



it courseware™

TRAINING MATERIALS FOR IT PROFESSIONALS

EVALUATION COPY

Unauthorized Reproduction or Distribution Prohibited

Windows Workflow Foundation Using C#

Rev. 4.8

Student Guide

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Object Innovations.

Product and company names mentioned herein are the trademarks or registered trademarks of their respective owners.



™ is a trademark of Object Innovations.

Author: Robert J. Oberg

Copyright ©2018 Object Innovations Enterprises, LLC All rights reserved.

Object Innovations
877-558-7246
www.objectinnovations.net

Published in the United States of America.

EVALUATION COPY

Unauthorized Reproduction or Distribution Prohibited

Table of Contents (Overview)

Chapter 1	Workflow Foundation Conceptual Overview
Chapter 2	Getting Started with WF 4.5
Chapter 3	Primitive and Control Flow Activities
Chapter 4	Custom Activities
Chapter 5	Workflow Hosting
Chapter 6	Collection and Parallel Activities
Chapter 7	More about Custom Activities
Chapter 8	Flowchart and State Machine
Chapter 9	Persistence
Chapter 10	Workflow Services
Chapter 11	Debugging and Error Handling
Appendix A	Learning Resources

Directory Structure

- **Install the course software by running the self-extractor *Install_WfCs_487.exe*.**
- **The course software installs to the root directory *C:\OIC\WfCs*.**
 - Example programs for each chapter are in named subdirectories of chapter directories **Chap01**, **Chap02** and so on.
 - The **Labs** directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.
 - The **Demos** directory is provided for performing in-class demonstrations led by the instructor.
- **The directory *C:\OIC\Data* contains a sample database and command files for creating and removing a SQL Server instance store for persistence of workflows.**
 - The database uses the LocalDB version of SQL Server 2016, which is bundled with Visual Studio 2015.
 - The command files are intended for use with SQL Server 2014 Express, which should be downloaded and installed separately. They are used in Chapter 9 on Persistence.

Table of Contents (Detailed)

Chapter 1: Workflow Foundation Conceptual Overview	1
What Is Workflow?.....	3
Windows Workflow Foundation.....	4
Workflows.....	5
Activities.....	6
Standard Activities.....	7
Runtime Services	8
Workflow Business Scenario.....	9
High Level Workflow	10
Details of While Activity.....	11
Structure of the Solution.....	12
Orders Folder.....	13
Manual Step in the Verification.....	14
Main Console Display	15
Issues Folder	16
Invoices Folder	17
Learning Microsoft's WF	18
Windows Workflow Foundation 3.....	19
Windows Workflow Foundation 4.....	20
Windows Workflow Foundation 4.5.....	21
Summary	22
Chapter 2: Getting Started with WF 4.5.....	23
Workflow Structure	25
Minimal Workflow Program.....	26
Sequence Activity	27
Visual Studio Workflow Projects	28
Workflow Designer Demo.....	29
Variables	33
Assign Activity	34
HelloAssign Workflow	35
C# and Visual Basic Expressions	36
Arguments.....	37
Argument Example	38
Lab 2A	39
Control Flow Activities	40
While Activity.....	41
Lab 2B.....	42
Summary	43
Chapter 3: Primitive and Control Flow Activities.....	53
Built-In Primitive Activities	55
InvokeMethod (Static)	56

Workflow (Static InvokeMethod).....	57
Variables in the Workflow.....	58
InvokeMethod (Instance).....	59
Workflow (Instance InvokeMethod)	60
Invoking .NET Framework Library.....	61
Input in Workflows	62
Prompt Example Workflow	63
Delay Activity.....	64
Lab 3A	65
Control Flow Activities	66
Parallel	67
Parallel Activity Example.....	68
If.....	69
If ... Else	70
While.....	71
DoWhile.....	72
Switch<T>	74
Switch<T> Example	75
Lab 3B.....	78
Summary	79
Chapter 4: Custom Activities.....	93
Why Custom Activities?.....	95
Authoring Custom Activities.....	96
Arguments in Custom Activities.....	97
Activity Class Hierarchy.....	98
CodeActivity	99
CodeActivityContext	100
CodeActivity<TResult>.....	101
CodeActivity Demo	102
Lab 4A	107
Composing Existing Activities	108
Demo: Composing Activities.....	109
Lab 4B.....	113
Summary	114
Chapter 5: Workflow Hosting	127
WorkflowInvoker.....	129
Specifying Input to a Workflow	130
Using Dictionaries	131
Output Argument	132
Invoking a Generic Activity	133
Result Output Argument.....	134
Reusing a Workflow	135
Workflow Timeout.....	136
Timeout with Idle Time	137

Long Computation without Idle Time	139
Invoking the Long Computation	141
Long Computation with Timeout.....	142
Lab 5A	143
Hosting a Workflow in Windows	144
Windows Host Code	145
WorkflowInvoker Instance Methods	146
WorkflowInvoker Asynchronous Methods.....	147
Asynchronous Demonstration.....	148
HelloAsync Code (Invoker).....	150
Example Workflow.....	152
Effect of Sleep.....	153
WorkflowApplication	154
WorkflowApplication Demo	155
Thread Synchronization.....	159
Arguments.....	160
WorkflowApplication Async Demo.....	161
HelloAsync Code (Application)	163
WorkflowApplication Delegates	164
Manual Control of Workflows.....	165
Stopping Workflow Execution	166
Workflow Manual Control Example	167
Enqueue Workflow	168
Dequeue Workflow	169
Host Code for Enqueue Workflow	170
Host Code for Dequeue Workflow	171
Lab 5B.....	172
Hosting a Workflow in ASP.NET	173
ASP.NET Workflow Host	174
Lab 5C.....	175
Summary	176
Chapter 6: Collection and Parallel Activities.....	185
Collection Activities	187
Collection Activities Example	188
Top-Level Workflow	189
Process Command Activity.....	190
Add and Show.....	191
Remove and Clear.....	192
ForEach<T> Activity	193
ParallelForEach<T>.....	194
Factor Workflow Example.....	195
Using ForEach<T>	196
Using ParallelForEach<T>	197
Using AsyncCodeActivity	198
Example Code.....	199

Host Code.....	200
Asynchronous Activities in WF.....	201
Factor.cs	202
FactorNumber.cs	203
Async Coding: BeginExecute	204
Async Coding: EndExecute	205
Lab 6	206
Summary	207
Chapter 7: More about Custom Activities	213
Waiting for Input.....	215
Bookmarks	216
NativeActivity	217
Bookmark Example	218
GetTwoInt Custom Activity	219
Host Program	220
Passing Data on Resume	221
Bookmark Options.....	222
Bookmarks and Threads	223
Threads in Host Code	224
Threads in Workflow	225
Sample Threading Output.....	226
Lab 7	227
A Compute Intensive Workflow.....	228
EnqueueLoop	229
DequeueLoop.....	230
FactorQueueBookmark Solution	231
FactorConsoleWF	232
More Experiments.....	234
Pick Activity	235
Pick Example	236
Get and Check Answer	237
Set Timer.....	238
Summary	239
Chapter 8: Flowchart and State Machine	249
Workflow Modeling Styles.....	251
Flowchart Activities.....	252
Flowchart Activity Designer.....	253
Demo: Absolute Value Flowchart	254
Auto-Connect.....	259
Using FlowDecision	260
Flowchart and Custom Activities	263
Lab 8A	264
State Machine Workflows in WF 4.5	265
State Machine Workflow Modeling	266

State Machine Workflow Example	267
Power On Transition	268
TransitionCommand Trigger	269
Warming State	270
Heated Transition.....	271
Host Program	272
State Machine Activity Designer.....	273
Demo: Timer State Machine.....	274
Timer in the Math Game.....	279
Do Problem State	280
TimeUp Transition.....	281
Time Out State	282
Complete Transition	283
Threading Issue.....	284
Threading Issue Resolved.....	286
Shared Trigger	287
Echo Transition.....	288
Quit Transition.....	289
Lab 8B.....	290
Summary	291
Chapter 9: Persistence.....	307
Long Running Workflows	309
Persistence and Bookmarks	310
Long-Running Workflow Example.....	311
Persistent Term Life Example	312
Workflow Persistence in WF 4.5.....	313
SQL Server Persistence Database.....	314
Host Code to Enable Persistence	315
Persistence Demo.....	316
AutoResetEvent	320
PersistableIdle Event.....	322
How to Persist a Workflow.....	324
Loading a Persisted Workflow.....	325
Unload and Load Example.....	326
StartAndUnloadInstance()	327
LoadAndCompleteInstance().....	328
Stopping and Starting the Host	329
StartWorkflow.....	330
StartWorkflow.....	331
Loading After Data Available.....	332
InitNames.....	333
GetString.....	334
String Commands.....	335
Host Program	337
Lab 9	338

Summary	339
Chapter 10: Workflow Services.....	345
What Is WCF?.....	347
WCF Services	348
WCF = ABC	349
Address, Binding, Contract.....	350
Workflow Services.....	351
Messaging Activities.....	352
Messaging Activity Templates	353
IIS Express Hosting	354
Demo – Creating a Workflow Service.....	355
WCF Test Client	359
Demo – Workflow Services Client	361
Configuration Manager.....	362
Workflow Services Client (Cont'd).....	363
Multiple Operations	366
Multiple Operations via Parallel.....	367
Hosting a Workflow Service.....	368
Lab 10	369
Summary	370
Chapter 11: Debugging and Error Handling.....	377
Debugging Workflows.....	379
Control Flow and Flowchart.....	380
Breakpoint Example.....	381
Breakpoint in XAML.....	382
Exceptions.....	383
Exception Demonstration	384
Account and Bank Classes.....	385
Use of Dictionary.....	386
Deposit and Withdraw	387
Code Activities	388
Composite Activities.....	389
Top-Level Workflow	390
Host Program	391
Unhandled Exceptions	392
Using WorkflowApplication.....	393
Sample Output	394
TryCatch Activity	395
TryCatch Demo.....	396
Transactions	401
Compensation	402
No Compensation.....	403
Using Compensation.....	404
Transfer.xaml	405

Compensable Withdrawal	406
Compensation Token	407
Canceling the Workflow	408
Exceptions and Compensation	409
Compensation Not Performed.....	410
Lab 11	411
Summary	412
Appendix A: Learning Resources.....	421

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Chapter 1

Workflow Foundation Conceptual Overview

Workflow Foundation Conceptual Overview

Objectives

After completing this unit you will be able to:

- **Explain what a workflow is and how Windows Workflow Foundation supports workflow applications.**
- **Describe a typical business scenario for workflow and illustrate with a WF application.**
- **Explain the concepts of workflows and activities.**
- **Describe runtime services provided in WF.**

What Is Workflow?

