Hands-on Lab 2.1:
Orchestrating the Commit Stage

Table of Contents

Objectives

In this HOL you learn how to customize the TFS default template in order to orchestrate the commit stage of the release pipeline. Orchestration for the commit stage includes:

- Adding steps that name the pipeline instance and the stage.
- Versioning the assemblies.
- Placing the compiled binaries in the drop location.
- Triggering the next stage if the commit stage succeeds or stopping the entire pipeline if the commit stage fails.

This HOL is part one of the four-part Orchestration HOL. Together, the four parts demonstrate how to use Microsoft Team Foundation Server (TFS) and Lab Management to orchestrate the stages of a release pipeline that will support continuous delivery. The subject of orchestration is covered in Chapter 3 of Building a Release Pipeline with Team Foundation Server 2012.

The example application and services that are used in some exercises in this lab are in the subfolders of the **Lab02-Orchestration\Start-Lab** folder. Visual Studio solutions that are the result of completing all of the tasks in an exercise are in the **Lab02-Orchestration\Completed-Lab** folder. You run the examples for this lab on your local computer.

Prerequisites

Here are the prerequisites for completing this lab.

- Complete Introduction.
- Complete Lab-01-StartingPoint.

Time

You should be able to complete all of the exercises in this lab in approximately 40 minutes. If you are familiar with Visual Studio Editor, TFS, and TFS build customization, you should be able to complete the four HOLs in the orchestration sequence in three to four hours.

Exercise 1: Add TfsBuildExtensions.Activities.dll to TreyResearchBuildCustomization.sln

In Lab 1 you added the TFS Build Extensions to the TreyResearch Team Project. In this exercise you'll add one of its DLLs to the **TreyResearchBuildCustomization.sln** located at C:\HOL\Lab02-Orchestration\Start-Lab\TreyResearchBuildCustomization. Here are the steps.

1. Create a Lib folder at the location C:\HOL\Lab02-Orchestration\Start-Lab\TreyResearchBuildCustomization\TreyResearchBuildCustomization.
2. Locate **TfsBuildExtensions.Activities.dll** in the TreyResearch project under the **Custom Assemblies** folder that was created in Lab 1.
3. Copy the **TfsBuildExtensions.Activities.dll and Ionic.Zip.Dll** to the new Lib folder. Navigate to the HOL\Lab02-Orchestration\Start-Lab\TreyResearchBuildCustomization and open the solution **TreyResearchBuildCustomization.sln**.
4. Navigate to the HOL\Lab02-Orchestration\Start-Lab\TreyResearchBuildCustomization and open the solution **TreyResearchBuildCustomization.sln**.
5. Add **TfsBuildExtensions.Activities.dll** to the Reference folder in the **TreyResearchBuildCustomization.sln**.
6. Save the solution.

Exercise 2: Adding Steps That Name the Pipeline Instance and the Commit Stage

In this exercise you will modify a customized version of the TFS default build template named Start_CDPipelineCommitStageProcessTemplate.xaml so that it names both the pipeline instance and the commit stage of that pipeline instance. These names make it easy to associate a particular stage (in this lab, the commit stage) with a particular pipeline instance.

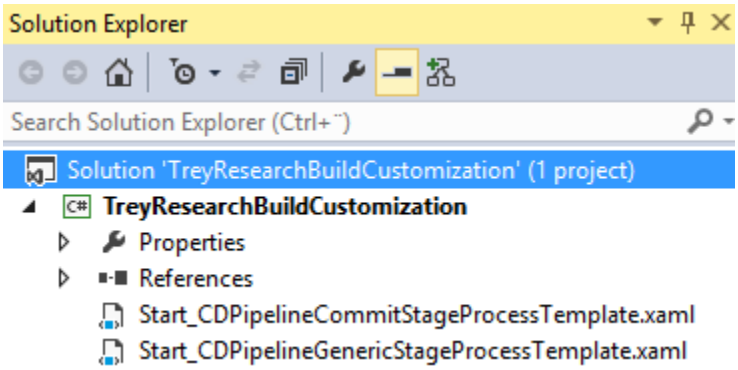
To name the pipeline instance, you use the **TfsVersion** activity that is found under the **TFS Build Extensions** tab in the Visual Studio Toolbox. The **TFSVersion** activity calculates the pipeline instance name. This name is based on the standard pattern *Major.Minor.Build.Release*.

> **Note:** If the **TfsVersion** activity does not appear in the Visual Studio **Toolbox**, you may not have downloaded the **TfsBuildExtensions** activity. See Introduction, and the section named "Third-party Libraries," for more information.

Task 1: Open the Solution File

In this task, you open the solution file so that you can edit the workflow.

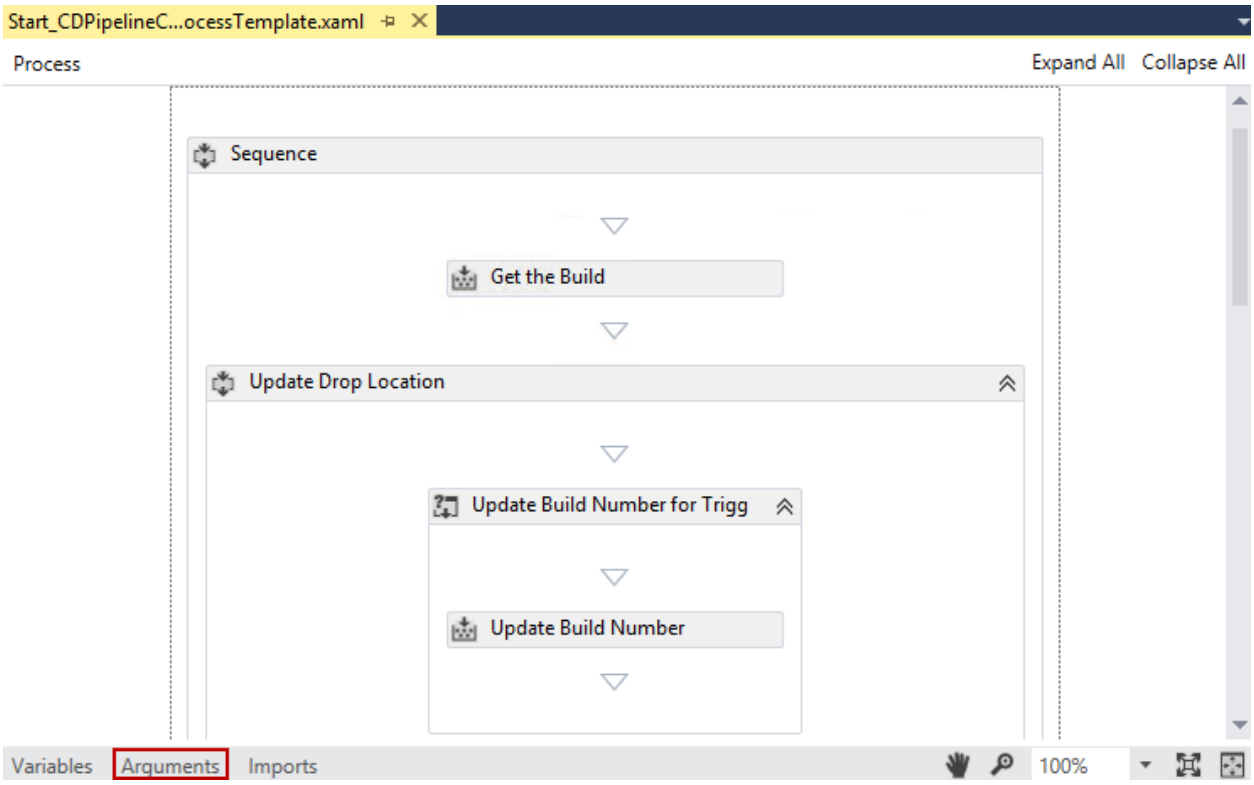
1. Navigate to HOL\Lab02-Orchestration\Start-Lab\ TreyResearchBuildCustomization.
2. Locate the solution file **TreyResearchBuildCustomization.sln**.
3. When you open the solution file you'll see the following solution layout.



Task 2: Edit the Workflow Arguments

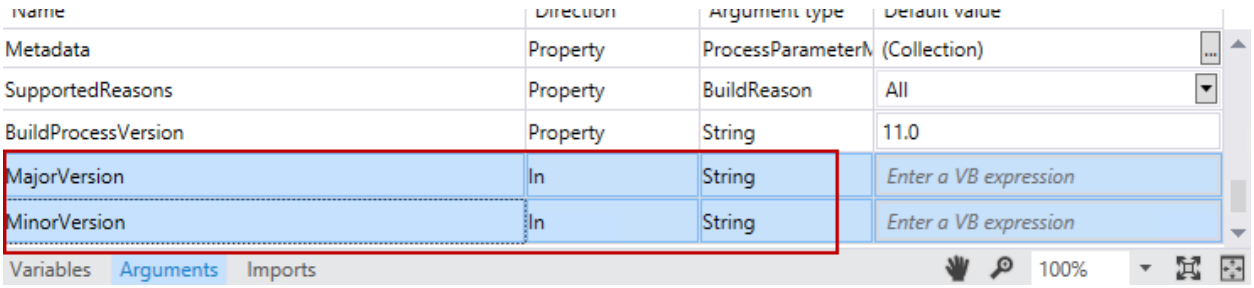
In this task you add 2 new arguments called **MajorVersion** and **MinorVersion**, which will contain versioning information. Then you edit the metadata for the existing, standard build process argument **BuildNumberFormat** and for the new **MajorVersion** and **MinorVersion** arguments**.**

1. From Solution Explorer, open **Start_CDPipelineCommitStageProcessTemplate.xaml**. It opens in the workflow editor.
2. Click the **Arguments** tab in the lower-left corner of the workflow editor. The **Arguments** pane opens.

