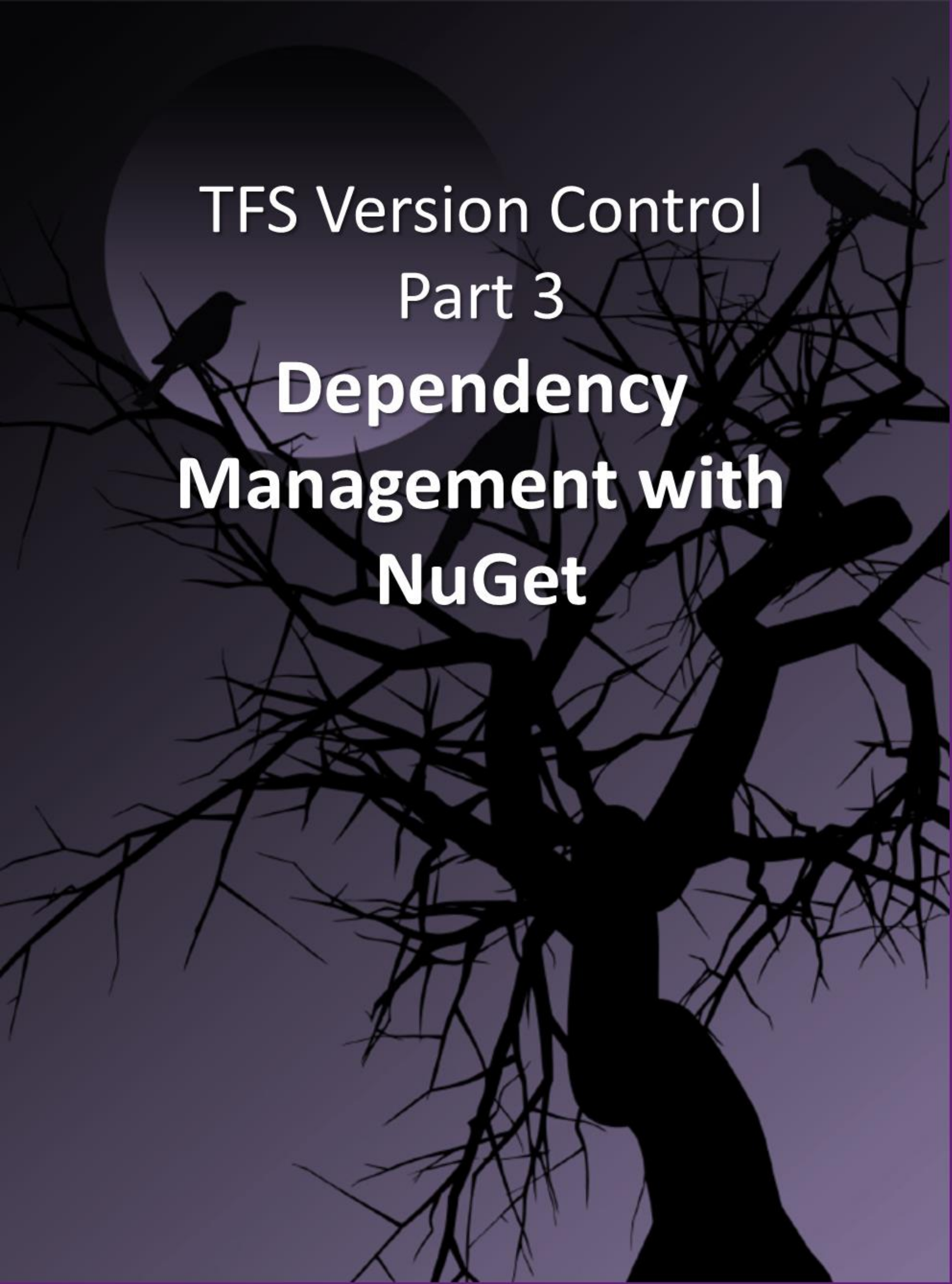
V3

TFS Version Control Part 3 Dependency Management with NuGet

Visual Studio ALM Rangers



Microsoft Visual Studio

Table of Contents

Foreword 4
Introduction 5
Understanding NuGet 6
  Managing Shared Resources with NuGet 6
  Using NuGet for Dependency management 11
  Additional references 15
Hands-on Lab (HOL) – Using NuGet 16
  Exercise 1 – Find and install a NuGet Package 16
  Exercise 2 – Using Package Manager Console 19
  Exercise 3 – NuGet Package Manager 20
In Conclusion 22

Foreword

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

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

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

Happy versioning!

Matthew Mittrik – Program Manager, Cloud Dev Services

Introduction

This guide aims to provide an insight and practical guidance on dependency management and managing shared resources, using NuGet with Visual Studio. It is one of a set of companion guides as outlined in chapter "What's New" in the **TFS Version Control Part 1 – Branching Strategies** guide.

Intended audience

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

Visual Studio ALM Rangers

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

Contributors

Alan Wills, Anna Galaeva, Howard Dierking, Krithika Sambamoorthy, Stawinski Fabio, Taavi Kõosaar and Tommy Sundling

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

Using the sample source code, Errata and support

All source code in and revisions of this guide are available for download from the CodePlex site Version Control (formerly Branching and Merging) Guide [3]. You can contact the team using the CodePlex discussion forum.

Additional ALM Rangers Resources

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

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

[1] http://vsarguidance.codeplex.com/releases/view/88001
[2] http://aka.ms/vsarindex
[3] http://aka.ms/treasure18

Understanding NuGet

Managing Shared Resources with NuGet

Sharing and managing resources between projects can be complicated. *NuGet* helps simplify the whole process. You have probably already used NuGet in Visual Studio to download extensions from the public NuGet gallery. However, you can also set up your own local repository to share resources in your project or your organization. "NuGet is a Visual Studio Extension that makes it easy to install and update libraries and tools in Visual Studio" (http://nuget.org). It comes pre-installed on Visual Studio 2012 and later versions, and you can install it as an extension in earlier versions. There is also a standalone tool, NuGet.exe, which you can use outside Visual Studio.

Here is what NuGet provides:

- A NuGet package can install shared resources, update project references, install Visual Studio extensions, and update project files. It can provide everything you need for doing development in a particular domain.
- Private repository – You can create your own NuGet package repository, either through file sharing or as a web server.
- Public repository – You can also access and contribute to the official NuGet gallery [4]. As well as other public repositories such as myget.org
- Easy overview of which installed packages are outdated
- Easy upgrade process when shared packages are updated, including for dependencies of those packages
- Command-line tool can run in the build process to automatically download the right versions of shared resources

We are basing this guidance on NuGet version 2.7.1. This guidance is complementary to the NuGet tool documentation[5].

The aim of this guidance is to offer an overall approach on how NuGet helps manage shared resources and dependencies when working with projects using Visual Studio and Team Foundation Server.

When should you consider NuGet to share resources?

- If the project uses NuGet for external dependency management (for example libraries such as Entity Framework, jQuery, or ASP.NET MVC), then you can simplify management using the same approach for internal dependencies.
- If several projects use the shared resources then NuGet helps simplify sharing and managing those resources including dependencies and versions.
- If a project's shared resources have complex dependencies between them.
- If there is a separate team (or teams) that build shared resources and many other projects use those resources.

As you see, most scenarios that involve references are a great fit for NuGet. However, if you are working with a single solution that has projects in that solution that depend on each other you can still simply leverage project references and forgo the NuGet package scenarios. You should likely only do this if you do not intend to share those projects beyond that single solutions boundary since NuGet would make it easier to package and share

[4] http://nuget.org/

[5] http://docs.nuget.org/

the project artifacts in a scalable way that can reach many solutions easily. It is not necessary to decide early on, because turning component(s) into sharable NuGet package(s) is straightforward.

Using NuGet for sharing resources

NuGet consists of the following components:

- **NuGet specification file (.nuspec)** – a specification for the creation of a NuGet package. This defines common information about the package such as title, description, version, etc. It also defines dependencies on other NuGet packages.
- **NuGet package (.nupkg)** – a package in NuGet format (Open Packaging Conventions based Zip) that contains the shared resources and libraries. The content of the package is determined by the NuGet specification file[6].
- **NuGet feed** – a local folder, UNC share or a NuGet RSS feed built using the NuGet.Server component.
- **NuGet.exe** – a standalone command line tool.
- **NuGet Package Manager and NuGet Package Manager Console** – a Visual Studio Extension with which you manage NuGet packages for a specific solution
- **NuGet.config** – NuGet configuration per machine or per solution[7]. Among other things, one can define common repositories (e.g. internal NuGet feeds) for the team or the location of packages for a specific solution.

