Hands-on Lab 6.2: (Advanced) Automating the Deployment of the Windows Phone 8 App

Table of Contents

Objectives

This HOL demonstrates how to automatically deploy the Trey Research Windows Phone 8 app to the target environments. To enable automatic deployment you must:

- Set up the target environments.
- Prepare the configuration files to match the target environments.
- Package the files that are required to deploy and run the Windows 8 Phone application as well as the configuration files.
- Deploy the Windows Phone 8 app to the target environments.

This HOL is part two of the two part AdvancedHOL. (The subject of automation is covered in Chapter 4 of Building a Release Pipeline with Team Foundation Server 2012.)

> **Note:** You cannot use the Brian Kellar VM with this lab.

Prerequisites

The prerequisite for completing this lab is to have completed the previous advanced lab.

> **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 used in the previous lab is your starting point for this lab, it should be 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, which is built out in Lab 3.1. Although this lab is about the Windows Phone 8 app, it doesn't change anything in terms of the workflow of the release pipeline.

You run the exercises for this lab on your local computer and on the computer(s) that act as the environments.

Time

This HOL 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 Windows Phone 8 emulator and so that they can be managed by the pipeline. The emulator is not mandatory. You can also use a physical Windows Phone 8 device plugged into the target machine. The procedure is largely identical.

The pipeline has four environments: development, testing, staging, and production. Because the development environment is isolated and only exists on development machines, it doesn't host an emulator. However, the three other environments each need a Windows Phone 8 emulator in order to deploy and run the Windows Phone 8 application.

Typically, these three emulators would reside on three different computers. For simplicity, this lab uses a single machine with one emulator.

Task 1: Install the Windows Phone 8 Emulator

In this task you install the Windows Phone 8 emulator on the computer that hosts the target environments. There are two prerequisites.

- The emulator requires Windows 8 or Windows Server 2012.
- You cannot use a virtual machine (VM) because the emulator runs on Hyper-V itself.

Download the Windows Phone 8 SDK and install it. The SDK includes the emulator.

Task 2: Set Up the Development Environment

In this task you install the Windows Phone 8 SDK on all the development machines, as well as the build machine(s) that host the build agent(s) that run the commit stage of the pipeline. There are two considerations:

- The build agents must be installed on a Windows 8 or Windows Server 2012 machine.
- A VM will work if you plan to use it only for building the application and not for running it inside the emulator.

Download the Windows Phone 8 SDK at http://developer.windowsphone.com/en-us/downloadsdk and install it.

Task 3: Set Up the Environment in Lab Managment

In this task you add the computer that hosts the Windows Phone 8 emulator to a Lab Management environment. The pipeline uses Lab Management to manage the environments and for automation.

Before you begin this task, make sure that there is a running test controller that is configured for Lab Management.

1. Open **Microsoft Test Manager**.
2. Select the team project that contains the build definitions for the pipeline orchestration. Select **Connect Now**.
3. Click **Connect to Lab**.
4. In the **Environments** dialog box, you should see the environment that you set up in Lab 3.1-Automating the Deployment of the WCF Service. Select it. Click **Next**.
5. In the **Machines** tab, select **Add Machine** to add the computer that will run the emulator to the environment.
6. In the **Computer name** field, provide either the NetBIOS or DNS name.
7. In the **Type Role** field enter **Client**. (You use a generic name here because the same computer is used for the WPF application.)
8. The **User name and Password** fields should already be filled in.
9. Click **Verify**.
10. If the verification succeeds, click **Finish**.
11. After a few minutes the agents are installed on the target machines and the environment is available to the pipeline.

Exercise 2: Preparing the Configuration Files

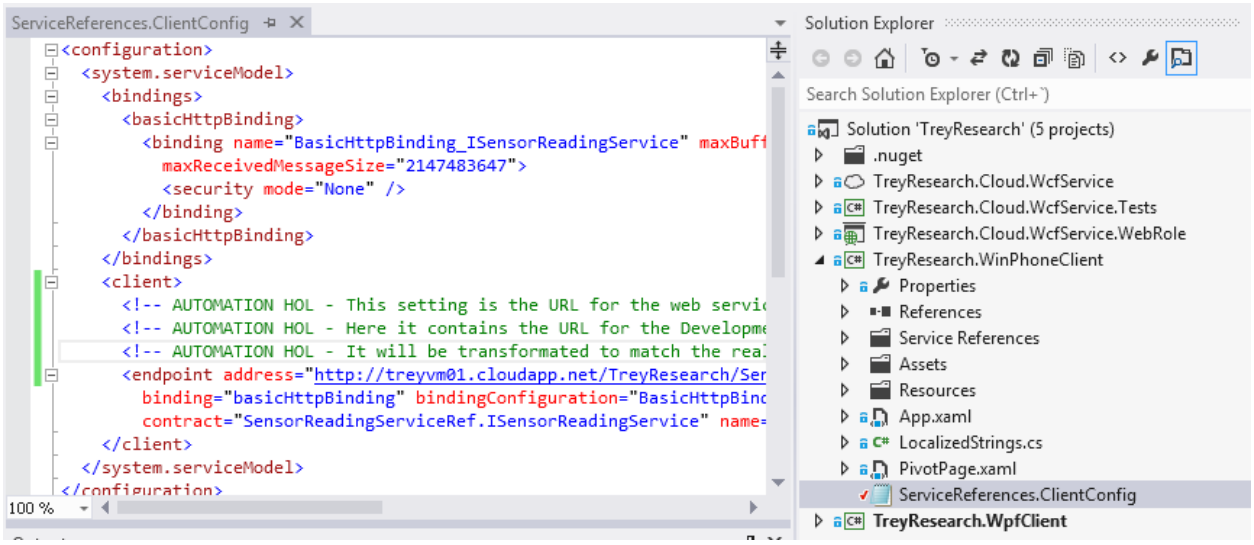
In this exercise you automate the process of creating configuration files that conform to a specific environment. In particular, the ServiceReferences.ClientConfig file for the Windows Phone 8 app uses a different URL in each environment to access the web services.

