Hands-on Lab 3.2: Automating the Deployment of the WPF App

Table of Contents

Objectives

In this HOL you learn how to automatically deploy the Windows Presentation Foundation (WPF) application that is part of the sample Trey Research project. Automating the deployment includes:

- Setting up the target environments.
- Preparing the configuration files to match the target environments.
- Packaging the files that are needed to deploy and run the WPF application, along with the configuration files.
- Deploying to the target environments.

This HOL is part two of the three-part Automation HOL. Together, the three parts demonstrate how to use Microsoft Visual Studio, Microsoft Team Foundation Server (TFS) and Lab Management to automate the deployment and testing of the following technologies:

- WCF services
- WPF

Automation is discussed in Chapter 4 of Building a Release Pipeline with Team Foundation Server 2012. This lab implements the pipeline changes that are discussed in that chapter.

Prerequisites

The prerequisite for this lab is to have completed all the previous labs.

NOTE: You can use either standard environments or SCVMM environments for this lab. Standard environments are much simpler to set up because you do not need to configure the network virtualization, but some features, such as snapshots, are not available.

The Trey Research application is in your TreyResearch folder. Visual Studio solutions that are the result of completing all of the tasks in an exercise are in the **Lab03-Automation\Completed-Lab** folder.

You run the examples for this lab on your local computer and the computer(s) that host the environments.

Time

This HOL (Lab3_2) takes approximately 60 minutes.

Exercise 1: Setting Up the Target Environments

In this exercise you set up the target environments so that they can run the WPF application and so that they can be managed by the pipeline.

The pipeline has four environments: development, test, staging, and production. Because the development environment is isolated and only exists on development machines, it doesn't host the WPF application. However, the other three environments do. Typically, these three environments would reside on three different computers. For simplicity, this lab uses one computer for all three environments.

Task 1: Set Up the Environments

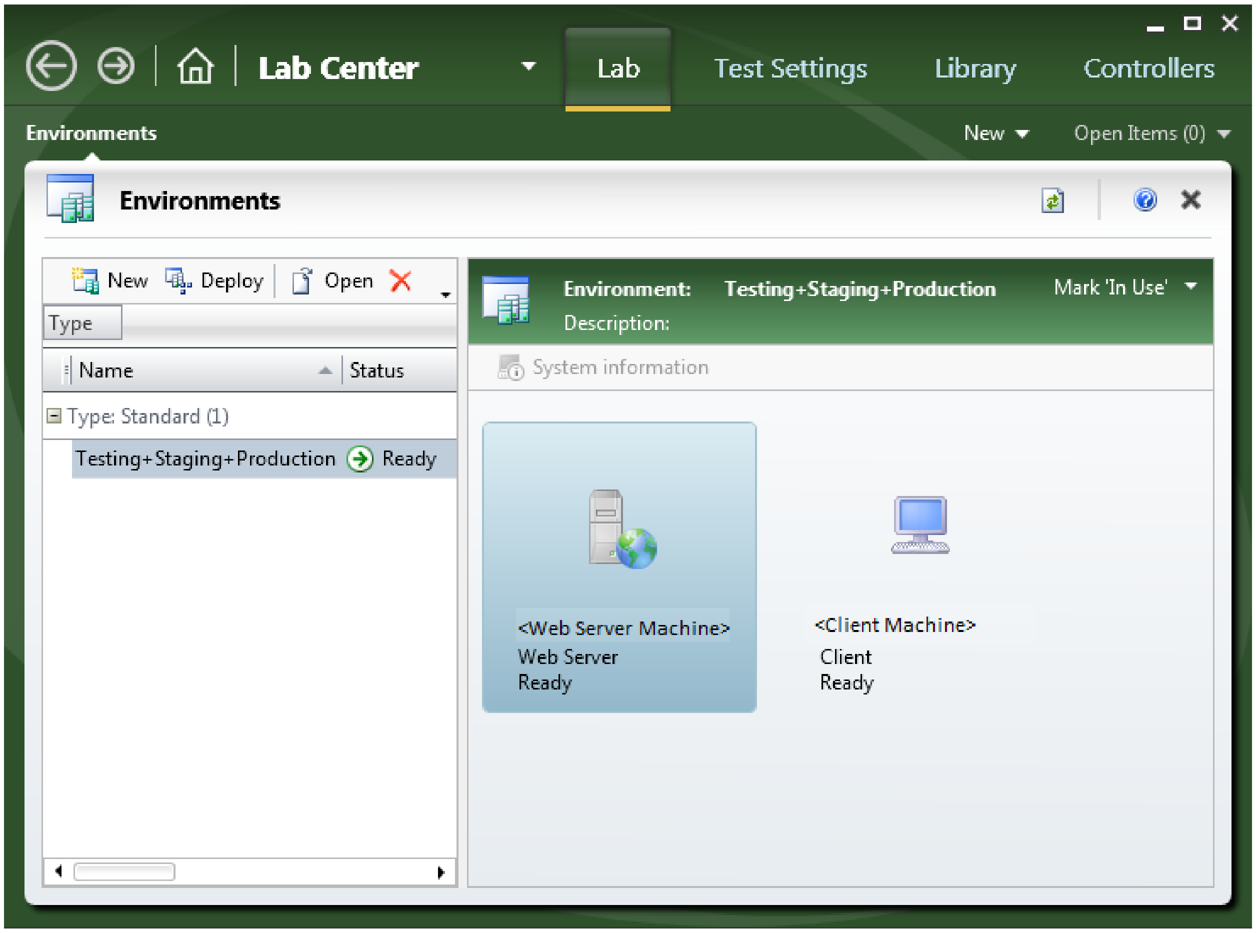
In this task you install the Windows Installer XML (WiX) tool set on the development machine, as well as on the build machine(s) that host the build agent(s) that run the commit stage of the pipeline. You use WiX to package and deploy the WPF application.

Go to http://wixtoolset.org/ to download and install WiX.

Task 2: Set Up the Environment in Lab Manager

Note: If you use Brain Kellar's VM, you use the same computer and environment that you set up in Lab 1 -- Starting Point.

From Lab Management, simply verify that the environment that contains the computer is still available. The following screenshot shows an example of what you should see.



Type: Standard (1)
Testing+Staging+Production Ready

Environment: Testing+Staging+Production
Description:
Mark 'In Use'
System information

<Web Server Machine>
Web Server
Ready

<Client Machine>
Client
Ready

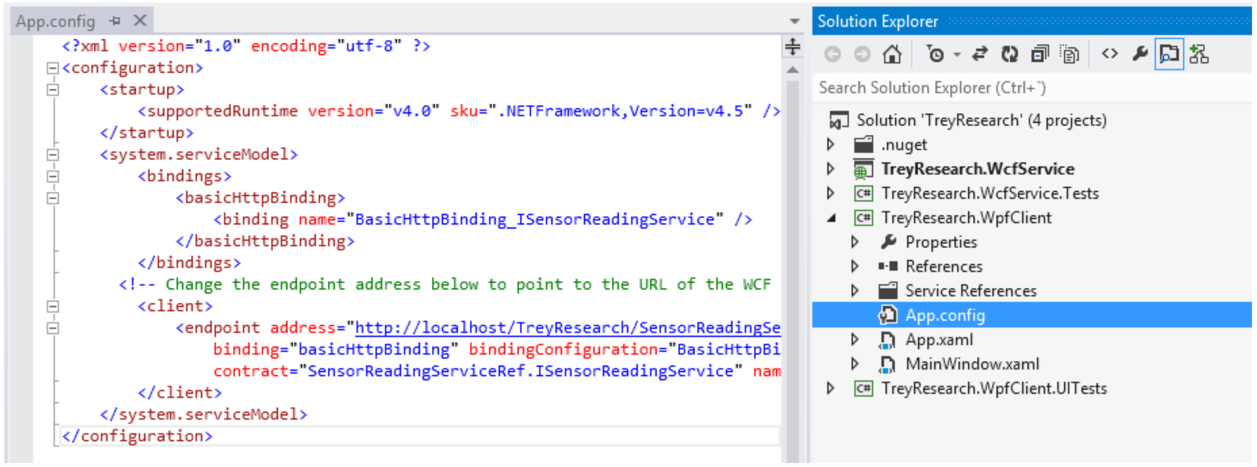
Exercise 2: Preparing the Deployment Configuration Files

In this exercise you automate the process of creating configuration files that conform to a specific environment. In particular, the App.config file of the WPF application uses a different URL in each environment to access the web services.

Task 1: Prepare the App.config Files

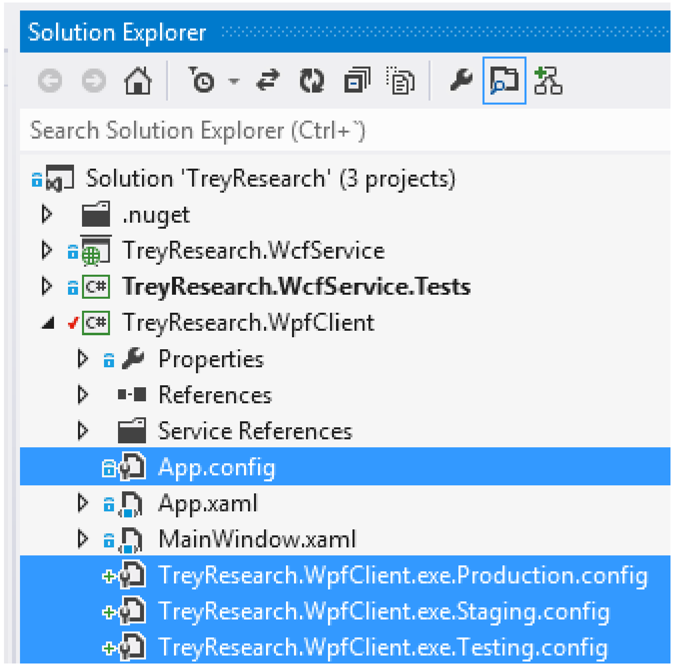
In this task you add the App.config files to the project. The pipeline uses these files when it deploys the WPF application to each environment. You need four files. Three are for the three environments. The fourth stores the base configuration that is transformed for each environment. This file already exists.

1. In Visual Studio, open **TreyResearch.sln**.
2. In Solution Explorer, under **TreyResearch.WpfClient**, find the **App.config file**.



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
    <system.serviceModel>
        <bindings>
            <basicHttpBinding>
                <binding name="BasicHttpBinding_ISensorReadingService" />
            </basicHttpBinding>
        </bindings>
      <!-- Change the endpoint address below to point to the URL of the WCF
        <client>
            <endpoint address="http://localhost/TreyResearch/SensorReadingSe
                binding="basicHttpBinding" bindingConfiguration="BasicHttpBi
                contract="SensorReadingServiceRef.ISensorReadingService" nam
        </client>
    </system.serviceModel>
</configuration>
```

3. Use the **Text File** template to add three additional configuration files. Name each one after the target environment, using the format **TreyResearch.WpfClient.exe.<*Environment*>.config**. The following screenshot shows the results.



Solution Explorer
Search Solution Explorer (Ctrl+`)
Solution 'TreyResearch' (3 projects)
.nuget
TreyResearch.WcfService
TreyResearch.WcfService.Tests
TreyResearch.WpfClient
Properties
References
Service References
App.config
App.xaml
MainWindow.xaml
TreyResearch.WpfClient.exe.Production.config
TreyResearch.WpfClient.exe.Staging.config
TreyResearch.WpfClient.exe.Testing.config

4. Add the following XML code to each file. Replace the **<IIS-Server>** tag with the name of the server for each different environment and config file.

XML

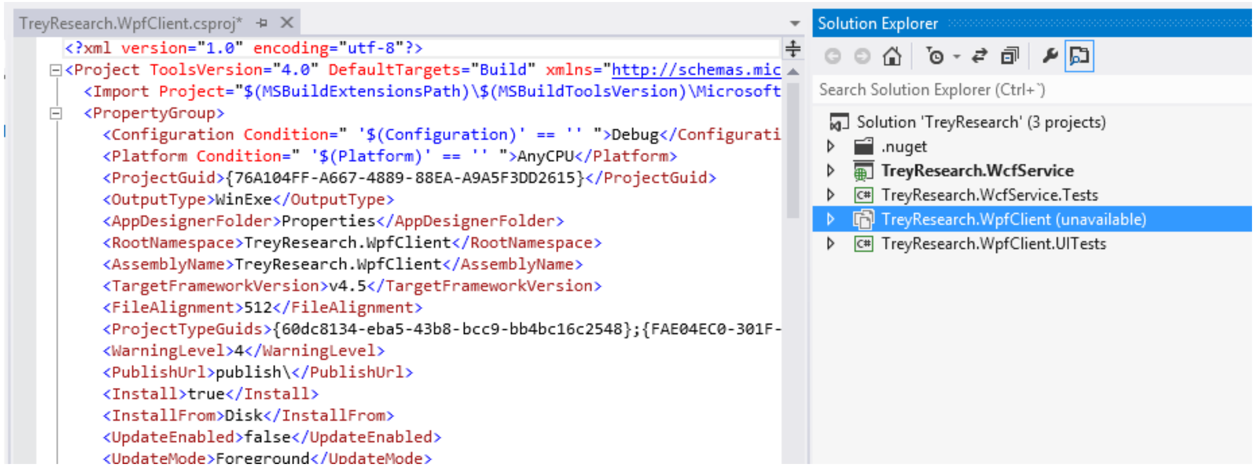
```
<?xml version="1.0" encoding="utf-8" ?>
<!-- For more information on using transformations
```