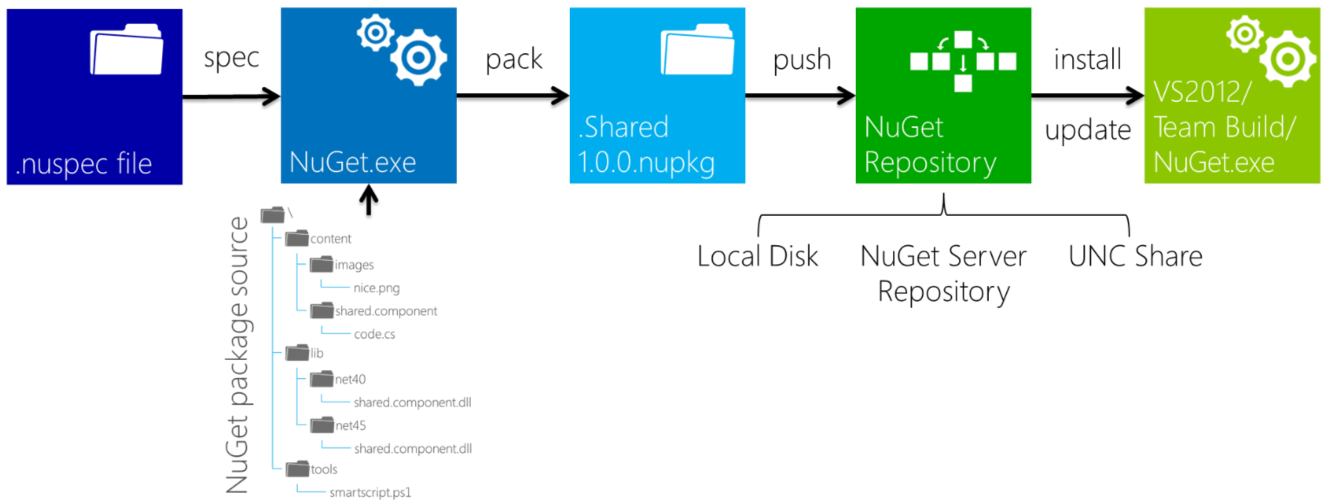


Figure 1 – The general flow with NuGet

Creating a NuGet package

Step	Instructions
1 Create package structure ☐ - Done	First, create a folder structure for packaging. You have to follow the documented NuGet package folder structure [8]. The structure shown below supports three type of resources – assemblies (dll's), tools (PowerShell scripts/programs that can be run from the Package Manager Console), content (different files that will become part of the target project on installation such as images, C# code files, web files, css) and build (MSBuild targets files that are inserted into the project). Note: Keep in mind you can also pass the

[6] http://docs.nuget.org/docs/creating-packages/creating-and-publishing-a-package#From_a_convention_based_working_directory

[7] http://docs.nuget.org/docs/release-notes/nuget-2.1#Hierarchical_Nuget.config

[8] http://docs.nuget.org/docs/creating-packages/creating-and-publishing-a-package#From_a_convention_based_working_directory

Step	Instructions
	proj file directly to a pack command that can help generate the nuspec based on the assembly data automatically via the command line. (See Step 3 below)

NuGet package structure
- \
- **\lib**net40\Shared.Component.dll
- **\lib**net45\Shared.Component.dll
- **\tools**SmartScript.ps1
- **\content**Images\nice.png
- **\content**Shared.Component\Code.cs

The lib folder can contain subfolders for specific framework versions (such as net40, net45). When someone installs your NuGet package, the assembly NuGet will be use depends on the target version of your project. All target versions will use libraries located directly in the lib folder. In NuGet v2.7.1 or later the content and tools folders can also contain framework specific folders.

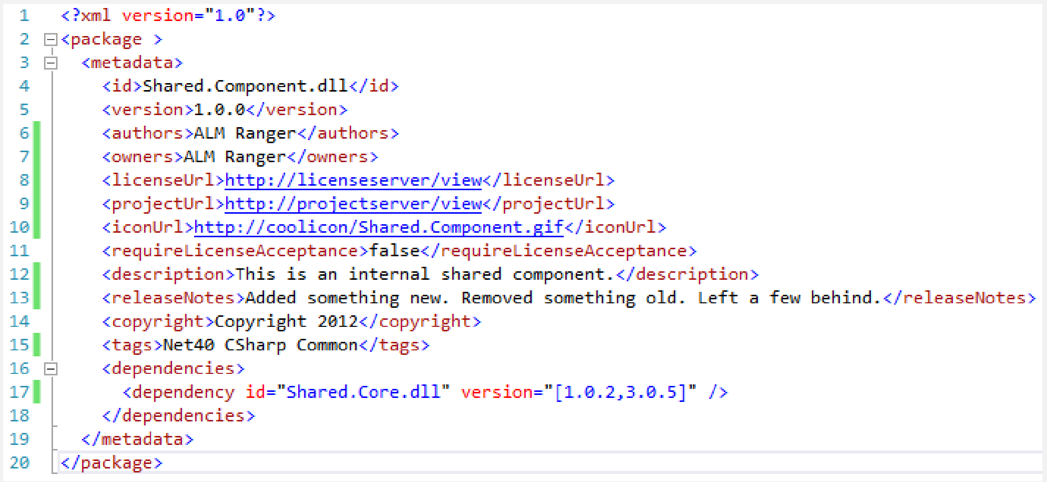
Creating this folder structure can be an automated build step.

Step	Instructions
2 Create the specification ☐ - Done	Create a NuGet package specification [9]. You can use Nuget.exe to do this (and there are other ways). Let us say you want to create a package that only contains Shared.Component.dll. Run:

```
nuget.exe spec "Shared.Component.dll"
```

That generates the xml file Shared.Component.dll.nuspec. It needs further modifications to include correct metadata about the package. You can find detailed .nuspec file information in NuGet Docs[10].

There is also a GUI Package Explorer[11] tool to help ease the creation and modification of NuGet packages. The illustration below shows a sample specification for the “Shared.Component.dll” component.



```xml
<?xml version="1.0"?>
<package >
  <metadata>
    <id>Shared.Component.dll</id>
    <version>1.0.0</version>
    <authors>ALM Ranger</authors>
    <owners>ALM Ranger</owners>
    <licenseUrl>http://licenseserver/view</licenseUrl>
    <projectUrl>http://projectserver/view</projectUrl>
    <iconUrl>http://coolicon/Shared.Component.gif</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>This is an internal shared component.</description>
    <releaseNotes>Added something new. Removed something old. Left a few behind.</releaseNotes>
    <copyright>Copyright 2012</copyright>
    <tags>Net40 CSharp Common</tags>
    <dependencies>
      <dependency id="Shared.Core.dll" version="[1.0.2,3.0.5]" />
    </dependencies>
  </metadata>
</package>
```

The specification defines the version of the component and any dependencies it might have on other NuGet packages. In the sample, there is a dependency on Shared.Core.dll from version 1.0.2 up to version 3.0.5 (included). You can find more information on versioning in NuGet documentation[12].

You can also create a specification based on a Visual Studio project by using the “spec” command within the project folder.

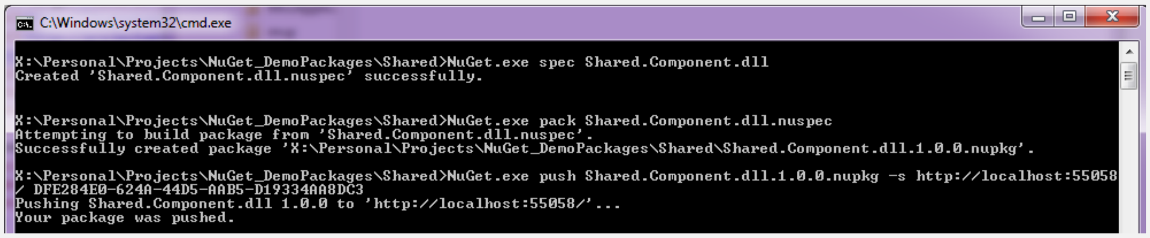
[9] http://docs.nuget.org/docs/creating-packages/creating-and-publishing-a-package#Create_the_manifest

[10] http://docs.nuget.org/docs/reference/nuspec-reference

[11] http://docs.nuget.org/docs/creating-packages/using-a-gui-to-build-packages

[12] http://docs.nuget.org/docs/reference/versioning

Step	Instructions
3 Create the package ☐ - Done	Generate the NuGet package: nuget.exe pack "Shared.Component.dll.nuspec" You can also skip the first 3 steps by using the project file as input for "pack" command to generate package structure, NuGet specification file, and resulting package based on assembly and project metadata.