- **In general terms, a workflow can be thought of as a flow of processes or tasks that produce some result.**
- **Workflows are often concerned with documents that flow through various activities and may spawn other documents as they are processed.**
- **Workflows can be manual, with paper documents being transmitted among people working at different desks in an office, each person performing defined tasks according to specified rules.**
- **We are concerned with workflows as software systems that define the flow of work, the activities performed, and the rules that are employed.**
 - Rules can be expressed declaratively or in code.
 - Activities may be entirely automated or may involve human interaction.
 - Workflows may be distributed among multiple computers in diverse locations.
 - Workflows are typically represented in a graphical manner.

Windows Workflow Foundation

- **Windows Workflow Foundation (WF) is a framework that supports creating and running workflow applications on Windows platforms.**
 - WF consists of a programming model, an engine, and tools.
 - The tools include designers for Visual Studio.
- **WF provides a consistent development experience with other .NET 4.x technologies, including WCF and WPF.**
- **The WF API contains support for both C# and Visual Basic, a special workflow compiler, workflow debugging support, and a visual workflow designer.**
- **Workflows can be developed completely in code or created in conjunction with XAML markup.**
- **The WF model and designer are extensible, enabling developers to create custom activities that encapsulate particular workflow functionality.**

Workflows

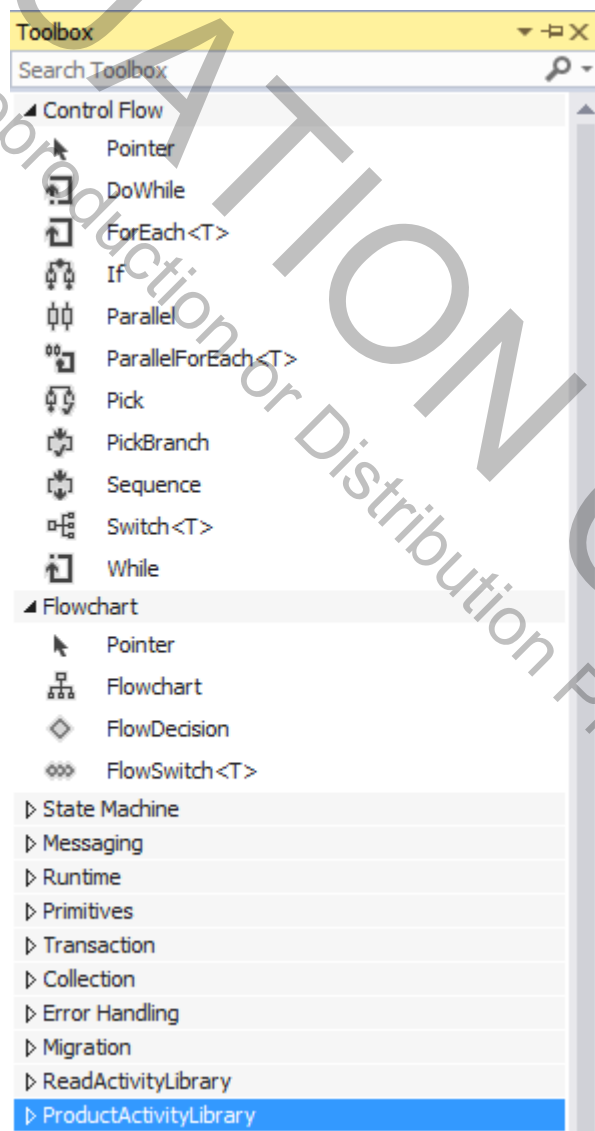
- **A *workflow* is a set of *activities* that are stored in a model describing a business (or other real-world) process.**
- **A workflow describes the order of execution and relationships between units of work.**
 - The units of work may run for a short time or a long time.
 - Activities may be performed by people or the computer.
- **A workflow instance is created and maintained by the *workflow runtime engine*.**
 - There can be several workflow engines within an application domain.
 - Each instance of the engine can support multiple workflow instances.
- **A compiled workflow model can be hosted inside any Windows process, including a console application, a Windows Forms application, a WPF application, a Windows service, an ASP.NET Web application, and a Web service.**

Activities

- **The units of work of a workflow are called *activities*.**
- **When a workflow instance starts, activities are executed in an order as defined by the workflow model.**
 - Both parallel and sequential orders of execution are supported.
 - Conditional and looping behavior of activities is supported.
 - Execution continues until the last activity completes, and the workflow then terminates.
- **Activities can be reused within a workflow and in other workflows.**
- **Activities usually have properties that are configurable.**
- **WF provides many standard activities out-of-the box, and custom activities can be created.**

Standard Activities

- **The Visual Studio Toolbox contains many standard activities that can be dragged onto the surface of the Workflow Designer.**
 - These are arranged in groups, such as Control Flow, Flowchart, State Machine, Messaging, and so on.
 - Custom activities can be provided in activity libraries.



Runtime Services

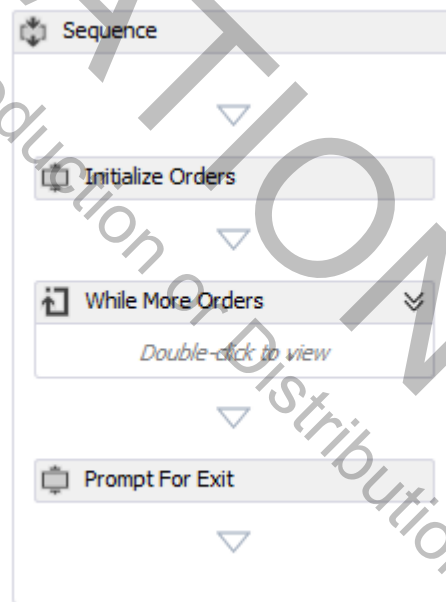
- WF provides a number of out-of-the box *runtime services* that are available in the workflow engine.
- *Persistence services* enable a developer to easily save a WF instance to external storage, such as a database or XML file.
 - This capability enables workflow applications to maintain state and be long-running, surviving application restarts.
- *Transaction services* enable you to maintain transactional integrity in workflow applications.
- *Tracking services* support monitoring and recording workflow execution.
- *Scheduling services* enable you to control how the WF runtime manages threads in your application.

Workflow Business Scenario

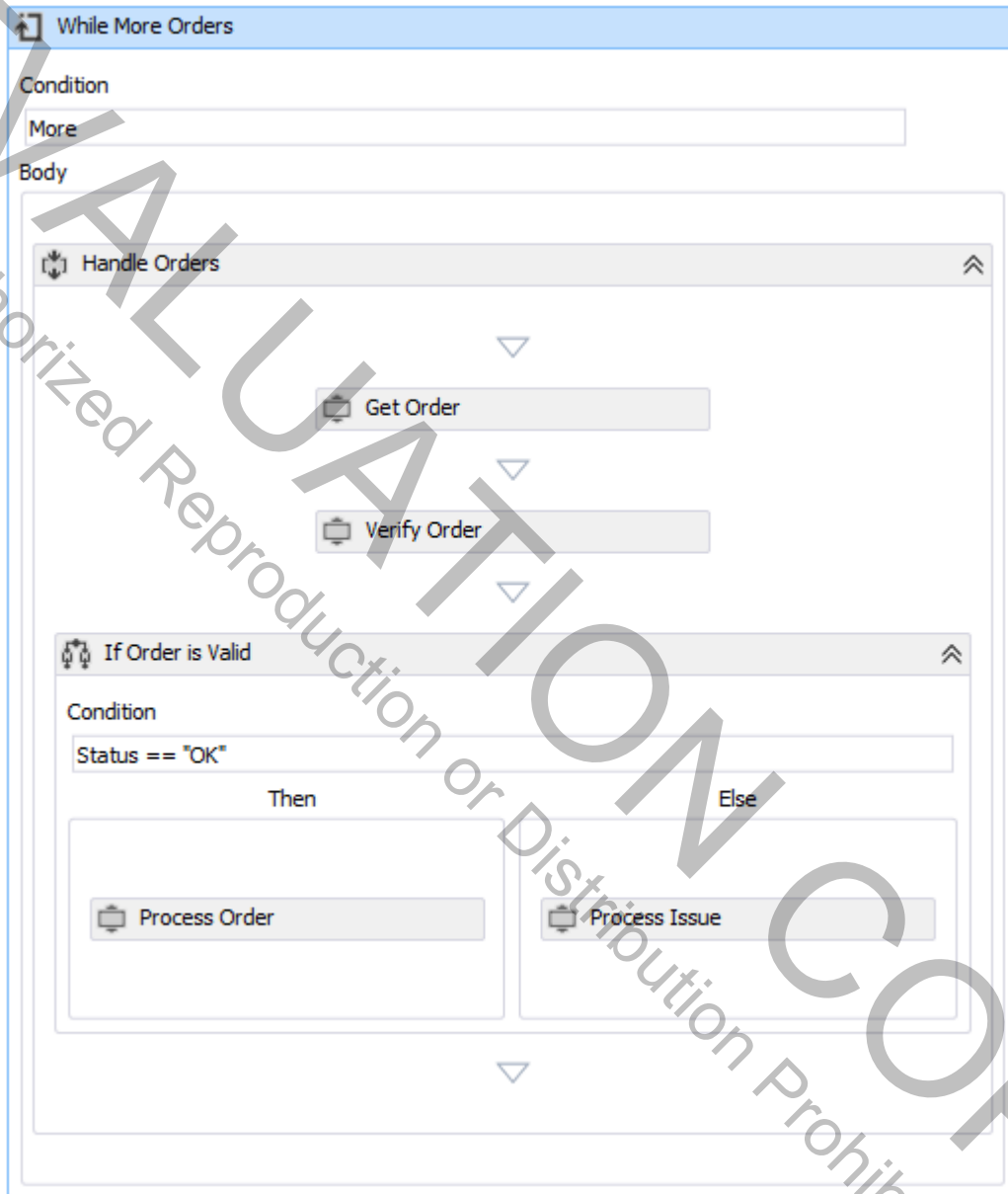
- **To illustrate workflows consider the following business scenario:**
 - As orders are created they are specified in XML files placed in the folder Orders.
 - Each order in the Orders folder is processed, beginning with getting the order information from the XML file.
 - The order is verified, which is done by a person who checks each item in the order for consistency (description furnished by customer matches the description in the vendor database or has a trivial error such as a misspelling). The system also does some verification, making sure that the item ID is found in the database.
 - If the order is valid, it is processed and an invoice is created. The invoice is specified in an XML, stored in the Invoices folder.
 - If the order is not valid, the issue with the order is specified in another XML file, stored in the Issues folder. (A customer service representative can follow up on such orders by emailing or calling the customer.)
- **This scenario is implemented by a workflow application.**
 - See **Chap01\OrderWorkflow**. See the file **Workflow1.xaml** in the **OrderWorkflow** project for a diagram of the workflow.

