

# Windows Workflow Foundation Tutorial

PDF: <http://ceres.napier.ac.uk/staff/bill/wwf.pdf>

Lecture: [http://ceres.napier.ac.uk/staff/bill/e\\_presentations/dotnet\\_workflow/dotnet\\_workflow.htm](http://ceres.napier.ac.uk/staff/bill/e_presentations/dotnet_workflow/dotnet_workflow.htm)

## A Sequential Workflows

- 1a. Create a Sequential Console Workflow, as defined in Figure 1, so that a user enters a value of *i* (codeActivity2), and the IfElseActivity1 branches to codeActivity1 if the value is less than 5, or to codeActivity3 if not.

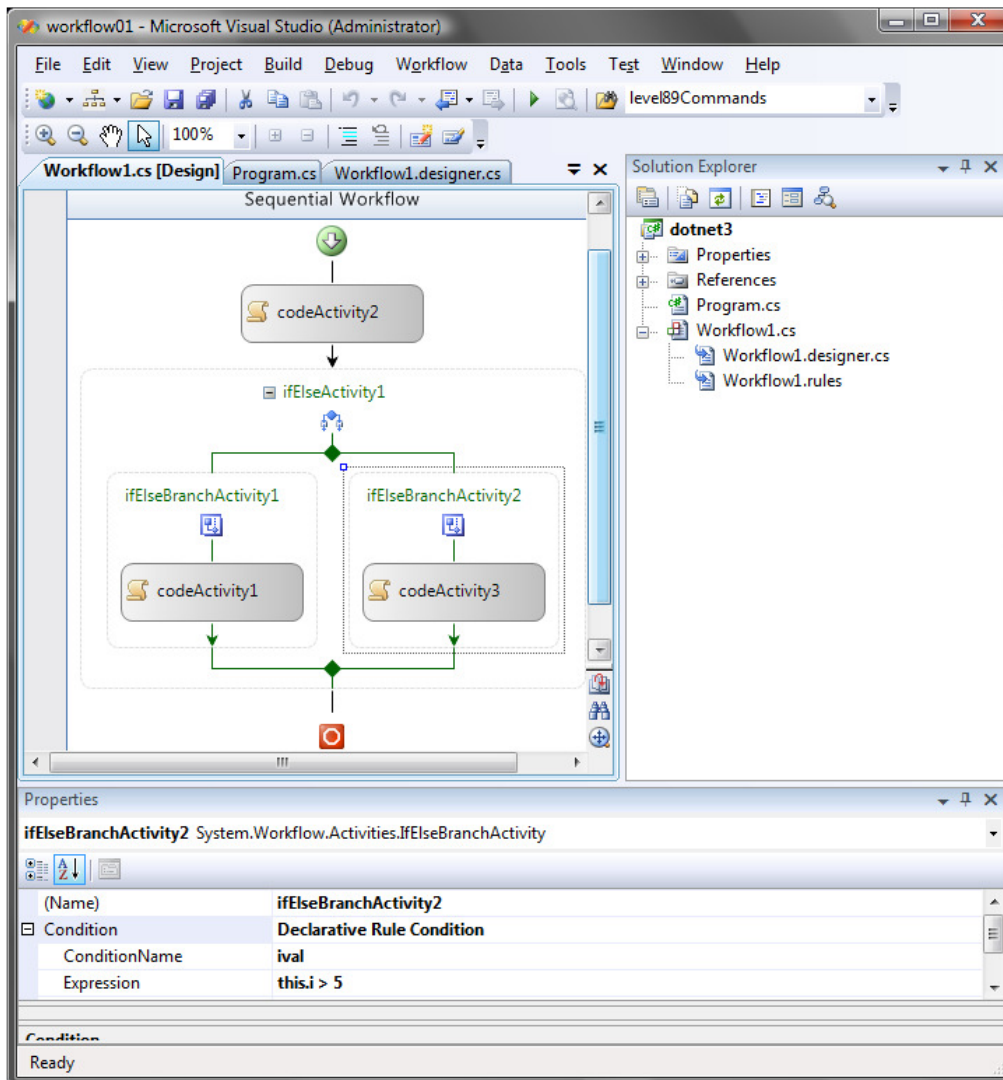


Figure 1:

An outline of the code is:

```
public sealed partial class Workflow1 : SequentialWorkflowActivity
{
```

```

int i;
public Workflow1()
{
    InitializeComponent();
}
private void codeActivity1_ExecuteCode(object sender, EventArgs e)
{
    Console.WriteLine("Value less than 5");
    Console.ReadLine();
}
private void codeActivity1_ExecuteCode_1(object sender, EventArgs e)
{
    Console.WriteLine("Enter i > >");
    i = Convert.ToInt32(Console.ReadLine());
}
private void codeActivity3_ExecuteCode(object sender, EventArgs e)
{
    Console.WriteLine("Value greater than 5");
    Console.ReadLine();
}

```

- 1b.** Run the program, and prove its operation.
- 2a.** Create a Sequential Console Workflow, as defined in Figure 2, so that it displays the number of prime numbers within a given range. With the parallelActivity1, codeActivity1 should calculate the primes from 3 to 100, and codeActivity2 should calculate the primes from 101 to 200. Each of the activities should set a flag on their completion, and the whileActivity1 will pick up when both flags are complete.

An outline of the code is:

```

public sealed partial class Workflow1 : SequentialWorkflowActivity
{
    ArrayList primes1 = new ArrayList();
    bool flag1 = false;
    ArrayList primes2 = new ArrayList();
    bool flag2 = false;
    public Workflow1()
    {
        InitializeComponent();
    }
    private void codeActivity1_ExecuteCode(object sender, EventArgs e)
    {
        for (ulong i = 3; i < 100; i++)
        {
            if (isprime(i)) primes1.Add(i);
        }
        this.flag1 = true;
    }
}

```

```

private void codeActivity2_ExecuteCode(object sender, EventArgs e)
{
    for (ulong i = 100; i < 200; i++)
    {
        if (isprime(i)) primes2.Add(i);
    }
    this.flag2 = true;
}
private void codeActivity3_ExecuteCode(object sender, EventArgs e)
{
    Console.WriteLine("0-100, No of primes=" +
        Convert.ToString(primes1.Count));
    Console.WriteLine(Convert.ToString(primes1[0]+"..." +
        Convert.ToString(primes1[primes1.Count-1]));
    Console.WriteLine("100-200, No of primes=" +
        Convert.ToString(primes2.Count));
    Console.WriteLine(Convert.ToString(primes2[0]+"..." +
        Convert.ToString(primes2[primes2.Count-1]));
    Console.ReadLine();
    flag1 = false;
}
public bool isprime(ulong val)
{
    for (ulong i=2;i<val/2;i++)
    {
        if ((val % i)==0) return false;
    }
    return true;
}
}
}

```

2b. Next determine the number of primes within the following ranges (the first one has been completed):

|           |                      |    |                          |     |
|-----------|----------------------|----|--------------------------|-----|
| 3-200     | <b>No of primes:</b> | 46 | <b>End prime number:</b> | 199 |
| 3-1,000   | <b>No of primes:</b> |    | <b>End prime number:</b> |     |
| 3-10,000  | <b>No of primes:</b> |    | <b>End prime number:</b> |     |
| 3-100,000 | <b>No of primes:</b> |    | <b>End prime number:</b> |     |

2c. Next determine the time taken for each of the parallel tasks to run. For this add the following when the workflow starts:

```
DateTime start = DateTime.Now;
```

and then the following on each of the task:

```
DateTime end1 = DateTime.Now;
```