4
Cope, move or publish the package
☐ - Done
	Move or publish the NuGet package[13] to a NuGet repository (feed).
	This screenshot shows the whole process.
	
	C:\Windows\system32\cmd.exe
	`X:\Personal\Projects\NuGet_DemoPackages\Shared>NuGet.exe spec Shared.Component.dll`
	`Created 'Shared.Component.dll.nuspec' successfully.`
	
	`X:\Personal\Projects\NuGet_DemoPackages\Shared>NuGet.exe pack Shared.Component.dll.nuspec`
	`Attempting to build package from 'Shared.Component.dll.nuspec'.`
	`Successfully created package 'X:\Personal\Projects\NuGet_DemoPackages\Shared\Shared.Component.dll.1.0.0.nupkg'.`
	
	`X:\Personal\Projects\NuGet_DemoPackages\Shared>NuGet.exe push Shared.Component.dll.1.0.0.nupkg -s http://localhost:55058`
	`/ DFE284E0-624A-44D5-AAB5-D19334AA8DC3`
	`Pushing Shared.Component.dll 1.0.0 to 'http://localhost:55058/'...`
	`Your package was pushed.`
	The NuGet documentation includes detailed information about the transformations[14] and automatically running PowerShell scripts during package installation and removal[15].

Getting an update of a NuGet package

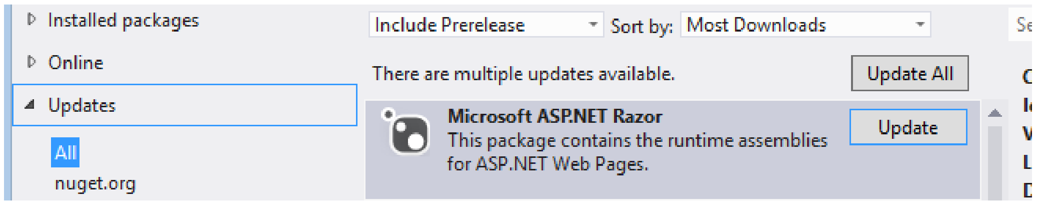
NuGet allows updating of package(s) on Project or Solution level. However, you should do thorough checks per package to verify that updating will not cause any unexpected problems within the solution or project. One approach is to use the –what if switch from the PS console to view the dependency chain that will be updated without actually performing the operation.

We recommend:

- Updating the package on the solution level to maintain the same version throughout projects within the solution.

We do **not** recommend:

- Updating all packages in the solution at the same time, unless you have done checks for all the packages.
- To have different versions of a NuGet package applied to different projects in the same solution. It could lead to build problems that would be difficult to track down. Exceptional cases may need different versions but mostly only temporarily. However if the packages are just for things like scripts in web applications such as jQuery, this may be ok in your scenario.



There are multiple updates available.

Microsoft ASP.NET Razor
This package contains the runtime assemblies for ASP.NET Web Pages.

Figure 23 – Updates available from Manage NuGet Packages window

[13] http://docs.nuget.org/docs/creating-packages/creating-and-publishing-a-package

[14] http://docs.nuget.org/docs/creating-packages/configuration-file-and-source-code-transformations

[15] http://docs.nuget.org/docs/creating-packages/creating-and-publishing-a-package#Automatically_Running_PowerShell_Scripts_During_Package_Installation_and_Removal

Before updating, check the release notes of the NuGet package. Does it contain any functional changes that might break your code? If it has new dependencies, how will that affect your project?

Starting with NuGet v2.8 it is possible to use the PowerShell standard –whatif switch[16] on PS Console with install-package, uninstall-package and update-package commands that provides a dry-run capability before applying the change.

The conventions of semantic versioning[17] provide a useful guide to the likely impact of a change on your project. Briefly, in the version number {Major}. {Minor}. {Patch}:

- Major updates may contain incompatible API changes
- Minor updates should contain only backwards-compatible changes
- Patch updates should contain only backwards-compatible bug fixes

Creating a NuGet feed

A NuGet repository is a feed that NuGet Package Manager, NuGet Package Console, NuGet.exe, or other NuGet tools can consume. The feed is a centralized repository for accessing the NuGet packages that you have pushed to the repository. There are three types of NuGet feeds, but only one of them supports comprehensive capabilities such as rss feed or pushing.

- **Local folder —**the simplest way to create a feed is to create a folder on local disk and copy the .nupkg into it. You can configure this location in Visual Studio as the NuGet feed location where your projects can consume the packages.
- **Shared folder—**Use a shared location such as a UNC share. Copy new packages to the share, as they become available Again it is a simple feed configuration in Visual Studio. Figure 3 illustrates the UI from where you can add new feeds.
- **NuGet server** - With your own NuGet web application, you get update notifications and other benefits. There is a NuGet package to help you. You can find detailed documentation in the NuGet website[18]. It is also possible to clone the full nuget.org gallery implementation[19].

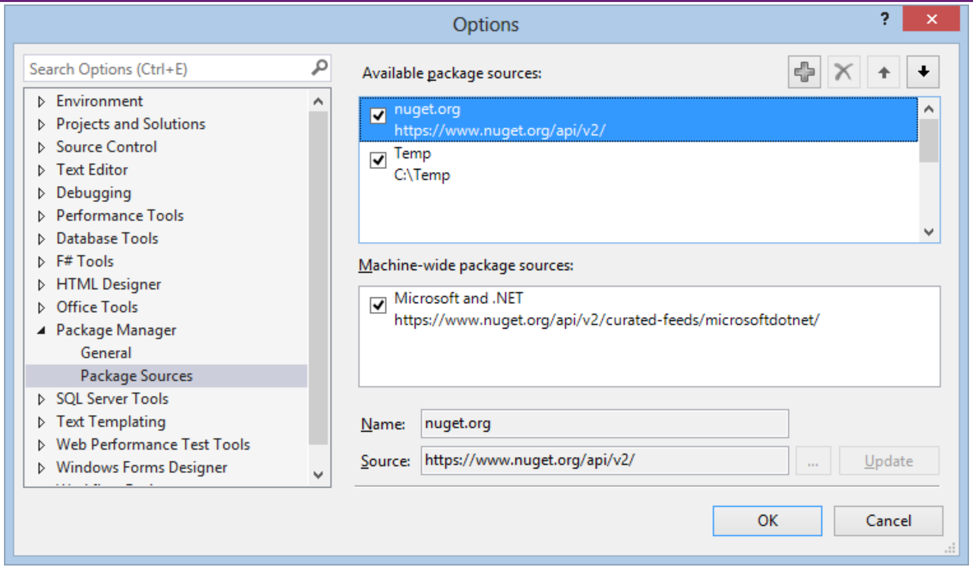
After you have set up any of these repositories, you can configure your Visual Studio installations to use them by opening Options, Package Manager, Package Sources. By default, Visual Studio 2013 is configured to read the public feed from NuGet.org.

[16] http://docs.nuget.org/docs/release-notes/nuget-2.8#Preview_NuGet_Operations_With_-whatif

[17] http://semver.org/

[18] http://docs.nuget.org/docs/creating-packages/hosting-your-own-nuget-feeds

[19] https://github.com/NuGet/NuGetGallery/blob/master/README.markdown



Available package sources:

- [✓] nuget.org https://www.nuget.org/api/v2/
- [✓] Temp C:\Temp

Machine-wide package sources:

- [✓] Microsoft and .NET https://www.nuget.org/api/v2/curated-feeds/microsoftdotnet/

Name: nuget.org

Source: https://www.nuget.org/api/v2/

Figure 3 – NuGet Package Manager configuration options in Visual Studio 2013

Using NuGet for Dependency management

NuGet makes it easy to share resources. NuGet has been around for a few years. It is constantly evolving, maturing, and gaining more and more ground[20] in the software development community.

Version Control Storage Locations for Shared Resources

Should you keep NuGet packages in version control?

For the source of the package, the answer is obviously yes – just as you would for the code of any assembly.

In the target projects where the packages are used, there are two approaches:

- Do not add NuGet packages to version control. Instead, use the NuGet Package Restore pattern, which we will describe in the next chapter.
- Add NuGet packages to version control like your source code. Configure NuGet to download one version of each package and keep it in a location that you can share across the team project.

