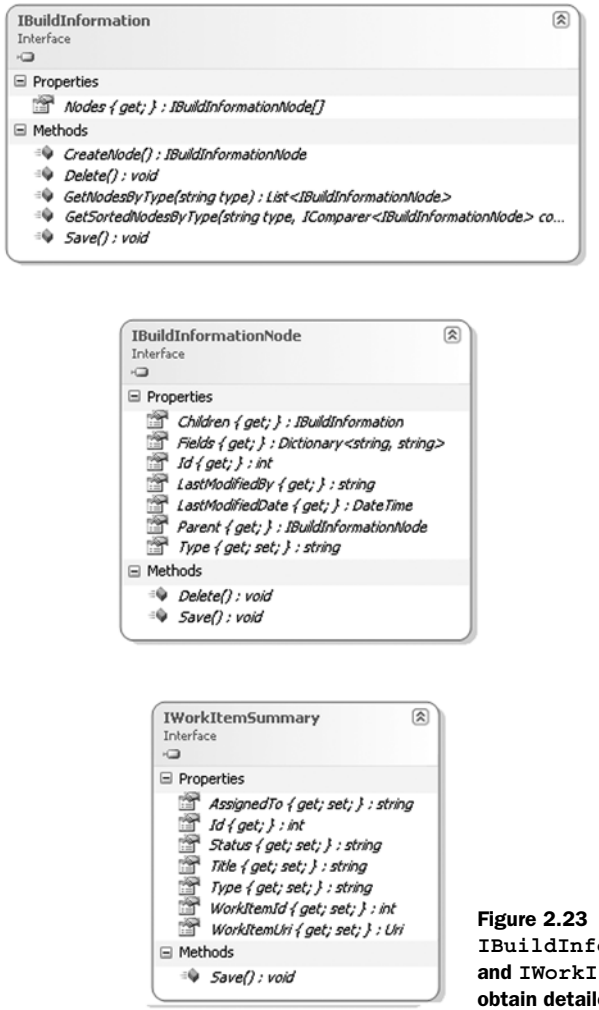

Figure 2.23 `IBuildInformation`, `IBuildInformationNode`, and `IWorkItemSummary` interfaces enable you to obtain detailed information about a specific build.

Figure 2.24 The `InformationNodeConverters` class offers useful methods to configure build execution as well as to retrieve information about completed builds.

actual work item number. We compare this number with the input work item number to identify the build we're searching for.

2.2 Team Foundation version control

Compared to Team Build, the changes in TFVC are incremental. But there are significant performance-related improvements that may not be immediately obvious, unless you're working on large projects. In this section, we discuss the features of TFVC that have undergone improvements in TFS 2008.

2.2.1 Get the latest on check-out

TFS 2005 didn't provide a way to automatically download the latest version when checking out a file. When you checked out a file, TFS 2005 simply made the current version of the file (the version that the server believed to be in your workspace) editable. If there was a later version of the file in the server, it wasn't download to the local workspace.

This behavior was not a bug but a feature. The idea was rooted in the fact that since files are often modified in a group (a feature or a change request could affect multiple files), downloading the latest version of a single file while leaving others unchanged could make the software in the local workspace inconsistent. Hence, the preferred way of downloading changes from the server was to deliberately fetch the appropriate version of all related files. This can be done (both in TFS 2005 and TFS 2008) at the file or folder level using the context menu. Right-click a file or folder and select Get Latest Version or Get Specific Version from the context menu (see figure 2.25). If you perform this

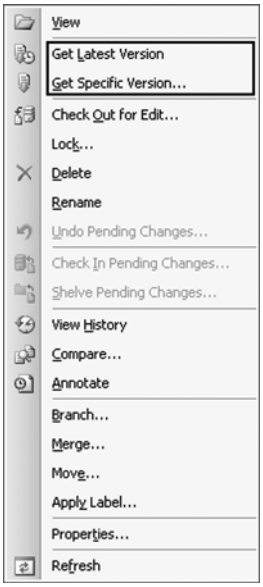

Figure 2.25 You can download a specific version of a file from TFVC using the context menu associated with files and folders.

operation at the folder level, the changes in that folder as well as in mapped subfolders will be downloaded recursively.