Task 1: Prepare the ServiceReferences.ClientConfig Files

In this task you add the ServiceReferences.ClientConfig files to the project. The pipeline uses these files when it deploys the Windows Phone 8 app 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 **TreyResearchWinPhoneClient**, find the **ServiceReferences.ClientConfig** file.



3. Use the Text File template to add three additional configuration files. Name each one after the target environment, using the format **ServiceReferences.<Environment Name>.ClientConfig**. For example, the file for the production environment is named ServiceReferences.Production.ClientConfig.

4. Add the following XML code to each file. Replace the **<IIS-Server>** tag with the name of the IIS server that you used in Lab 3.1.

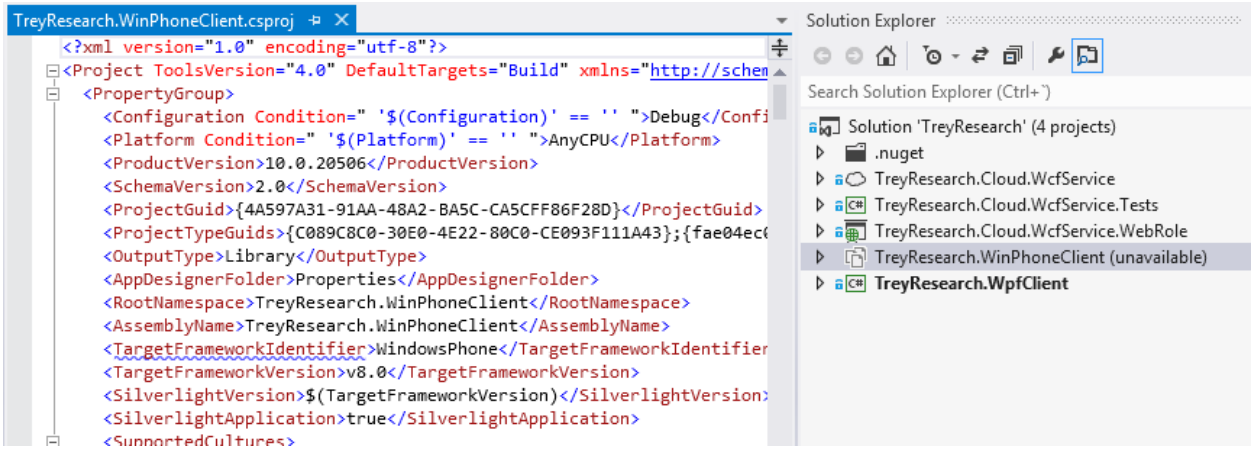
XML

```
<?xml version="1.0" encoding="utf-8"?>
<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>:9000/SensorReadingService.svc"
        name="BasicHttpBinding_ISensorReadingService"
            xdt:Transform="SetAttributes(address)" xdt:Locator="Match(name)"
/>
    </client>
  </system.serviceModel>
</configuration>
```

Task 2: Set the Dependency on the Base File

In this task you make all the **ServiceReferences.<Environment Name>.ClientConfig** files dependent on the base configuration file, **ServiceReferences.ClientConfig**. 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. WinPhoneClient** project and select **Unload project**.
2. Right-click on the project again and select **Edit TreyResearch.WinPhoneClient.csproj**. The MSBuild code that makes up the csproj project file appears.



3. Locate the following XML code.

XML

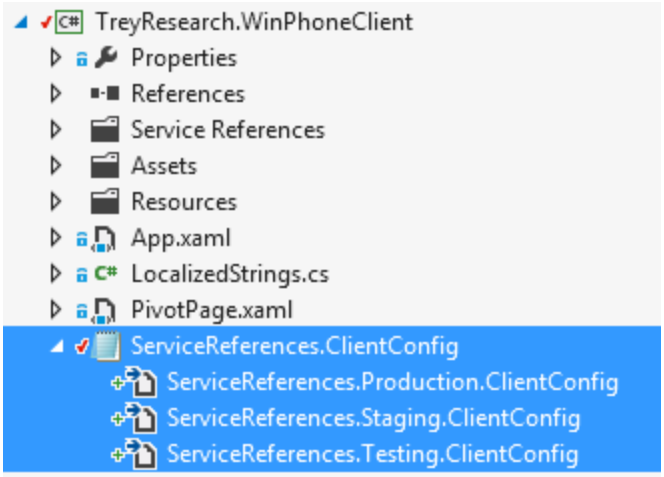
```
<Content Include="ServiceReferences.Production.ClientConfig" />
<Content Include="ServiceReferences.Testing.ClientConfig" />
<Content Include="ServiceReferences.Staging.ClientConfig" />
```

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

XML

```
<None Include="ServiceReferences.Production.ClientConfig">
      <DependentUpon>ServiceReferences.ClientConfig</DependentUpon>
</None>
<None Include="ServiceReferences.Testing.ClientConfig">
      <DependentUpon>ServiceReferences.ClientConfig</DependentUpon>
</None>
<None Include="ServiceReferences.Staging.ClientConfig">
      <DependentUpon>ServiceReferences.ClientConfig</DependentUpon>
</None>
```

5. Save the file. Right-click on the **TreyResearch. WinPhoneClient** 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.