Types of resources that can be shared with NuGet

NuGet packages can contain four types of resources[21] – assemblies, tools, content (such as images, css, JavaScript, razor views, c# code files) and build (MSBuild targets files that you insert into the project). It is possible to make packages that contain only compiled assemblies or only content files or both. Additionally you can structure each assembly or content for NuGet to be install only for specific .Net framework versions[22]. Another important aspect to remember is that NuGet packages can contain useful custom installation/uninstallation and configuration transformation scripts per package and even per framework version.

Dependency Management Pattern with Central NuGet feed(s)

The main pattern for shared resources and dependency management with NuGet is to use internal and external NuGet feed(s). By default, NuGet has knowledge of the external public NuGet Gallery feed, through which you

[20] http://stats.nuget.org/

[21] http://docs.nuget.org/docs/creating-packages/creating-and-publishing-a-package#From_a_convention_based_working_directory

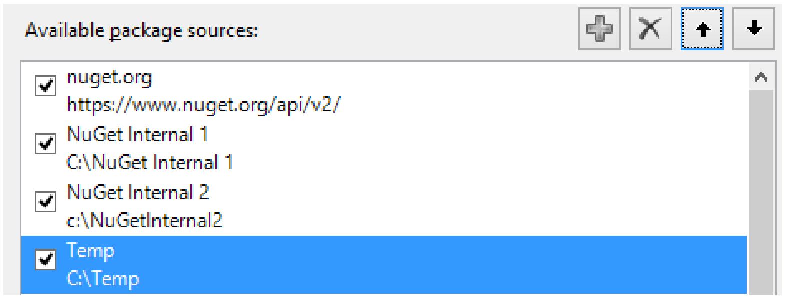
[22] http://docs.nuget.org/docs/creating-packages/creating-and-publishing-a-package#Supporting_Multiple_.NET_Framework_Versions_and_Profiles

can find standard publicly released packages such as NHibernate, jQuery, Entity Framework, ASP.NET MVC. Using the NuGet tools (such as Package Manager) you can install new NuGet packages to the project or update existing ones. The NuGet Visual Studio Extension will notify you when new versions of installed packages become available. When you install a NuGet package to your project, NuGet will automatically install all dependencies.

This pattern has three threads.

- **Provide a NuGet feed.** It can be either local directory, UNC share or a real NuGet.Server based repository. To have a feed that is available to everyone in the team, use a UNC share or NuGet.Server feed. It is very easy to publish a package into a UNC share as you just copy it into the shared directory. Creating a NuGet.Server feed requires a small amount of development work (or at least the setup of cloned nuget.org gallery), and you use the NuGet utility to publish packages. The main advantage of the NuGet server is that it can notify users of new versions and will scale better for searching and browsing, as there are more packages in the repository since package metadata is available for querying.
- **Populate the feed** with NuGet packages. Using the command-line utility nuget.exe, you can automate the creation and publication of new packages with the help of MSBuild or Team Foundation Build.
- **Listen to the feed.** In case you plan to store the packages in Version Control, then in each project that uses a package, just one team member should have responsibility for subscribing to the feed, installing the package, and subsequently getting updated versions from the feed. That user should install the package in his own environment, and then check the package into version control, just like any other changes. Other team members get the package contents through version control—they do not subscribe to the feed.
  This method ensures that everyone in the project is using the same version of the assemblies or other resources from the package. Optionally, you could set permissions that prevent other team members from updating the resources from the package.
  You can also decide to avoid storing packages in Version Control and use “Dependency Management Pattern using Package Restore.” In this scenario, all development environments would access the feed and download missing packages when needed. Keeping dependencies, and often this may mean binaries out of version control can provide several benefits.

You can set Visual Studio to work with more than one NuGet repository at the same time. You can now add packages from both internal (visible within company) and external repositories. One scenario for an organization might be to mirror “approved” packages from a public feed to a private feed so that developers only work with approved packages. In your internal repository, you can also include packages of internal components that depend on packages from any of the repositories (incl. external such as the NuGet feed) and NuGet will automatically install those dependencies, as you would expect. Additionally when searching for a package, NuGet will look for it in each repository in turn. If you enable package restoration, NuGet uses this same behavior of sequential search in each repository to get dependent packages.



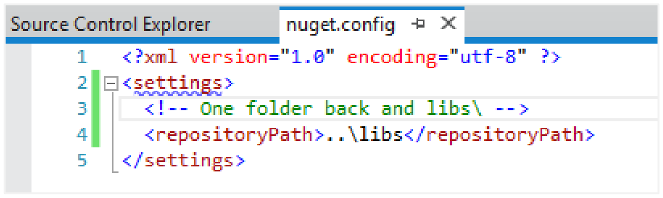
- ✓ nuget.org
  https://www.nuget.org/api/v2/
- ✓ NuGet Internal 1
  C:\NuGet Internal 1
- ✓ NuGet Internal 2
  c:\NuGetInternal2
- ✓ Temp
  C:\Temp

Figure 47 – List of available package sources

The downloaded NuGet packages location

By default, NuGet places all downloaded packages within the solution folder. However, depending on the version control structure and requirements in the project, you might want to place the packages in a different location.

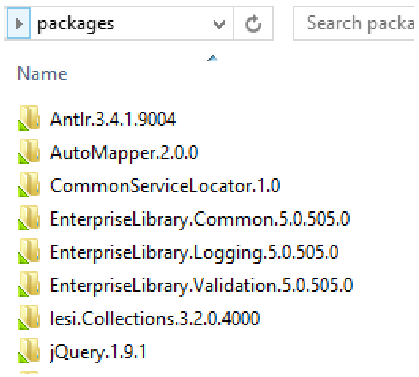
The location where to store the packages (among other options) can be changed using nuget.config[23] (Figure 5) and placed at a suitable location in the folder hierarchical structure.


Source Control Explorer | nuget.config

```
<?xml version="1.0" encoding="utf-8" ?>
<settings>
  <!-- One folder back and libs\ -->
  <repositoryPath>..\libs</repositoryPath>
</settings>
```

Figure 5 – Nuget.config file to configure different repository path

When you add a package to the project, NuGet will add it in a folder named following the pattern "PackageName.Version."


packages

Name

Antlr.3.4.1.9004
AutoMapper.2.0.0
CommonServiceLocator.1.0
EnterpriseLibrary.Common.5.0.505.0
EnterpriseLibrary.Logging.5.0.505.0
EnterpriseLibrary.Validation.5.0.505.0
Iesi.Collections.3.2.0.4000
jQuery.1.9.1

Figure 69 – "Packages" folder and its content Dependency Management Pattern with NuGet Package Restore

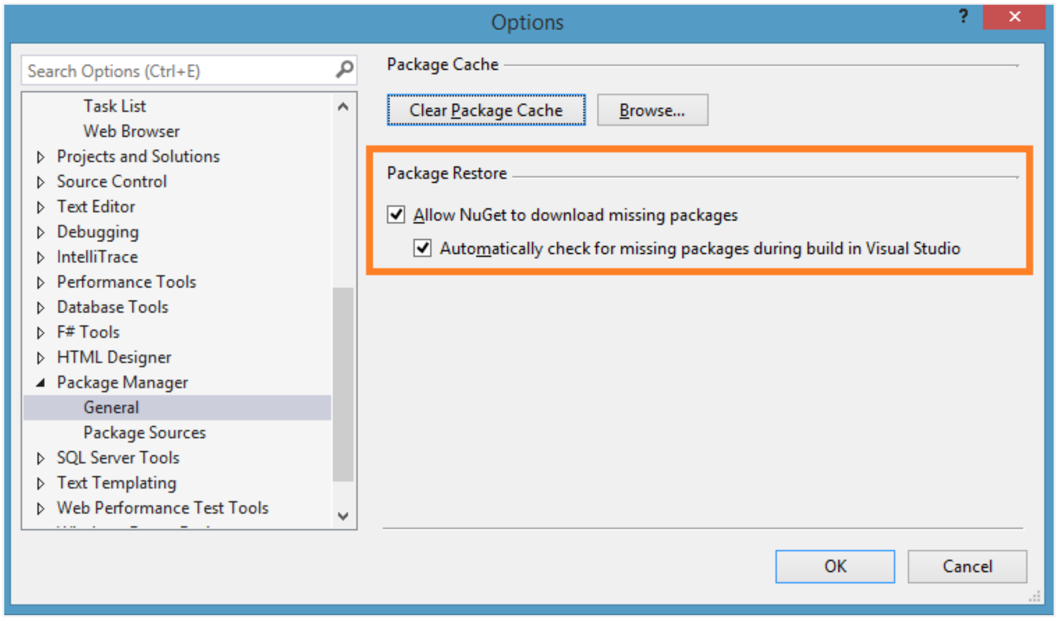
In this pattern, you will not add packages to version control. Instead, each development environment downloads packages from the repository according to the packages.config that you define per project in a solution. Starting with v2.7 NuGet enables Package Restore in Visual Studio by default. By default, you give consent to download automatically missing packages (Tools -> Options -> Package Manager - Figure 8).