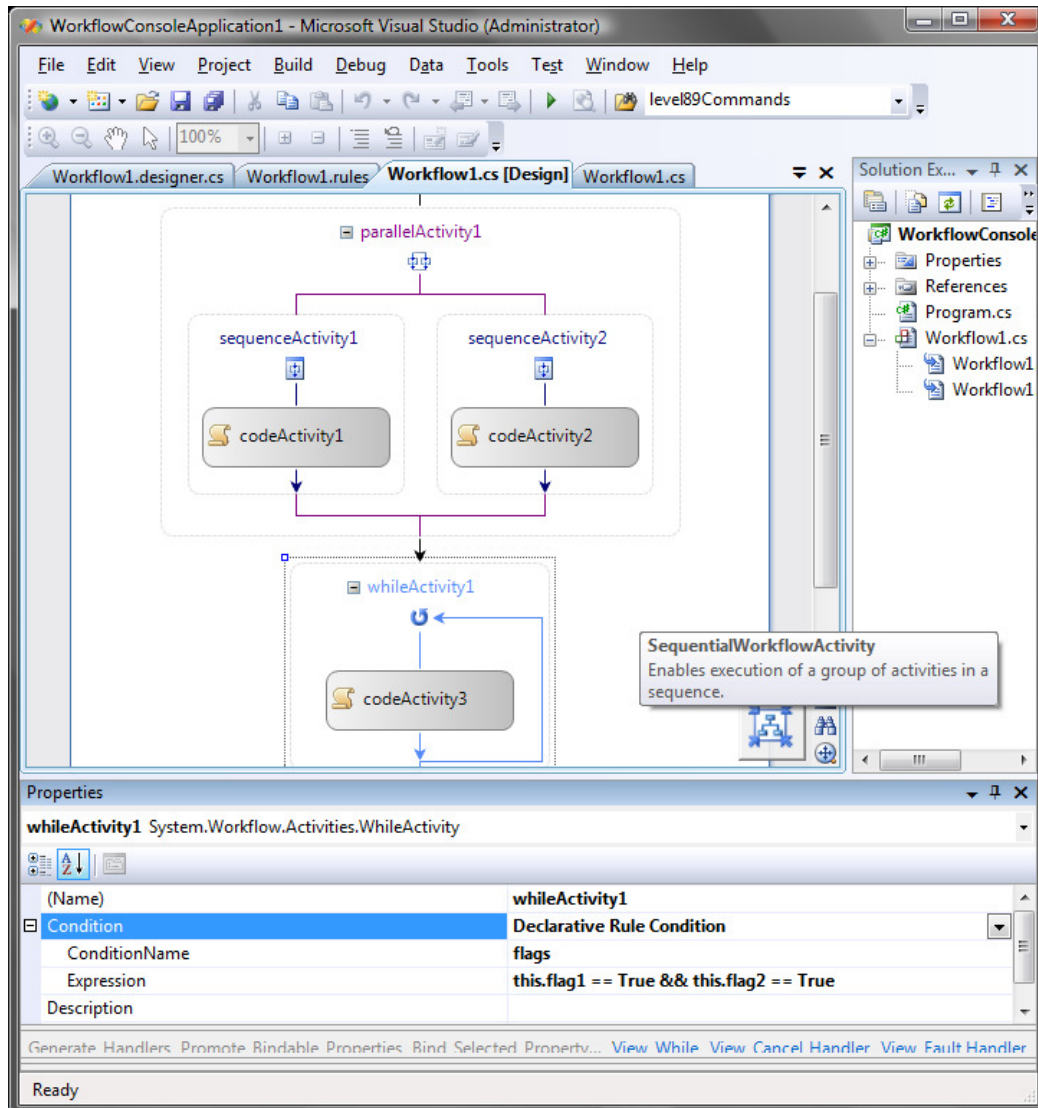
```
DateTime end2 = DateTime.Now;
```

and then the following on each of the task:

```
TimeSpan duration = end1 - start;  
Console.WriteLine("Activity 1: "+duration);
```

and complete the following:

|           |                     |
|-----------|---------------------|
| 3-200     | <b>Time to run:</b> |
| 3-1,000   | <b>Time to run:</b> |
| 3-10,000  | <b>Time to run:</b> |
| 3-100,000 | <b>Time to run:</b> |
| 3-500,000 | <b>Time to run:</b> |