Task 3: Set Up the Transformation for the ServiceReferences.ClientConfig Files

In this task you use the **TransformXml** MSBuild task to transform the base **ServiceReferences.ClientConfig** configuration file. You use the **ServiceReferences.<Environment Name>.ClientConfig** files to define the transformation. When the transformation is done, you will have the environment-specific configuration files.

1. Unload and edit the **TreyResearch. WinPhoneClient.csproj** file just 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="ServiceReferences.*.ClientConfig" />
    </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\WindowsPhone"
Condition="!Exists('$(OutDir)\ConfigFiles\WindowsPhone')" />
```

```
    <!-- AUTOMATION HOL - We call the TransformXml task for all the
configuration files using MSBuild batching (the @() syntax)  -->
    <TransformXml Source="ServiceReferences.ClientConfig"
Transform="@(TransformationFiles)"
Destination="$(OutDir)\ConfigFiles\WindowsPhone\%(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.WinPhoneClient.csproj**.
4. Test to see if the transformations are performed. Run the following command.

CMD

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

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

NOTE: The **Version** parameter could also be set for each environment. This parameter is stored in the application manifest, WMAppManifest.xml. You could transform the parameter so that it matches the instance of the pipeline that generated each XAP package.

To do this, you would use a similar approach to the one used to transform the ServiceReferences.ClientConfig files. An additional step is that the commit stage would pass the version number as a parameter to MSBuild.

Other than its similarity to what you've already seen, this HOL doesn't set the **Version** parameter because the binaries are already versioned, so versioning the package isn't a critical step. However, if you want to distribute a package to end users then it could be useful to know the package's version before installing it.

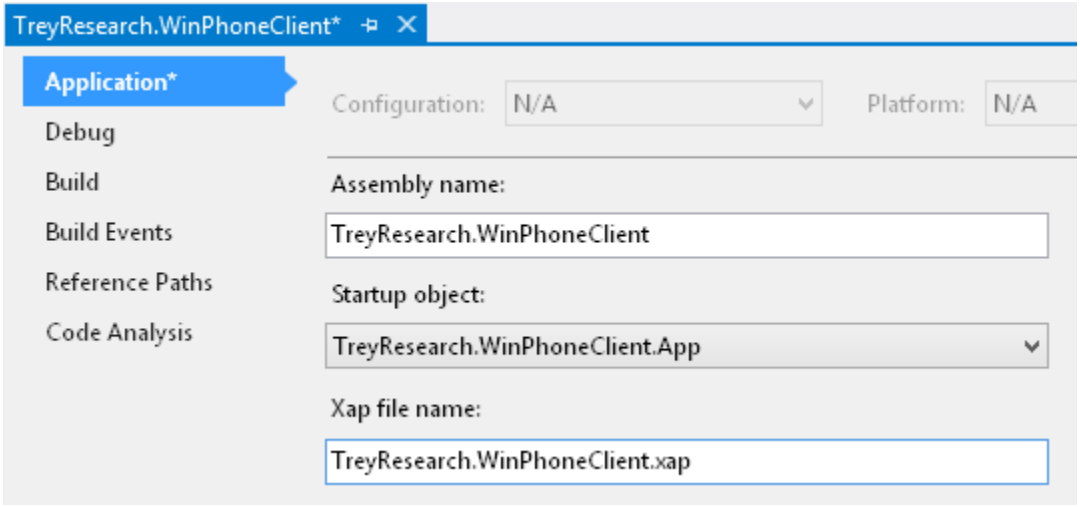
Exercise 3: Deployment Automation: Packaging the Files for Deployment

In this exercise, you only need you change the default name of the XAP package so that it is meaningful, no matter the environment. By default, Windows Phone 8 projects are always packaged during the build process. MSBuild generates the XAP package without requiring any special configuration. This is true even when the build is done by TFS Build in the commit stage.

Task 1: Change the Default Name of the Generated XAP Package

In this task you change the default name of the XAP package by removing a substring.

1. In Solution Explorer, select the **Properties** folder for the **TreyResearch.WinPhoneClient** project.
2. In the **Application** section, remove the text **_Debug_AnyCPU** from the **Xap file name** field. The following screenshot shows the modified XAP file name.



3. Save the project.

Exercise 4: Performing the Automated Deployment

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

In general, the same process shown in Lab 3.1 Automating the Deployment of the WCF Service also applies to Windows Phone 8 apps. The deployment is done locally by the Lab Management deployment agent, which uses a script that you provide.

However, deployment scripts for Windows Phone 8 packages must address two issues. The first issue is how to replace the base configuration file with the transformed file. The second has to do with starting the Windows Phone 8 emulator.

Replacing the ServicesReferences.ClientConfig File

You need to provide a mechanism that replaces the ServiceReferences.ClientConfig file embedded in the XAP package with a transformed file that corresponds to an environment. Because XAP packages are actually .zip files with a different extension, you can extract the contents of the package, replace the configuration file, and then repackage the contents before deploying them. This exercise shows how to use a PowerShell script to do this.