Automatic Package Restore[24] will restore missing packages before building any MSBuild projects. This makes sure that NuGet had downloaded all the necessary items (including build targets/properties that are included in the nuget package) and placed them in the correct location. NuGet does this by hooking into Visual Studio build events.

For Team Build 2013 and Visual Studio Online NuGet Package Restore workflow works by default, however for previous versions of Team Build some customization is necessary to integrate using the NuGet Command Line Package Restore[25].

Please note that if you have enabled NuGet Package Restore on solution level before v2.7, then we recommended migrating it to latest "Automatic Package Restore[26]."

[23] http://docs.nuget.org/docs/reference/nuget-config-file#NuGet_config_extensibility_point
[24] http://docs.nuget.org/docs/reference/package-restore
[25] http://docs.nuget.org/docs/reference/package-restore-with-team-build
[26] http://docs.nuget.org/docs/workflows/migrating-to-automatic-package-restore



Package Restore

☑ Allow NuGet to download missing packages

☑ Automatically check for missing packages during build in Visual Studio

Figure 7 Visual Studio Package Manager Package Restore default settings

Getting NuGet packages with package restore

Add and update packages using NuGet package manager in the same way as you did before.

Do not commit the binaries into source control.

As before, only one team member needs maintain the packages configurations for projects, and check them into version control (and you might want to restrict write access to the file). When any team member presses F5, the first time, NuGet will automatically download the packages. This will also happen in the build server.

Pros and Cons regarding NuGet package restore

Like with any feature, there are always advantages and disadvantages to using it and the same applies to NuGet package restore[27]. Keep in mind that although NuGet package restore depends on a network connection to a NuGet feed, NuGet can rely on package cache scenarios to work offline. Also in today's development, environments the network feed to NuGet may be mostly reliable.

Using NuGet Package restore

Advantages

- You will use less storage space in TFS Source Control because no you do not need to store binaries in the repository.
- Less data in TFS means less to back up for TFS DR scenarios
- Get operations may be optimized since you are pulling less data over the wire

Disadvantages

- Developers can be blocked if any of the NuGet repositories is unavailable after a package update.
- TFS Build would fail or be postponed if a NuGet repository is unavailable (especially when the setting to clean workspace before build is turned on) if no cache version of the package was available.

[27] http://docs.nuget.org/docs/workflows/using-nuget-without-committing-packages

Additional references

- How to use NuGet with TFS Version Control [28]
- Introducing NuGet Package Management for .NET [29]
- Manage Project Libraries with NuGet [30]
- NuGet Documentation [31]
- NuGet FAQ [32]
- NuGet Gallery [33]
- NuGet In Depth: Empowering Open Source on the .NET Platform [34]
- NuGet Videos [35]
- Packages in Visual Studio Templates [36]
- NuGet Package Restore with Team Foundation Build [37]

[28] http://loicbaumann.fr/en/2012/02/07/how-to-use-nuget-with-the-tfs-version-control/

[29] http://www.hanselman.com/blog/IntroducingNuGetPackageManagementForNETAnotherPieceOfTheWebStack.aspx

[30] http://msdn.microsoft.com/en-us/magazine/hh547106.aspx

[31] http://docs.nuget.org/

[32] http://docs.nuget.org/docs/start-here/nuget-faq

[33] http://www.nuget.org/

[34] http://channel9.msdn.com/Events/MIX/MIX11/FRM0

[35] http://docs.nuget.org/docs/start-here/videos

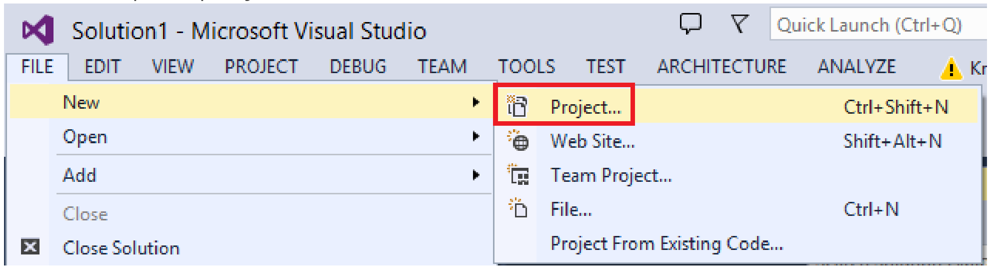
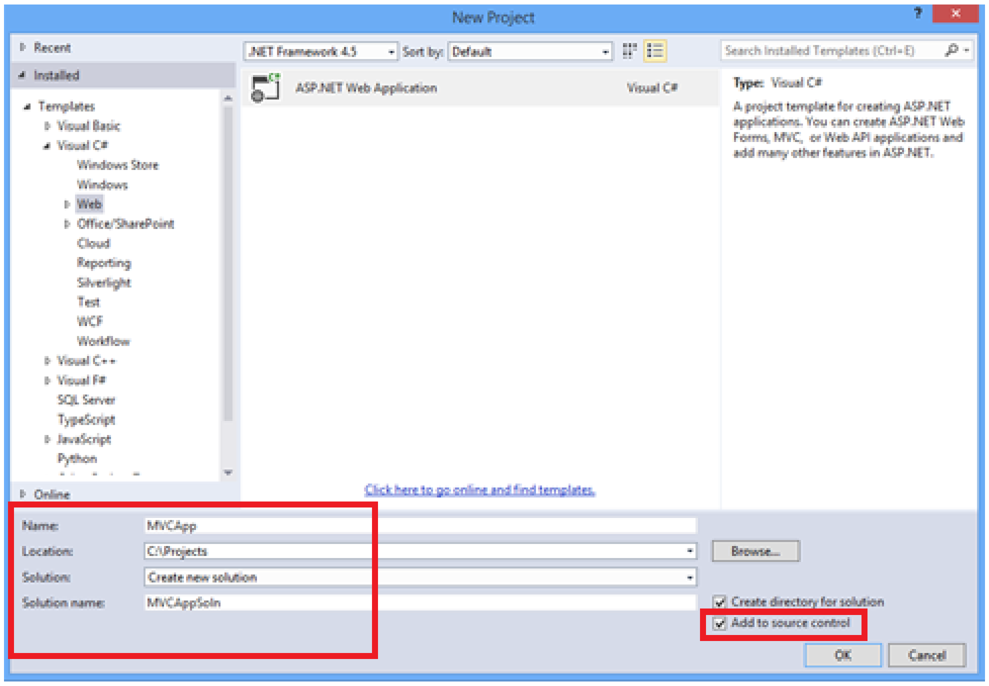
[36] http://docs.nuget.org/docs/reference/packages-in-visual-studio-templates