The default check-out behavior remains unchanged in TFS 2008. But TFS 2008 provides an option (turned off by default) to automatically download the latest version when checking out a file or folder. Many users have voiced this need when attempting to ensure that developers are always working on the latest version of a file; people often forget to consciously download the latest version. In the Tools menu, click Options. In the Options dialog box, expand the Source Control node in the tree and click the Visual Studio Team Foundation Server element; make your selection using the Get Latest Version of Item on Check Out check box (see figure 2.26).

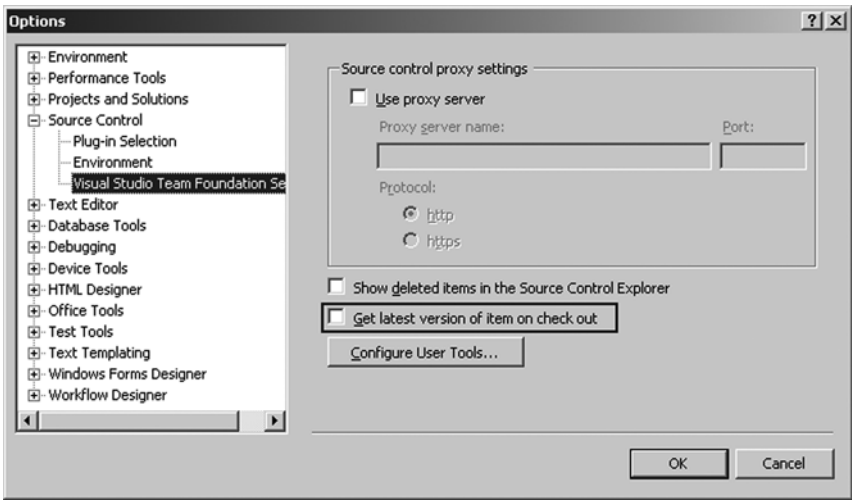

Figure 2.26 The Options dialog box enables you to configure whether checking out a file will automatically download the latest version.

If this check box is selected, TFS 2008 will automatically download the latest version of a file or folder when you click Check Out for Edit from the context menu. If the check box is cleared, TFS 2008 will simply make the current version of the file or folder editable (by pending an edit). However, when you double click the file to open it (or select View from the right-click context menu), TFS 2008 will inform you that there's a mismatch between the server and local versions and ask which version you want to open (see figure 2.27). Furthermore, you can find out which files in your workspace are outdated by looking at the Latest column in the Source Control Explorer (see figure 2.28).

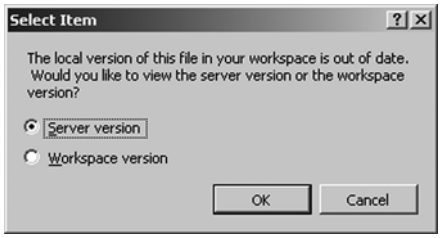

Figure 2.27 TFVC displays the Select Item dialog box when there's a mismatch between the server and workspace versions of a file.

In the Select Item dialog box (see figure 2.27), you have the option of working with either the server version or the local version of

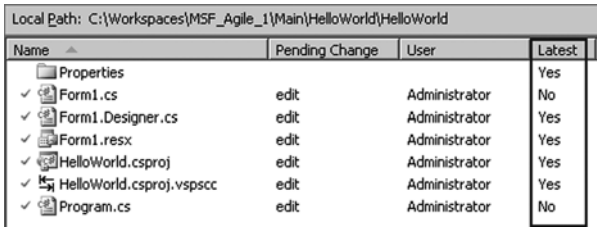

Figure 2.28 The Latest column in Source Control Explorer displays the status of files in the workspace relative to the server.

a file. Recall that this dialog box is displayed for any file that has a mismatch between its workspace and server versions.

If you choose to work with the workspace version, you'll encounter a conflict when checking in the changes (since the server version is different), and you'll be asked to resolve the version conflicts (see figure 2.29). When version conflicts are detected during check-in, TFS 2008 halts the check-in process so that you can resolve the conflicts and verify the changes (otherwise, untested code will be uploaded to the server). After you're satisfied that the changes are valid, you can check in the changes.

On the other hand, if you choose to work with the server version, you'll encounter an error when trying to save the file (since the workspace version is different and is still read-only); you can choose to overwrite the local file or save the file using a different filename. Regardless of whether you work with the server version or the workspace version, ultimately you'll need to synchronize the two versions when you try to check in the changes.