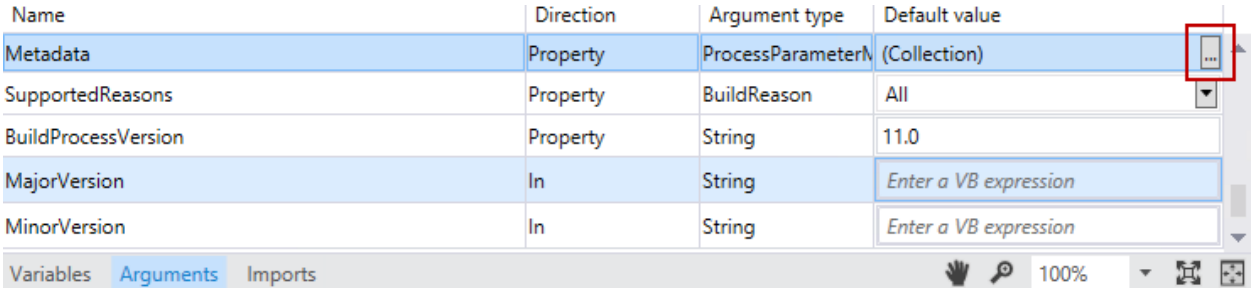


3. In the Arguments pane, click ***Create Argument***. In the **Name** column enter **MajorVersion**.

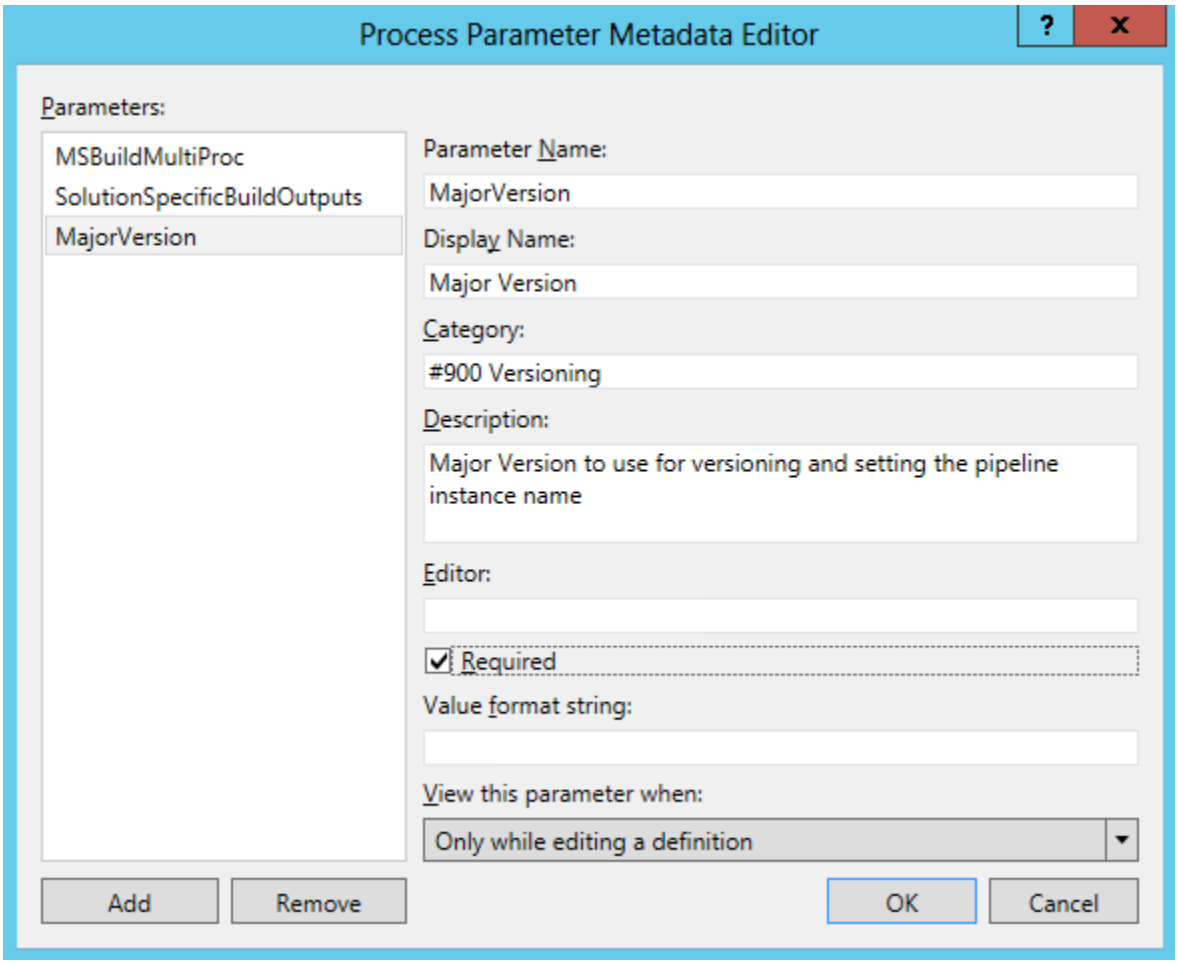
4. Click ***Create Argument***. In the **Name** column enter **MinorVersion**.



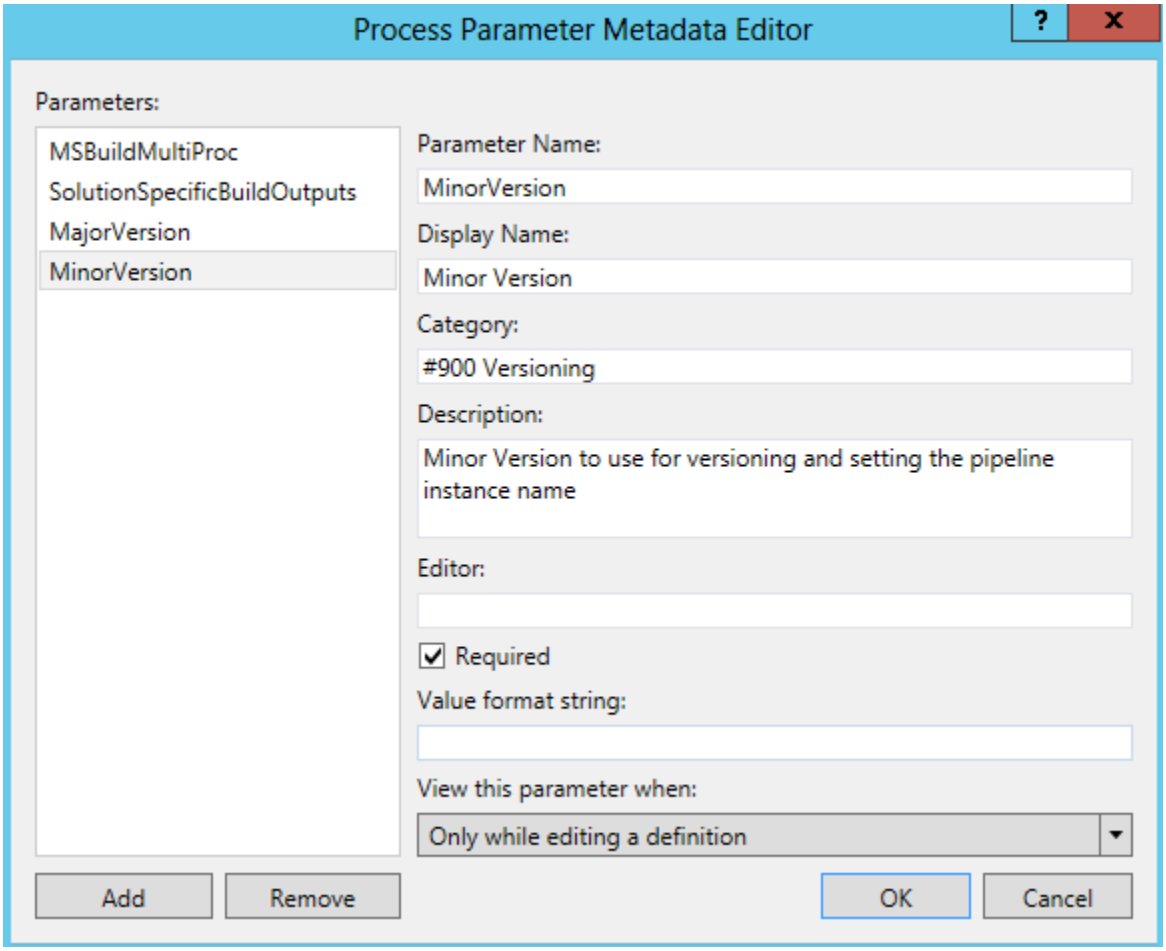
5. Add the metadata for the **MajorVersion** and **MinorVersion** arguments and for the standard build process argument **BuildNumberFormat**. To do this, click the **Edit** button in the **Default value** column of the **Metadata** row.