```
     see the web.config examples at
http://go.microsoft.com/fwlink/?LinkId=214134. -->
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-
Transform">
  <system.serviceModel>
    <client>

      <!-- AUTOMATION HOL - This defines the way the value of the "endpoint"
setting gets transformed for the Testing environment. -->
      <!-- AUTOMATION HOL - In the config file for the Testing environment,
the endpoint will get this value -->
      <endpoint address="http://<IIS-Server:port>/SensorReadingService.svc"
                name="BasicHttpBinding_ISensorReadingService"
                xdt:Locator="Match(name)"
                xdt:Transform="SetAttributes(address)">
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>
```

Task 2: Set the Dependency on the Base File

In this task you make all the **TreyResearch.WpfClient.exe.<*Environment Name*>.config** files dependent on the base configuration file, **App.config**. Although this step is not required to perform the transformations, it helps to keep the project organized and understandable.

1. Right-click on the **TreyResearch.WpfClient** project and select **Unload project**.
2. Right-click on the project again and select **Edit TreyResearch.WpfClient.csproj**. The MSBuild code that makes up the csproj project file appears.


TreyResearch.WpfClient.csproj*
Solution Explorer
Search Solution Explorer (Ctrl+`)

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.mic
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configurati
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{76A104FF-A667-4889-88EA-A9A5F3DD2615}</ProjectGuid>
    <OutputType>WinExe</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>TreyResearch.WpfClient</RootNamespace>
    <AssemblyName>TreyResearch.WpfClient</AssemblyName>
    <TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
    <ProjectTypeGuids>{60dc8134-eba5-43b8-bcc9-bb4bc16c2548};{FAE04EC0-301F-
    <WarningLevel>4</WarningLevel>
    <PublishUrl>publish\</PublishUrl>
    <Install>true</Install>
    <InstallFrom>Disk</InstallFrom>
    <UpdateEnabled>false</UpdateEnabled>
    <UpdateMode>Foreground</UpdateMode>
```

Solution 'TreyResearch' (3 projects)
- .nuget
- **TreyResearch.WcfService**
- TreyResearch.WcfService.Tests
- TreyResearch.WpfClient (unavailable)
- TreyResearch.WpfClient.UITests

3. Locate the following XML code.

XML

```
<None Include="TreyResearch.WpfClient.exe.Production.config" />
```

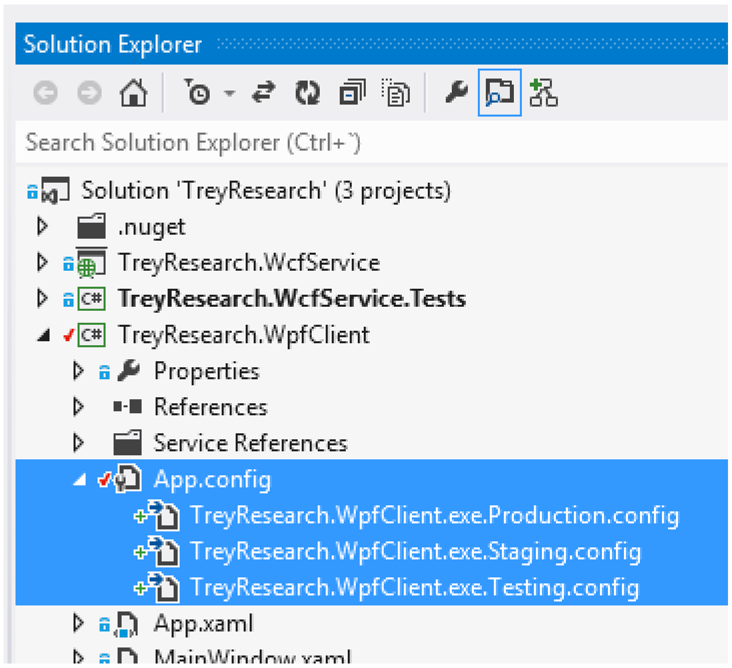
```
<None Include="TreyResearch.WpfClient.exe.Staging.config" />
<None Include="TreyResearch.WpfClient.exe.Testing.config" />
```

4. Replace the XML with the following code, which makes the environment-specific files dependent upon the base configuration file.

XML

```
<None Include="TreyResearch.WpfClient.exe.Production.config">
       <DependentUpon>App.config</DependentUpon>
</None>
<None Include="TreyResearch.WpfClient.exe.Staging.config">
      <DependentUpon>App.config</DependentUpon>
</None>
<None Include="TreyResearch.WpfClient.exe.Testing.config">
      <DependentUpon>App.config</DependentUpon>
</None>
```

5. Save the file. Right-click on the **TreyResearch.WpfClient** project and select **Reload Project**. Your project should have the structure shown in the following screenshot. Note that the environment-specific files are nested under the base configuration file.


Solution Explorer
Search Solution Explorer (Ctrl+`)
Solution 'TreyResearch' (3 projects)
.nuget
TreyResearch.WcfService
TreyResearch.WcfService.Tests
TreyResearch.WpfClient
Properties
References
Service References
App.config
TreyResearch.WpfClient.exe.Production.config
TreyResearch.WpfClient.exe.Staging.config
TreyResearch.WpfClient.exe.Testing.config
App.xaml
MainWindow.xaml

Task 3: Set Up the Transformations for the TreyResearch.WpfClient.exe.Config Files

In this task you use the **TransformXml** MSBuild task to transform the base **App.config** configuration file. You use the **TreyResearch.WpfClient.exe.<*Environment Name*>.config** files to define the transformations. When the transformations are done, you will have the environment-specific configuration files.

1. Unload and edit the **TreyResearch.WpfClient.csproj** file, the same way as you did in task 2. Near the bottom of the file, just before the closing **</Project>** tag, insert the following code.

XML

```
  <!-- AUTOMATION HOL - Referencing the TransformXml task so we can use it -->
  <UsingTask TaskName="TransformXml"
AssemblyFile="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v11.0\Web\Micr
osoft.Web.Publishing.Tasks.dll" />
  <!-- AUTOMATION HOL - We are triggering the transformations just after
building -->
  <!-- AUTOMATION HOL - Checking whether we are BuildingInsideVisualStudio
makes that this will be triggered only in the Commit Stage of the pipeline (or
in command-line builds), not while working inside Visual Studio -->
  <!-- AUTOMATION HOL - Checking whether we are building the Release
configuration makes the configuration files available only for Release builds,
the ones being used by the pipeline -->
  <Target Name="AfterBuild" Condition="('$(BuildingInsideVisualStudio)' !=
'true') And ('$(Configuration)' == 'Release')">
    <ItemGroup>
      <TransformationFiles Include="TreyResearch.WpfClient.exe.*.config" />
    </ItemGroup>
    <!-- AUTOMATION HOL - We generate the transformated configuration files in
a subfolder of $(OutDir). That way, they will get copied directly to the
BinariesFolder in the TFS build agent, and in turn, to the Drop folder of the
Commit Stage, without having to explicitly copying them -->
    <MakeDir Directories="$(OutDir)\ConfigFiles\WpfClient"
Condition="!Exists('$(OutDir)\ConfigFiles\WpfClient')" />
    <!-- AUTOMATION HOL - We call the TransformXml task for all the
configuration files using MSBuild batching (the @() syntax)  -->
    <TransformXml Source="App.config" Transform="@(TransformationFiles)"
Destination="$(OutDir)\ConfigFiles\WpfClient\%(TransformationFiles.Identity)"
/>
  </Target>
```

2. Reload the project. It opens in Solution Explorer.

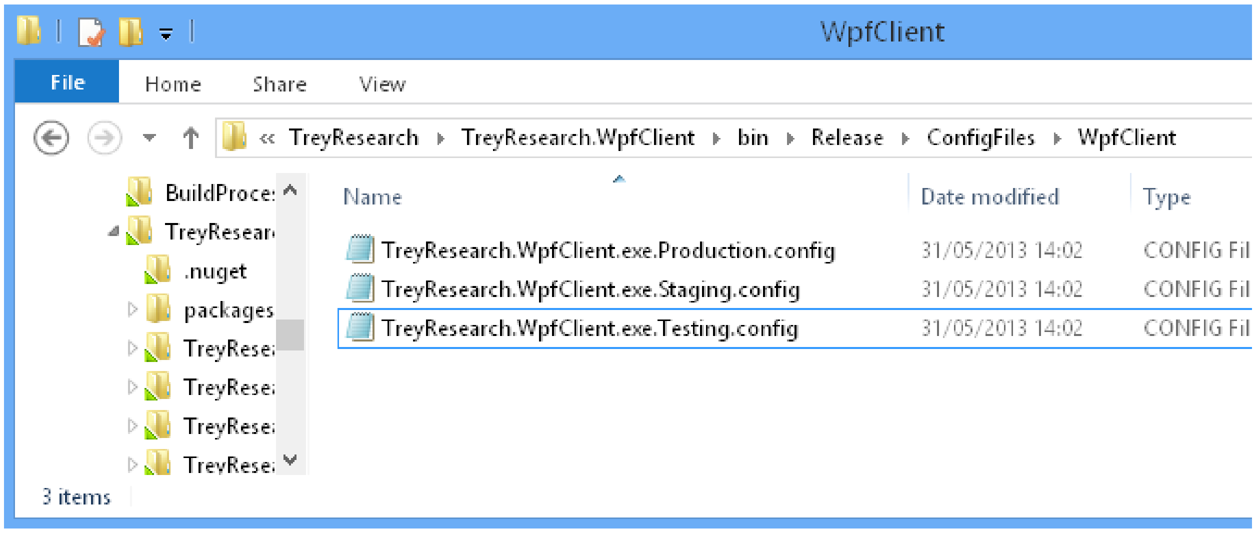
3. From the Windows Start menu (or in Windows 8, the **Start** screen) open a **Developer Command Prompt for VS2012**. Change the directory to the location of **TreyResearch. WpfClient.csproj**.

4. Test to see if the transformations were performed. Run the following command.

CMD

```
msbuild /p:Configuration=Release TreyResearch.WpfClient.csproj
```

5. After MSBuild finishes, you should find a new folder named **ConfigFiles** under the **bin\Release** subfolder. The folder contains the three transformed configuration files. If you open any of them, you'll see that the content has been transformed to match the target environment.



Name	Date modified	Type
TreyResearch.WpfClient.exe.Production.config	31/05/2013 14:02	CONFIG Fil
TreyResearch.WpfClient.exe.Staging.config	31/05/2013 14:02	CONFIG Fil
TreyResearch.WpfClient.exe.Testing.config	31/05/2013 14:02	CONFIG Fil

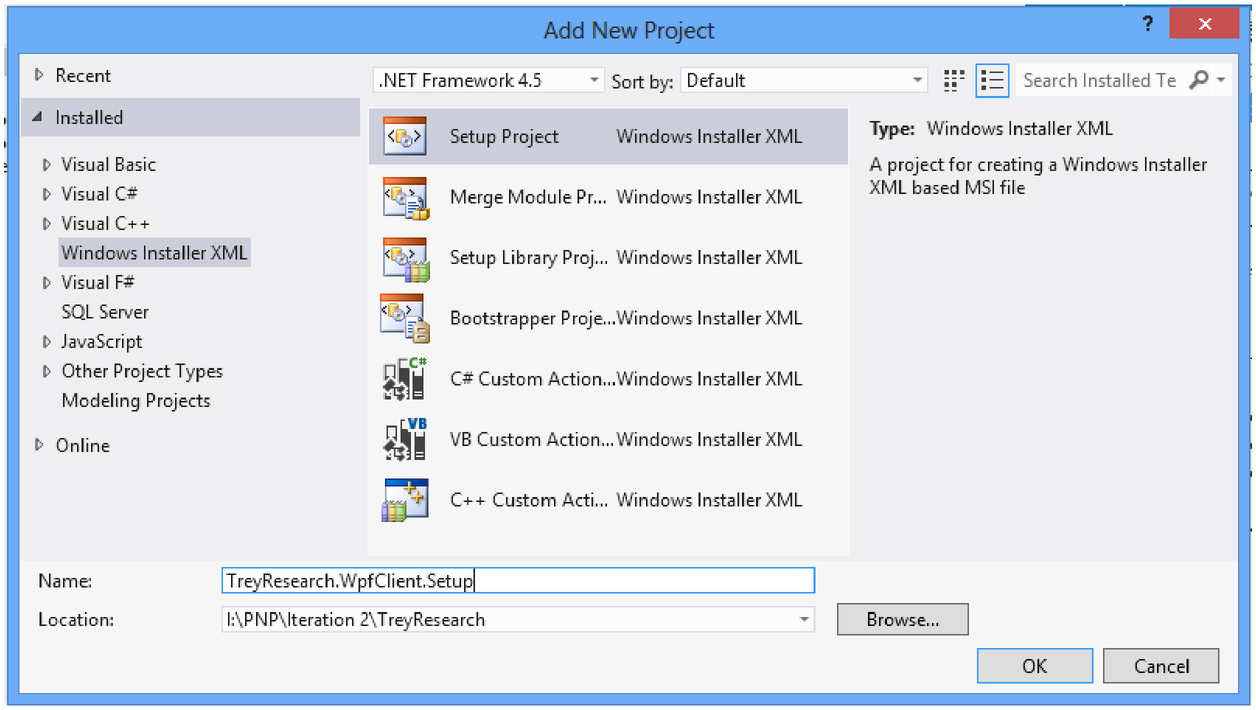
Exercise 3: Deployment Automation: Packaging the Files for Deployment

In this exercise you use the Windows Installer to package the files that will be deployed. There are several tools that generate Windows Installer packages. This lab uses the WiX tool set. WiX allows you to build Windows installation packages from XML source code. WiX supports completely automated installations and uninstallations, and is integrated with Visual Studio and MSBuild. For more information about the Windows Installer, see Windows Installer. For more information about writing WiX files, refer to the WiX tutorial or the WiX manual.