High Level Workflow

- **Open up the file *Workflow1.xaml*.**
 - You must build the solution before the **Workflow1.xaml** file will load properly in the visual designer.
 - The Workflow Designer opens up, showing a diagram of the workflow.
 - Examine the workflow at a high level by collapsing the **While More Orders** activity.



Details of While Activity



- **The heart of the workflow is a loop that gets and verifies orders.**
 - If order is valid, it is processed.
 - If order is not valid, an issue is processed.

Structure of the Solution

- **The solution consists of three projects:**
 - **OrderLibrary** is an activity library that defines custom activities such as **InitializeOrders**, **GetOrder**, and so on.
 - **OrLib** is an ordinary class library defining the classes that are used in the implementation of the custom activities. This includes code to access a database.
 - **OrderWorkflow** is a console application that contains the workflow itself. It has a graphical representation specified in the XAML file **Workflow1.xaml**.
- **Three folders are provided for XML data files:**
 - The **Orders** folder contains orders to be processed.
 - The **Invoices** folder contains invoices that are created for valid orders.
 - The **Issues** folder contains files describing issues for orders that are not valid.
- **A database of products is provided in the SQL Server database file *Product.mdf* in the *C:\OIC\Data* folder.**
 - The database uses the LocalDB version of SQL Server 2016, which is bundled with Visual Studio 2017.
 - The database access code is encapsulated within the **OrLib** class library.

Orders Folder

- The *Orders* folder contains XML files representing orders. The loop goes through all the files in this folder.

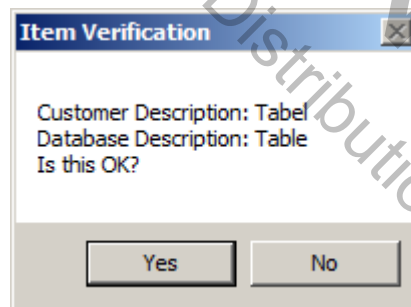
- As an example, consider **1002.xml**. This order is basically valid, but one of the items is slightly questionable.

```
<?xml version="1.0" encoding="utf-8"
standalone="yes"?>
<Order>
  <OrderId>1002</OrderId>
  <Customer>
    <Name>Mary Smith</Name>
    <Email>mary@bar.com</Email>
  </Customer>
  <Item>
    <ItemId>104</ItemId>
    <Description>Sofa</Description>
    <Quantity>1</Quantity>
  </Item>
  <Item>
    <ItemId>102</ItemId>
    <Description>Tabel</Description>
    <Quantity>2</Quantity>
  </Item>
  <Item>
    <ItemId>103</ItemId>
    <Description>Lamp</Description>
    <Quantity>2</Quantity>
  </Item>
</Order>
```

- An automated verification might reject item 102 because of the misspelling of the description.

Manual Step in the Verification

- As is typical in workflow applications, steps in the workflow can be carried out either by the computer or a person.
- As part of the processing of the *VerifyOrder* activity a message box will be displayed for a human to verify each item.
- Build and run the solution.
 - **OrderWorkflow** is the startup project, so this console application will run.
 - As orders are verified, a message box will pop up for each item of each order.
- Here is the message box for item 102 of order 1002.



- The person viewing this will clearly see that this is benign and will click Yes to approve this item.

Main Console Display

- As the workflow is executed, a running report on each item is displayed on a main console.
- Here is the display for order 1002:

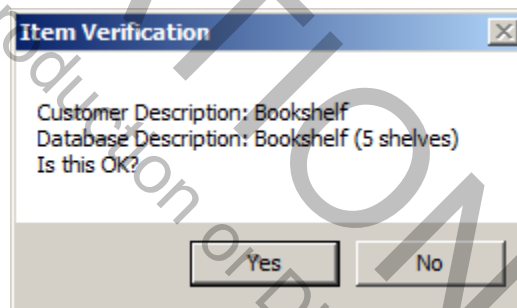
```
C:\OIC\WfCs\Chap01\OrderWorkflow\Orders\1002.xml
OrderId = 1002
Verification message = OK
OrderId = 1002
CustomerName = Mary Smith
CustomerEmail = mary@bar.com
ItemId   Description   Quantity   ...   Price   Extension
104      Sofa           1          ...   $500.00 $500.00
102      Tabel           2          ...   $200.00 $400.00
103      Lamp            2          ...   $50.00  $100.00
Total = $1,000.00
File 1002.xml has been created in folder Invoices
```

- And here is the display for orders 1003 and 1004 for which there were genuine issues.

```
file = 1003.xml
C:\OIC\WfCs\Chap01\OrderWorkflow\Orders\1003.xml
OrderId = 1003
Verification message = Description does not match
Create issue: Description does not match
File 1003.xml has been created in folder Issues
file = 1004.xml
C:\OIC\WfCs\Chap01\OrderWorkflow\Orders\1004.xml
OrderId = 1003
Verification message = Item Not Found
Create issue: Item Not Found
File 1004.xml has been created in folder Issues
Press Enter to exit
```

Issues Folder

- The *Issues* folder will contain an XML file for each order having genuine issues.
- Our example illustrates two kinds of problems.
 - The item is not found in the database, a problem that can be detected automatically, illustrated in order 1004.
 - The item's description indicates a genuine question, such as an ambiguity that should be verified with the customer, illustrated in order 1003.