6. The **Process Parameter Metadata Editor** opens. Click **Add**. Enter the **MajorVersion** metadata shown in the following screenshot.



7. Click **Add**. Enter the **MinorVersion** metadata that is shown in the following screenshot.

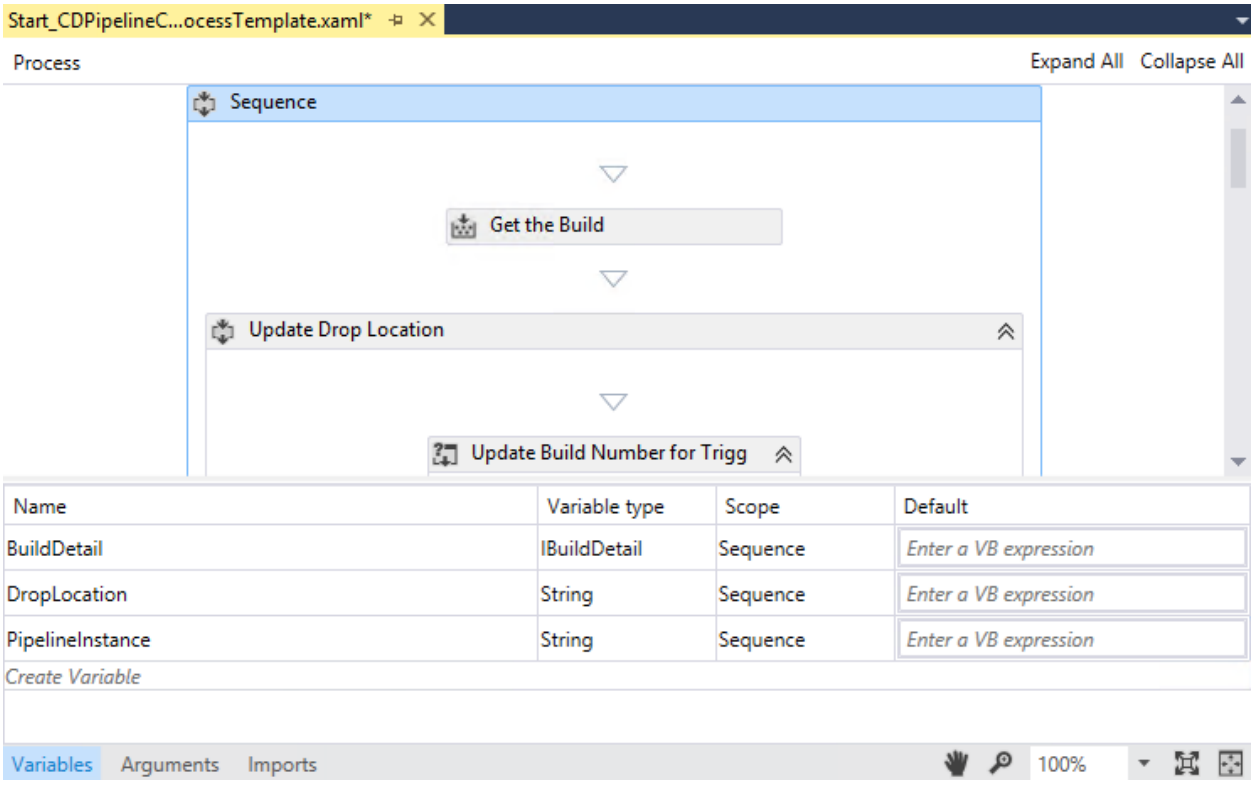


8. Click **Add**. Enter the **BuildNumberFormat** metadata that is shown in the following screenshot. Note that **Never** means that this parameter is hidden.

Task 3: Add the PipelineInstance Variable

In this task you add the **PipelineInstance** variable and scope it to the top-level workflow sequence.

1. Scroll to the top of the workflow file and click in the first **Sequence** activity to highlight it.
2. Click **Variables** in the lower-left corner. The variable pane for the top-level scope opens.
3. Click ***Create Variable***.
4. Create a variable named **PipelineInstance**. The steps for this task are shown in the following screenshot.



Task 4: Add the Set PipelineInstance Sequence to the Workflow

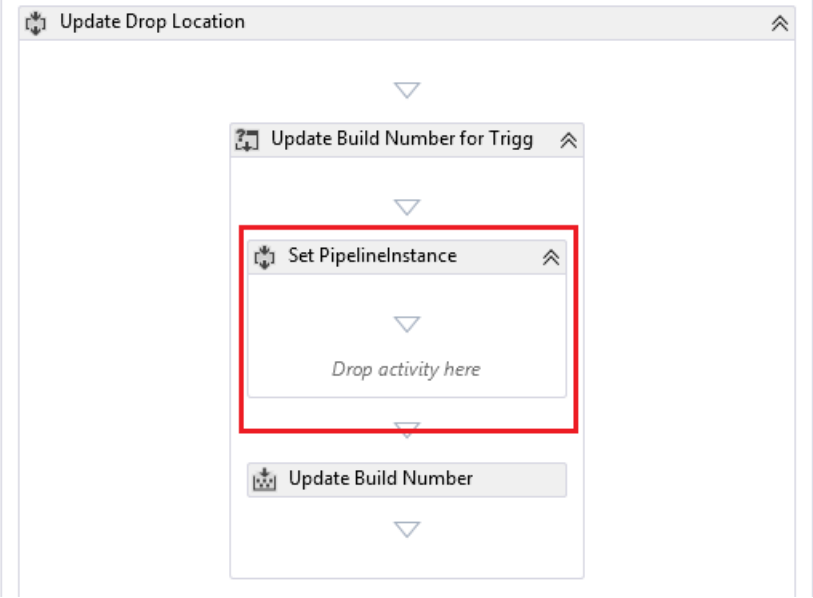
In this task you add the **Set PipelineInstance** sequence to the workflow. This sequence will include the steps related to naming the commit stage and the pipeline instance.

1. Locate the activity named **Update Build Number for Triggered Builds**. It is the first activity inside the **Update Drop Location** sequence, near the top of the workflow.
2. Add a **Sequence** above the **Update Build Number** activity and within the **Update Build Number for Triggered Builds**.

> **TIP**: All activities can be found in the Toolbox by using the search option at the top.

3. Name this sequence **Set PipelineInstance** by changing the **DisplayName** property in the **Properties** window.

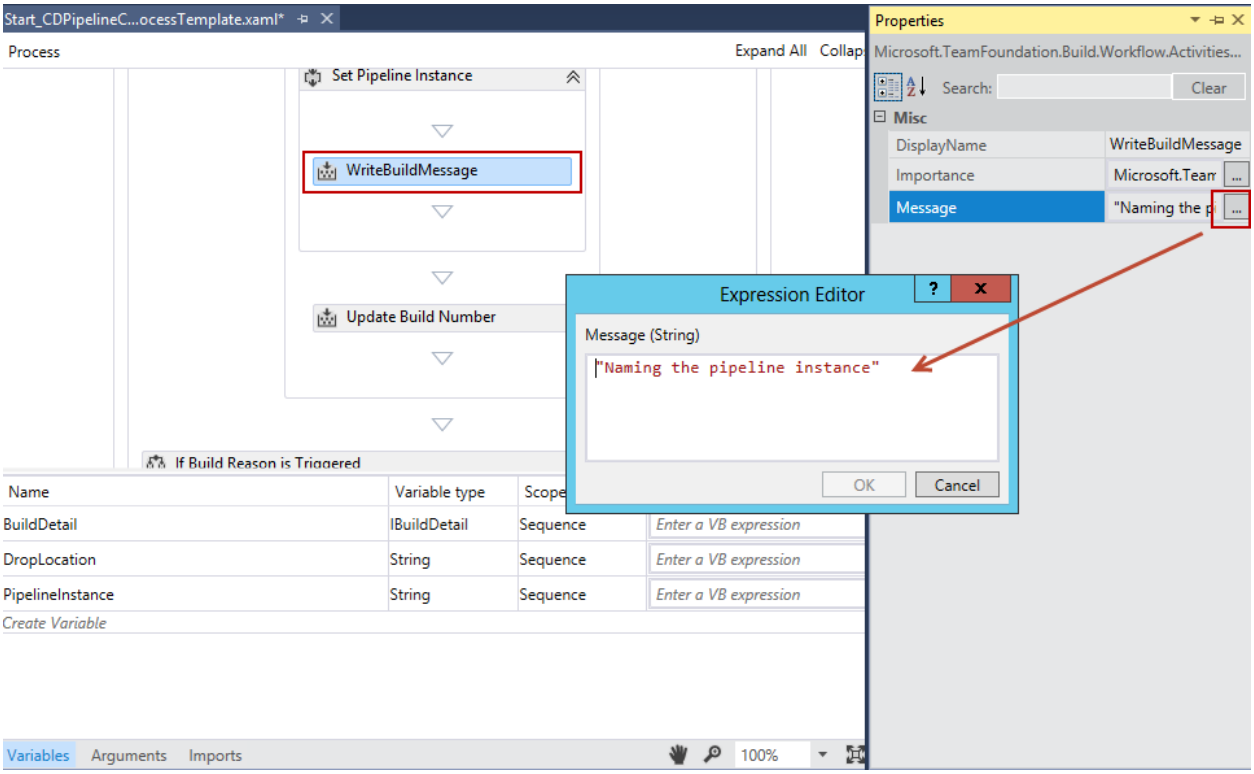
The following screenshot shows the location of the **Set PipelineInstance** sequence within the workflow.