Task 1: Create the WiX Visual Studio Project

In this task you create a WiX Visual Studio project.

1. In Visual Studio, add a new project to the **TreyResearch** solution. In the **Add New Project** dialog box, click **Windows Installer XML**. Click the **Setup Project** template. Name the project **TreyResearch.WpfClient.Setup**.


Add New Project
Recent
Installed
Visual Basic
Visual C#
Visual C++
Windows Installer XML
Visual F#
SQL Server
JavaScript
Other Project Types
Modeling Projects
Online
.NET Framework 4.5
Sort by: Default
Search Installed Te
Setup Project Windows Installer XML
Merge Module Pr... Windows Installer XML
Setup Library Proj... Windows Installer XML
Bootstrapper Proje...Windows Installer XML
C# Custom Action...Windows Installer XML
VB Custom Action...Windows Installer XML
C++ Custom Acti... Windows Installer XML
Browse...
OK
Cancel

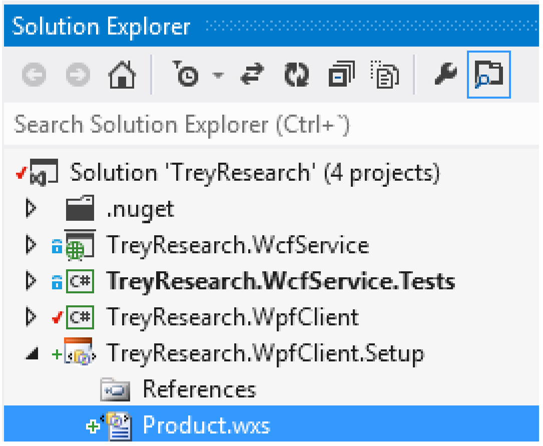
Type: Windows Installer XML

A project for creating a Windows Installer XML based MSI file

Name: TreyResearch.WpfClient.Setup

Location: I:\PNP\Iteration 2\TreyResearch

2. In the **TreyResearch.WpfClient.Setup** project, open the **Product.wxs** file. This file describes the behavior of the Windows Installer.


Solution Explorer
Search Solution Explorer (Ctrl+`)
Solution 'TreyResearch' (4 projects)
.nuget
TreyResearch.WcfService
TreyResearch.WcfService.Tests
TreyResearch.WpfClient
TreyResearch.WpfClient.Setup
References
Product.wxs

3. Replace the existing code with the following code.

XML

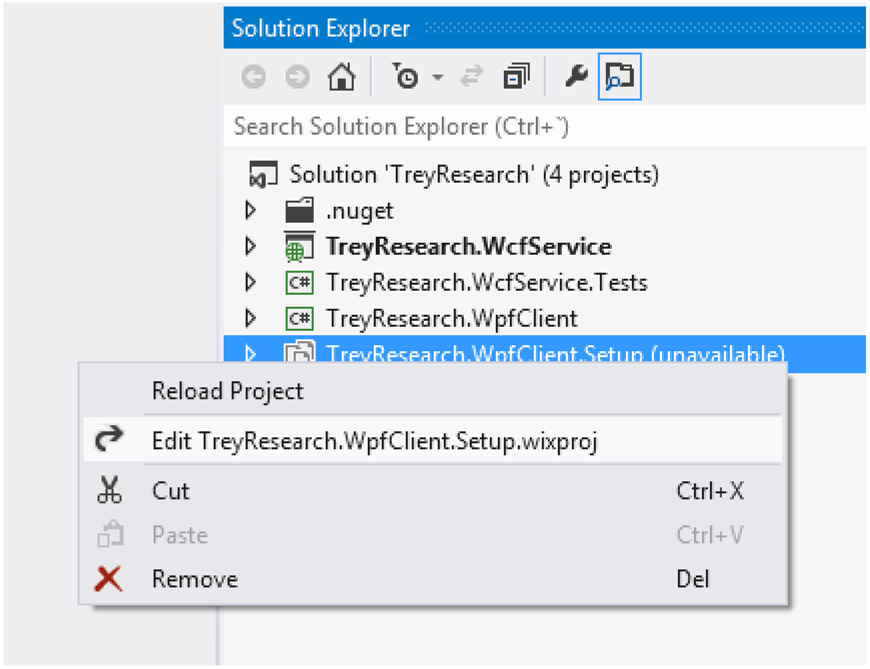
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- AUTOMATION HOL - This is the Product.wxs file that describes how the
Windows Installer package is generated and how it will behave during
deployment. -->

<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
```

```
  <!-- AUTOMATION HOL - For the Product section below, there are a couple of
attributes worth mentioning. -->
    <!-- The UpgradeCode attribute uniquely identifies our product in the
installation database of the target computer for upgrades. It is a GUID that
we must generate and provide. -->
    <!-- It can be generated in Visual Studio: "Tools->Create GUID->Registry
format" (just remove the curly braces). -->
    <!-- The Version attribute relates the installer package to the product
its contains, and allows us to use Windows Installer update features if
needed.  -->
  <Product Id="*" Name="TreyResearch.WpfClient" Language="1033"
           Version="$(var.ProductVersion)" Manufacturer="TreyResearch"
UpgradeCode="8365f624-ffc5-4719-ac25-f403a54e593c">

    <Package Keywords="Installer" Description="TreyResearch.WpfClient Wix
Installer" Comments="Installer package for the TreyResearch.WpfClient
application" Manufacturer="TreyResearch" InstallerVersion="200"
Compressed="yes" />

    <!-- AUTOMATION HOL - The following two sections are added to support
upgrades. -->
    <Upgrade Id="8365f624-ffc5-4719-ac25-f403a54e593c">
      <UpgradeVersion OnlyDetect="yes" Minimum="$(var.ProductVersion)"
Property="NEWERVERSIONDETECTED" IncludeMinimum="no" />
      <UpgradeVersion OnlyDetect="no" Maximum="$(var.ProductVersion)"
Property="OLDERVERSIONBEINGUPGRADED" IncludeMaximum="no" />
    </Upgrade>
    <InstallExecuteSequence>
      <RemoveExistingProducts After="InstallInitialize" />
    </InstallExecuteSequence>

    <MediaTemplate EmbedCab="yes" />

    <!-- AUTOMATION HOL - In the Feature subsection, we describe the
components that are going to be installed on the target machine. -->
      <!--In this case, it is going to be just the product components
(binaries and configuration files), and a couple of shortcuts to help the user
locate and run the application. -->
    <Feature Id="Complete" Title="TreyResearch.WpfClient Complete" Level="1">
      <ComponentGroupRef Id="ProductComponents" />
      <ComponentGroupRef Id="Shortcuts" />
    </Feature>

  </Product>

  <Fragment>
    <!-- AUTOMATION HOL - Here we specify the target directories that will be
modified during the installation. -->
    <Directory Id="TARGETDIR" Name="SourceDir">
```

```
      <Directory Id="ProgramFilesFolder">
        <!-- AUTOMATION HOL - The folder Program Files\TreyResearch.WpfClient 
will be created. -->
        <Directory Id="INSTALLFOLDER" Name="TreyResearch.WpfClient" />
      </Directory>
      <Directory Id="ProgramMenuFolder" Name="Programs">
        <!-- AUTOMATION HOL - The Program Menu folder will be modified to 
receive the TreyResearch.WpfClient shortcut. -->
        <Directory Id="ProgramMenuDir" Name="TreyResearch.WpfClient" />
      </Directory>
      <!-- AUTOMATION HOL - The Desktop will be modified as well, to receive 
the TreyResearch.WpfClient shortcut. -->
      <Directory Id="DesktopFolder" Name="Desktop" />
    </Directory>
  </Fragment>

  <Fragment>
    <ComponentGroup Id="ProductComponents" Directory="INSTALLFOLDER">
      <Component Id="MainExecutable">
        <!-- AUTOMATION HOL - The main executable for the application will be 
copied to INSTALLFOLDER. -->
        <!-- AUTOMATION HOL - The Source attribute is specified by means of a 
couple of parameters, so we can easily provide them during an automated build 
in the build server. -->
        <File Id="TreyResearch.WpfClient.exe" 
Name="TreyResearch.WpfClient.exe" 
Source="$(var.SourcePath)\$(var.MainExecutable)" KeyPath="yes">
          <!-- AUTOMATION HOL - A couple of shortcuts are created for the main 
executable: one in the start menu (star screen in Windows 8) and another one 
in the desktop. -->
          <Shortcut Id='startmenuTreyResearch.WpfClient' 
Directory='ProgramMenuDir' Name='TreyResearch.WpfClient' 
WorkingDirectory='INSTALLDIR' Icon='TreyResearchIcon.exe' Advertise='yes' />
          <Shortcut Id='desktopTreyResearch.WpfClient' 
Directory='DesktopFolder' Name='TreyResearch.WpfClient' 
WorkingDirectory='INSTALLDIR' Icon='TreyResearchIcon.exe' Advertise='yes' />
        </File>
      </Component>
      <Component Id="ConfigFile">
        <!-- AUTOMATION HOL - The configuration file for the application will 
be copied to INSTALLFOLDER. -->
        <!-- AUTOMATION HOL - The Source attribute is specified by means of a 
couple of parameters, so we can easily provide them during an automated build 
in the build server. -->
        <File Id="TreyResearch.WpfClient.exe.config" 
Name="TreyResearch.WpfClient.exe.config" 
Source="$(var.SourcePath)\ConfigFiles\WpfClient\$(var.EnvironmentConfigFile)" 
/>
      </Component>
    </ComponentGroup>
    <ComponentGroup Id='Shortcuts' Directory='ProgramMenuDir'>
      <Component Id="ProgramMenuShortcut">
        <!-- AUTOMATION HOL - These keys allow to remove the application
shortcuts on uninstall. -->
        <RemoveFolder Id='ProgramMenuDir' On='uninstall' />
        <RegistryValue Root='HKCU' Key='Software\[Manufacturer]\[ProductName]'
Type='string' Value='' KeyPath='yes' />
      </Component>
    </ComponentGroup>
  </Fragment>

  <Fragment>
    <!-- AUTOMATION HOL - Here we specify the icon for the application; it is
just the one embedded into the executable. -->
    <Icon Id="TreyResearchIcon.exe"
SourceFile="$(var.SourcePath)\$(var.MainExecutable)" />
  </Fragment>

</Wix>
```

4. Save the file and close it.

In Solution Explorer, right-click on the **TreyResearch.WpfClient.Setup** project. Click **Unload Project**. Click **Edit TreyResearch.WpfClient.Setup.wixproj**.


Solution Explorer
Search Solution Explorer (Ctrl+`)
Solution 'TreyResearch' (4 projects)
.nuget
TreyResearch.WcfService
TreyResearch.WcfService.Tests
TreyResearch.WpfClient
TreyResearch.WpfClient.Setup (unavailable)
Reload Project
Edit TreyResearch.WpfClient.Setup.wixproj
Cut Ctrl+X
Paste Ctrl+V
Remove Del

5. Inside the first **<PropertyGroup>** section just before the **</PropertyGroup>** begins, add the following line of code.

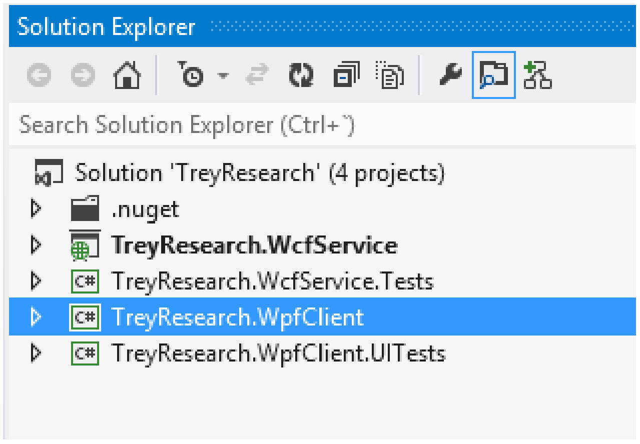
XML