- The human verifier will reject this item, because the company has several bookshelves having different numbers of shelves, and the customer should be queried to make sure she gets the desired bookshelf.
- Here is the generated issue file **1003.xml**:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Issue>
  <OrderId>1003</OrderId>
  <CustomerName>Bill Jones</CustomerName>
  <CustomerEmail>bill@forest.net</CustomerEmail>
  <Message>Description does not match</Message>
</Issue>
```

Invoices Folder

- **For valid orders an XML file is created from which an invoice can be generated.**
 - The actual invoice that will be sent to the printer is created by a separate subsystem, which could be another workflow, whose input is the files in the **Invoices** folder.
- **As an example, here is the invoice XML file for order 1002.**

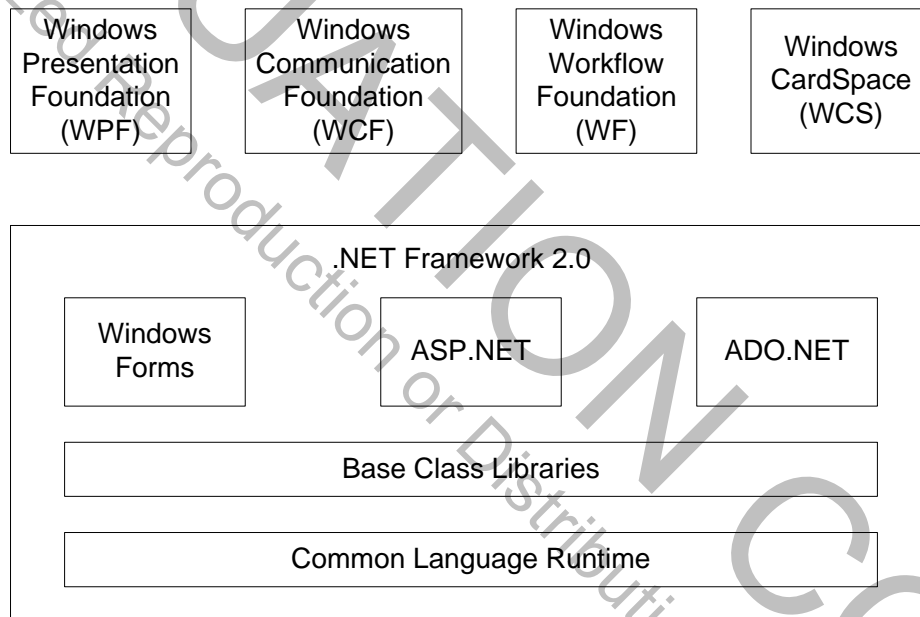
```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Invoice>
  <OrderId>1002</OrderId>
  <CustomerName>Mary Smith</CustomerName>
  <CustomerEmail>mary@bar.com</CustomerEmail>
  <LineItem>
    <ItemId>104</ItemId>
    <Description>Sofa</Description>
    <Quantity>1</Quantity>
    <Price>500.0000</Price>
    <Extension>500.0000</Extension>
  </LineItem>
  <LineItem>
    <ItemId>102</ItemId>
    <Description>Table</Description>
    <Quantity>2</Quantity>
    <Price>200.0000</Price>
    <Extension>400.0000</Extension>
  </LineItem>
  <LineItem>
    <ItemId>103</ItemId>
    <Description>Lamp</Description>
    <Quantity>2</Quantity>
    <Price>50.0000</Price>
    <Extension>100.0000</Extension>
  </LineItem>
  <Total>1000.0000</Total>
</Invoice>
```

Learning Microsoft's WF

- **There are two main challenges in learning to use Windows Workflow Foundation.**
- **The first challenge is to understand the nature of workflow applications, which are quite different in structure from conventional applications.**
 - For this purpose studying a miniature example illustrating an actual business scenario is invaluable.
 - The program **OrderWorkflow** is such an example. (By contrast, a conventional console application is provided in **OrderConsole** in the chapter folder.)
- **The second challenge is to understand the classes and tools in the actual framework.**
 - For this purpose studying small standalone examples, divorced from the complexities of a business situation, will be most helpful.
 - We'll largely follow this second approach in the remainder of the course, beginning with a simple "Hello Workflow" example in Chapter 2.

Windows Workflow Foundation 3

- **Windows Workflow Foundation was introduced as part of .NET Framework 3.0 (formerly called WinFX).**
- **.NET Framework 3.0 layers on top of .NET Framework 2.0 and has the components shown in the diagram.**



- **.NET Framework 3.5 added some important new features, notably Language Integrated Query or LINQ, but remained layered on top of .NET 2.0.**
 - WF 3.5 used the same Workflow programming model as WF 3.0.

Windows Workflow Foundation 4

- **.NET 4.0 is a major revision of the .NET Framework, and the WinFX technologies are no longer layered on top of .NET 3.0.**
- **Windows Workflow Foundation in .NET 4.0 is a complete re-architecting of WF.**
 - WF 4 has a completely new set of assemblies: **System.Activities.*** that are used in place of the **System.Workflow.*** assemblies from WF 3.
 - But WF 4 retains the **System.Workflow.*** assemblies, so WF 3 workflows will run unchanged in WF 4.
- **Major changes in WF 4 include:**
 - A new visual designer, built with WPF, supports the ability to work with much larger workflows.
 - Data flows, which were opaque in WF 3, can now be clearly specified as variables and arguments with familiar typing and scoping.
 - A new Flowchart activity is provided.
 - The programming model has been revamped, making Activity a core base type used for both workflows and activities. The model is now fully declarative, expressible in XAML, with no code-beside.
 - Integration with WCF has been improved, with new messaging activities, and fully declarative service definition.

Windows Workflow Foundation 4.5

- **Windows Workflow Foundation in .NET 4.5/4.6 and Visual Studio 2017 contains new activities, designer capabilities, and a new workflow development model.**
- **A major enhancement for C# programmers is C# expressions.**
 - Prior to .NET 4.5 expressions in workflows had to be written in Visual Basic.
- **.NET 4.5 provides for a state machine development model out-of-the box.**
 - WF 3 supported state machines but WF 4 did not
- **There are a number of new designer capabilities:**
 - Auto-surround with Sequence; Dragging a second activity into a block expecting a single activity will cause the designer to automatically insert a Sequence.
 - There is an auto-connect feature in flowcharts.
 - Breakpoints can be set on states in state machines.
- **Consult MSDN documentation for other enhancements in WF 4.5.**

Summary

- A workflow can be thought of as a flow of processes or tasks that produce some result.
- Windows Workflow Foundation (WF) is a framework that supports creating and running workflow applications on Windows platforms.
- The units of work of a workflow are called *activities*.
- Workflows can be defined using a visual designer.
- Standard activities are provided in a Toolbox, and custom activities can be created in an activity library, also made available in the Toolbox.
- Workflow Foundation 4 is a complete re-architecting of WF.
- WF 4.5 contains new activities, designer capabilities, and a new state machine development model. C# expressions are supported in workflows.

Chapter 2

Getting Started with WF 4.5

Getting Started with WF 4.5

Objectives

After completing this unit you will be able to:

- **Describe the overall structure of workflow programs.**
- **Implement workflows using C# code.**
- **Use the Sequence activity to implement workflows consisting of multiple activities.**
- **Create Workflow projects using Visual Studio 2017.**
- **Implement workflows declaratively in XAML by using the Visual Studio Workflow Designer.**
- **Use the WriteLine and Assign activities in your workflow programs.**
- **Describe the use of variables, arguments and C# expressions in workflow programs.**
- **Describe the Control Flow activities and illustrate using the While activity.**

Workflow Structure

- **The fundamental unit of work in a workflow application is called an *activity*.**
 - In WF an activity is represented by a class that derives ultimately from the class **Activity**.
 - Activities can be composed together to form larger activities.
- **An activity that is used as a top-level entry point is called a *workflow*.**
- **A workflow is executed by a runtime engine.**
 - The simplest way to invoke the runtime engine is through the **WorkflowInvoker** class.
 - This will start the workflow at the top-level activity, just as a CLR program will start at the top-level entry point **Main()**.

Minimal Workflow Program

- See *Minimal* in the chapter folder.

```
using System;
using System.Activities;
using System.Activities.Statements;

namespace HelloCode
{
    class Program
    {
        static void Main(string[] args)
        {
            Activity wf = new WriteLine
            {
                Text = "Hello, world"
            };

            WorkflowInvoker.Invoke(wf);
        }
    }
}
```

- This workflow consists of a single activity **WriteLine**.
- The activity is instantiated, its **Text** property initialized, and passed as a parameter to the static **Invoke()** method of the **WorkflowInvoker** class.

Sequence Activity

- **In most cases a top-level activity will act as a container of other activities.**
 - Examples of such container activities include **Sequence**, **Parallel** and **Flowchart**.
- **A Sequence activity, as its name suggests, executes its contained activities sequentially.**
 - An illustration is provided by the **Sequence** example.