**Figure 2:**

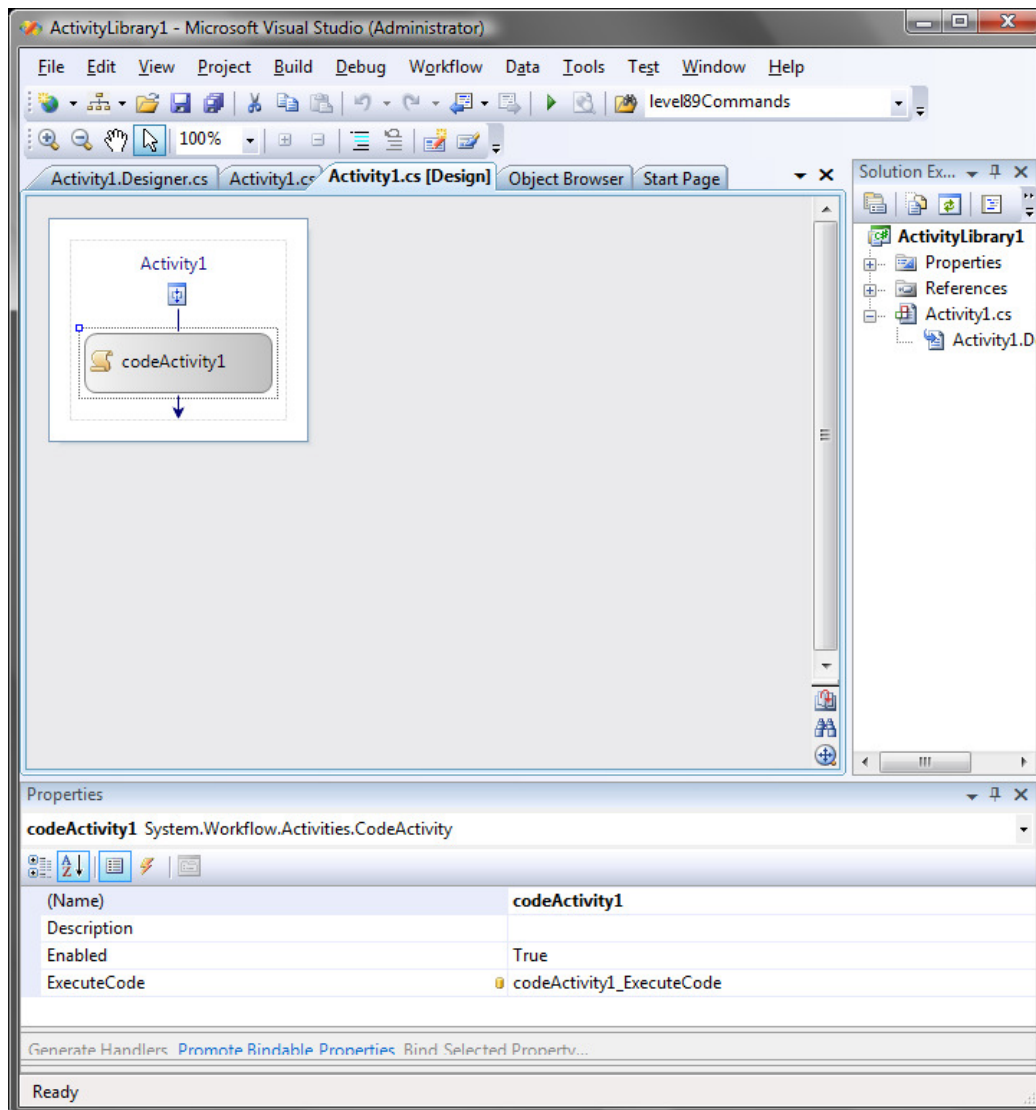
- 3a. The aim of this tutorial is to setup a sequential workflow library, which will then be called from a standard Windows program. First create Windows Workflow Activity, as shown in Figure 3, and add the code which will determine the number of prime numbers within a given range for a given maximum number:

```
public partial class Activity1 : SequenceActivity
{
    public ulong max=100;
    public ArrayList primes1 = new ArrayList();
    public int primes
    {
        get { return primes1.Count; }
    }
    public object setmax
    {
        set { this.max = (ulong)value; }
        get { return this.max; }
    }
    public Activity1()
    {
        InitializeComponent();
    }

    private void codeActivity1_ExecuteCode(object sender, EventArgs e)
    {
        for (ulong i = 3; i < this.max; i++)
        {
            if (isprime(i)) primes1.Add(i);
        }
    }
    public bool isprime(ulong val)
    {
        for (ulong i = 2; i < val / 2; i++)
        {
            if ((val % i) == 0) return false;
        }
        return true;
    }
}
```

Next build the library, and note the location of the DLL produced.