Starting the Emulator

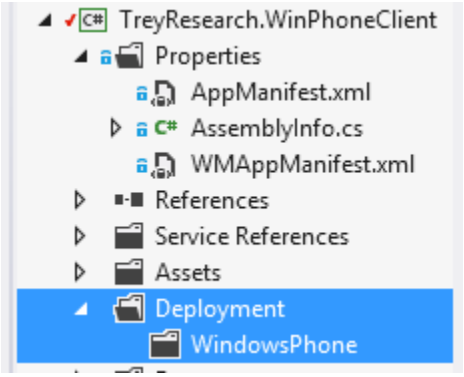
The final deployment step is to start the Windows Phone 8 emulator. This operation must be done in the context of a Windows session that has a graphical user interface (GUI). However, the deployment agent is a Windows service, which means there is no GUI. In other words, the deployment agent can't start the emulator in order to deploy the package. Even if you configure the agent's Windows service so that it can interact with the desktop, the emulator would be launched under the lab service account, which is different from the one that is used by someone who wants to run the emulator.

Although this issue makes it impossible to automate the entire deployment, it is not a critical limitation. Testing the Windows Phone 8 app within the pipeline must be done manually in any case. A tester can run the last deployment step just before starting the actual tests.

To make the last step as easy as possible, this exercise shows you how to write an additional script that the pipeline copies to the target machine. A user can select this script to open the emulator with the application deployed inside it.

Task 1: Creating the Deployment Script for the Agent

1. In Solution Explorer, create a new folder named **Deployment** under the **TreyResearch.WinPhoneClient** project. This is where you will store the deployment script.
2. Under the Deployment folder, create a subfolder named **WindowsPhone** so that the scripts are better organized when they are copied to the drop folder and the target environments. The following screenshot shows the folder hierarchy.



3. Add a new script file to the **WindowsPhone** subfolder. Right-click on the folder and add a new item of type **Text File**. Change the name to **DeployWinPhoneClient.cmd**.
4. In the **DeployWinPhoneClient.cmd** properties, set the **Build Action** to **None**. Set **Copy to Output Directory** to **Copy always**.
5. Paste the following script into the file.

CMD Script

```
REM AUTOMATION HOL - This script copies the needed files from the pipeline Drop
folder to the target machine and runs the PowerShell script that leaves the
package prepared for deployment

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 configuration file 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 files and run
the deployment
set deploymentLocation="%tempfolder%\WindowsPhone\%environment%"
if exist "%deploymentLocation%" rd /s /q "%deploymentLocation%"
mkdir "%deploymentLocation%"

REM AUTOMATION HOL - Copying the PowerShell script used for repackaging,  from the
Drop location to the local deployment folder
copy
"%packageLocation%\Release\Deployment\WindowsPhone\RepackageWinPhoneClient.ps1"
"%deploymentLocation%"

REM AUTOMATION HOL - Running the PowerShell script to change the configuration
file and repackage
cmd /c Powershell.exe -ExecutionPolicy bypass -File
"%deploymentLocation%\RepackageWinPhoneClient.ps1" "%packageLocation%\Release"
"%deploymentLocation%" "%environment%"
```

6. The script copies a PowerShell script that modifies the XAP package to have the correct parameters (you create the PowerShell script in the next task), and runs it.
7. Save the **.cmd** file using the encoding scheme **UTF-8 without signature**. Ignore the source control warning.

Task 2: Create the PowerShell Script

In this task you create a PowerShell script that copies the package and configuration files from the drop location, extracts the contents of the package, replaces the configuration file, and repackages the application.

1. Right-click on the **WindowsPhone** subfolder. Add a new item of type **Text File**. Change the name to **RepackageWinPhoneClient.ps1**.
2. Open the **RepackageWinPhoneClient.ps1** properties. Set the **Build Action** to **None**. **Set Copy to Output Directory** to **Copy always**.
3. Copy the following script into the file and save it.

Powershell Script

```
<#
AUTOMATION HOL - This script copies the needed files from the pipeline Drop
folder to the target machine,
extracts the package, replaces the configuration file, and repackages the
application again.
#>
param(
       [parameter(Mandatory=$true)]
       [string]
       # AUTOMATION HOL - Path where the package to deploy (Xap file) and
configuration files are located
       $packageLocation,
       [parameter(Mandatory=$true)]
       [string]
       # AUTOMATION HOL - Temporal path to use for deployment
       $deploymentLocation,
       [parameter(Mandatory=$true)]
       [string]
       # AUTOMATION HOL - Environment where deployment is made, used to pick
the corresponding configuration file from the available ones
       $environment
     )

# AUTOMATION HOL - Function to extract the contents of the xap file
function ExtractXap
{
param($sourceXapFile, $targetFolder)

[System.Reflection.Assembly]::Load("System.IO.Compression.FileSystem,Version=4
.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089") | Out-Null
    [System.IO.Compression.ZipFile]::ExtractToDirectory($sourceXapFile,
$targetFolder)
}

# AUTOMATION HOL - Function to create the xap file back from the extracted
contents
function PackageXap
{
param($destinationXapFile, $sourceFolder = '')
    [System.IO.Compression.ZipFile]::CreateFromDirectory($sourceFolder,
$destinationXapFile)
}

# AUTOMATION HOL - Copying the package and configuration files from the
package location
Copy-Item "$($packageLocation)\TreyResearch.WinPhoneClient.xap"
$($deploymentLocation)
```

```
Copy-Item
"$($packageLocation)\ConfigFiles\WindowsPhone\ServiceReferences.$($environment
).ClientConfig" $deploymentLocation

# AUTOMATION HOL - Creating the temporary folder to copy the files, extract
the xap and apply the target configuration for the environment
$tempFolder = "$($deploymentLocation)\temp"
if (!(Test-Path -path $tempFolder)) {New-Item $tempFolder -Type Directory}

# AUTOMATION HOL - Extracting the package
$packageFile = $deploymentLocation + "\TreyResearch.WinPhoneClient.xap"
ExtractXap -sourceXapFile $packageFile -targetFolder $tempFolder

# AUTOMATION HOL - Overwriting the configuration file with the one
corresponding to the target environment
$sourceServiceConfig = $deploymentLocation + "\ServiceReferences." +
$environment + ".ClientConfig"
$targetServiceConfig = $tempFolder + "\ServiceReferences.ClientConfig"
Copy-Item $sourceServiceConfig -Destination $targetServiceConfig -Force

# AUTOMATION HOL - Regenerating the xap package with the new configuration
file
Remove-Item -Path $packageFile
PackageXap $packageFile -sourceFolder $tempFolder

# AUTOMATION HOL - Copying the script that runs the emulator from the package
location, to leave it ready for the user
Copy-Item
"$($packageLocation)\Deployment\WindowsPhone\DeployAndLaunchEmulator.cmd"
$($deploymentLocation)
```