```
Activity wf = new Sequence
{
    Activities =
    {
        new WriteLine
        {
            Text = "Hello, world"
        },
        new WriteLine
        {
            Text = "Goodbye."
        }
    }
};
```

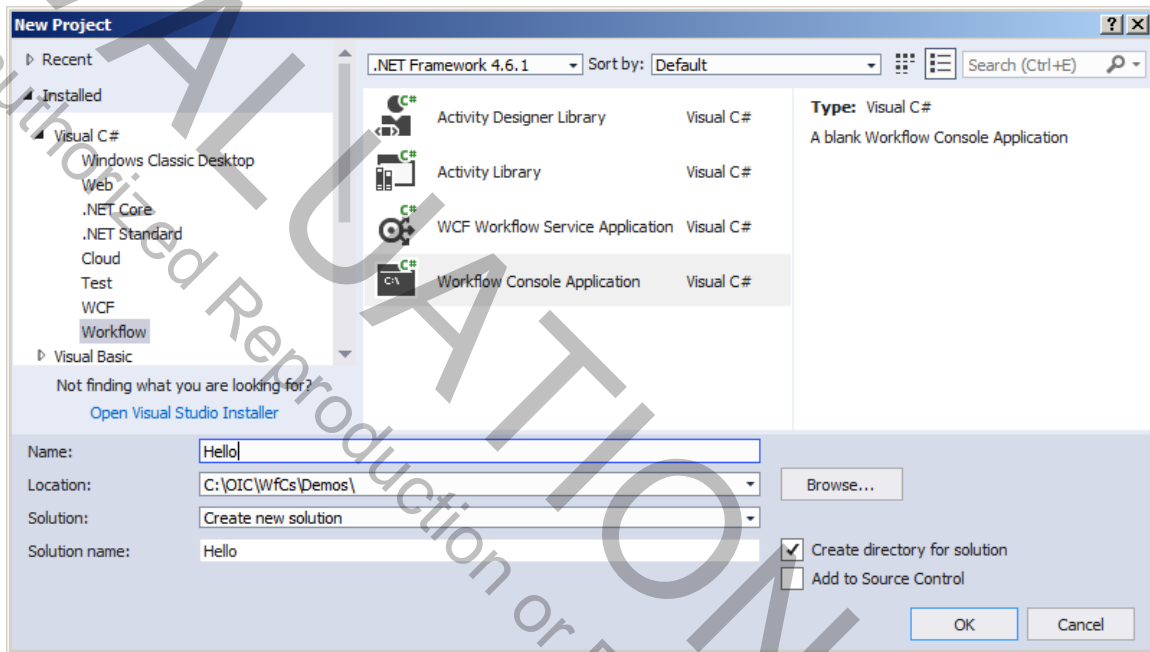
Visual Studio Workflow Projects

- While it is certainly possible to create workflow applications using only C# code, it is much more productive to use Visual Studio¹ and the Workflow Designer.
- Visual Studio provides several workflow project templates:
 - Activity Designer Library
 - Activity Library
 - WCF Workflow Service Application
 - Workflow Console Application
- We'll begin with Workflow Console Applications.
- Let's use Visual Studio to create a "Hello" application, and we'll use the visual Workflow Designer after our project has been created.

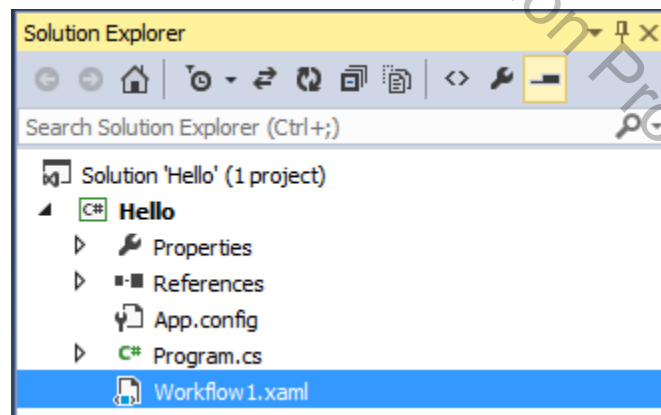
¹ The Workflow Designer is now available in the free Visual Studio Community 2017.

Workflow Designer Demo

1. Create a new Workflow Console Application **Hello** in the **Demos** folder. The necessary template is in the Workflow group in the Toolbox.

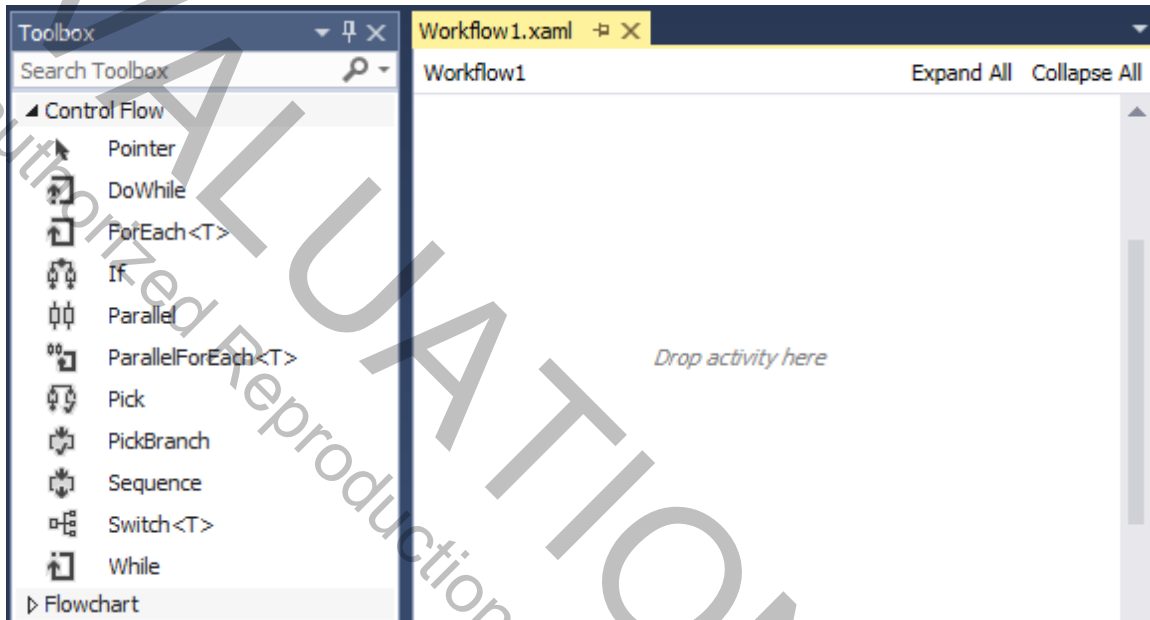


2. Examine the files of your new project in Solution Explorer. The workflow will be defined declaratively in the file **Workflow1.xaml**. It is open in the visual Workflow Designer.

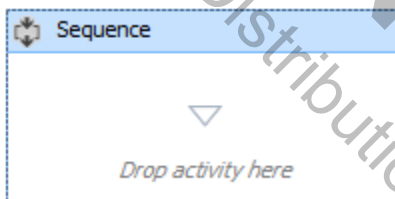


Workflow Designer Demo (Cont'd)

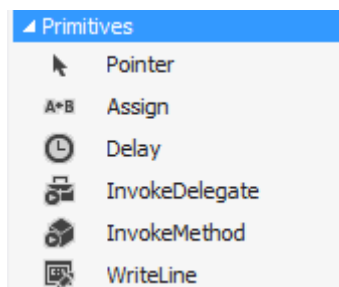
3. There is a Toolbox containing a palette of built-in activities. Expand the Control Flow group.



4. Drag a Sequence activity onto the center of the design canvas.

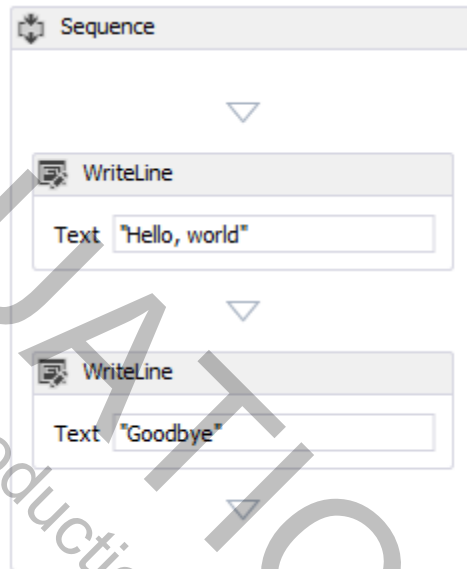


5. Expand the Primitives group in the Toolbox.



Workflow Designer Demo (Cont'd)

6. Drag two WriteLine activities onto the Sequence activity. Specify the Text property as “Hello, world” and “Goodbye”.



7. Build and run (without debugging). You'll see this output:

```
Hello, world
Goodbye
Press any key to continue . . .
```

8. Examine the C# code in **Program.cs**.

```
class Program
{
    static void Main(string[] args)
    {
        WorkflowInvoker.Invoke(new Workflow1());
    }
}
```

Workflow Designer Demo (Cont'd)

9. The workflow has been specified declaratively in the file **Workflow1.xaml**. Normally you will work with the visual representation. If you want to see the XAML, right-click over the file in Solution Explorer and choose View Code ... from the context menu. To go back to the visual representation, choose View Designer.

```
<Activity mc:Ignorable="sap sap2010 sads" x:Class="Hello.Workflow1" sap2010:ExpressionActivity>
  xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:sads="http://schemas.microsoft.com/netfx/2010/xaml/activities/debugger"
  xmlns:sap="http://schemas.microsoft.com/netfx/2009/xaml/activities/presentation"
  xmlns:sap2010="http://schemas.microsoft.com/netfx/2010/xaml/activities/presentation"
  xmlns:scg="clr-namespace:System.Collections.Generic;assembly=mscorlib"
  xmlns:sco="clr-namespace:System.Collections.ObjectModel;assembly=mscorlib"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <TextExpression.NamespacesForImplementation>...</TextExpression.NamespacesForImplementation>
    <TextExpression.ReferencesForImplementation>...</TextExpression.ReferencesForImplementation>
    <Sequence sap2010:WorkflowViewState.IdRef="Sequence_1">
      <WriteLine sap2010:WorkflowViewState.IdRef="WriteLine_1" Text="Hello, world" />
      <WriteLine sap2010:WorkflowViewState.IdRef="WriteLine_2" Text="Goodbye" />
      <sads:DebugSymbol.Symbol>dy1D01xPSUNCv3BmVmJcRGVtb3NcSGVsbG9cSGVsbG9cV29ya2Zsb3cxLnhhbWwFJ,