**Name and location of the DLL:**



**Figure 3:**

- 3b.** Now create a new Windows project, and add, as a reference, the DLL of the workflow. Add a button and textbox, so that the button calls up the Workflow created in 3a, and displays the result to the text box. Next add the code which will start the Workflow engine, and invoke the workflow:

```

int primes = 0;
bool flag1 = false;
ulong max = 0;

public Form1()
{
    InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{

```

```

WorkflowRuntime workflowRuntime = new WorkflowRuntime();
workflowRuntime.WorkflowCompleted += OnWorkflowCompleted;
workflowRuntime.StartRuntime();
Dictionary <string,object> parameters = new Dictionary<string,object>();
parameters.Add("setmax", (ulong)100);

WorkflowInstance workflowInstance =
workflowRuntime.CreateWorkflow(typeof(ActivityLibrary1 .Activity1), parameters);
workflowInstance.Start();
while (flag1 == false)
{
    Thread.Sleep(100);
}
this.textBox1.Text = "The number of primes is " + Convert.ToString(primes)+
    " and "+Convert.ToString(this.max);
}
void OnWorkflowCompleted(object sender, WorkflowCompletedEventArgs e)
{
    primes = (int)e.OutputParameters["primes"];
    max = (ulong)e.OutputParameters["setmax"];
    flag1 = true;
}

```

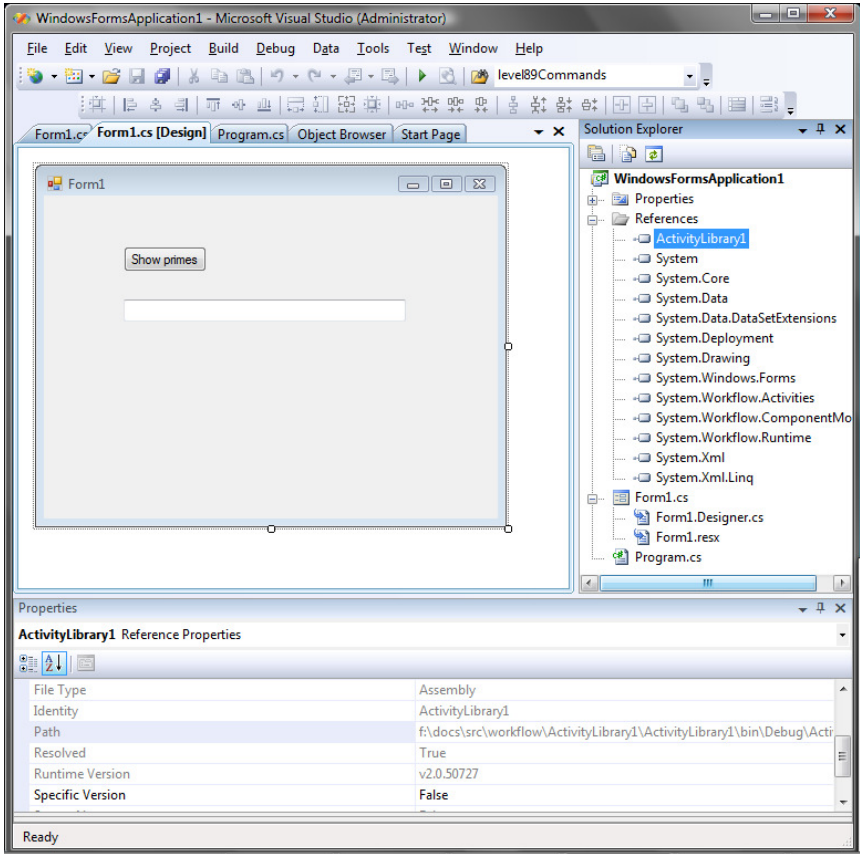


Figure 4:

- 3c. Prove the operation of the program, and then modify the Windows program, so that the user enters the maximum value for the prime number search range from a text box on the Windows form, and the program uses this to pass to the workflow. Prove its operation.
- 3d. Next modify the workflow library so that it returns the ArrayList of prime numbers for a given range. This should be returned to the Windows program, and is then displayed to the Window's form.