```
    <!-- AUTOMATION HOL - Here we define the parameters that can be passed
when generating the installer. -->
    <!-- ProductVersion: version number for the product and for the installer.
-->
    <!-- SourcePath: root path where the files to be added to the installer
are located.  -->
    <!-- EnvironmentConfigFile: environment-specific configuration file that
will be included in the installer, instead of the default App.config. -->
    <!-- MainExecutable: this is the executable for the WPF application. -->
    <!-- Additionally, we will pass the OutputPath property as a parameter so
we can change the place where the generated installer is copied. But we don't
need to specify it here since it is a regular MSBuild property. -->

<DefineConstants>ProductVersion=$(ProductVersion);SourcePath=$(SourcePath);Env
ironmentConfigFile=$(EnvironmentConfigFile);MainExecutable=$(MainExecutable)</
DefineConstants>
```

This code defines the parameters so that they are available when MSBuild creates the installer, either from the command line or from within the commit stage of the pipeline.

6. Reload the **TreyResearch.WpfClient.Setup.wixproj** project. It opens in Solution Explorer.

7. Remove the **TreyResearch.WpfClient.Setup.wixproj** from the Visual Studio solution. This project does not build inside Visual Studio because it is configured to be built from the command line or by Team Build, and by passing in the necessary parameters. Removing it from the solution avoids confusion and prevents build errors. In Solution Explorer, right-click on the project. Click **Remove**. The following screenshot shows an example of what you should see after you remove the project..

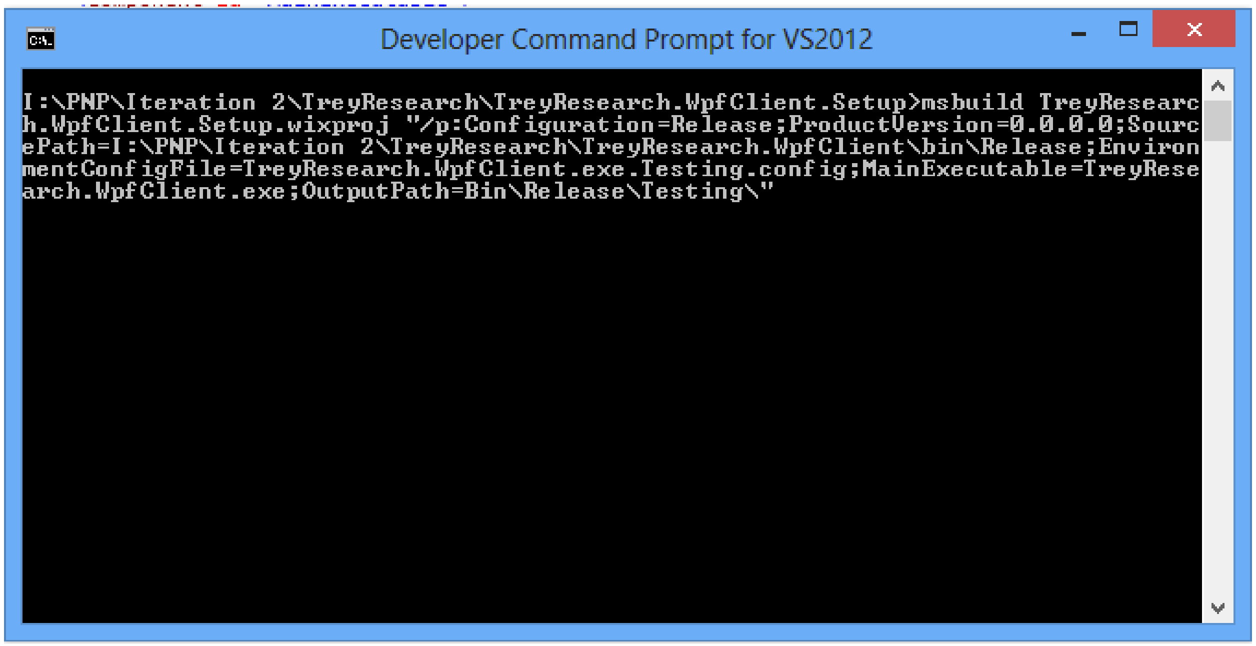

Solution Explorer
Search Solution Explorer (Ctrl+`)
Solution 'TreyResearch' (4 projects)
.nuget
TreyResearch.WcfService
TreyResearch.WcfService.Tests
TreyResearch.WpfClient
TreyResearch.WpfClient.UITests

8. Generate the installer. From the Windows **Start** menu (or in Windows 8, the **Start** screen) open a **Developer Command Prompt for VS2012**. Change the directory to the location of the WiX project. The following command generates the installer for the test environment. Before running the following command, replace the highlighted path with your current working directory for the TreyResearch solution.

CMD

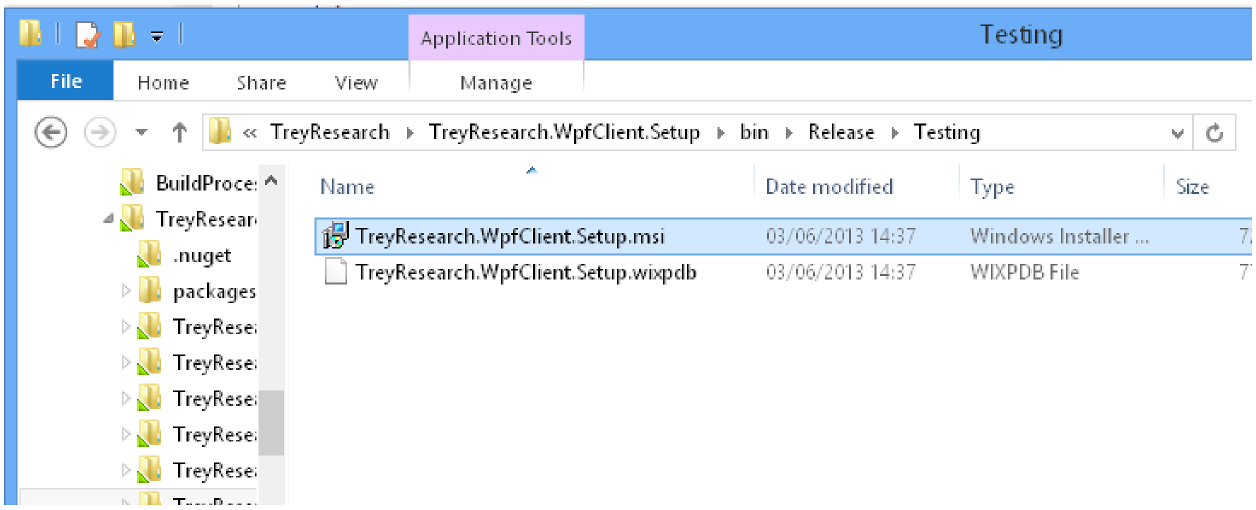
```
msbuild TreyResearch.WpfClient.Setup.wixproj
"/p:Configuration=Release;ProductVersion=0.0.0.0;SourcePath=<Folder containing
the Visual Studio solution
file>\TreyResearch.WpfClient\bin\Release;EnvironmentConfigFile=TreyResearch.Wp
fClient.exe.Testing.config;MainExecutable=TreyResearch.WpfClient.exe;OutputPat
h=Bin\Release\Testing\"
```

The following screenshot shows the command with an example of the source path.