Task 3: Create the Script That Launches the Emulator and Deploys the Application

In this task, you create a script that launches the emulator and deploys the Windows Phone 8 app to it. The script invokes the Windows Phone SDK command line deployment tool in order to deploy the application and launch the emulator.

For more information about deploying and running Windows Phone 8 applications in the emulator, see [How to deploy and run a Windows Phone app].

1. Right-click on the **WindowsPhone** subfolder. Add a new item of type **Text File**. Change the name to **DeployAndLaunchEmulator.cmd**.
2. Open the **DeployAndLaunchEmulator.cmd** properties. Set the **Build Action** to **None**. Set **Copy to Output Directory** to **Copy always**.
3. Copy the following code into the file.

XML

```
REM AUTOMATION HOL - This script launches the Windows Phone 8 emulator and
deploys the package on it, leaving it opened for the user

"C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v8.0\Tools\XAP
Deployment\XapDeployCmd.exe"  /installlaunch TreyResearch.WinPhoneClient.xap
/targetdevice:xd
```

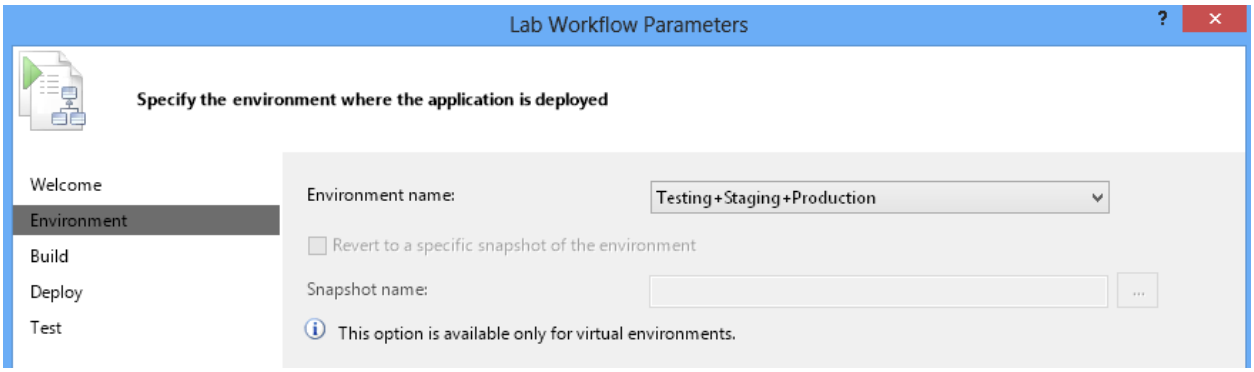
4. Save the **.cmd** file using the encoding scheme **UTF-8 without signature**. Ignore the source control warning.

At this point, you have made all the required changes to the**TreyResearch.WinPhoneClient** project. Make sure that you've saved all the files and checked them in to the TFS version control system.