Task 5: Add Activities to the Set PipelineInstance Sequence

In this task you add the activities that generate the pipeline instance name within the **Set PipelineInstance** sequence.

1. Add the **WriteBuildMessage** activity to the **Set PipelineInstance** sequence. This activity logs the operations that occur within the sequence.
2. Add the message string "Naming the pipeline instance" by using the Expression Editor. You can open the editor from the **Properties** window for the **WriteBuildMessage** activity. The following screenshot shows the message. It is highlighted in red.



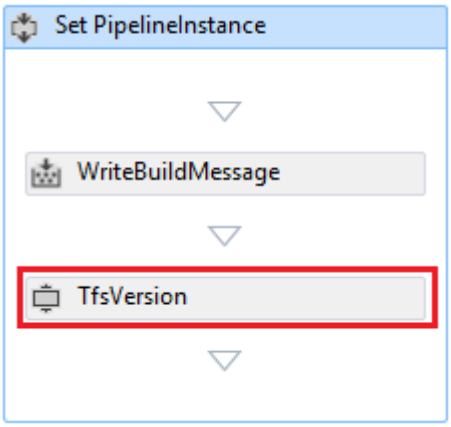
Task 6: Add the TfsVersion Activity

In this task you create the **TfsVersion** activity, which generates the actual version name in the ***Major.Minor.Build.Revision*** format.

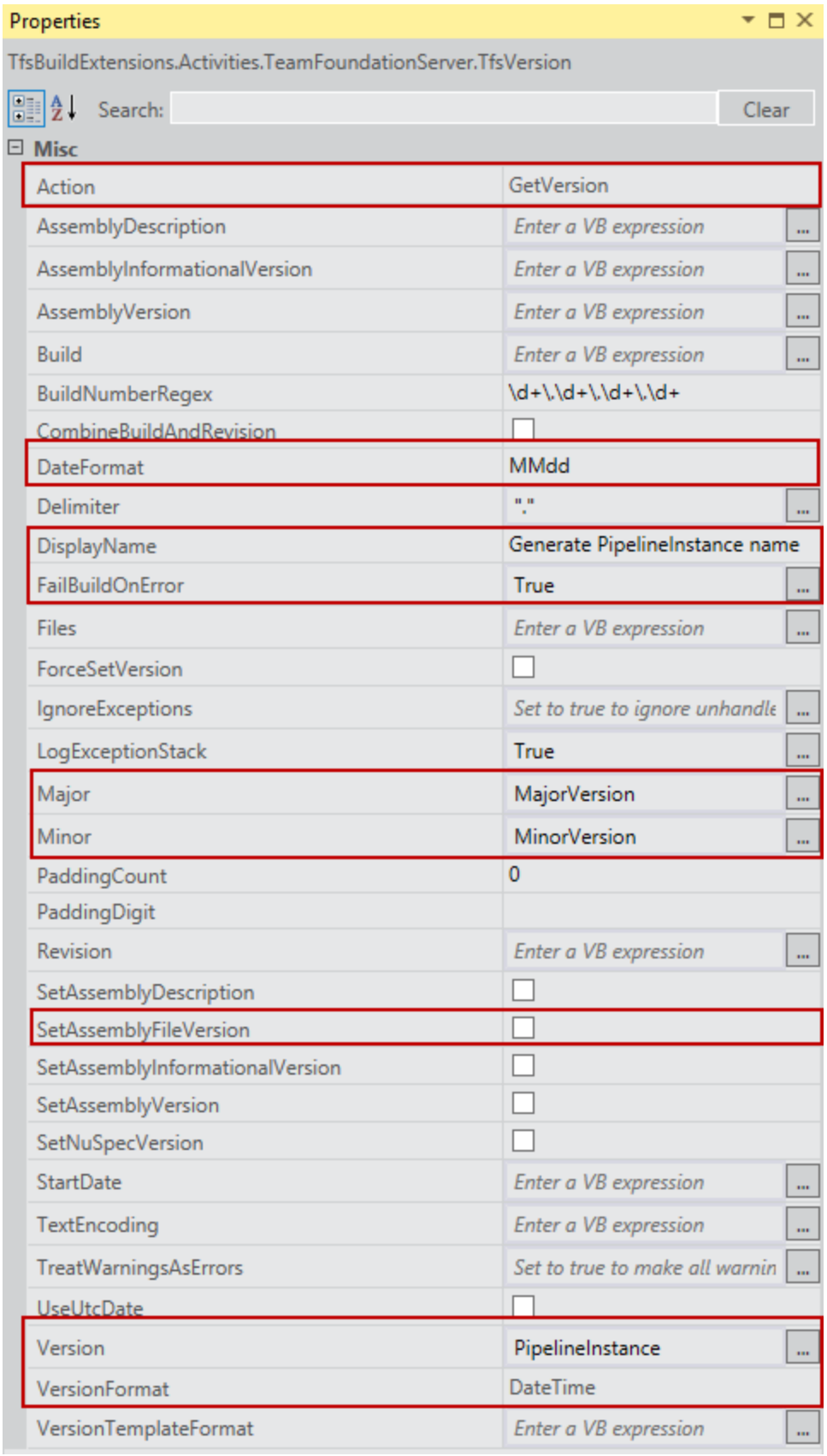
- The ***Major*** and ***Minor*** values are provided by the user.
- The ***Build*** value depends on the current date.
- The ***Revision*** value is automatically generated and incremented by TFS each time the commit stage runs.

> **Note:** This naming schema for the pipeline is not mandatory. You can use any schema you like, as long as it generates unique names.

1. Add a **TfsVersion** activity after the **WriteBuildMessage** activity.



2. Update the values in the **TfsVersion** activity properties. The correct values are highlighted in red in the following screenshot.


Properties
TfsBuildExtensions.Activities.TeamFoundationServer.TfsVersion
Search:
Clear
Misc
Action
GetVersion
AssemblyDescription
Enter a VB expression
AssemblyInformationalVersion
Enter a VB expression
AssemblyVersion
Enter a VB expression
Build
Enter a VB expression
BuildNumberRegex
\d+\.\d+\.\d+\.\d+
CombineBuildAndRevision
DateFormat
MMdd
Delimiter
"."
DisplayName
Generate PipelineInstance name
FailBuildOnError
True
Files
Enter a VB expression
ForceSetVersion
IgnoreExceptions
Set to true to ignore unhandle
LogExceptionStack
True
Major
MajorVersion
Minor
MinorVersion
PaddingCount
0
PaddingDigit
Revision
Enter a VB expression
SetAssemblyDescription
SetAssemblyFileVersion
SetAssemblyInformationalVersion
SetAssemblyVersion
SetNuSpecVersion
StartDate
Enter a VB expression
TextEncoding
Enter a VB expression
TreatWarningsAsErrors
Set to true to make all warnin
UseUtcDate
Version
PipelineInstance
VersionFormat
DateTime
VersionTemplateFormat
Enter a VB expression

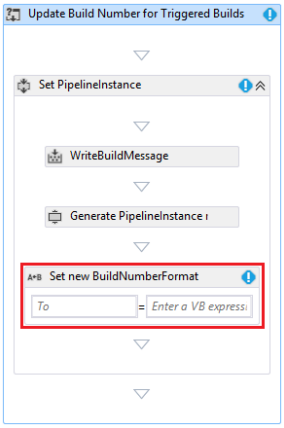
3. Rename the **TfsVersion** activity to **GeneratePipelineInstance name**.

Task 7: Add an Assign Activity

In this task, you add an **Assign** activity that generates the entire instance name for the pipeline and assigns it to the **BuildNumberFormat** argument.

1. Add an **Assign** activity below the **GeneratePipelineInstance** activity.

> **Note**: In the following image the ! sign occurs because the **Assign** activity properties still need to be entered.