If you've checked out a file (there is a pending edit on it), and someone else changes the file on the server (while it's still checked out to you), TFS 2008 will display a merge window if you try to get the latest version. The merge window provides a mechanism for you to selectively (or automatically) incorporate the server-side changes with the version in your workspace.

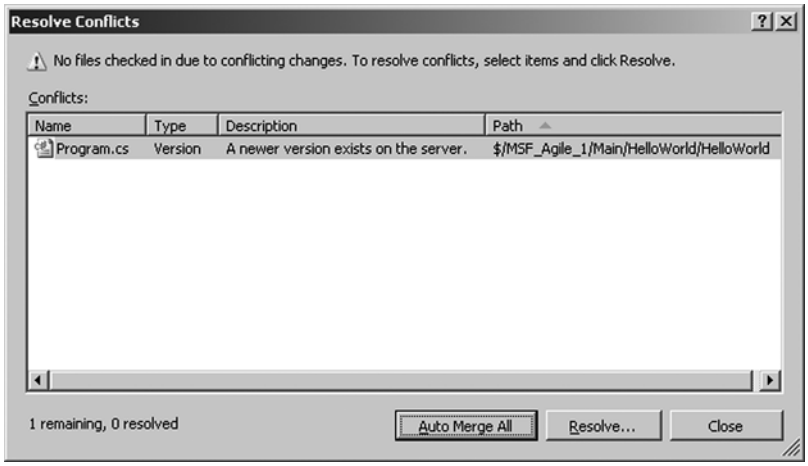

Figure 2.29 The Resolve Conflicts dialog box enables you to resolve conflicts between the workspace and server versions.

If you haven't checked out the file (there is no pending edit on it), and someone else modifies the file on the server, you won't get a merge window when you try to get the latest version; TFS 2008 will simply overwrite the workspace version with the server version. Consequently, be careful when trying this if you don't have pending edits—the operation will unconditionally overwrite your workspace files with files downloaded from the server. You can get into this situation if you go offline (see section 2.2.2), make changes, and then set the read-only attribute. If you wish to merge server-side changes with the version in your workspace, check out the file (pend an edit) and then get the latest version.

2.2.2 Working offline

TFS 2005 didn't provide an option in the interface to work offline. If you needed to work offline in TFS 2005, you had to remove the read-only attributes of the source files, make necessary changes, and then use the `tfpt` power tool (with the `online` option) to pend the changes (once the server was back online). In TFS 2008, although the general approach remains unchanged from TFS 2005, offline support is integrated into the Visual Studio IDE and you don't need to use a command-line tool.

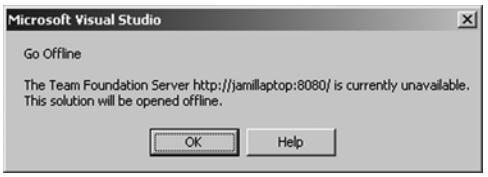

Figure 2.30 Visual Studio 2008 can switch to offline mode if TFVC is unavailable.

Figure 2.31 Click the Go Online icon to synchronize local changes with TFVC.

When opening a project or solution, the IDE checks whether TFS is available and switches to offline mode if it isn't (figure 2.30 shows the message box that's displayed). When in offline mode, the Solution Explorer window also displays an extra icon (named Go Online) in the toolbar (see figure 2.31). When connectivity to TFS is restored, click the Go Online icon to pend appropriate changes to the files in your workspace. You can also select the Go Online option from the right-click context menu associated with the solution or file in Solution Explorer. TFS 2008 supports edit, add, and delete operations in offline mode, but not rename operations.

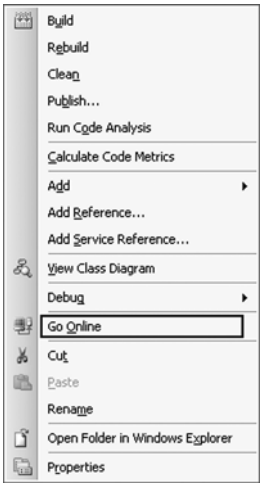

Figure 2.32 The Go Online option is also available from the context menu associated with a solution or file in Solution Explorer.