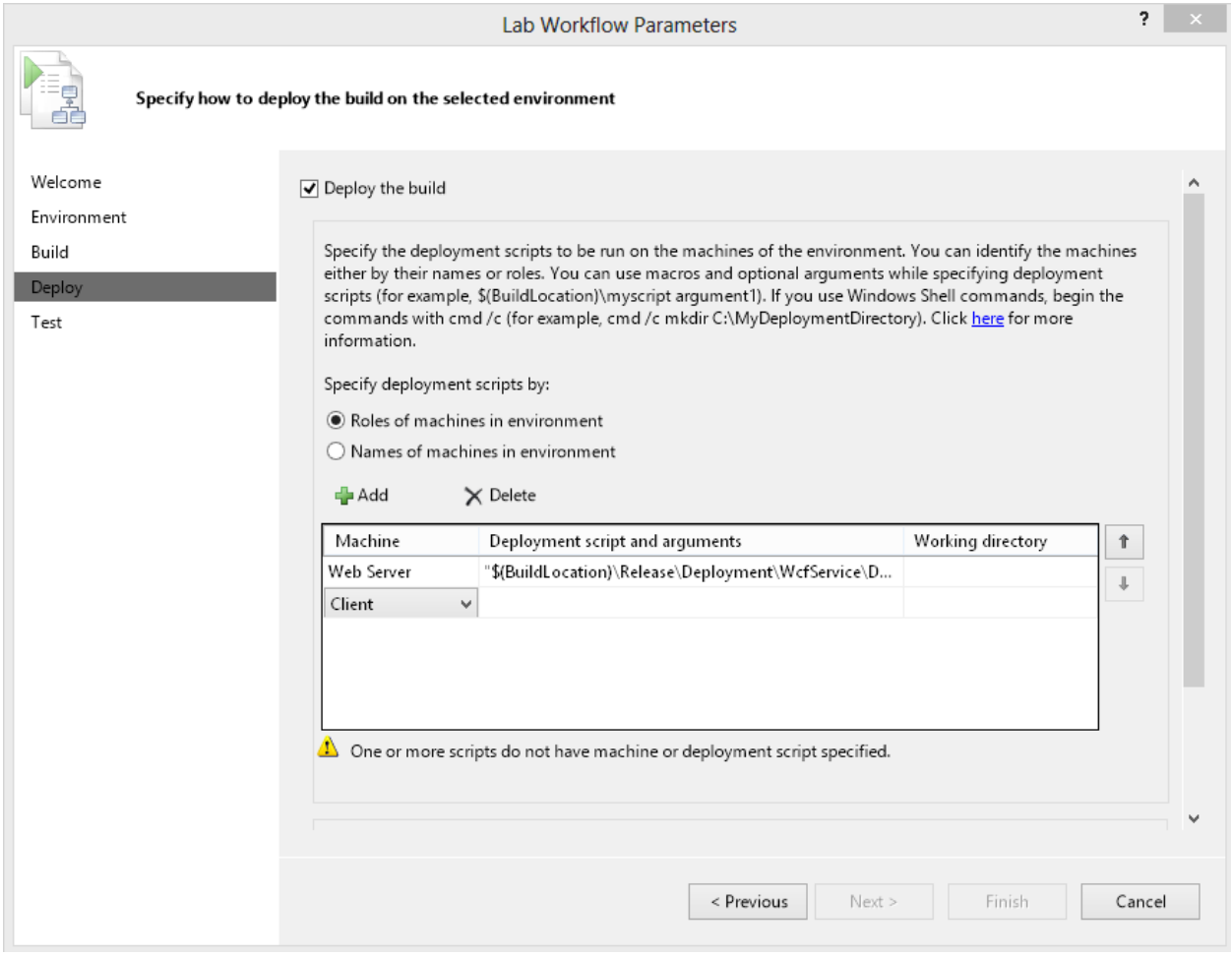
Task 4: Configure the Pipeline Stages to Run the Deployment Script

In this task you configure the release and UAT stages of the pipeline to run the deployment script. You won't configure the acceptance test stage because you can't automate Windows Phone 8 testing inside the emulator.

1. Open the build definition for **03a Release Stage**, which deploys to the production environment. Under the **Process** tab, open the **Lab Process Settings**.
2. The **Lab Workflow Parameter** dialog box opens. In the **Environment** section, make sure that the environment used in Lab 3.1 is selected.



Select the **Deploy** tab. Click **Add** to incorporate the deployment script into the build definition.