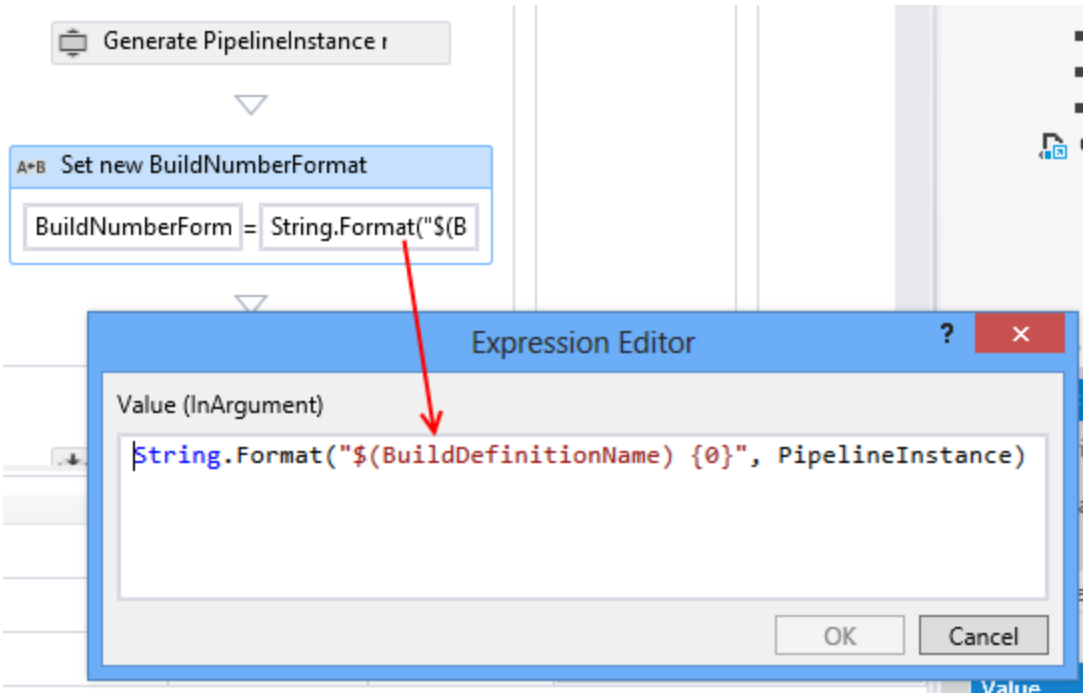


2. Rename the **Assign** activity to **Set new BuildNumberFormat**.
3. In the **Set new BuildNumberFormat To** variable box, enter **BuildNumberFormat**. Insert the following code into the value argument.

Visual Basic

```
String.Format("$(BuildDefinitionName) {0}", PipelineInstance)
```

The following screenshot shows how to set the variable and the argument.



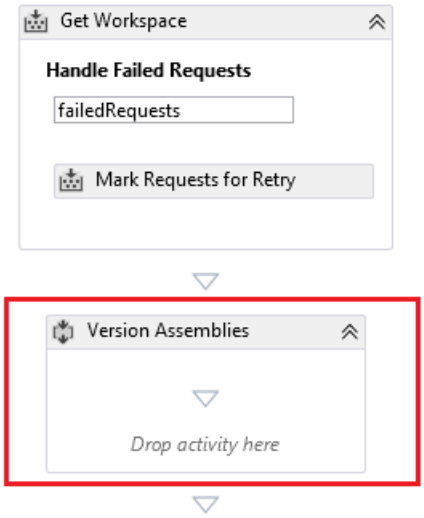
Exercise 3: Versioning the Assemblies

In this exercise you create the steps that version the assemblies to match the pipeline instance name. With this versioning strategy, you always know the instance that generated any specific binary. Because, by default, TFS also labels the code after the build number, you will also be able to identify the source code that generated the binary.

Task 1: Add the Versions Assemblies Sequence

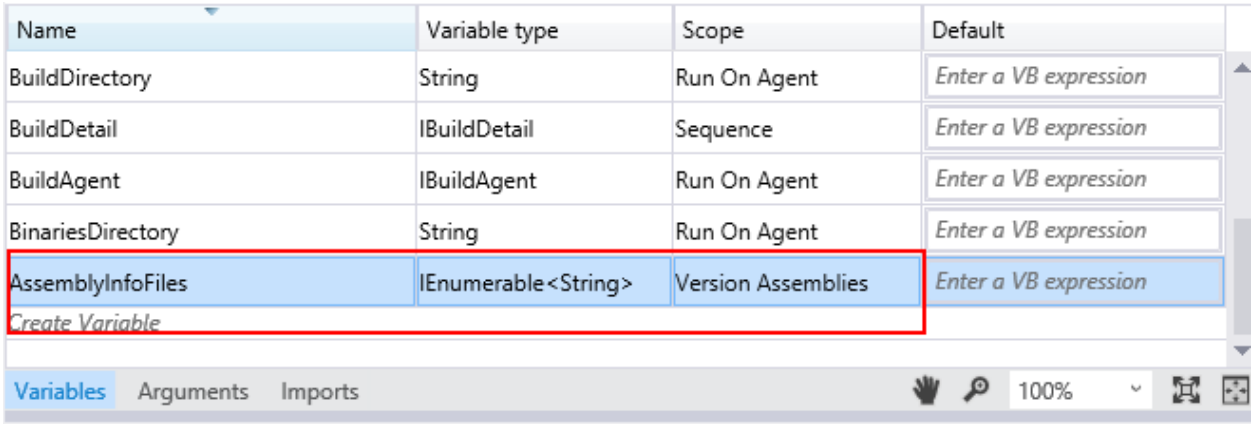
In this task, you add a sequence that embeds the version number in the AssemblyInfo files in the source code. The version number is the same as the pipeline instance name. Versioning the assemblies occurs just after getting the source code from the version control system, but before compiling and building it.

1. Locate the **Get Workspace** activity inside the workflow. It's towards the end of the file.
2. Add a **Sequence** and name it **Version Assemblies**. The following screenshot shows the location of the **Version Assemblies** sequence.



3. Add a variable scoped to the **Version Assemblies** sequence. Name it **AssemblyInfoFiles**. The variable type is **IEnumerable<String>**. This variable stores the list of AssemblyInfo files to be changed.

The following screenshot shows the **Variables** pane with the **AssemblyInfoFiles** variable.



Task 2: Add the FindMatchingFiles Activity

In this task, you add activities to the **Version Assemblies** sequence in order to retrieve all the AssemblyInfo files that need to be changed.

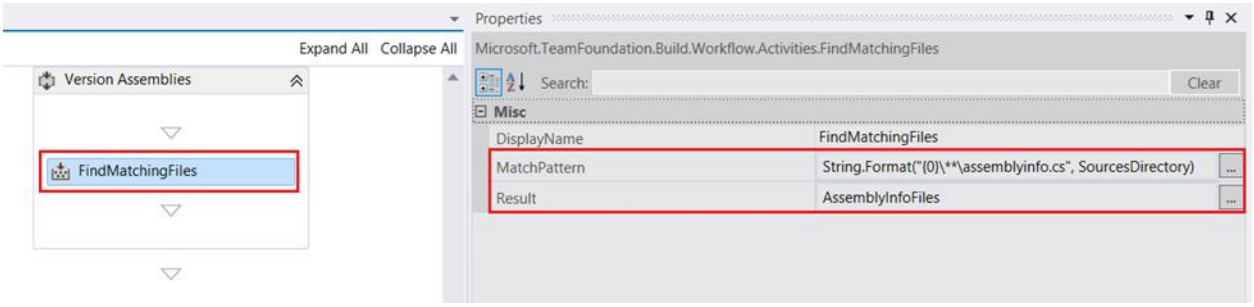
1. Add the **FindMatchingFiles** activity to the **Version Assemblies** sequence.
2. Open the associated **Properties** dialog box for this activity.
3. Set the **MatchPattern** argument to the following code.

Visual Basic

```
String.Format("{0}\**\assemblyinfo.cs", SourcesDirectory)
```

4. Set the **Result** argument to **AssemblyInfoFiles**.

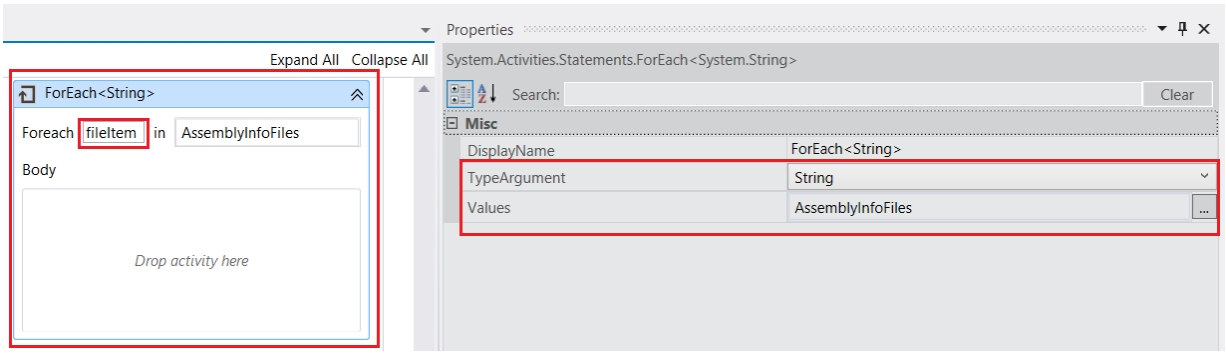
The following screenshot shows how to set the arguments.



Task 3: Add a ForEach Activity

In this task you add a **ForEach** activity that goes through the list of files included in the **AssemblyInfoFiles** variable, assigns them to the **fileItem** variable and logs the file name.

1. Add a **ForEach** activity after the **FindMatchingFiles** activity.
2. Rename the default item value in the **ForEach** box to **fileItem**.
3. Open the **ForEach** activity **Properties** dialog box.
4. Set the **TypeArgument** argument to **String**.
5. Set the **Values** argument to **AssemblyInfoFiles**. The following screenshot shows how to set the arguments.