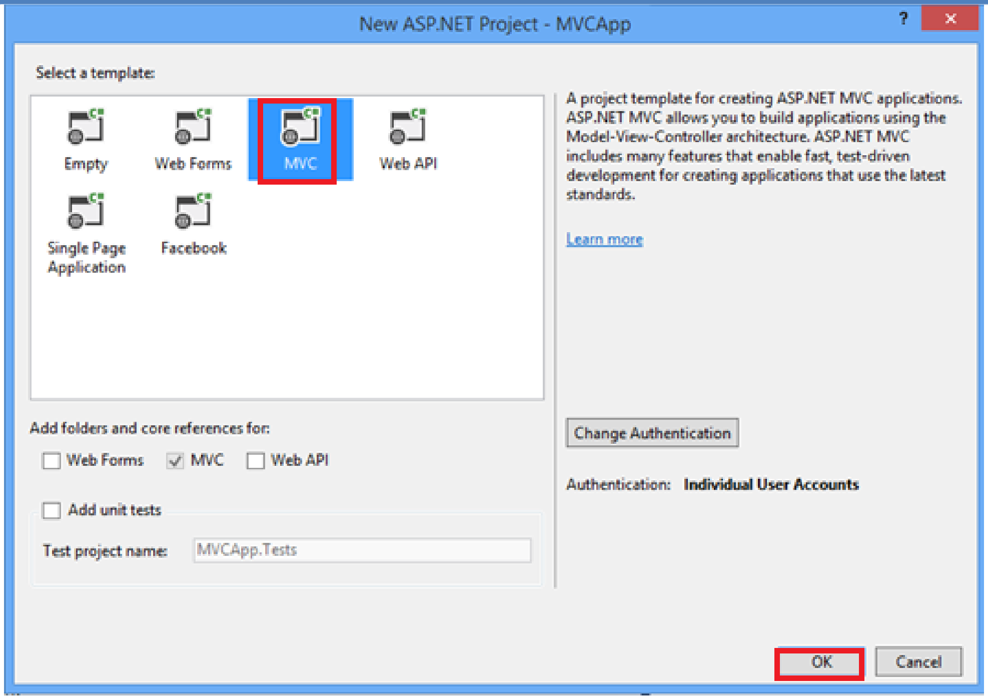
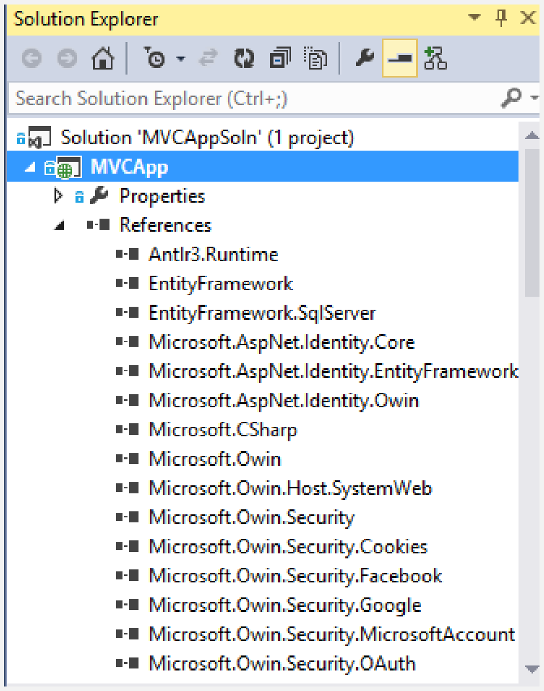
[37] http://docs.nuget.org/docs/reference/package-restore-with-team-build

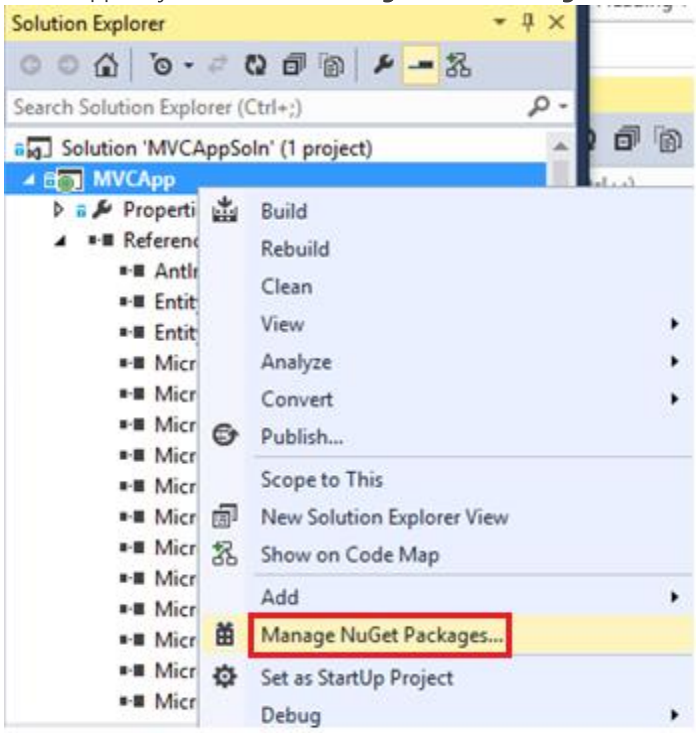
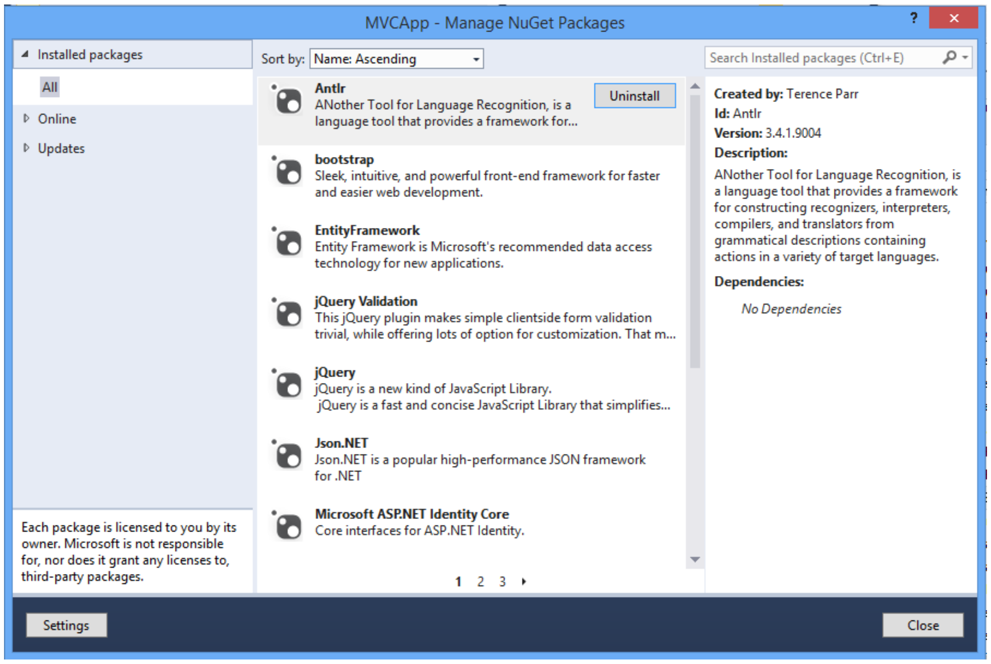
Hands-on Lab (HOL) – Using NuGet

Exercise 1 – Find and install a NuGet Package

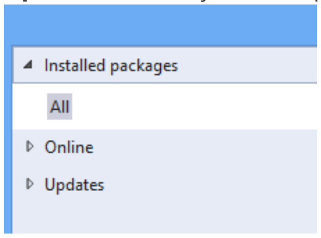
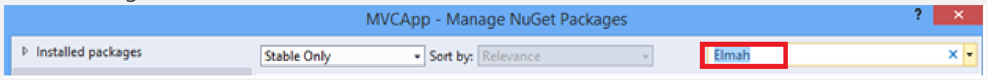
NuGet is a Visual Studio Extension that makes it easy to install and update libraries and tools in Visual Studio.

Step	Instructions
1 Create sample solution ☐ - Done	• Open Visual Studio 2013. • Choose File | New | Project.  Solution1 - Microsoft Visual Studio Quick Launch (Ctrl+Q) FILE EDIT VIEW PROJECT DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE New Open Add Close Close Solution Project... Ctrl+Shift+N Web Site... Shift+Alt+N Team Project... File... Ctrl+N Project From Existing Code... • Select **ASP.NET Web Application** from the **Web** template. Make sure to check **Add to source control** to add the solution to TFS.  New Project Recent Installed Templates Visual Basic Visual C# Windows Store Windows Web Office/SharePoint Cloud Reporting Silverlight Test WCF Workflow Visual C++ Visual F# SQL Server TypeScript JavaScript Python Online .NET Framework 4.5 Sort by: Default Search Installed Templates (Ctrl+E) ASP.NET Web Application Visual C# Click here to go online and find templates. Browse... OK Cancel Type: Visual C# A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET. Name: MVCApp Location: C:\Projects Solution: Create new solution Solution name: MVCAppSoln ☑ Create directory for solution ☑ Add to source control • Select **MVC** template from New ASP.NET Project dialog box. Click **OK**.