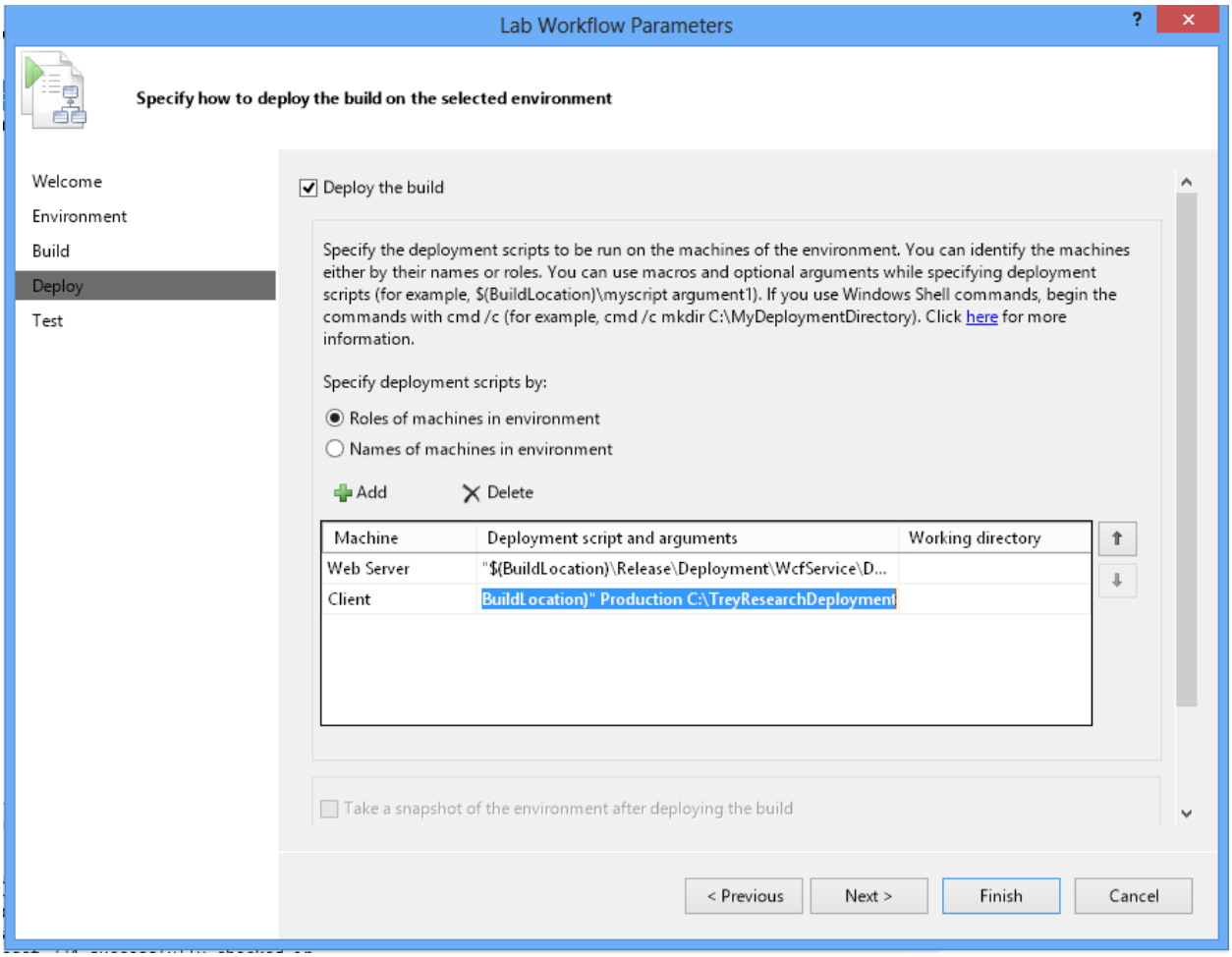


3. In the **Machine** drop down list, select **Client**. The deployment will be done on all the clients in the environment (in this lab there is only one).
4. In the **Deployment script and arguments** column, add the following command line, which will execute the script.

CMD

```
"$(BuildLocation)\Release\Deployment\WindowsPhone\DeployWinPhoneClient.cmd"
"$(BuildLocation)" Production C:\TreyResearchDeployment
```

The deployment agent retrieves the script directly from the drop location, where it is placed by the specific instance of the pipeline. Use the **$(BuildLocation)** built-in variable to compose the path. The first parameter is the drop location. The second parameter is the name of the target environment, which in this case is the production environment. The third parameter is the temporary folder in the target computer that is used by the deployment scripts. The following screenshot shows the completed dialog box.



5. Click **Finish** and save the changes.
6. Repeat steps 1 through 5 for the **03b UAT Stage**. In the fourth step, the second parameter is **Staging**, because the UAT stage deploys to the staging environment. Here is the code.

CMD

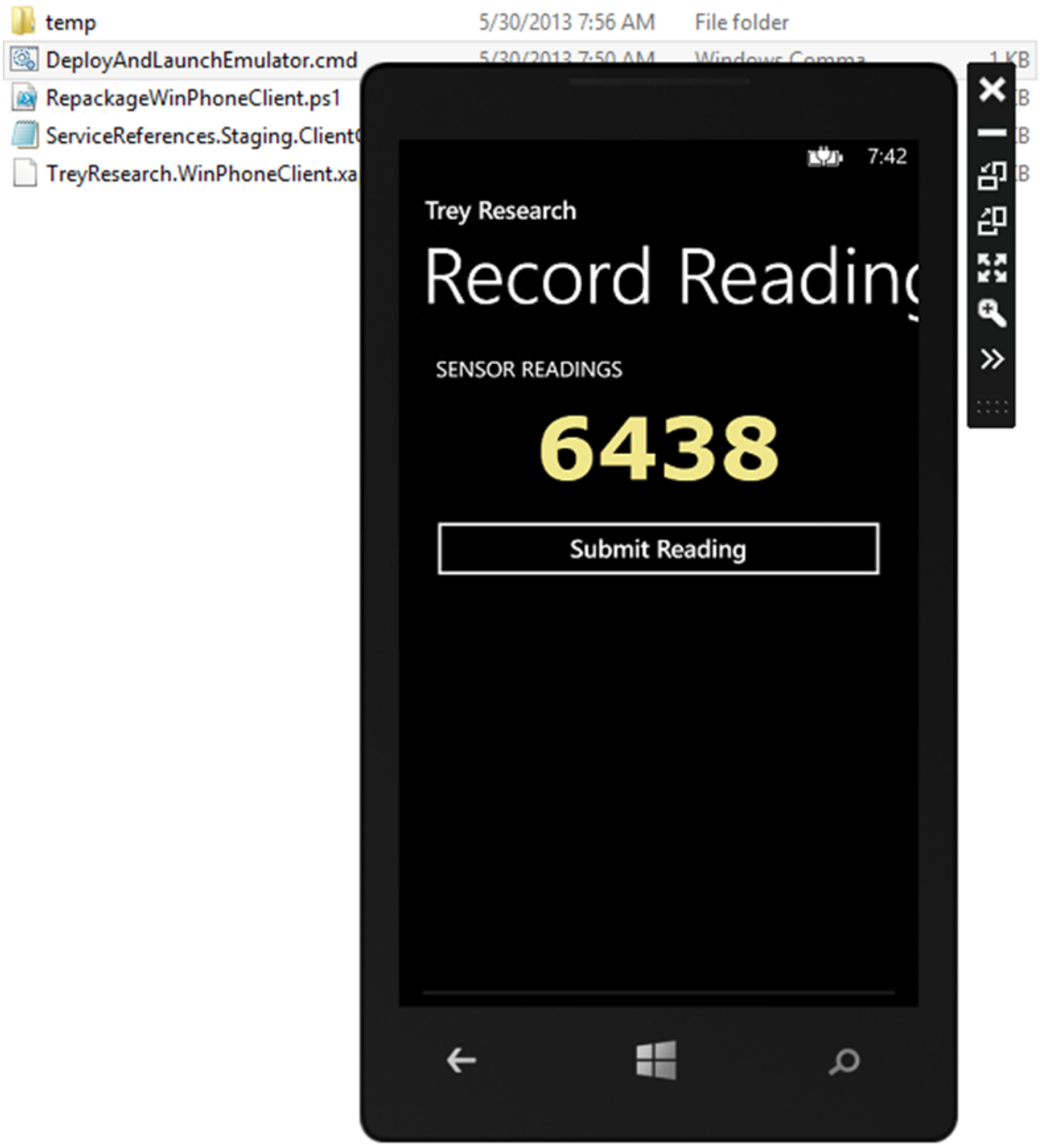
```
"$(BuildLocation)\Release\Deployment\WindowsPhone\DeployWinPhoneClient.cmd"
"$(BuildLocation)" Staging C:\TreyResearchDeployment
```