```
I:\PNP\Iteration 2\TreyResearch\TreyResearch.WpfClient.Setup>msbuild TreyResearc
h.WpfClient.Setup.wixproj "/p:Configuration=Release;ProductVersion=0.0.0.0;Sourc
ePath=I:\PNP\Iteration 2\TreyResearch\TreyResearch.WpfClient\bin\Release;Environ
mentConfigFile=TreyResearch.WpfClient.exe.Testing.config;MainExecutable=TreyRese
arch.WpfClient.exe;OutputPath=Bin\Release\Testing\"
```

9. Go the folder you specified in the **OutputPath** parameter. You should find the .msi installer for the test environment. The following screenshot shows an example.



Name	Date modified	Type	Size
TreyResearch.WpfClient.Setup.msi	03/06/2013 14:37	Windows Installer ...	7
TreyResearch.WpfClient.Setup.wixpdb	03/06/2013 14:37	WIXPDB File	7

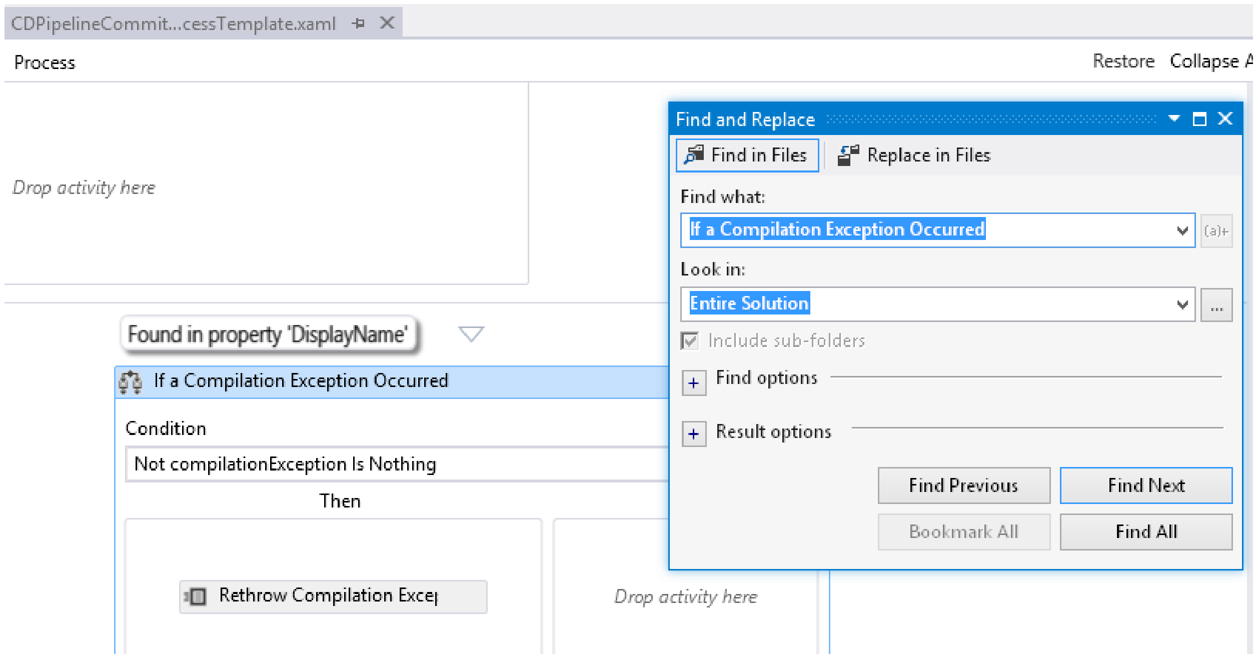
Task 2: Configure the Commit Stage To Generate the Installers

In this task you modify the commit stage's build workflow so that it generates the installers for all of the environments.

NOTE: In the first lab, the commit stage prepares the configuration files and packages for the WCF service. During the deployment, the other stages of the pipeline replace the configuration file with the one corresponding to the target environment. The advantage to this approach is that the commit stage remains short and finishes quickly. A fast commit stage is important because it runs every time there is a check-in, and you want to know as soon as possible if the check-in has caused any problems.

However, for the WPF Windows installer, it is easier to prepare all the installers with the correct configuration files during the commit stage. This removes a dependency on WiX in the later stages of the pipeline.

1. In Visual Studio, open the **TreyResearchBuildCustomization** solution that you created in the Orchestration HOL.
2. Open the **CDPipelineCommitStageProcessTemplate.xaml** file.
3. Open **Find and Replace** (Ctrl + F). Search for **If a Compilation Exception Occurred**. Close **Find and Replace**.



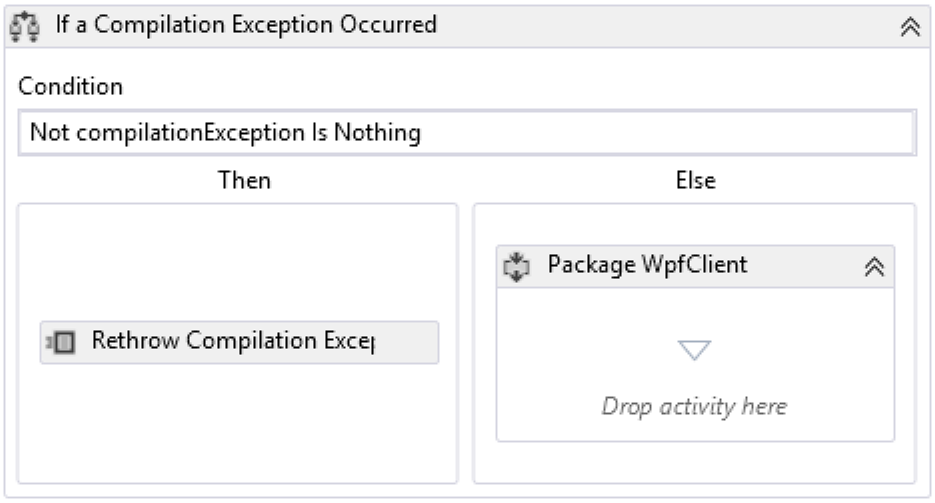
Find what: If a Compilation Exception Occurred

Look in: Entire Solution

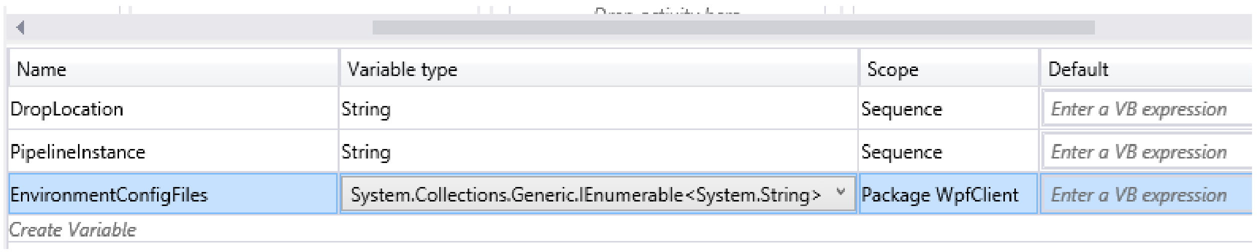
Condition

Not compilationException Is Nothing

4. Add a **Sequence** activity to the **Else** section of the **If a Compilation Exception Occurred** activity and name it **Package WpfClient**. You must perform the packaging after the WpfClient project is compiled, so that the binaries and configuration files are available, but before the results are copied to the drop folder.


If a Compilation Exception Occurred
Condition
Not compilationException Is Nothing
Then
Else
Rethrow Compilation Excep
Package WpfClient
Drop activity here

5. Add a new variable named **EnvironmentConfigFiles**, of type **IEnumerable<String>**, scoped to the **Package WpfClient** sequence. This variable stores the list of environment-specific configuration files.

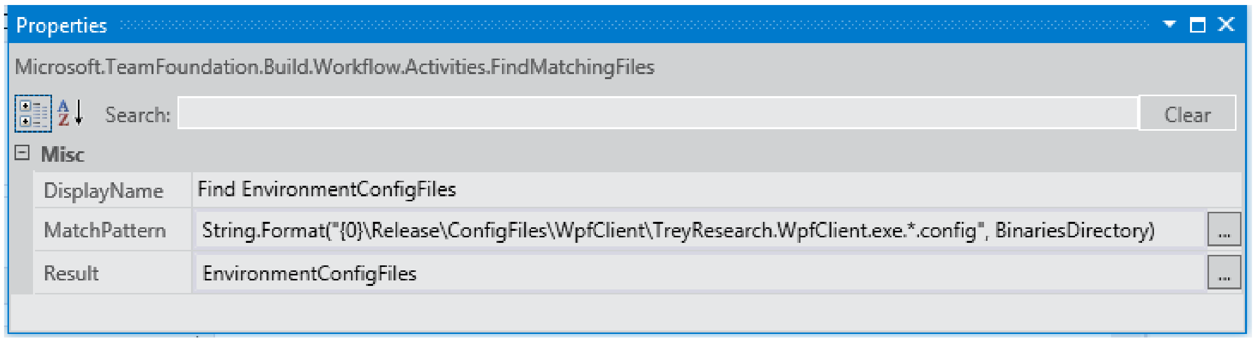


Name	Variable type	Scope	Default
DropLocation	String	Sequence	Enter a VB expression
PipelineInstance	String	Sequence	Enter a VB expression
EnvironmentConfigFiles	System.Collections.Generic.IEnumerable<System.String>	Package WpfClient	Enter a VB expression
Create Variable			

6. Add a **FindMatchingFiles** activity to the sequence and set its properties to the following code.

Visual Basic
```
String.Format("{0}\Release\ConfigFiles\WpfClient\TreyResearch.WpfClient.exe.*.
config",BinariesDirectory)
```

The following screenshot shows the **MatchPattern** property.



Microsoft.TeamFoundation.Build.Workflow.Activities.FindMatchingFiles

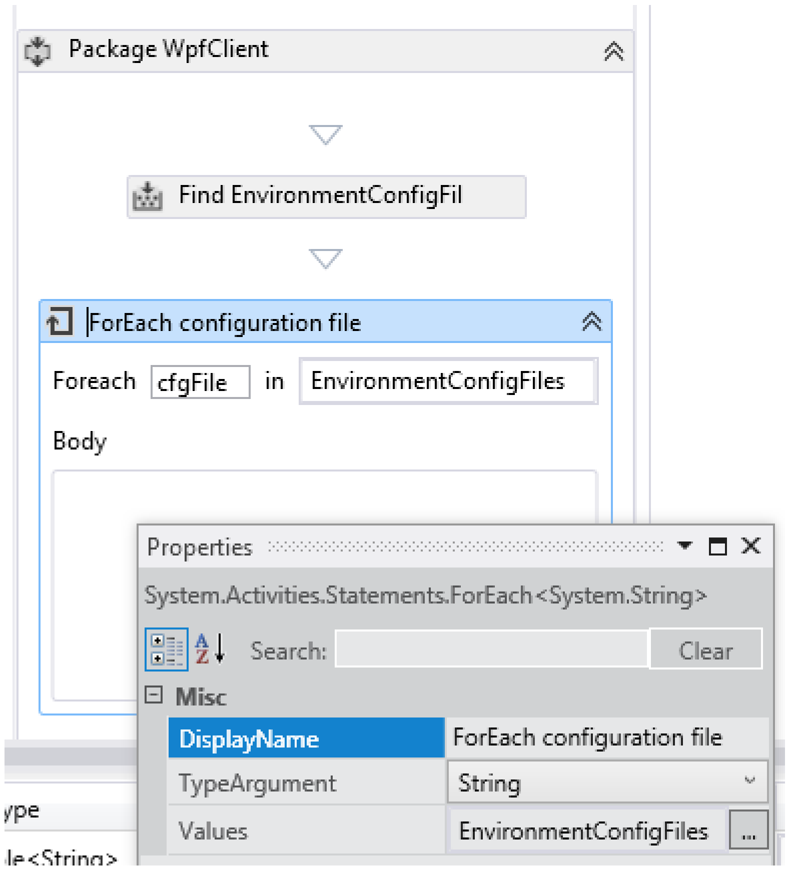
Misc

DisplayName	Find EnvironmentConfigFiles
MatchPattern	String.Format("{0}\Release\ConfigFiles\WpfClient\TreyResearch.WpfClient.exe.*.config", BinariesDirectory)
Result	EnvironmentConfigFiles

The **MatchPattern** property specifies how to search for the configuration files. In this case, the search occurs inside the directory that was specified during Exercise 2, Task 3, step 5.

NOTE: For simplicity, the path and name of the configuration files in the **MatchPattern** are hardcoded. An improvement would be to specify the **MatchPattern** property when defining the build, expose it as an argument and specify its metadata. This is what you did for the arguments in the Orchestration HOL. You could do the same for the rest of the properties in this section, as well.

7. Add a **ForEach** activity and rename it **ForEach configuration file**. This activity iterates through the configuration files. Make sure the properties match what is shown in the following screenshot.

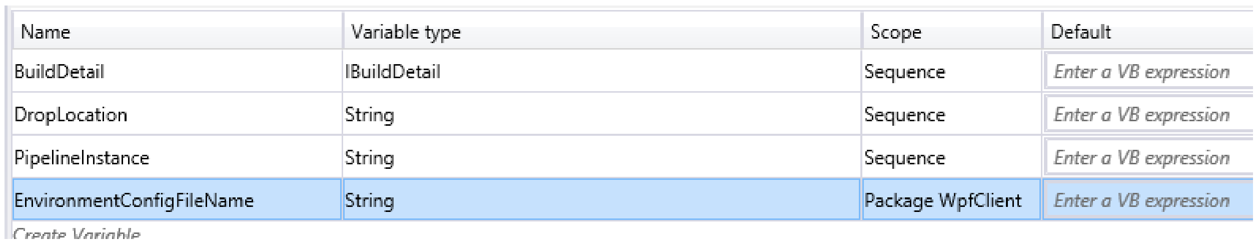


Foreach cfgFile in EnvironmentConfigFiles

System.Activities.Statements.ForEach<System.String>

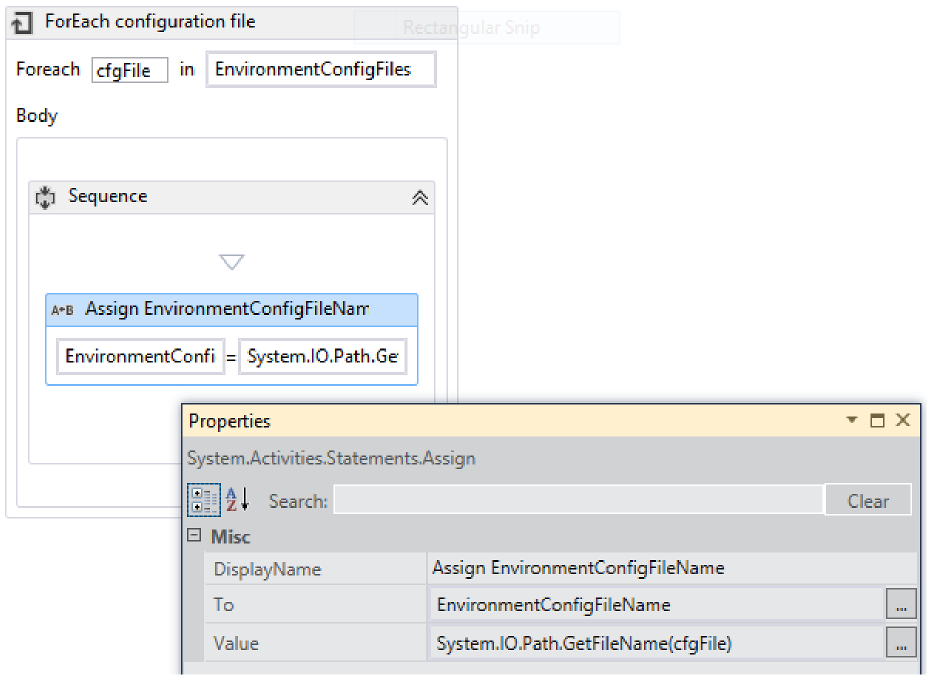
DisplayName	ForEach configuration file
TypeArgument	String
Values	EnvironmentConfigFiles

8. Inside the **Body** of the **ForEach** activity, create a variable named **EnvironmentConfigFileName**, which will store the name of the configuration file.



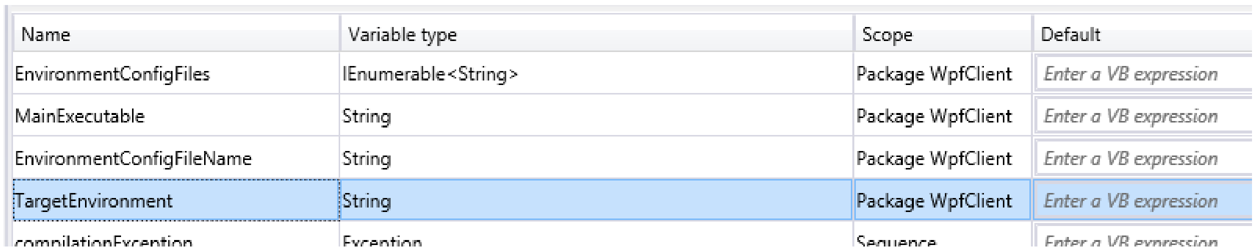
Name	Variable type	Scope	Default
BuildDetail	IBuildDetail	Sequence	Enter a VB expression
DropLocation	String	Sequence	Enter a VB expression
PipelineInstance	String	Sequence	Enter a VB expression
EnvironmentConfigFileName	String	Package WpfClient	Enter a VB expression
Create Variable			

9. Inside the **Body** of the **ForEach** activity, add an **Assign** activity and name it **Assign EnvironmentConfigFileName**. Set the **To** property to **EnvironmentConfigFileName**. Set the **Value** property to **System.IO.Path.GetFileName(cfgFile)**.


ForEach configuration file
Foreach cfgFile in EnvironmentConfigFiles
Body
Sequence
Assign EnvironmentConfigFileNam
EnvironmentConfi = System.IO.Path.Ge
Properties
System.Activities.Statements.Assign
Search:
Clear
Misc

DisplayName	Assign EnvironmentConfigFileName
To	EnvironmentConfigFileName
Value	System.IO.Path.GetFileName(cfgFile)

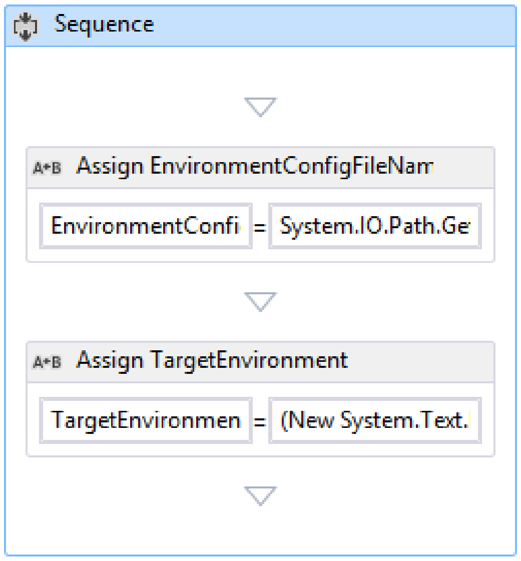
10. Create a variable named **TargetEnvironment**. Set the **Variable type** to **String**. Set the **Scope** to **PackageWpfClient**.



Name	Variable type	Scope	Default
EnvironmentConfigFiles	IEnumerable<String>	Package WpfClient	Enter a VB expression
MainExecutable	String	Package WpfClient	Enter a VB expression
EnvironmentConfigFileName	String	Package WpfClient	Enter a VB expression
TargetEnvironment	String	Package WpfClient	Enter a VB expression
compilationException	Exception	Sequence	Enter a VB expression

The value of the **TargetEnvironment** variable will be the target output path. Each installer will be generated in a subfolder that is named after the environment where the installer will run.

11. Add another **Assign** activity after the **Assign EnvironmentConfigFileName** activity you added in step 9. Name it **Assign TargetEnvironment**.


Sequence
Assign EnvironmentConfigFileNam
EnvironmentConfi = System.IO.Path.Ge
Assign TargetEnvironment
TargetEnvironmen = (New System.Text.

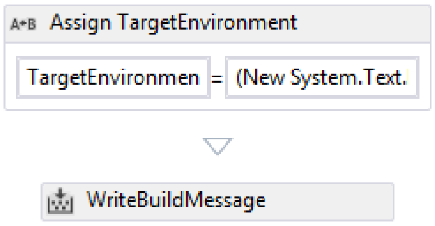
12. Set the **To** property to **TargetEnvironment**. Set the **Value** property to the following code.

Visual Basic