In offline mode, you can't check out a file from the server, since the server isn't available. You can only work on the files in your local workspace. Since the files in your workspace are read-only (unless you've performed a check-out), when you try to save your changes, you'll encounter a warning screen saying that the file can't be saved (see figure 2.30). Click Overwrite to remove the read-only attribute and save your changes.

When you click Go Online (after server connectivity is restored), Visual Studio will scan your local workspace and display a dialog box indicating appropriate changes (see

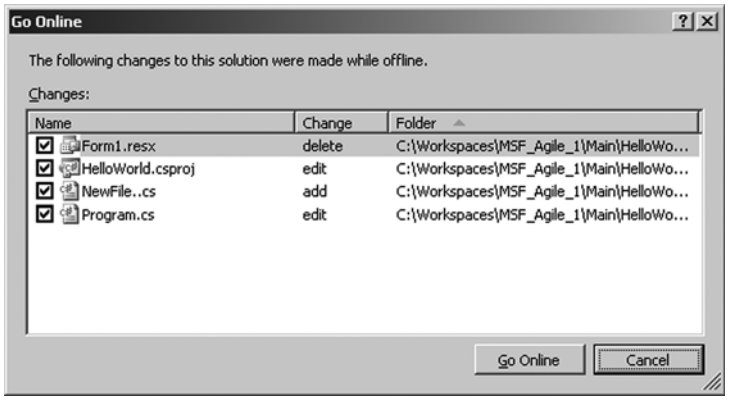

Figure 2.33 When synchronizing local changes with the server, TFVC marks each locally modified file with the appropriate type of change.

figure 2.33). Keep in mind that the logic responsible for determining edit and add operations checks the file attributes in the workspace. For edits and adds, the read-only attribute on the impacted files shouldn't be set. If you set the read-only attribute to `true`, the file won't be displayed in the Go Online dialog box (and won't have edits or adds pended for check-in).

2.2.3 Folder comparison

In TFS 2005, you couldn't compare folders (server or local) from the Visual Studio IDE. This commonly requested feature has been implemented in TFS 2008. Right-click a folder in the Source Control Explorer and select Compare from the context menu. In the Compare dialog box, specify the source and target folders (see figure 2.34).

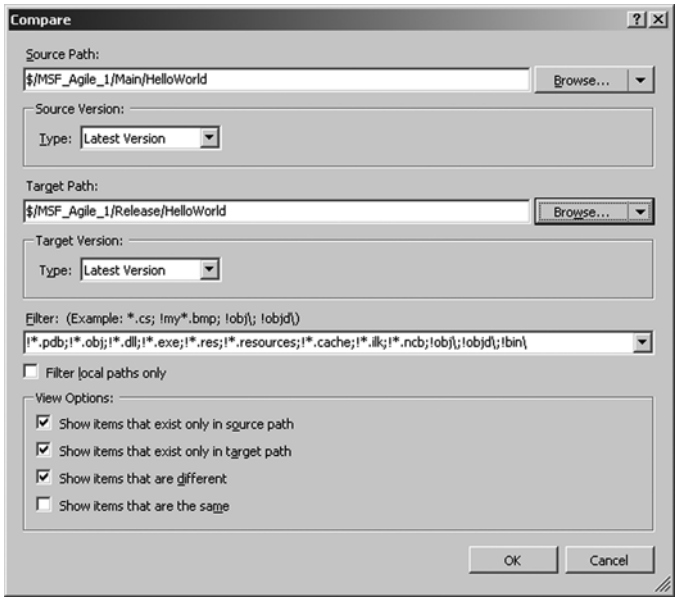

Figure 2.34 The Compare dialog box enables you to compare server or workspace folders and specify various options.

Use the Browse buttons to navigate to the chosen folders—you can specify server or local folders. You can also specify the source and target version types, based on labels, changesets, dates, latest versions, and so on.

Furthermore, you can filter the files that appear in the output list by typing excluded file names, file extensions, or folder names in the Filter box. For example, if you want to exclude all .config files, type `!*.config` in the Filter box. If you want to exclude a folder named "settings", type `!settings\` in the Filter box. Separate the entries using a semicolon (`;`). The output of the Compare operation is displayed in figure 2.35.