Exercise 5: Testing the deployment

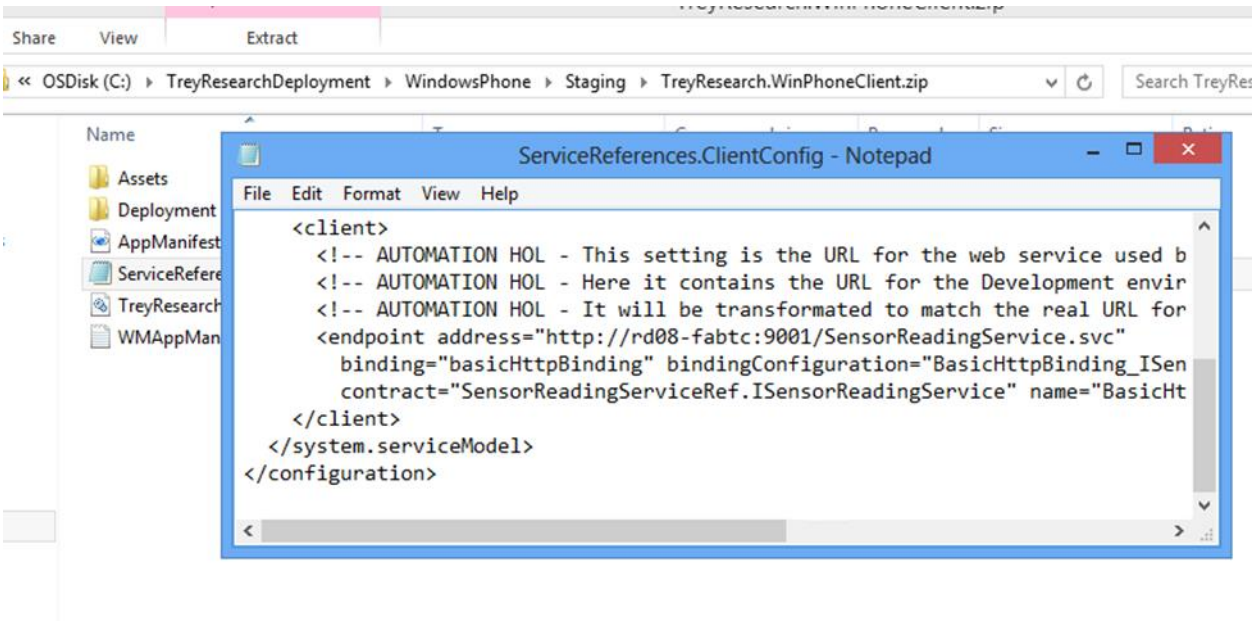
In this exercise you test to see if the Windows Phone 8 app is correctly deployed to the different environments.

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 progagated through the pipeline, so the 02 Acceptance Test Stage is triggered automatically. You can monitor the progress from the Build Explorer window.
3. 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.
4. Verify that the deployment occurs automatically. Log on to the target machine and run the script that triggers the emulator.
5. Check if the deployment can be made automatically by logging on to the target machine and running the script that triggers the emulator. From Windows Explorer, select **C:\TreyResearchDeployment\WindowsPhone\Staging\DeployAndLaunchEmulator.cmd**. The following screenshot shows an example of what you should see.



6. Check that the application uses the web service URL that corresponds to the correct environment. For example, you can open the **ServiceReferences.ClientConfig** file that is embedded in the C:\TreyResearchDeployment\WindowsPhone\Staging\TreyResearch.WinPhoneClient.xap file. To do this, change the file extension from .xap to .zip. The following screenshot shows an example of what you should see.



7. When you are ready to release to the production environment, trigger the 03a Release Stage. Follow the same steps that you did for the staging environment to make sure that the deployment occurred correctly.

Summary

In this HOL you automated the deployment of Trey Research's Windows 8 phone app to the test, staging and production environments. You set up the target environments and then the Lab Management environment. You then prepared the configuration files. You created three files for the three environments. The base configuration file already existed.

Next, you prepared the configuration files. You set the dependency on the base file, and added the code that performs the transforms. You then automated the step for packaging the files by changing the default name of the XAP package.

You then created a deployment script and another script that launches the emulator. You then configured the UAT and release stages to run the scripts. Finally, you tested the pipeline to ensure that the deployments happened correctly.

Copyright