```
(New
System.Text.RegularExpressions.Regex("TreyResearch.WpfClient.exe.(?<TargetEnvi
ronment>.+).config")).Match(EnvironmentConfigFileName).Groups("TargetEnvironme
nt").Value
```

This code retrieves the name of the environment from the configuration file name.

13. Add a **WriteBuildMessage** activity after the **Assign** activity in order to perform logging.



14. Set the **WriteBuildMessage Importance** property.

Visual Basic

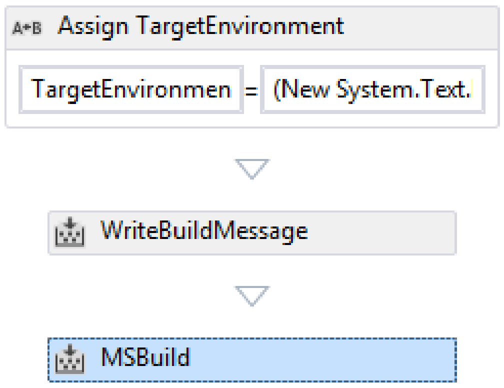
```
Microsoft.TeamFoundation.Build.Client.BuildMessageImportance.High
```

15. Set the **WriteBuildMessage Message** property.

Visual Basic

```
String.Format("Generating the installer for {0} configuration file, {1}
environment", EnvironmentConfigFileName, TargetEnvironment)
```

16. Add an **MSBuild** activity. This activity invokes MSBuild to generate the installers.


A+B Assign TargetEnvironment
TargetEnvironmen = (New System.Text.
WriteBuildMessage
MSBuild

17. Set the **CommandLineArguments** property to the following code.

Visual Basic

```
String.Format("/p:Configuration=Release;ProductVersion={0};SourcePath=""{1}\Re
lease"";EnvironmentConfigFile=""{2}"";MainExecutable=TreyResearch.WpfClient.ex
e;OutputPath=""{3}\Release\Deployment\WpfClient\Installers\{4}\\""",
PipelineInstance, BinariesDirectory, EnvironmentConfigFileName,
BinariesDirectory, TargetEnvironment)
```

18. Set the **Project** property to the following code.

Visual Basic

```
String.Format("{0}\TreyResearch.WpfClient.Setup\TreyResearch.WpfClient.Setup.w
ixproj", SourcesDirectory)
```

19. Set the **Targets** property to the following code.

Visual Basic

```
New String() {"Rebuild"}
```

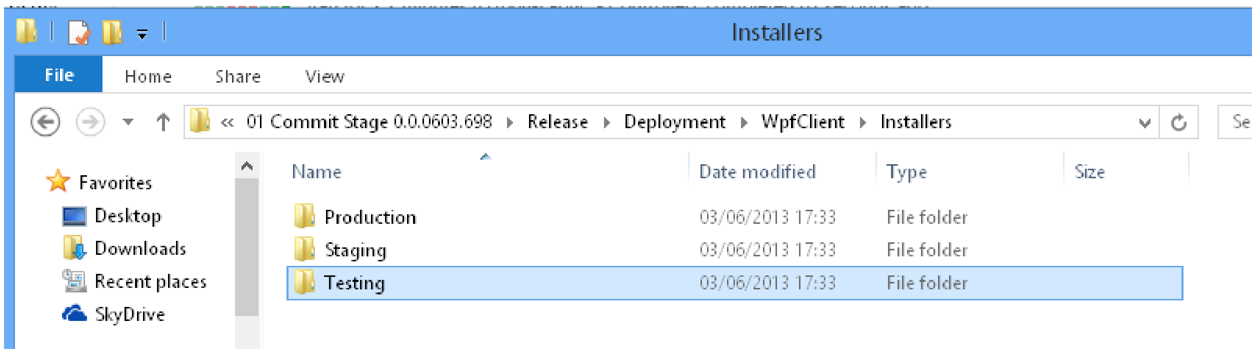
This code ensures that the installers are generated each time MSBuild is called from within the **ForEach configuration file** loop.

Task 3: Test the Configuration Transforms and Package Generation

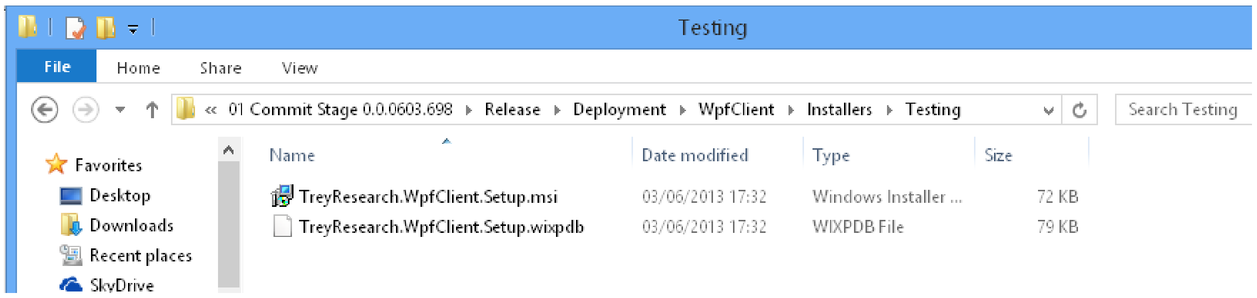
In this task you test to see if the configuration files are transformed correctly and if the install packages are generated correctly.

1. Make sure that the modified build process template, **CDPipelineCommitStageProcessTemplate.xaml**, along with all the other changes you've made, are checked into version control. These changes include the WiX Visual Studio project and the changes you made to the WpfClient project.

2. The commit stage should be triggered automatically. After it finishes successfully, open the drop folder. You should see a subfolder for each environment. Each subfolder should contain a different .msi installer for each environment.


Installers
File
Home
Share
View
« 01 Commit Stage 0.0.0603.698 ▸ Release ▸ Deployment ▸ WpfClient ▸ Installers
Favorites
Desktop
Downloads
Recent places
SkyDrive

Name	Date modified	Type	Size
Production	03/06/2013 17:33	File folder	
Staging	03/06/2013 17:33	File folder	
Testing	03/06/2013 17:33	File folder	


Testing
File
Home
Share
View
« 01 Commit Stage 0.0.0603.698 ▸ Release ▸ Deployment ▸ WpfClient ▸ Installers ▸ Testing
Search Testing
Favorites
Desktop
Downloads
Recent places
SkyDrive

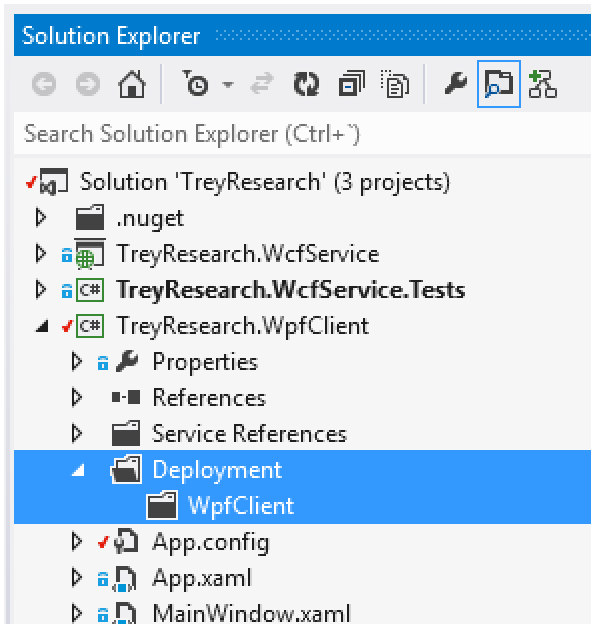
Name	Date modified	Type	Size
TreyResearch.WpfClient.Setup.msi	03/06/2013 17:32	Windows Installer ...	72 KB
TreyResearch.WpfClient.Setup.wixpdb	03/06/2013 17:32	WIXPDB File	79 KB

Exercise 4: Performing the Automated Deployment

In this exercise you set up the pipeline so that it can automatically deploy the installer packages to the environments.

Task 1: Create the Deployment Script for The Agent

1. In Solution Explorer, under the **TreyResearch.WpfClient** project, create a new folder named **Deployment**. Create a subfolder named **WpfClient**.



2. Add a new script file to the **WpfClient** subfolder. Right-click on the subfolder and add a new item of type **Text File**. Change the name to **DeployWpfClient.cmd**.

3. Open the **DeployWpfClient.cmd** file properties. Set **Build Action** to **None**, and **Copy to Output Directory** to **Copy always**. This means that, when the project is built, the script will be copied to the output directory, and in turn to the drop location, so that the deployment agent can retrieve and run it.

4. Open the **DeployWpfClient.cmd** file. Paste the following code into it.

XML

```
REM AUTOMATION HOL - This script copies the msi installer for the new version
and environment, and runs it

REM AUTOMATION HOL - Path where the deployment package is stored (the Drop
location for the pipeline instance)
set packageLocation=%~1
REM AUTOMATION HOL - Name of the environment where the deployment is made.
This is used to pick the corresponding installer from the available ones
set environment=%~2
REM AUTOMATION HOL - Name of the temporal folder to use in the target computer
set tempfolder=%~3

REM AUTOMATION HOL - Preparing the local temp directory to copy the installer
and run the deployment and store log files
set deploymentLocation=%tempfolder%\WpfClient\%environment%
if exist "%deploymentLocation%" rd /s /q "%deploymentLocation%"
mkdir "%deploymentLocation%"
```

```
REM AUTOMATION HOL - Copying the msi installer to the deployment location (the
one corresponding to the target environment).
copy
"%packageLocation%\Release\Deployment\WpfClient\Installers\%environment%\TreyR
esearch.WpfClient.Setup.msi" "%deploymentLocation%"