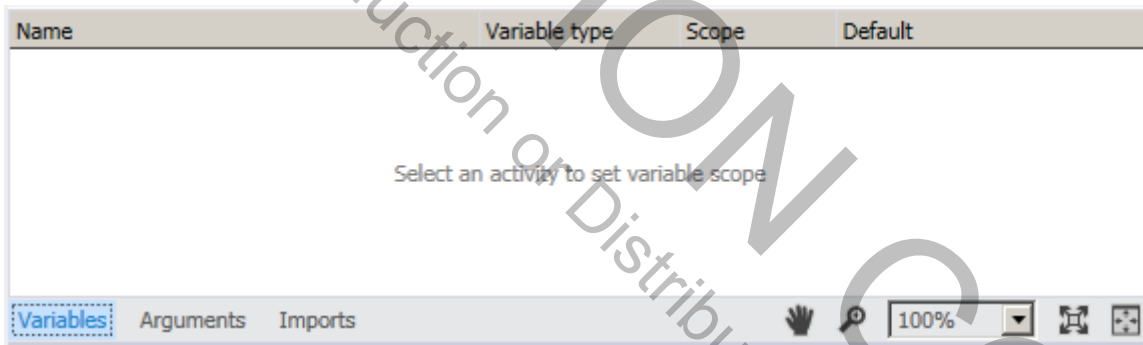
```

10. You have created a simple workflow without writing any C# code! The program is saved in **HelloXaml** in the chapter folder.

Variables

- You can use *variables* in workflow programs.
 - Define variables using the Workflow Designer. See the Variables tab at the bottom of the design surface.
 - You can assign values to variables using the **Assign** activity, which is part of the Primitives group.
- Let's illustrate by creating another version of our Hello program. Continue your work in the *Demos* folder.

1. Click on the Variables tab.



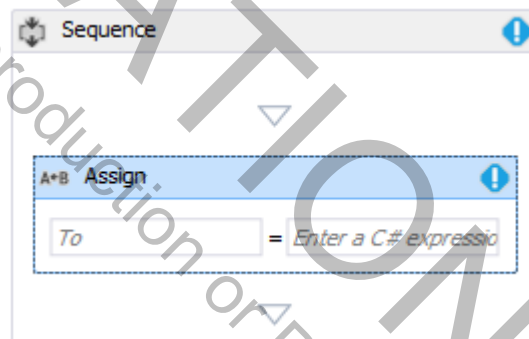
2. Note that in WF variables have **scope** as in programming languages. Click on the **Sequence** activity to specify a scope of the whole sequence, including the contained **WriteLine** activities.
3. You can now create a variable. Call it **Name**, with type **String** and Default value of “world”.

Name	Variable type	Scope	Default
Name	String	Sequence	"world"

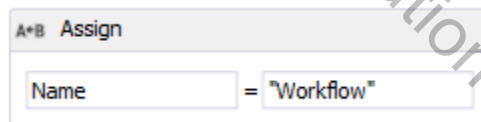
Create Variable

Assign Activity

- Another useful primitive activity is *Assign*.
 - This activity assigns a value to a variable.
 - The semantics of the Assign activity require that the **To** (left-hand side) must be a variable, but the right-hand side may be a variable, a literal, or an expression².
4. Drag an Assign activity from the Toolbox to the top of the Sequence.



5. Specify **Name** for the To and **“Workflow”** for the C# expression.



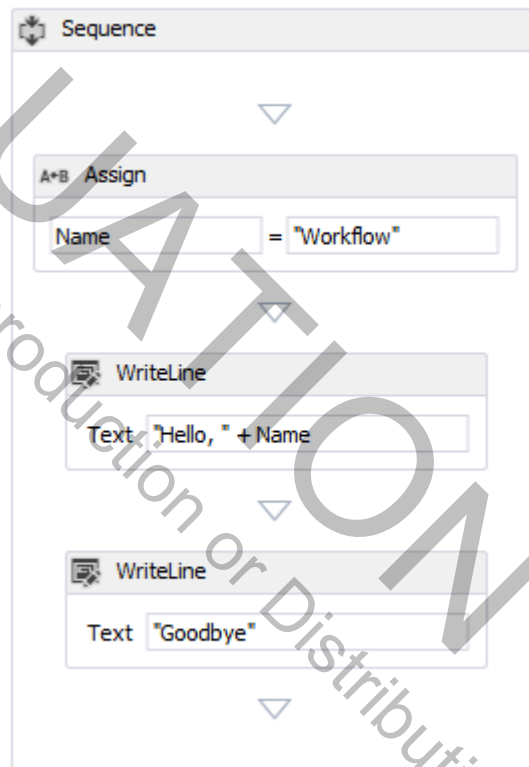
6. Edit the following WriteLine activity to provide this expression for the Text argument:

"Hello, " + Name

² In WF 4.5 you use C# expression for C# Workflow projects. Previously VB expressions were used.

HelloAssign Workflow

- The completed demo is saved as *HelloAssign* in the chapter folder.
 - Here is the final workflow:

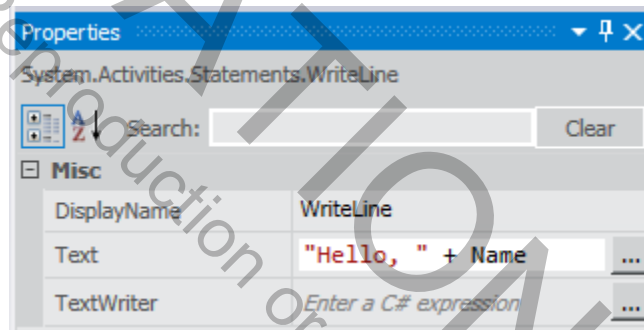


- **Build and run (without debugging):**

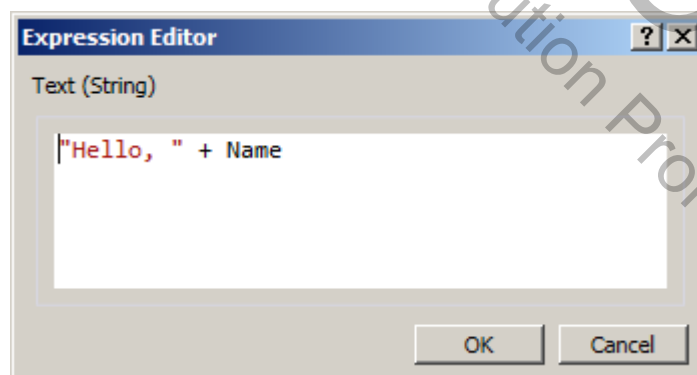
```
Hello, Workflow
Goodbye
Press any key to continue . . .
```

C# and Visual Basic Expressions

- In WF 4.5 new workflows created with C# use C# expression syntax and workflows created with Visual Basic use VB syntax.
 - In WF 4 expressions always used VB syntax.
- Besides using the special interface provides by the Workflow Designer for the WriteLine activity, you could enter the expression via the Properties window.

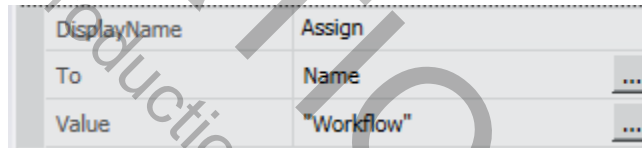


- For long expressions, click the  button to bring up the Expression Editor.

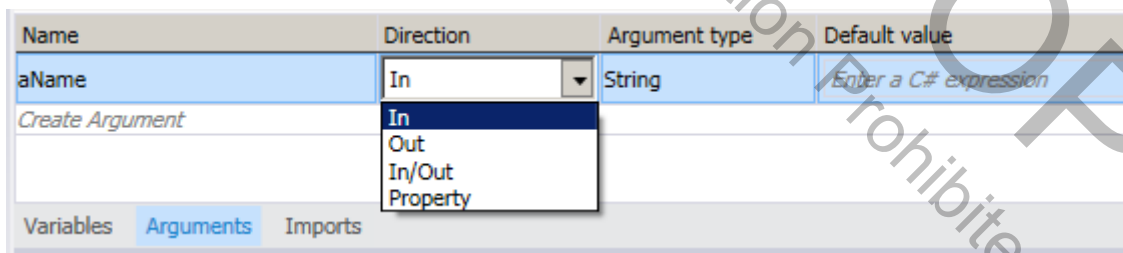


Arguments

- Besides variables and expressions, the data flow model in Workflow Foundation uses *arguments*.
- Arguments specify the inputs and outputs of activities.
- For example, the Assign activity has input argument *Value* and output argument *To*.
 - There is also a **DisplayName** property that is used by the Designer in the visual display of the activity.



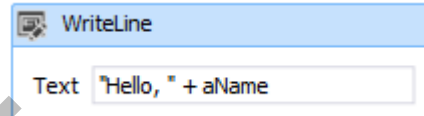
- You can also define your own arguments for a workflow in the Designer.
 - Click the Arguments tab at the bottom of the Designer window.



- We are creating an argument **aName** whose direction is **In**.

Argument Example

- Modifying our “Hello” example, we delete the Assign activity and the variable *Name*. Instead we will use the argument *aName* in the first WriteLine activity.



- We pass the argument in a Dictionary in a second parameter to the *Invoke()* method in *Program.cs*.

```
class Program
{
    static void Main(string[] args)
    {
        WorkflowInvoker.Invoke(new Workflow1(),
            new Dictionary<string, object>
            {
                {"aName", "Argument demo"}
            });
    }
}
```

- Note that the **Dictionary** class is in the namespace **System.Collections.Generic**.

- **Build and run:**

```
Hello, Argument demo
Goodbye
```

- Our example program is *HelloArgument* in the chapter folder.

Lab 2A

A Calculator Workflow

In this lab you will implement a very simple calculator workflow that will add and subtract two integers. As a variation, you will allow the user of the program to enter the integers at the command line, and the results will be passed back by arguments.

Detailed instructions are contained in the Lab 2A write-up at the end of the chapter.

Suggested time: 30 minutes

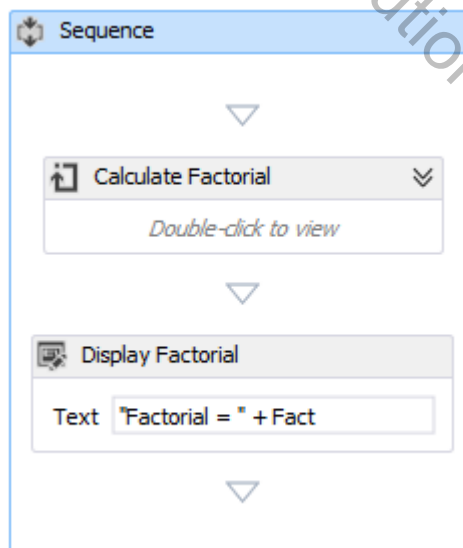
Control Flow Activities

- **The Built-In Activity Library provides a number of activities to support a variety of control flow patterns.**

- The simplest is Sequence, which provides for the sequential execution of activities.
- Looping behavior can be implemented with While, DoWhile and ForEach activities.
- Conditional execution can be implemented with If and Switch activities.
- These and other control flow activities are discussed in Chapter 3.

- **Let's illustrate control flow using While.**

- See **FactorialWhile** in the chapter folder, which calculates the factorial of a positive integer N.
- Here is the top-level workflow:



While Activity

- The logic for calculating the factorial of N can be expressed in this C# code:

```
int N = 5;
int Fact = 1;
while (N > 1)
{
    Fact = Fact * N;
    N = N - 1;
}
```

- Our workflow program implements the same logic using the While activity.
 - Expand the first activity of the Sequence.

The screenshot shows a workflow editor window titled "Calculate Factorial". It has a "Condition" field with the text "N > 1" and a "Body" section. Inside the "Body" is a "Sequence" activity. The "Sequence" activity contains two "Assign" steps. The first "Assign" step has the variable "Fact" on the left and the expression "Fact * N" on the right. The second "Assign" step has the variable "N" on the left and the expression "N - 1" on the right. The workflow editor has a blue header bar and a light blue background. A large, diagonal watermark reading "Unauthorized Production or Distribution Prohibited" is overlaid on the image.

Lab 2B

Exercises Using While and DoWhile

In this lab you will implement several exercises using the While and DoWhile activities.

Detailed instructions are contained in the Lab 2B write-up at the end of the chapter.

Suggested time: 45 minutes

Summary

- The fundamental unit of work in a workflow application is called an *activity*.
- An activity that is used as a top-level entry point is called a *workflow*.
- It is possible to implement workflows using only C# code.
- Visual Studio 2017 has several templates for creating workflow applications.
- You can implement workflows declaratively in XAML by using the Visual Studio Workflow Designer.
- With the Sequence activity you can implement workflows consisting of multiple activities.
- There are a number of primitive built-in activities, including WriteLine and Assign.
- WF 4.5 has a clear model for data flow using variables, arguments and expressions.
- There are a number of built-in Control Flow activities such as While and If.

Lab 2A

A Calculator Workflow

Introduction

In this lab you will implement a very simple calculator workflow that will add and subtract two integers. As a variation, you will allow the user of the program to enter the integers at the command line, and the results will be passed back by arguments. Some hints are provided after the statements of the exercises.

Suggested Time: 30 minutes

Root Directory: OIC\WfCs

Directories:	Labs\Lab2A	(do your work here)
	Chap02\Calculator\Step1	(answer to Part 1)
	Chap02\Calculator\Step2	(answer to Part 2)
	Chap02\Calculator\Step3	(answer to Part 3)

Part 1

Create a Workflow Console Application **Calculator** with the following features:

- A Sequence that contains two Assign and two WriteLine activities.
- Integer variables X and Y with default values.
- Integer variables Sum and Diff
- Calculate the sum and difference in the two Assigns and display them in the two WriteLines.

The finished workflow is shown on the following page.

Part 2

Modify your program so that X and Y are now arguments rather than variables. The values you previously used as default values should now be passed in a dictionary that is used as a second parameter. These values are hardcoded in **Program.cs**.

Part 3

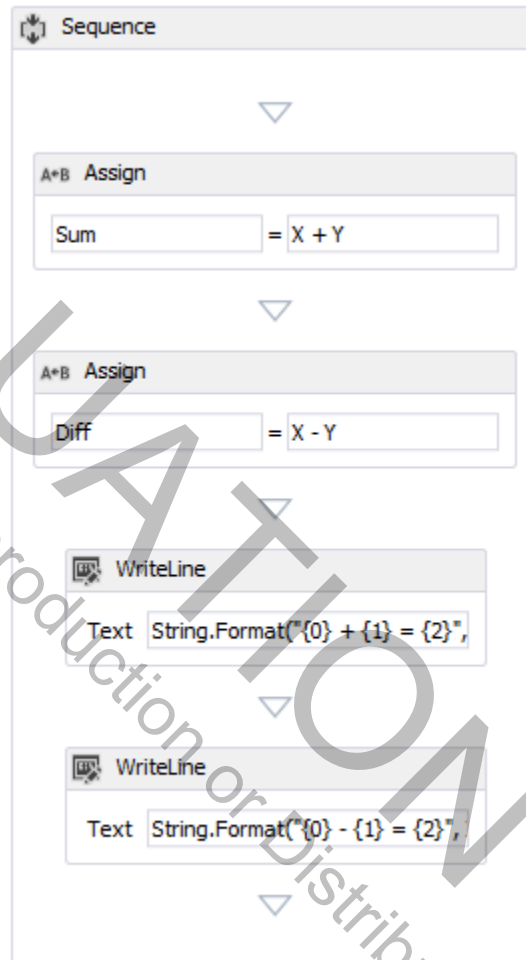
Modify your program so that instead the values to use for X and Y are passed as command-line parameters. You can then start a command prompt from Windows, navigate to the **bin\Debug** folder, and enter a command such as this:

```
Calculator 13 8
```

As a final touch, change the **DisplayName** of your activities to be more descriptive.

Hints for Part 1

Here is the workflow:



Here are the definitions of the variables:

Name	Variable type	Scope	Default
X	Int32	Sequence	13
Y	Int32	Sequence	8
Sum	Int32	Sequence	<i>Enter a C# expression</i>
Diff	Int32	Sequence	<i>Enter a C# expression</i>
<i>Create Variable</i>			

Here are the C# expressions for the two WriteLine:

```
String.Format("{0} + {1} = {2}", X, Y, Sum)
```

```
String.Format("{0} - {1} = {2}", X, Y, Diff)
```

Hints for Part 2

X and Y are deleted from the list of Variables and are now in the list of Arguments.

Name	Direction	Argument type	Default value
X	In	Int32	<i>Enter a C# expression</i>
Y	In	Int32	<i>Enter a C# expression</i>
<i>Create Argument</i>			

The code in **Program.cs** is modified to create a dictionary of input arguments. This dictionary is passed as a second parameter to the **WorkflowInvoker.Invoke()** method.

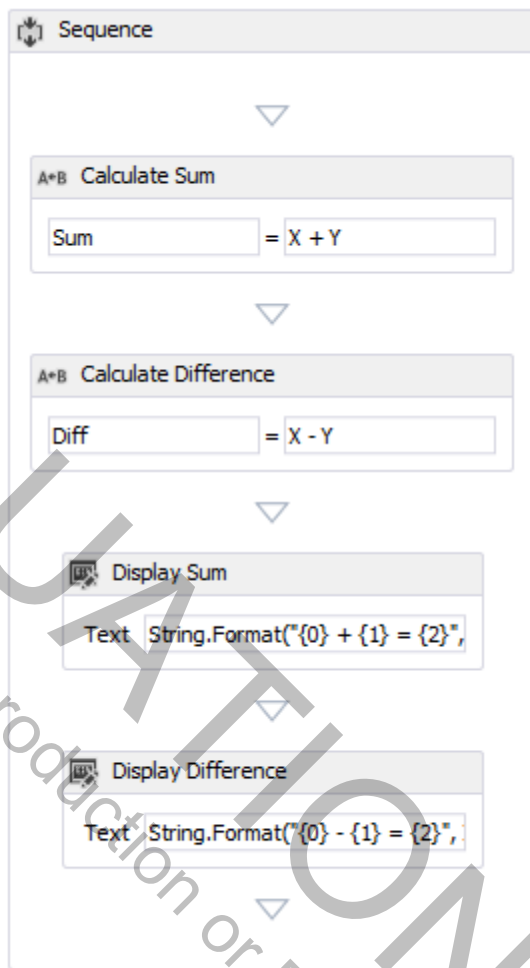
```
class Program
{
    static void Main(string[] args)
    {
        WorkflowInvoker.Invoke(new Workflow1(),
            new Dictionary<string, object>
            {
                {"X", 13},
                {"Y", 8}
            });
    }
}
```

Hints for Part 3

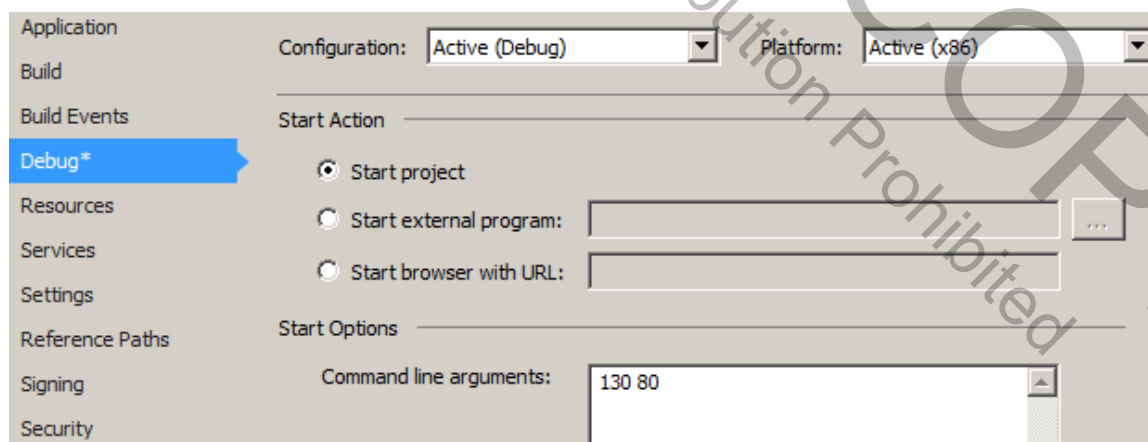
In place of hardcoding values for “X” and “Y”, we now obtain the values from the command line. Since the **Main()** method receives arguments as strings, we must parse these arguments to obtain integers. Here is the code in **Program.cs**:

```
class Program
{
    static void Main(string[] args)
    {
        int x = Int32.Parse(args[0]);
        int y = Int32.Parse(args[1]);
        WorkflowInvoker.Invoke(new Workflow1(),
            new Dictionary<string, object>
            {
                {"X", x},
                {"Y", y}
            });
    }
}
```

The workflow with more descriptive DisplayNames is shown on the next page.



You can run the program at the command line and provide arguments then, or you may specify command line arguments in the Debug options of the project properties.



Lab 2B

Exercises Using While and DoWhile

Introduction

In this lab you will implement several exercises using the While and DoWhile activities.

Suggested Time: 45 minutes

Root Directory: OIC\WfCs

Directories:	Labs\Lab2B	(do your work here)
	Chap02\ChristmasDoWhile	(answer to Exercise 1)
	Chap02\ArrayWF\Step1	(answer to Exercise 2)
	Chap02\ArrayWF\Step2	(answer to Exercise 3)

Exercise 1

Create a Workflow Console Application **Christmas** that will display the number of gifts giving each day in the “twelve days of Christmas” and the total number of gifts given in all. Recall from the song that

On day 1 there is 1 gift
 On day 2 there are $1 + 2 = 3$ gifts
 On day 3 there are $1 + 2 + 3 = 6$ gifts
 And so on.

Use variables Day, Gifts and Total. Use a DoWhile activity. The logic can be expressed in this C# code:

```
int Day = 0;
int Gifts = 0;
int Total = 0;
do
{
    Day += 1;
    Gifts += Day;
    Total += Gifts;
    Console.WriteLine("On day {0} number of gifts = {1}",
        Day, Gifts);
}
while (Day < 12);
Console.WriteLine("Total number of gifts = {0}", Total);
```

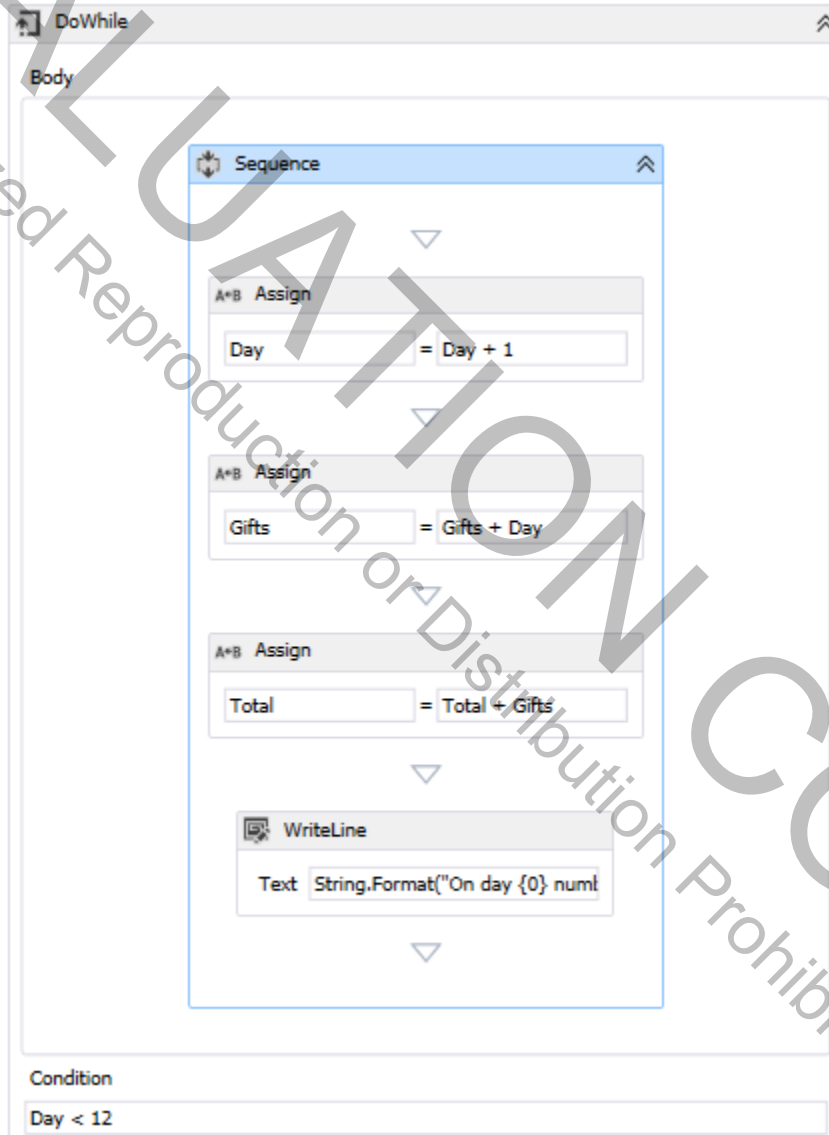
The finished workflow is shown on the following page.

Exercise 2

Create a Workflow Console Application **ArrayWF** that will calculate the sum and product of all numbers in an integer array. Your logic should handle the case of an empty array as well as an array with several integers.

Hints for Exercise 1

The heart of the workflow is this DoWhile activity:



Within the loop this C# expression displays number of gifts given on a day:

```
String.Format("On day {0} number of gifts = {1}", Day, Gifts)
```

After the loop this C# expression displays the total number of gifts given:

```
String.Format("Total number of gifts = {0}", Total)
```

Hints for Exercise 2

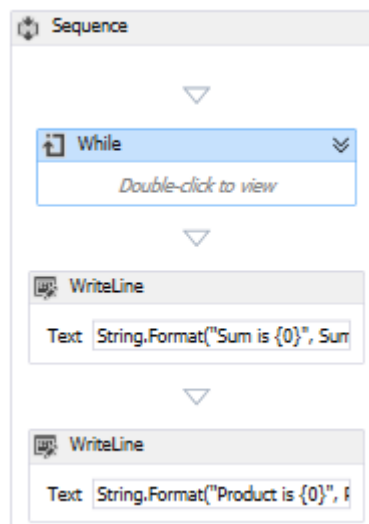
The logic is expressed in this C# code. Commented out is initialization of an empty array. Since we want to allow for the possibility of going through the loop zero times, we should use a while loop rather than a do while.

```
int[] arr = { 2, 3, 5 };
//int[] arr = { };
int sum = 0;
int prod = 1;
int i = 0;
while (i < arr.Length)
{
    sum += arr[i];
    prod *= arr[i];
    i += 1;
}
Console.WriteLine("Sum = {0}", sum);
Console.WriteLine("Product = {0}", prod);
```

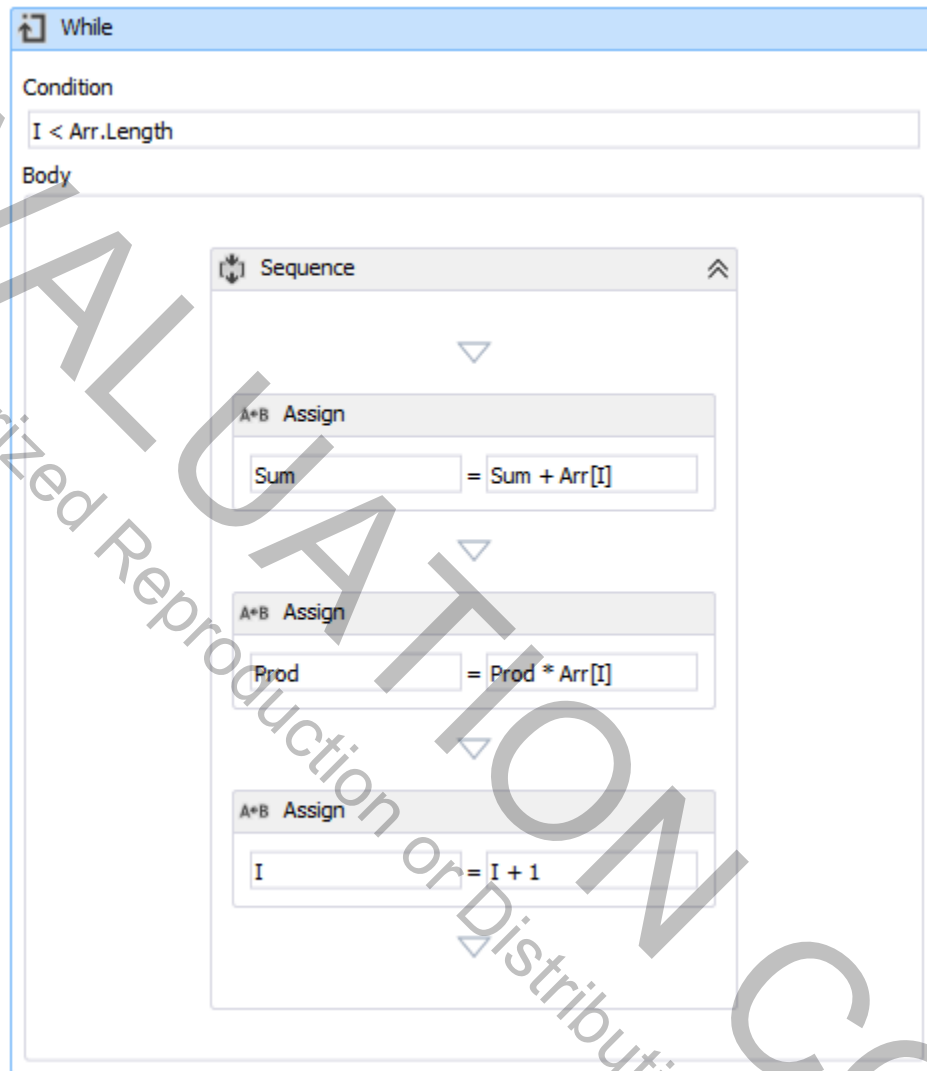
Here are suitable variables, scoped to the top-level Sequence. The syntax for the array initialization is a little tricky.

Name	Variable type	Scope	Default
Arr	Int32[]	Sequence	new int[] { 2, 3, 5 }
Sum	Int32	Sequence	0
Prod	Int32	Sequence	1
I	Int32	Sequence	0

Here is the top-level workflow:



Here is the workflow of the While activity:



Exercise 3 (Optional)

Modify your program to test via passing array elements in at the command line. Your **Program.cs** should allow for both empty and non-empty argument lists.

Hint for Exercise 3

You need to initialize an array in **Program.cs** and pass this array to the workflow as an argument in a dictionary. You could first initialize the array by simple assignment:

```
int[] arr = { 2, 3, 5 };
```

Here is the code to pass the array to the workflow via an argument in a dictionary:

```
WorkflowInvoker.Invoke(new Workflow1(),
    new Dictionary<string, object>
    {
        {"Arr", arr}
    });
```

You also have to create a String argument **Arr** in **Workflow1.xaml**.

Name	Direction	Argument type	Default value
Arr	In	Int32[]	<i>Enter a C# expression</i>
<i>Create Argument</i>			

When this is working, you can create the C# code to build the array from arguments entered at the command line.

```
int[] arr;
if (args.Length > 0)
{
    arr = new int[args.Length];
    for (int i = 0; i < args.Length; i++)
    {
        arr[i] = Int32.Parse(args[i]);
    }
}
else
    arr = new int[] {};
```



7400 E. Orchard Road, Suite 1450 N
Greenwood Village, Colorado 80111
Ph: 303-302-5280
www.ITCourseware.com