Step	Instructions
	 New ASP.NET Project - MVCApp Select a template: Empty, Web Forms, [✓ MVC], Web API, Single Page Application, Facebook A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards. Learn more Add folders and core references for: ☐ Web Forms ☑ MVC ☐ Web API ☐ Add unit tests Test project name: MVCApp.Tests Change Authentication Authentication: **Individual User Accounts** OK Cancel • Check in pending changes.
2 Add references ☐ - Done	• Open the solution you just created in **Solution Explorer**. • In **Solution Explorer**, expand **References**. • As part of the **MVC** template, several NuGet packages such as **EntityFramework** are available for use in the solution.  Solution Explorer Search Solution Explorer (Ctrl+;) Solution 'MVCAppSoln' (1 project) MVCApp Properties References Antlr3.Runtime EntityFramework EntityFramework.SqlServer Microsoft.AspNet.Identity.Core Microsoft.AspNet.Identity.EntityFramework Microsoft.AspNet.Identity.Owin Microsoft.CSharp Microsoft.Owin Microsoft.Owin.Host.SystemWeb Microsoft.Owin.Security Microsoft.Owin.Security.Cookies Microsoft.Owin.Security.Facebook Microsoft.Owin.Security.Google Microsoft.Owin.Security.MicrosoftAccount Microsoft.Owin.Security.OAuth

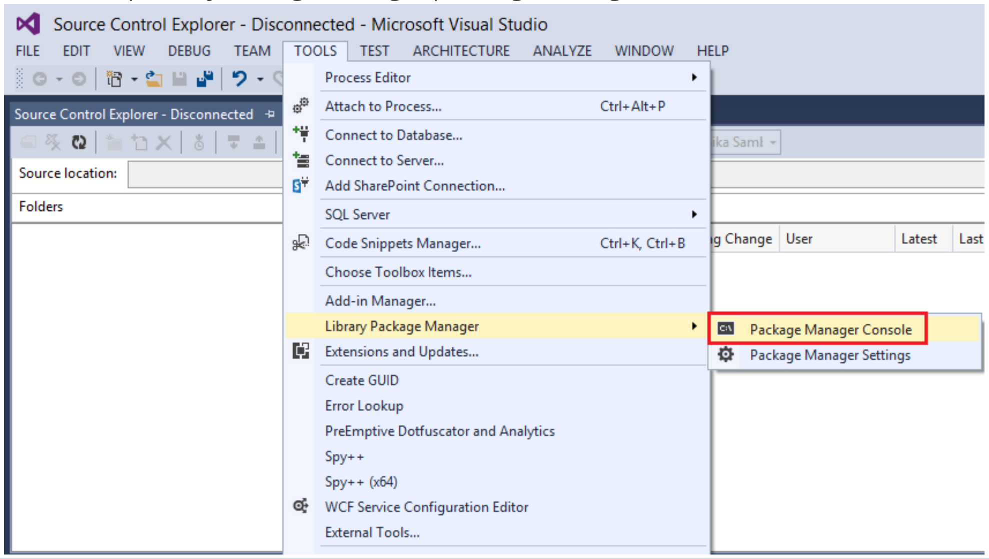
Step	Instructions
3 Manage NuGet Packages ☐ - Done	• Right click MVCApp Project and select **Manage NuGet Packages**.  Solution Explorer Search Solution Explorer (Ctrl+;) Solution 'MVCAppSoln' (1 project) MVCApp Properti Referenc Antlr Entit Entit Micr Build Rebuild Clean View Analyze Convert Publish... Scope to This New Solution Explorer View Show on Code Map Add Manage NuGet Packages... Set as StartUp Project Debug ○ • **Manage NuGet Packages** dialog box appears displaying the NuGet packages installed as part of the MVCApp project. You can also right click MVCAppSoln and manage NuGet packages for the entire solution.  MVCApp - Manage NuGet Packages Installed packages All Online Updates Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages. Sort by: Name: Ascending Search Installed packages (Ctrl+E) **Antlr** ANother Tool for Language Recognition, is a language tool that provides a framework for... Uninstall **bootstrap** Sleek, intuitive, and powerful front-end framework for faster and easier web development. **EntityFramework** Entity Framework is Microsoft's recommended data access technology for new applications. **jQuery Validation** This jQuery plugin makes simple clientside form validation trivial, while offering lots of option for customization. That m... **jQuery** jQuery is a new kind of JavaScript Library. jQuery is a fast and concise JavaScript Library that simplifies... **Json.NET** Json.NET is a popular high-performance JSON framework for .NET **Microsoft ASP.NET Identity Core** Core interfaces for ASP.NET Identity. 1 2 3 ▸ **Created by:** Terence Parr **Id:** Antlr **Version:** 3.4.1.9004 **Description:** ANother Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages. **Dependencies:** *No Dependencies* Settings Close • You can use **Manage NuGet Packages** dialog box to uninstall packages as well. • On the left pane of the **Manage NuGet Packages** dialog, there are three sections. ○ **Installed packages** – Displays installed packages available as part of your project or solution.

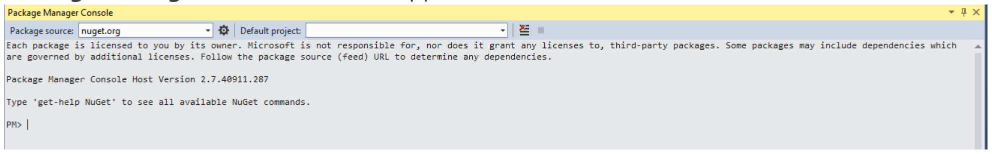
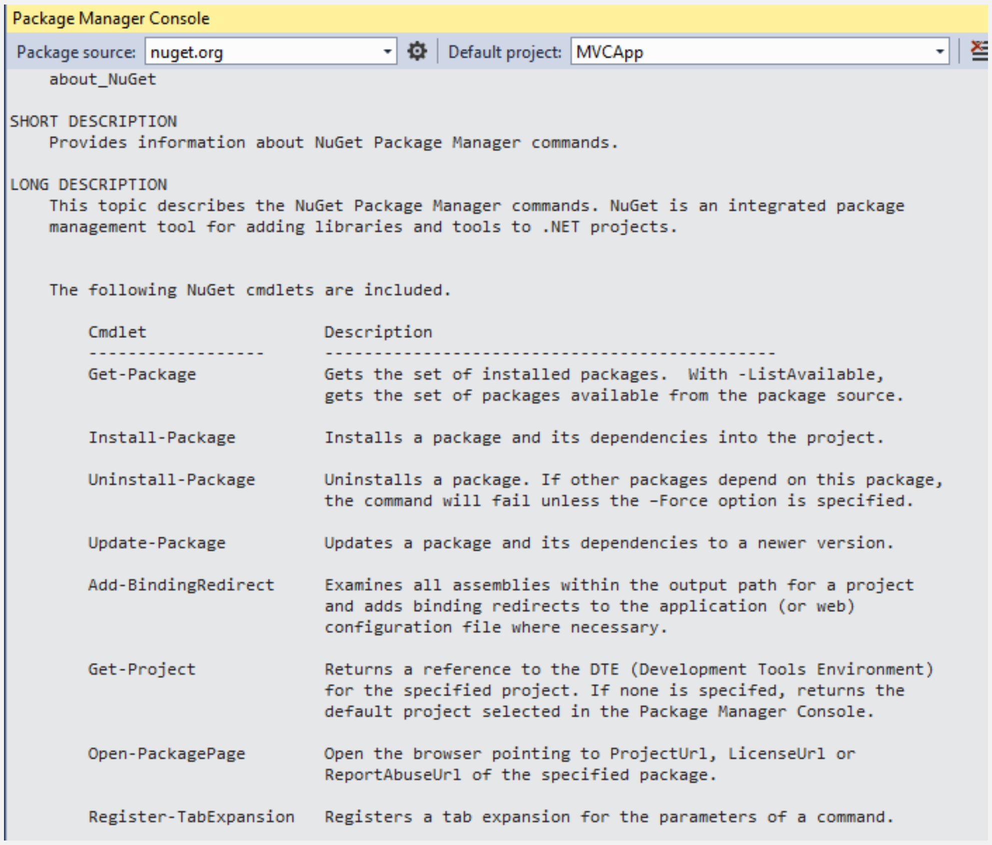
Microsoft

Step	Instructions
	○ **Online** – On NuGet.org or your private feeds (if you have any set up). You can search for a specific Displays NuGet packages that are available package and install it from this section. ○ **Updates** – Lists any installed packages that have updates. 
4 Add Elmah ☐ - Done	• Expand **Online** on the left pane and select **nuget.org**. Use the Search field on the top right corner of the dialog to find **Elmah**.  MVCApp - Manage NuGet Packages Installed packages Stable Only Sort by: Relevance Elmah • Elmah (Error Logging Modules and Handlers) is a logging framework available for ASP.NET. • Install **ELMAH** package by clicking on the **Install** button. Alternatively, you can use the **Package Manager Console** to install NuGet packages. • Go back to References to double check that you have installed **Elmah**. • **Elmah** framework is now ready for use.