REM AUTOMATION HOL - Running the deployment
msiexec.exe /i "%deploymentLocation%\TreyResearch.WpfClient.Setup.msi" /Lv*
"%deploymentLocation%\install.log" /passive ALLUSERS="1"
```

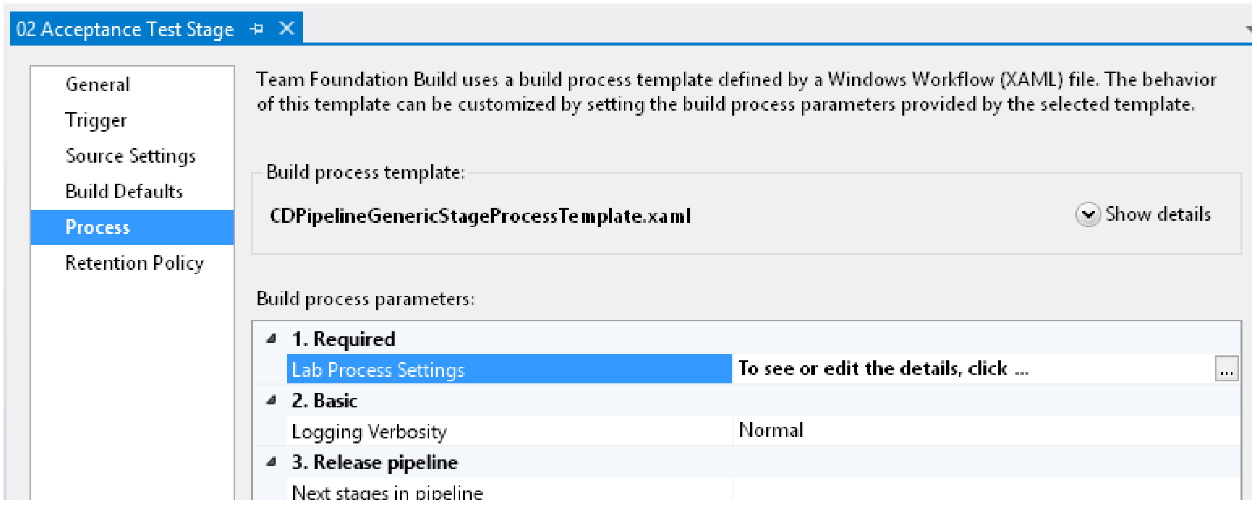
The script copies the installer from the drop location and runs the installer with the correct parameters. If there is a previous version of the application already installed, it is automatically updated by the installer.

5. Save the **.cmd** file using the encoding **UTF-8 without signature**. Select **Save As** from the Visual Studio **File** menu. Select the **UTF-8 without signature** option from the drop-down menu. Ignore the source control warning that. Save the file and check it in.

Task 2: Configure the Pipeline Stages to run the Deployment script

In this task you configure the remaining stages of the pipeline to run the deployment script.

1. Edit the **02 Acceptance Test Stage** build definition so that the stage deploys to the test environment. Under the **Process** tab, open the **Lab Process Settings**. The **Lab Workflow Parameters** dialog box opens.



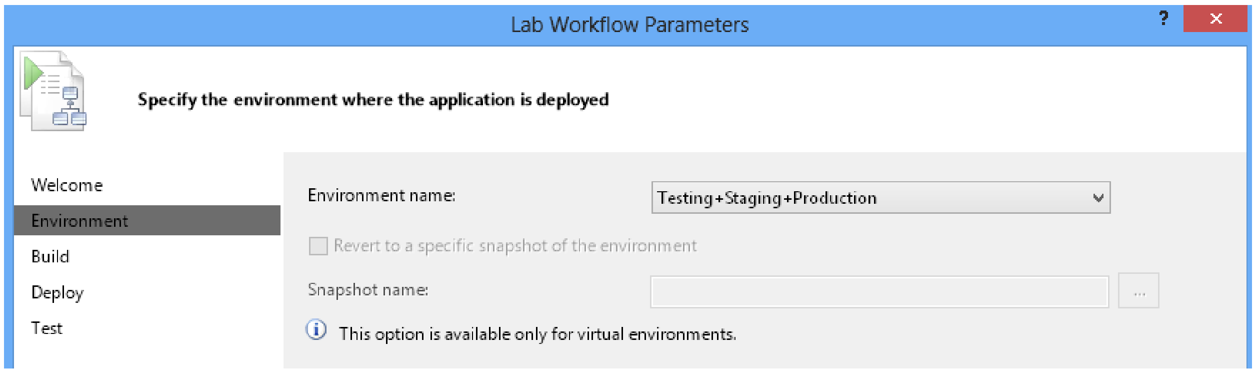
Team Foundation Build uses a build process template defined by a Windows Workflow (XAML) file. The behavior of this template can be customized by setting the build process parameters provided by the selected template.

Build process template:
CDPipelineGenericStageProcessTemplate.xaml

Build process parameters:

1. Required	
Lab Process Settings	To see or edit the details, click ...
2. Basic	
Logging Verbosity	Normal
3. Release pipeline	
Next stages in pipeline	

2. Under the **Environment** tab, select the environment you used in Lab3_1.


Lab Workflow Parameters

Specify the environment where the application is deployed

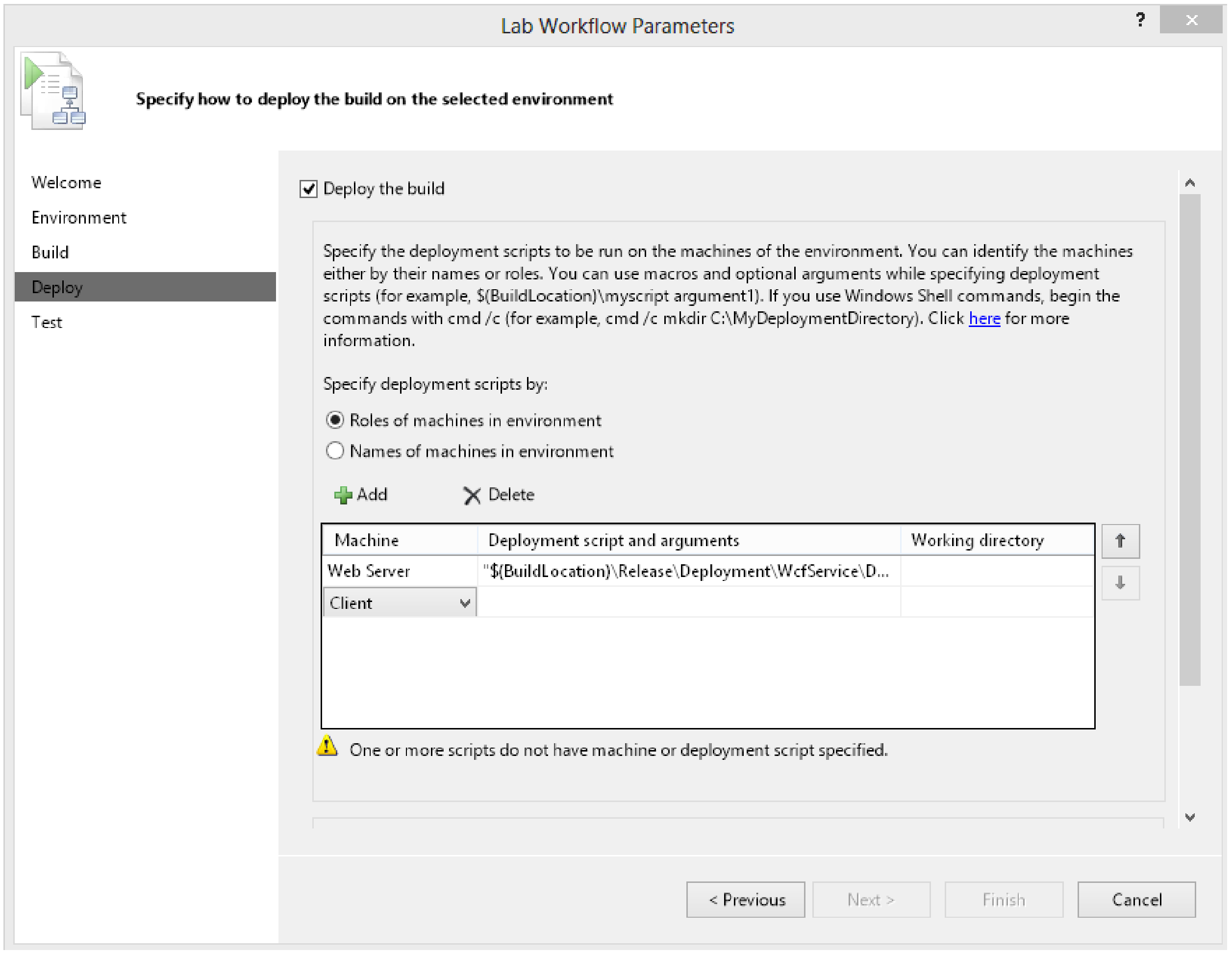
Environment name: Testing+Staging+Production

Revert to a specific snapshot of the environment

Snapshot name:

This option is available only for virtual environments.

3. Select the **Deploy tab**. Select **Deploy the build**. Click **Add**. This incorporates the deployment script into the build definition.
4. Select **Client** from the **Machine** drop-down list.


Lab Workflow Parameters

Specify how to deploy the build on the selected environment

[✓] Deploy the build

Specify the deployment scripts to be run on the machines of the environment. You can identify the machines either by their names or roles. You can use macros and optional arguments while specifying deployment scripts (for example, $(BuildLocation)\myscript argument1). If you use Windows Shell commands, begin the commands with cmd /c (for example, cmd /c mkdir C:\MyDeploymentDirectory). Click here for more information.

Specify deployment scripts by:

● Roles of machines in environment

○ Names of machines in environment

Machine	Deployment script and arguments	Working directory
Web Server	"$(BuildLocation)\Release\Deployment\WcfService\D...	
Client		

One or more scripts do not have machine or deployment script specified.

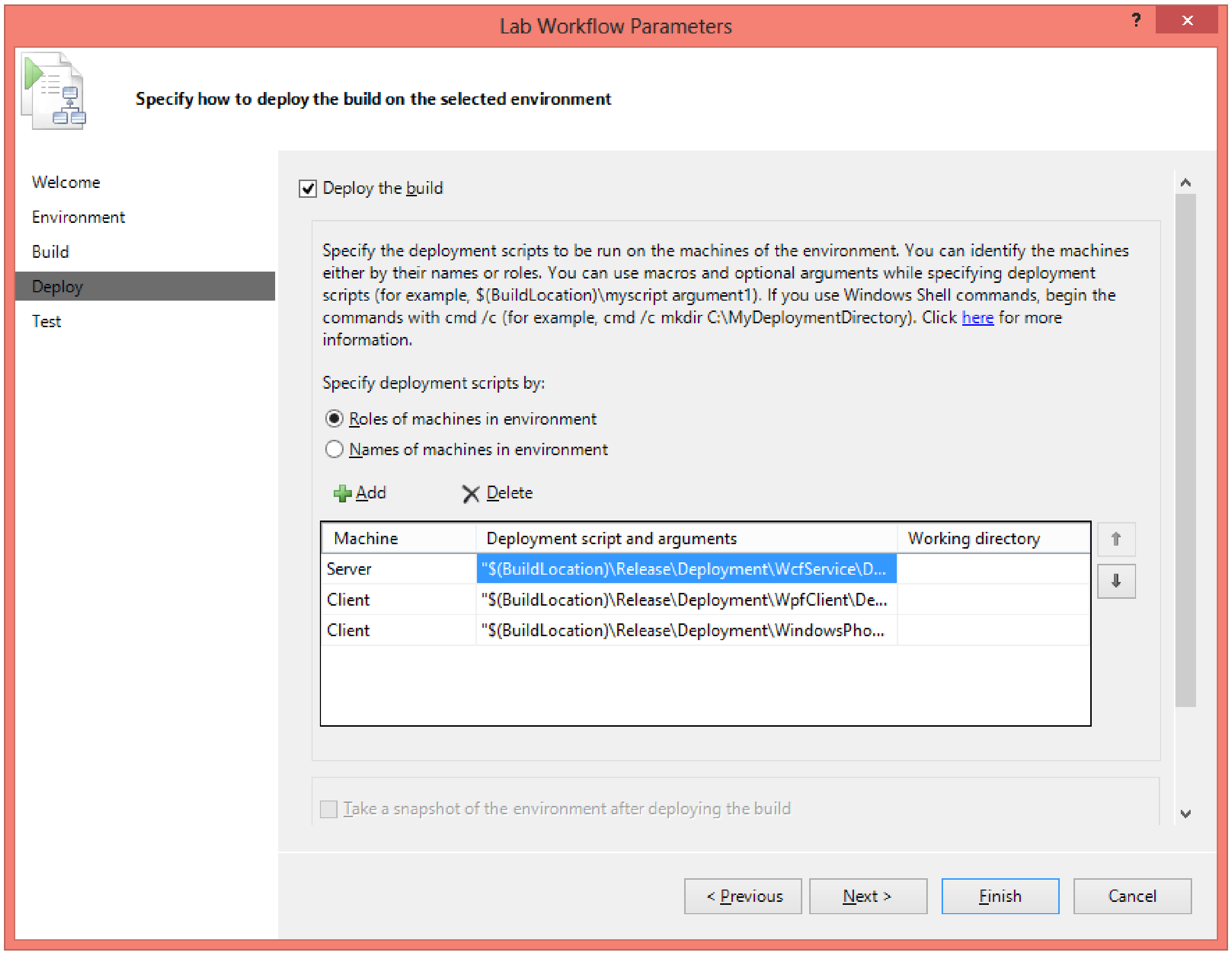
5. In the **Deployment script and arguments** column, add the following command line, which causes the deployment script to execute.

CMD