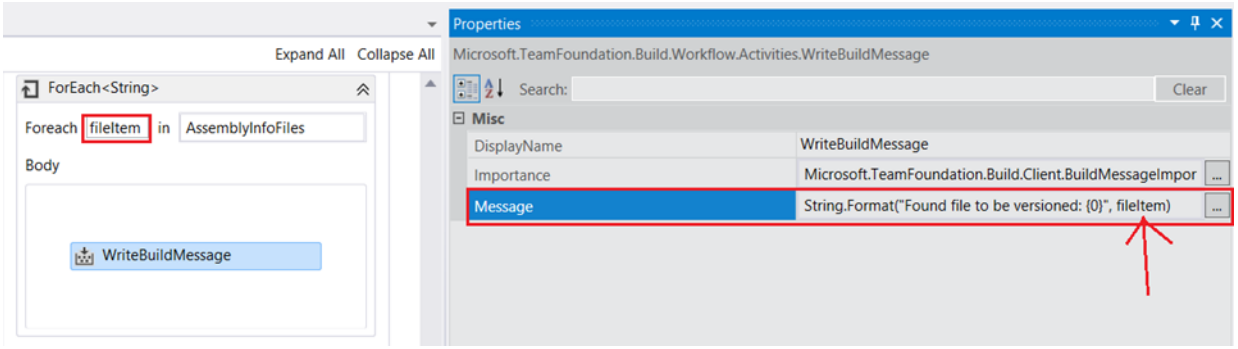


6. Add a **WriteBuildMessage** activity inside the **ForEach** loop. The activity logs the files that are selected.
7. Set the **Message** property to the following code.

Visual Basic

```
String.Format("Found file to be versioned: {0}", fileItem)
```

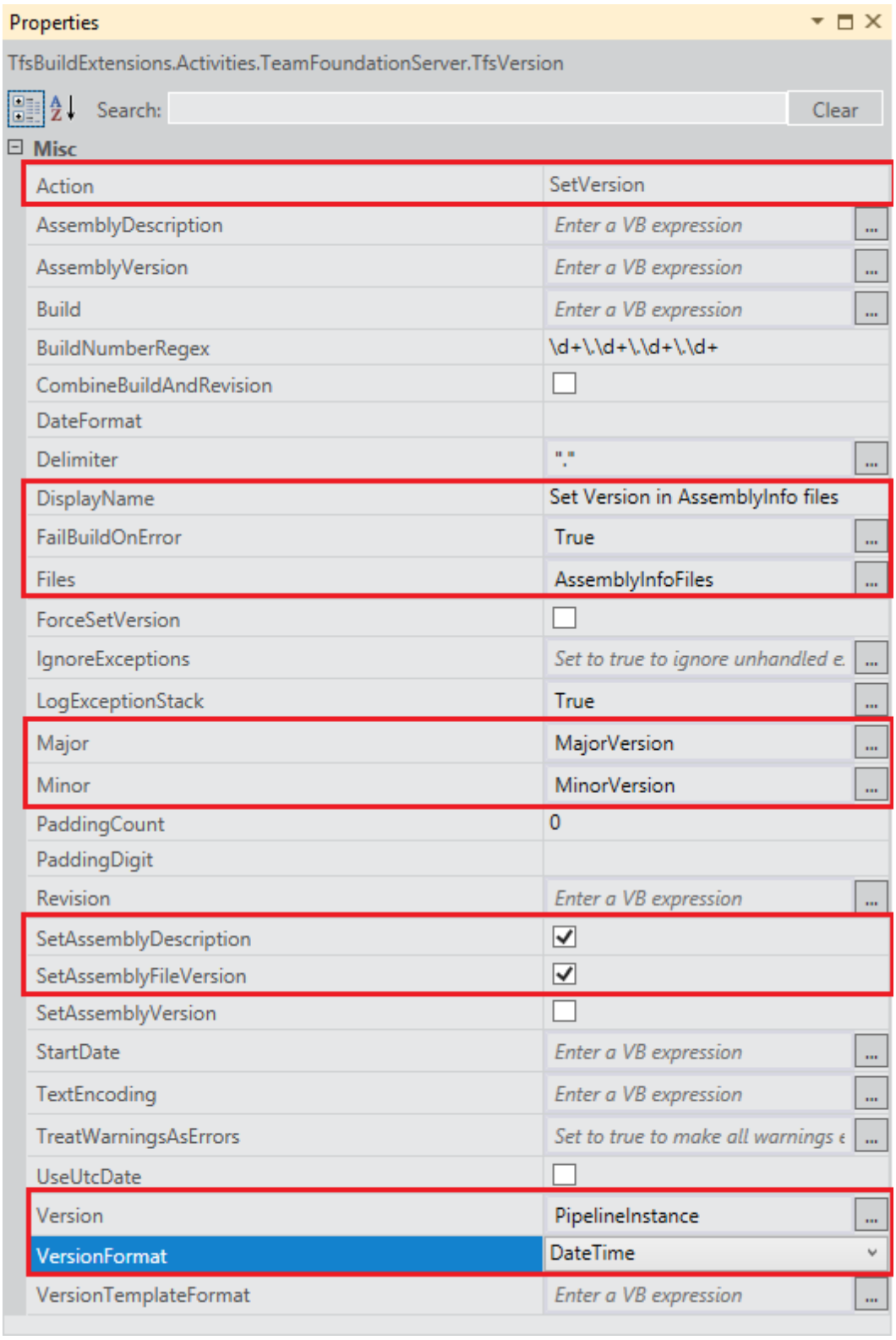
The following screenshot shows the **ForEach** loop properties.



Task 4: Add a TfsVersion Activity

In this task, you add a **TfsVersion** activity that updates the files with the name of the pipeline instance.

1. Add a **TfsVersion** activity after the **ForEach** activity.
2. Open the **TfsVersion** activity **Properties** dialog box.
3. Update the **TfsVersion** activity properties with the values highlighted in red in the following screenshot.


Properties
TfsBuildExtensions.Activities.TeamFoundationServer.TfsVersion
Search:
Clear
Misc
Action
SetVersion
AssemblyDescription
Enter a VB expression
AssemblyVersion
Enter a VB expression
Build
Enter a VB expression
BuildNumberRegex
\d+\.\d+\.\d+\.\d+
CombineBuildAndRevision
DateFormat
Delimiter
"."
DisplayName
Set Version in AssemblyInfo files
FailBuildOnError
True
Files
AssemblyInfoFiles
ForceSetVersion
IgnoreExceptions
Set to true to ignore unhandled e.
LogExceptionStack
True
Major
MajorVersion
Minor
MinorVersion
PaddingCount
0
PaddingDigit
Revision
Enter a VB expression
SetAssemblyDescription
SetAssemblyFileVersion
SetAssemblyVersion
StartDate
Enter a VB expression
TextEncoding
Enter a VB expression
TreatWarningsAsErrors
Set to true to make all warnings e
UseUtcDate
Version
PipelineInstance
VersionFormat
DateTime
VersionTemplateFormat
Enter a VB expression

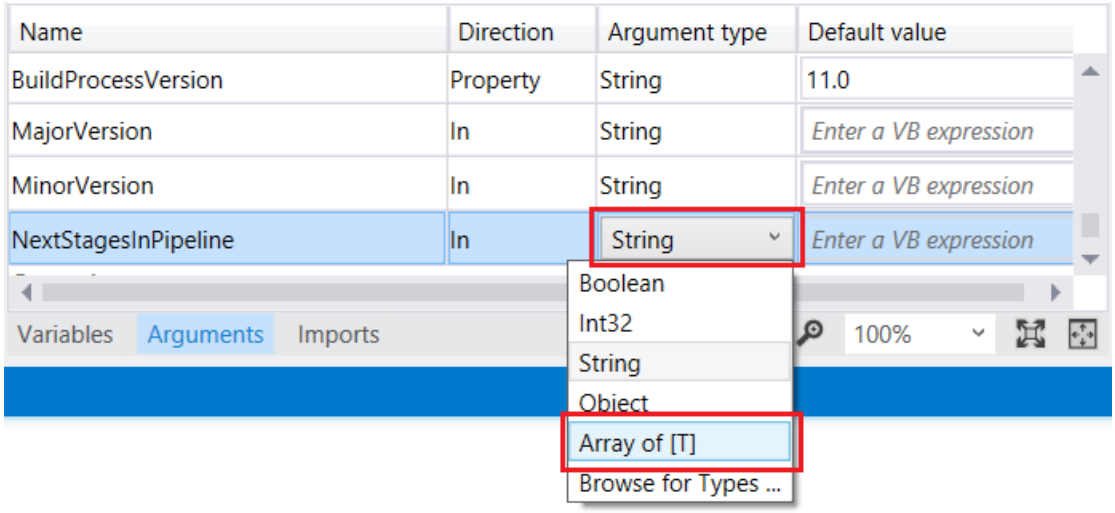
Exercise 4: Orchestrate the Stage to Propagate Changes or Stop the Pipeline

In the final exercise you orchestrate the commit stage so that, if it succeeds, it automatically triggers the next stage in the pipeline. If the commit stage fails, the entire pipeline stops.