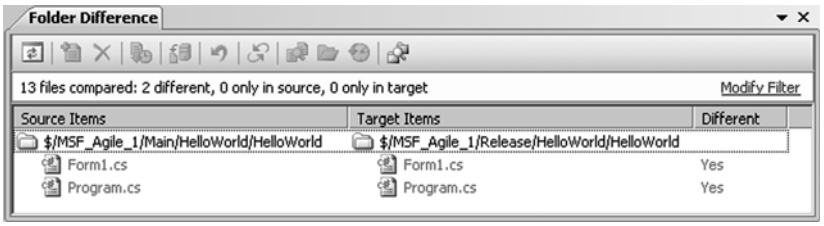

Figure 2.35 The Folder Difference window displays the files which differ between the specified folders.

2.2.4 Annotate

In TFS 2008, the annotate feature of the `tfpt` power tool has been integrated into the user interface. To invoke the annotate feature, right-click a file and select Annotate from the context menu.

The annotate functionality enables you to see the changes made to a file by various users, along with corresponding changeset information (see figure 2.36). You can view additional information about a changeset by clicking the changeset number. The annotation information serves as an audit trail regarding the changes to a file and enables you to see who did what, and when. In case of pending edits, the changes that have occurred in the workspace but haven't yet been checked in are marked Local.

Figure 2.36 The annotated file displays granular audit information regarding its change history.

You can gather additional information about a change by hovering the mouse pointer over an annotation entry. You can also right-click an annotation entry and select the options available in the context menu (see figure 2.37; the context menu options are different for entries marked Local). For example, click Annotate This Version to open the version indicated by the changeset number and display annotation information for that version. Select Compare with Previous Version to compare the current version of the file with the version indicated by the changeset number.

Figure 2.37 The context menu associated with annotation entries contains options to obtain additional information about the change.

2.3 Summary

As we saw in this chapter, TFS 2008 contains a number of significant improvements that make the product more stable, efficient, and usable. We looked at new features in Team Build such as support for various kinds of builds, a new object model, better security, and so on. We also reviewed new features of TFVC, such as offline support, folder comparison, better annotation support, and more.

The next version of TFS (codenamed *Rosario*) promises even more significant enhancements. Upcoming features in Rosario include support for managing multiple TFS instances and multiple projects, creating hierarchical work items, improved test management, relating tests and test results back to requirements, better traceability and ability to conduct impact analysis, and so on.

In the next chapter, we look at another major component of VSTS 2008 that's now available as an integrated product—Visual Studio Team System 2008 Database Edition. VSTS 2008 Database Edition brings database professionals into the mainstream development process by providing support for schema extraction, versioning, comparison, and deployment.

TEAM FOUNDATION SERVER 2008 IN ACTION

Jamil Azher

Free ebook
SEE INSERT

In a project with dozens of developers, hundreds of builds, and thousands of opportunities for mistakes, managing the process is as important as writing the code. Team Foundation Server 2008 is Microsoft's application lifecycle management (ALM) tool. Using it, you can effectively track work items, create test cases, manage source files, and generate builds. Ramp-up time is minimal because it's fully integrated into Visual Studio.

Team Foundation Server 2008 in Action shows developers and managers how to collaborate on complex software projects using TFS 2008. You'll follow real-life scenarios as you learn how to support your preferred development methodology. The techniques are actionable and the solutions are experience-based. You'll master the out-of-the-box functionalities in TFS as well as learn to customize its source code management, database development, build, and reporting capabilities.

What's Inside

- TFS for database professionals
- Customizations and workarounds
- Practical tips and tricks

"A must-have book for anyone who wants to learn TFS."
—Robert Horvick, Microsoft

"Tips and tricks only experience can provide!"
—Eric Swanson
Enterprise Architecture Consultant

"An impressive toolkit for mastering Team Foundation Server."
—David Barkol
Author of *ASP.NET AJAX in Action*

"Clear, distilled, nuggets of advice based on real-world experience."
—Benjamin Day
Benjamin Day Consulting, Inc.

About the Author

Jamil Azher is a technology architect at Microsoft with 18 years of experience in programming, global project management, and entrepreneurial ventures. Jamil earned a Master's degree in IT from Harvard and a BS degree in engineering from Caltech.

For online access to the author, code samples, and a free ebook for owners of this book, go to www.manning.com/TeamFoundationServer2008inAction

ISBN-13: 978-1933988597
ISBN-10: 1933988592
54999
9 781933 988597

MANNING $49.99 / Can $49.99 [INCLUDING eBOOK]