```
"$(BuildLocation)\Release\Deployment\WpfClient\DeployWpfClient.cmd"
"$(BuildLocation)" Testing C:\TreyResearchDeployment
```

6. Click **Finish** and save the changes.

7. Repeat steps 1 through 6 for **03a Release Stage** and **03b UAT Stage**. In the fifth step, make sure that the second parameter passed to the deployment script is **Production** for **03a Release Stage** and **Staging** for **3b UAT Stage**.



[✓] Deploy the build

Specify the deployment scripts to be run on the machines of the environment. You can identify the machines either by their names or roles. You can use macros and optional arguments while specifying deployment scripts (for example, $(BuildLocation)\myscript argument1). If you use Windows Shell commands, begin the commands with cmd /c (for example, cmd /c mkdir C:\MyDeploymentDirectory). Click here for more information.

Specify deployment scripts by:

[● Roles of machines in environment]
○ Names of machines in environment

Machine	Deployment script and arguments	Working directory
Server	"$(BuildLocation)\Release\Deployment\WcfService\D...	
Client	"$(BuildLocation)\Release\Deployment\WpfClient\De...	
Client	"$(BuildLocation)\Release\Deployment\WindowsPho...	

☐ Take a snapshot of the environment after deploying the build

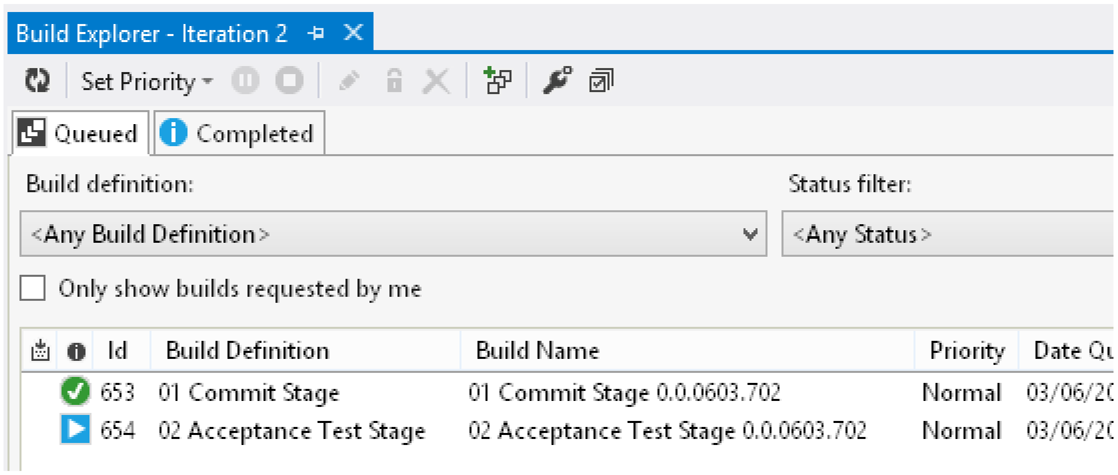
Note: The last client entry is for the Windows Phone 8 componet. You will only see this if you perform the advanced labs.

Exercise 5: Testing the Deployment

In this exercise you test to see if the automated deployment works correctly.

Task 1: Test the Deployment

1. Create a new instance of the pipeline by running the **01 Commit Stage**. You can either make some changes in the code and check them in, or, in Team Explorer, queue the build definition.
2. After the commit stage finishes successfully, the changes are propagated through the pipeline, so **02 Acceptance Test Stage** is triggered automatically. You can monitor the progress from the Build Explorer window.



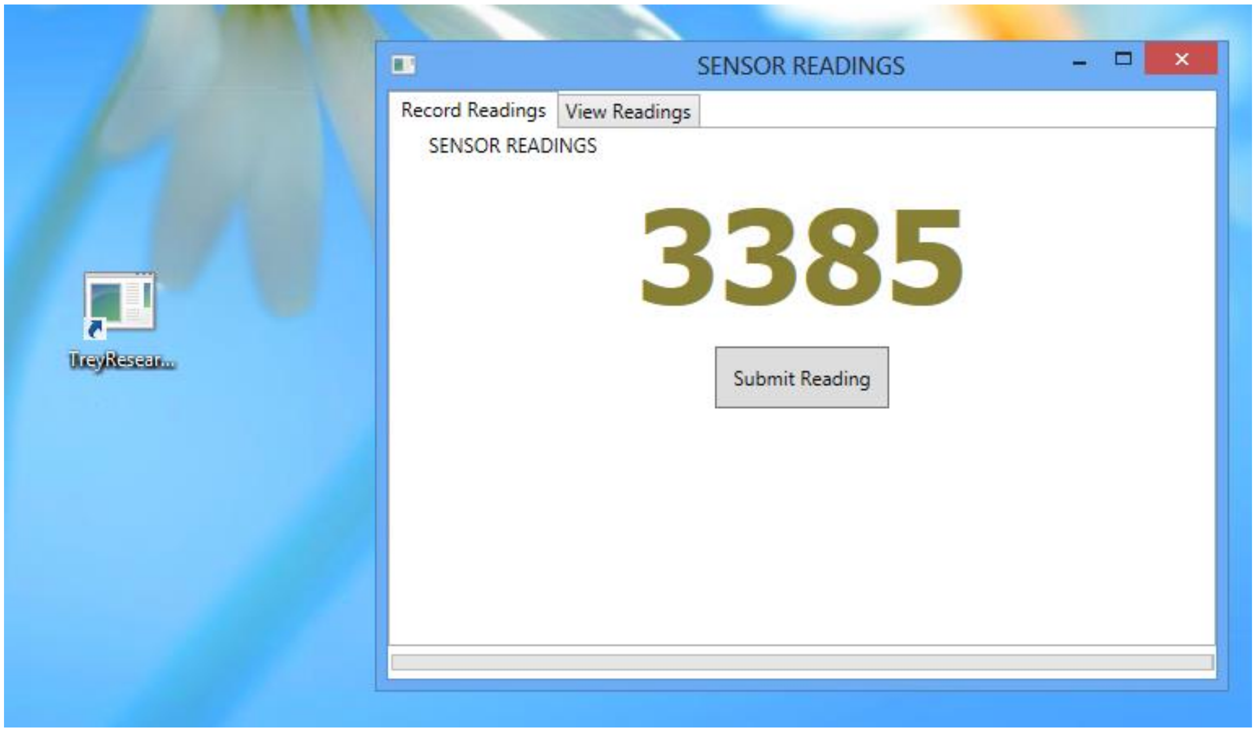
Build definition: <Any Build Definition>

Status filter: <Any Status>

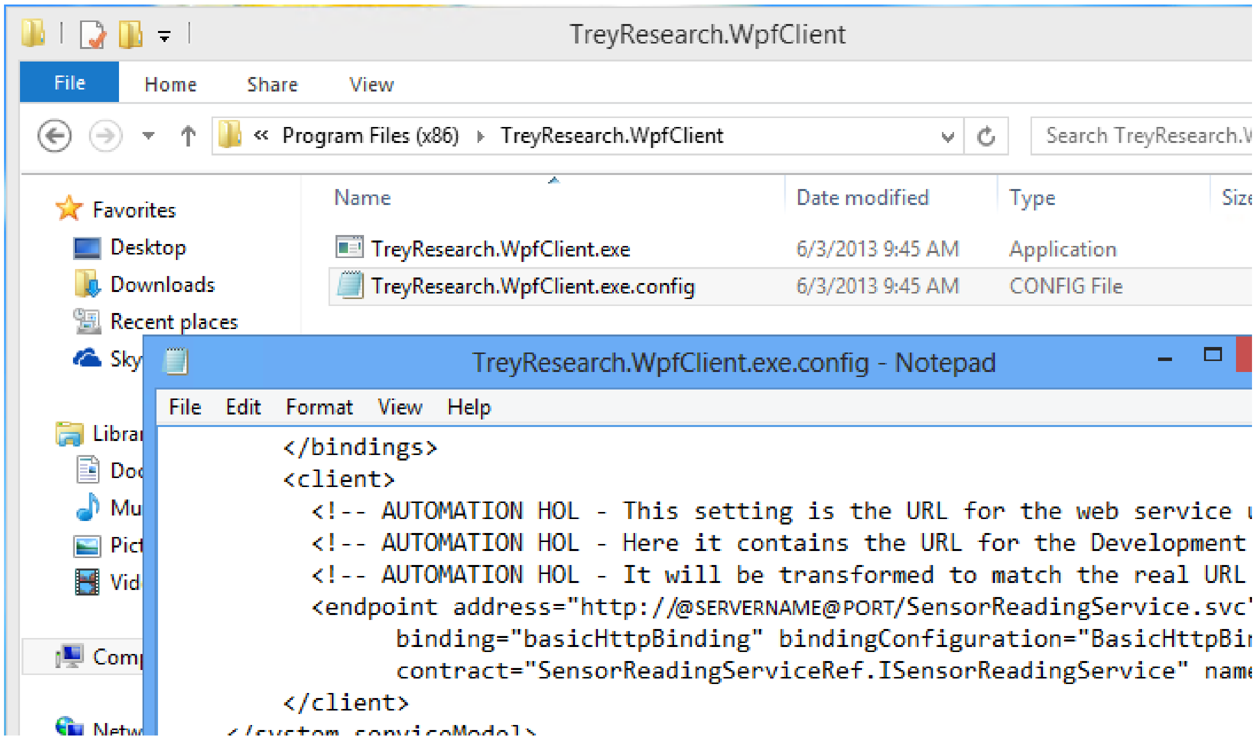
Only show builds requested by me

Id	Build Definition	Build Name	Priority	Date Qu
653	01 Commit Stage	01 Commit Stage 0.0.0603.702	Normal	03/06/20
654	02 Acceptance Test Stage	02 Acceptance Test Stage 0.0.0603.702	Normal	03/06/20

3. When the commit stage finishes, all the artifacts required for the deployment (the package, the configuration files and the scripts) have been generated and copied to the drop folder. When the acceptance stage finishes, you can log in to the target computer, go to the test environment, and run the WPF Client application from the shortcuts that have been created. The following screenshot is an example of what you should see.



4. You can check that the application is using the web service URL for the test environment by opening the **TreyResearch.WpfClient.exe.config** file that is located in **C:\Program Files (x86)\TreyResearch.WpfClient**.



Name	Date modified	Type	Size
TreyResearch.WpfClient.exe	6/3/2013 9:45 AM	Application	
TreyResearch.WpfClient.exe.config	6/3/2013 9:45 AM	CONFIG File	

```
        </bindings>
        <client>
          <!-- AUTOMATION HOL - This setting is the URL for the web service u
          <!-- AUTOMATION HOL - Here it contains the URL for the Development
          <!-- AUTOMATION HOL - It will be transformed to match the real URL
          <endpoint address="http://@SERVERNAME@PORT/SensorReadingService.svc"
                binding="basicHttpBinding" bindingConfiguration="BasicHttpBi
                contract="SensorReadingServiceRef.ISensorReadingService" name
        </client>
    </system.serviceModel>
```

5. At this point, if you want to perform UAT, queue the **03b UAT Stage** build definition and provide the values for the pipeline instance and the drop location. The WPF application will be automatically deployed to the staging environment.
6. You can check that the WPF application is using the web service URL that corresponds to the staging environment by opening the **TreyResearch.WpfClient.exe.config** file that is located in **C:\Program Files (x86)\TreyResearch.WpfClient**.
7. When you are ready to release the WPF application to production, trigger **03a Release Stage**.

Summary

In this lab you automated the deployment of the WPF application. To do this, you installed the WiX tool set. You were able to use the same Lab Management environment that you used in the previous labs.

Next, you prepared the App.config files that set up the environments. There is a base configuration file and three other files that store the differences between the base configuration and the environments. You made those three files dependent on the base file. Finally, you added the code that performs the transformations.

You then packaged all the files required for the deployment, using the WiX tool set to create the Windows install package. To do this, you created a WiX project in Visual Studio. Within that project, you created the .wxs file that describes the behavior of the Windows Installer.

The next step was to configure the commit stage of the pipeline to generate the installers for all the environments. You did this by modifying the build workflow. You tested to see that the configuration files and installers were generated correctly by triggering a build and examining the newly created subfolders

You then configured the pipeline to perform the deployment automatically by creating a Lab Management deployment script. Then, you edited the build definitions for the pipeline stages (other than the commit stage) to deploy to the correct environment. Finally, you tested the automated deployment.

Copyright