Exercise 2 – Using Package Manager Console

Package Manager Console is a PowerShell enabled window in Visual Studio. PowerShell commands allow you to install packages when you do not have the solution open. You will also need to use it for packages that create commands that require PowerShell.

Step	Instructions
1 Open Package Manager Console ☐ - Done	• Open Visual Studio 2013 • Click **Tools** | **Library Package Manager** | **Package Manager Console**  Source Control Explorer - Disconnected - Microsoft Visual Studio FILE EDIT VIEW DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP Source Control Explorer - Disconnected Source location: Folders Process Editor Attach to Process... Ctrl+Alt+P Connect to Database... Connect to Server... Add SharePoint Connection... SQL Server Code Snippets Manager... Ctrl+K, Ctrl+B Choose Toolbox Items... Add-in Manager... Library Package Manager Extensions and Updates... Create GUID Error Lookup PreEmptive Dotfuscator and Analytics Spy++ Spy++ (x64) WCF Service Configuration Editor External Tools... Package Manager Console Package Manager Settings Change User Latest Last

Step	Instructions
	• **Package Manager Console** window appears. 
2 NuGet PowerShell cmdlets ☐ - Done	• **Package Manager Console** is a PowerShell enabled window. You can run NuGet specific PowerShell cmdlets to perform a variety of operations. • To see a list of the available NuGet commands, type *Get-Help NuGet* in the **Package Manager Console**  • Type *Get-Package* to view the list of installed packages. • Type *Uninstall-Package Elmah –Force* to uninstall Elmah and its dependencies. • Double check to make sure NuGet has uninstalled Elmah and its dependencies.

Package Manager Console

Package source: nuget.org Default project:

```
Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may include dependencies which
are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 2.7.40911.287

Type 'get-help NuGet' to see all available NuGet commands.

PM> |
```

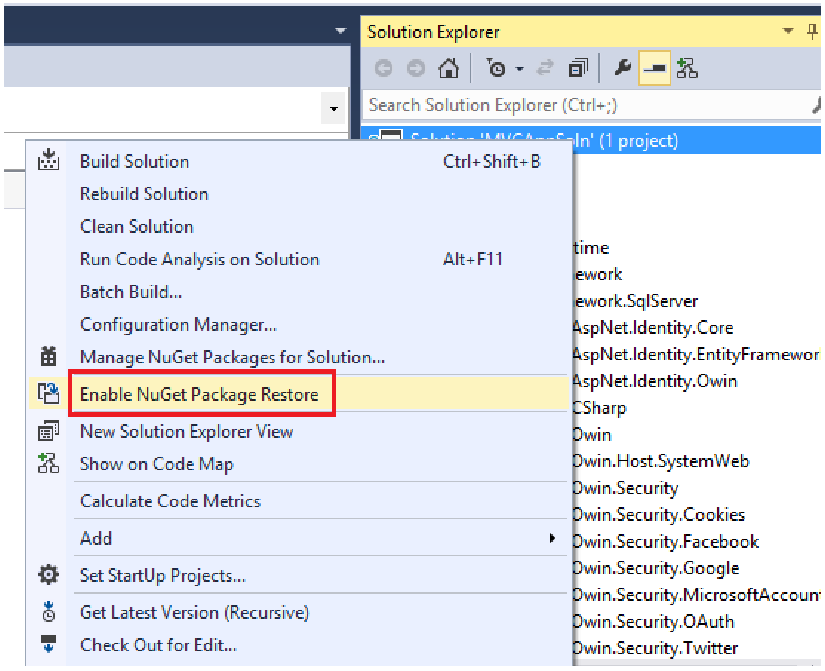
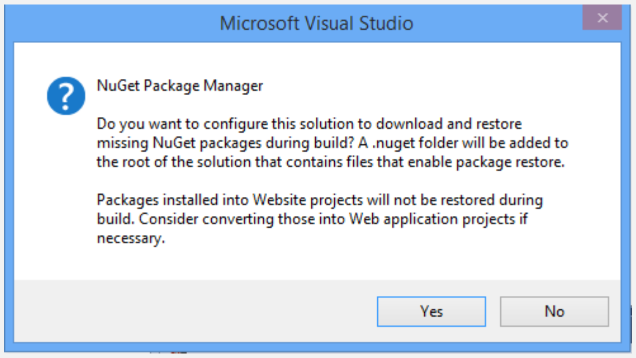
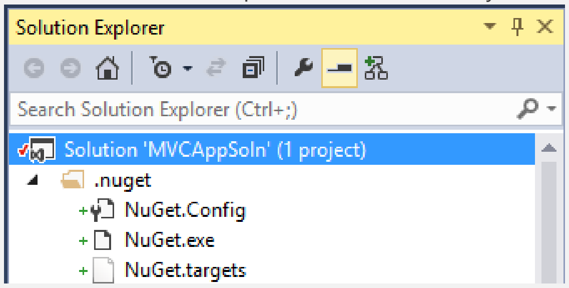
Package Manager Console

Package source: nuget.org Default project: MVCApp

```
    about_NuGet

SHORT DESCRIPTION
    Provides information about NuGet Package Manager commands.

LONG DESCRIPTION
    This topic describes the NuGet Package Manager commands. NuGet is an integrated package
    management tool for adding libraries and tools to .NET projects.


    The following NuGet cmdlets are included.

        Cmdlet                  Description
        ------------------      ----------------------------------------------
        Get-Package             Gets the set of installed packages.  With -ListAvailable,
                                gets the set of packages available from the package source.

        Install-Package         Installs a package and its dependencies into the project.

        Uninstall-Package       Uninstalls a package. If other packages depend on this package,
                                the command will fail unless the -Force option is specified.

        Update-Package          Updates a package and its dependencies to a newer version.

        Add-BindingRedirect     Examines all assemblies within the output path for a project
                                and adds binding redirects to the application (or web)
                                configuration file where necessary.

        Get-Project             Returns a reference to the DTE (Development Tools Environment)
                                for the specified project. If none is specifed, returns the
                                default project selected in the Package Manager Console.

        Open-PackagePage        Open the browser pointing to ProjectUrl, LicenseUrl or
                                ReportAbuseUrl of the specified package.

        Register-TabExpansion   Registers a tab expansion for the parameters of a command.
```

Exercise 3 – NuGet Package Manager

NuGet has package restoration that on building is capable of downloading all packages that have been installed and configured for the project, but cannot be found from local disk. This does not change the fact that NuGet needs to install packages for the project and installs updates the same way. NuGet enables Package Restore at the solution level.

Step	Instructions
1 Enable NuGet Package Restore ☐ - Done	• Right click MVCAppSoln and select **Enable NuGet Package Restore**.  Solution Explorer Search Solution Explorer (Ctrl+;) Solution 'MVCAppSoln' (1 project) Build Solution Ctrl+Shift+B Rebuild Solution Clean Solution Run Code Analysis on Solution Alt+F11 Batch Build... Configuration Manager... Manage NuGet Packages for Solution... Enable NuGet Package Restore New Solution Explorer View Show on Code Map Calculate Code Metrics Add Set StartUp Projects... Get Latest Version (Recursive) Check Out for Edit... time ework ework.SqlServer AspNet.Identity.Core AspNet.Identity.EntityFramewor AspNet.Identity.Owin CSharp Owin Owin.Host.SystemWeb Owin.Security Owin.Security.Cookies Owin.Security.Facebook Owin.Security.Google Owin.Security.MicrosoftAccoun Owin.Security.OAuth Owin.Security.Twitter
2 Restore missing NuGet packages ☐ - Done	• Click Yes. •  NuGet Package Manager Do you want to configure this solution to download and restore missing NuGet packages during build? A .nuget folder will be added to the root of the solution that contains files that enable package restore. Packages installed into Website projects will not be restored during build. Consider converting those into Web application projects if necessary. • Go back to Solution Explorer to view the newly added .nuget folder.  Solution Explorer Search Solution Explorer (Ctrl+;) Solution 'MVCAppSoln' (1 project) .nuget NuGet.Config NuGet.exe NuGet.targets

In Conclusion

This concludes our adventure into using NuGet. We have touched on basic theory of NuGet, managing shared resources, dependency management, and concluded with a practical Hands-on lab (HOL) exploring NuGet.

We hope that you have found this guide useful.

Sincerely

The Microsoft Visual Studio ALM Rangers