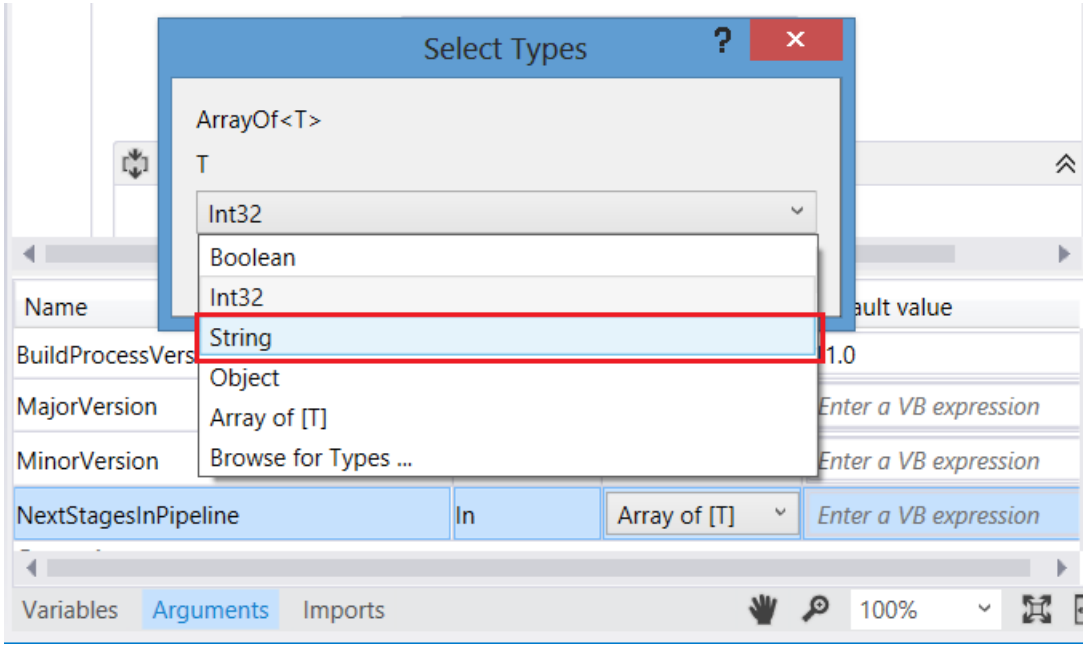
Task 1: Add the NextStagesInPipeline Variable

In this task, you add a variable that holds the list of stages to be triggered if the commit stage succeeds.

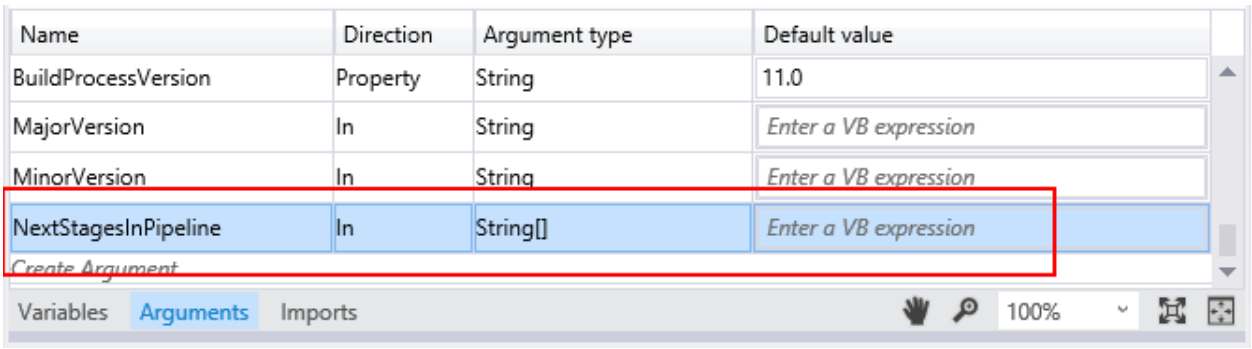
1. Add a new argument to the workflow named **NextStagesInPipeline**, of type **String[]**. In the **Argument type** drop-down box, select **Array of [T]**.



2. The **Select Types** dialog box opens. Select **String** from the drop-down list.

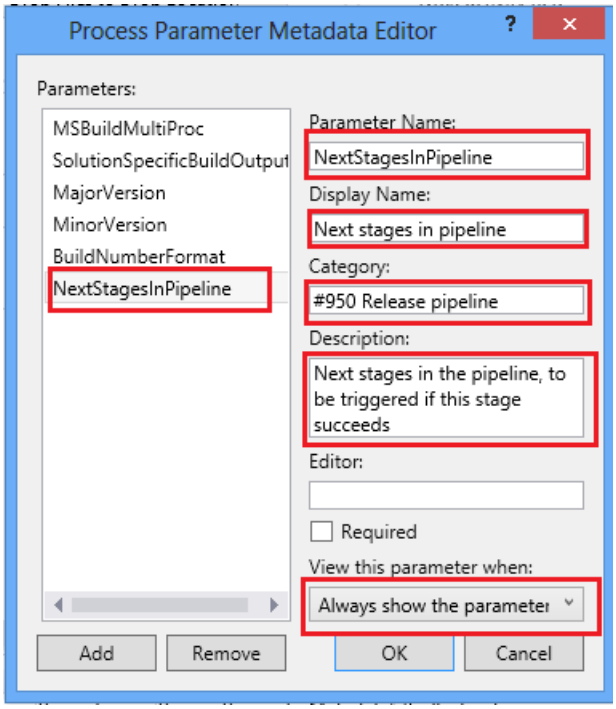


3. The **NextStagesInPipeline** Argument type should be set to **String[]**.



4. Click the **Edit** button in the **Default value** column of the **Metadata** row to launch the **Process Parameter Metadata Editor**.

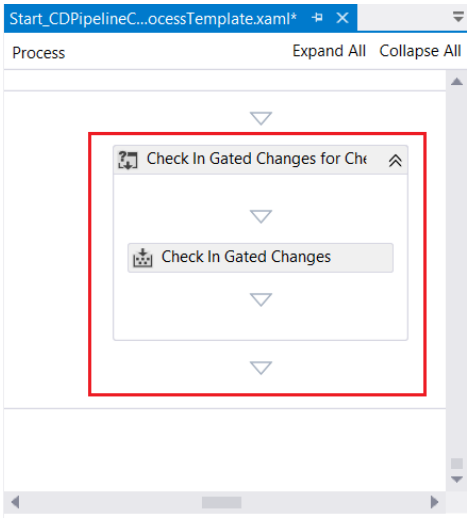
5. The **Process Parameter Metadata Editor** opens. Click **Add**. Enter the **NextStagesInPipeline** metadata, as shown in the following screenshot.



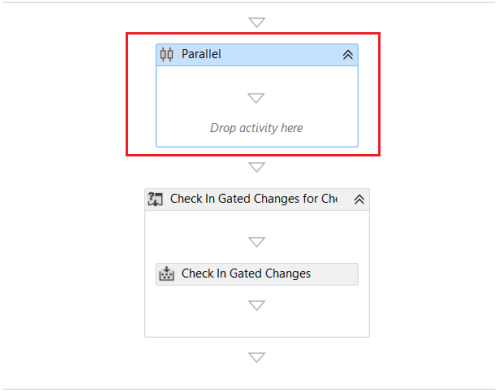
Task 2: Add a New Parallel Activity

In this task you add a new parallel activity at the end of the entire workflow that determines if the next stage can be triggered or if the pipeline should stop.

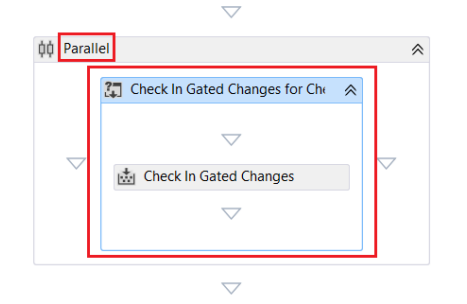
1. Go to the **Check In Gated Changes for CheckInShelveset Builds** activity at the end of the workflow.



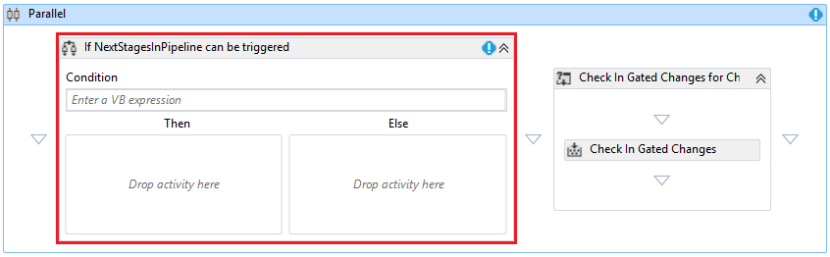
2. Add a **Parallel** activity just before the **Check In Gated Changes for CheckInShelveset Builds** activity.



3. Move the **Check In Gated Changes for CheckInShelveset Builds** activity inside the newly created **Parallel** activity.



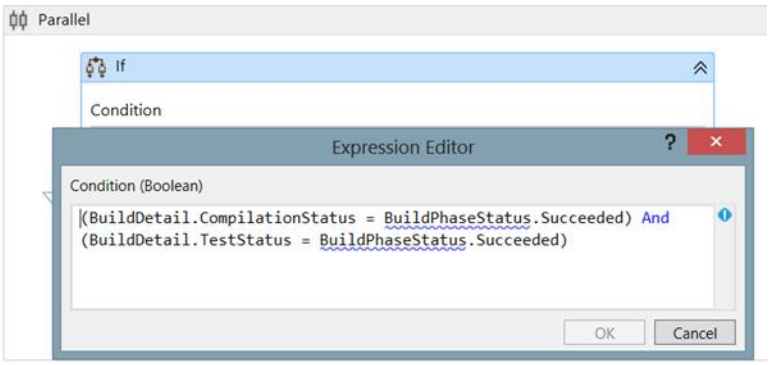
4. Inside the **Parallel** activity, add an **If** activity and name it **If NextStagesInPipeline can be triggered**. The **If** activity will check to see if the commit stage has succeeded or failed.



5. In the **Condition** box insert the following code, which checks to see if the commit stage was successful.

Visual Basic

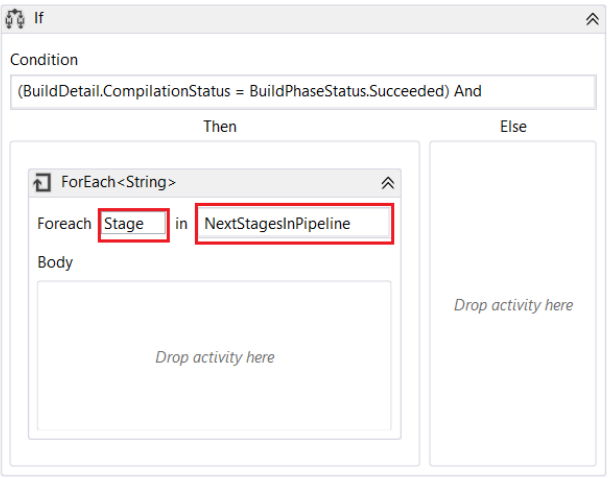
```
(BuildDetail.CompilationStatus = BuildPhaseStatus.Succeeded) And
(BuildDetail.TestStatus = BuildPhaseStatus.Succeeded)
```



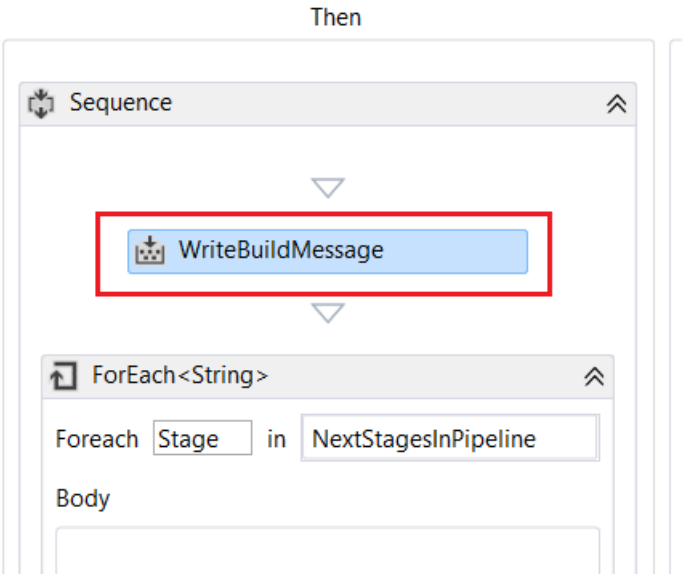
Task 3: Add a ForEach Activity to the If Activity.

In this task you add a **ForEach** activity to the **If Then** box located within the **Parallel** activity. The loop will iterate through the list of stages contained in the **NextStagesInPipeline** variable to find what stages should be triggered if the commit stage succeeds. In addition, you add logging to the work flow.

1. Insert a **ForEach** activity into the **If Then** box within the **Parallel** activity.
2. In the **ForEach** box, rename the default **item** value to **Stage**.
3. Open the **ForEach** activity's **Properties** dialog box.
4. Set the **TypeArgument** argument to **String**.
5. Set the **Values** argument to the **NextStagesInPipeline** variable.



6. Add a **WriteBuildMessage** activity above the **ForEach** activity. This automatically generates a **Sequence** that includes the **WriteBuildMessage** and the **ForEach** activity.



7. Set the message text for this **WriteBuildMessage** activity with the following code, which logs the triggering of subsequent stages.

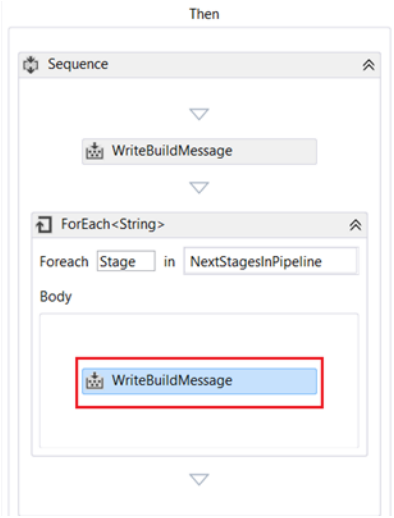
Visual Basic

```
String.Format("Triggering subsequent stages in the pipeline: {0} stages to be triggered", NextStagesInPipeline.Length)
```

8. Add another **WriteBuildMessage** activity inside the **ForEach** activity body.
9. Define the message text for the **WriteBuildMessage** activity with the following code, which logs the stage that is triggered.

Visual Basic

```
String.Format("Triggering stage: {0}", Stage)
```



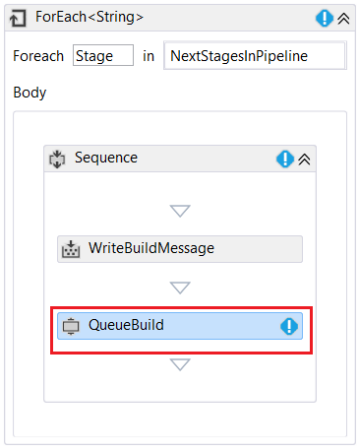
Task 4: Add Activities to Trigger Subsequent Stages

In this task you use the **QueueBuild** activity from the Community TFS Build Extensions to set some properties and to trigger the next stage in the pipeline. You pass two parameters to each subsequent stage of the pipeline. Here are the parameters.

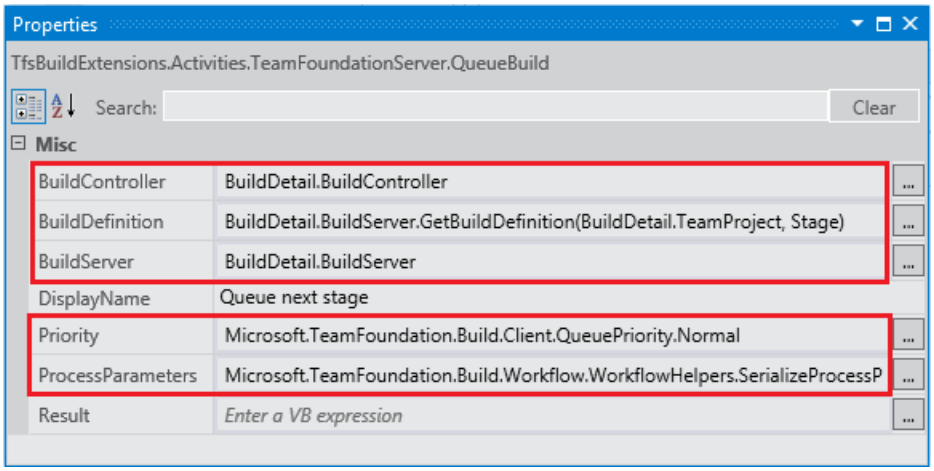
- **PipelineInstance:** This parameter relates the subsequent stage to the pipeline instance and gives the stage the same name as that instance.
- **DropLocation**: Because continuous delivery pipelines only build once, you need this parameter to tell the subsequent stages where to find the binaries.

---

1. Within the **ForEach** activity you added in the previous task, add a **QueueBuild** activity below the **WriteBuildMessage** activity.



2. Open the **Properties** window of the **QueueBuild** activity to define the values that are highlighted in red in the following screenshot. The code for each highlighted property is found in the next steps.



3. Add the following code for the **BuildController** property.

Visual Basic

```
BuildDetail.BuildController
```

4. Add the following code for the **BuildDefinition** property.

Visual Basic

```
BuildDetail.BuildServer.GetBuildDefinition(BuildDetail.TeamProject,Stage)
```

5. Add the following code for the **BuildServer** property.

Visual Basic

```
BuildDetail.BuildServer
```

6. Add the following code for the **Priority** property.

Visual Basic

```
Microsoft.TeamFoundation.Build.Client.QueuePriority.Normal
```

7. Add the following code for the **ProcessParameters** property.

Visual Basic

```
Microsoft.TeamFoundation.Build.Workflow.WorkflowHelpers.SerializeProcessParameters
(New Dictionary(Of String, Object) From {{"PipelineInstance", PipelineInstance},
{"PipelineInstanceDropLocation", BuildDetail.DropLocation}})
```

Task 5: Save Everything

Save your work. You have completed the first HOL in the orchestration sequence.

Summary

In this HOL you created and orchestrated the commit stage of the pipeline by customizing the work flow in the Start_CDPipelineCommitStageProcessTemplate.xaml file. Here are some points to remember.

- There is no way to create a "pipeline" in TFS. Instead, a pipeline is a set of orchestrated stages.
- Each stage of the pipeline is implemented by a build definition.
- Most of the steps inside the stages are implemented by using workflow foundation activities.
- A pipeline instance occurs when the stages execute.
- To check your work, examine the files found in the TreyResearchBuildCustomization.zip file that is located in the directory HOL\Lab02\Completed-Lab.

Copyright