## B State-based Workflows

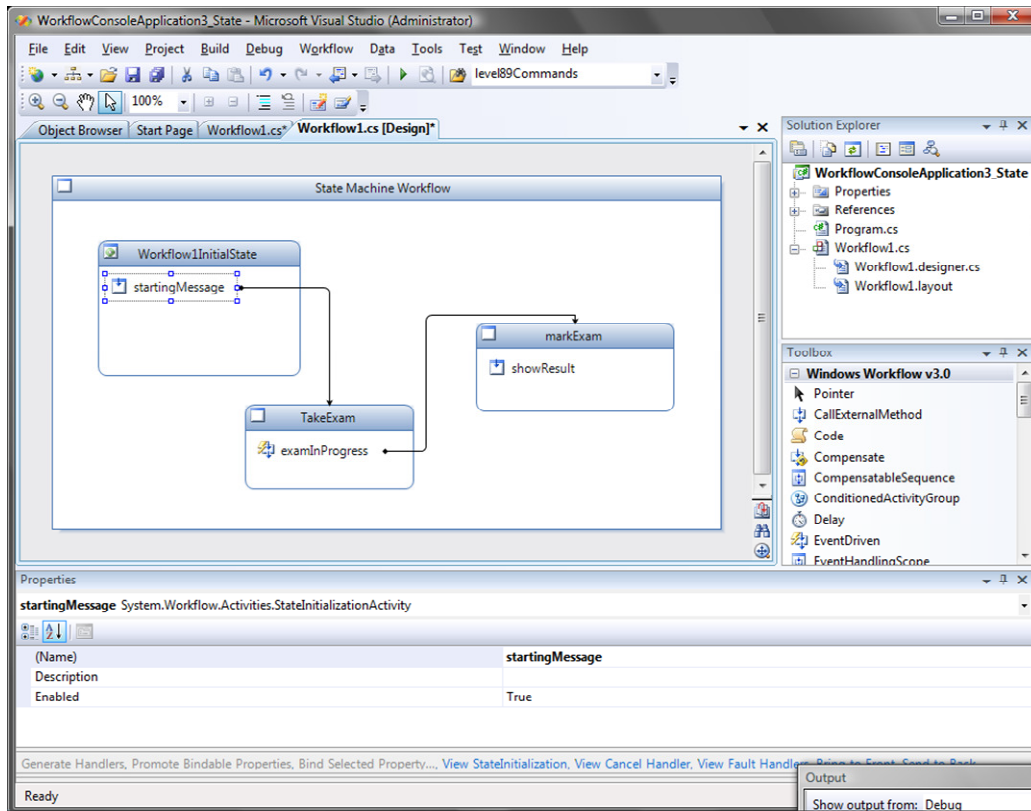
- 4a. Create a State-based Console Workflow, as defined in Figure 5. Then in each state add the following activities:

```
bool endexam = false;
public Workflow1()
{
    InitializeComponent();
}

private void codeActivity1_ExecuteCode(object sender, EventArgs e)
{
    Console.WriteLine("Started...");
}

private void codeActivity2_ExecuteCode(object sender, EventArgs e)
{
    do
    {
        Console.WriteLine("What is the capital of Scotland?");
        string input = Console.ReadLine();
        if (input == "edinburgh") endexam = true;
        else endexam = false;
    } while (endexam==false);
}

private void codeActivity3_ExecuteCode(object sender, EventArgs e)
{
    Console.WriteLine("You have passed the exam");
    Console.ReadLine();
}
```



**Figure 5:**

- 4b. Run the program, and prove its operation.
- 5a. Create a State-based Console Workflow, as defined in Figure 6. Then in each state add the following activities:

```

int correct = 0;
public Workflow1()
{
    InitializeComponent();
}
private void codeActivity2_ExecuteCode(object sender, EventArgs e)
{
    Console.WriteLine("startMessage");
}
private void codeActivity3_ExecuteCode_1(object sender, EventArgs e)
{
    Console.WriteLine("What is the capital of Scotland?");
    string input = Console.ReadLine();
    if (input == "edinburgh") correct++;

    Console.WriteLine("What is the capital of England?");
    string input = Console.ReadLine();
    if (input == "london") correct++;
}

```

```

Console.WriteLine("What is the capital of Wales?");
string input = Console.ReadLine();
if (input == "cardiff") correct++;

Console.WriteLine("What is the capital of Ireland?");
string input = Console.ReadLine();
if (input == "dublin") correct++;
}
private void codeActivity4_ExecuteCode(object sender, EventArgs e)
{
    Console.WriteLine("Fail");
}
private void codeActivity5_ExecuteCode(object sender, EventArgs e)
{
    Console.WriteLine("Pass");
}
}

```

The PassExam state should be called when the number of correct answers is three or more, otherwise the FailExam state should be call.

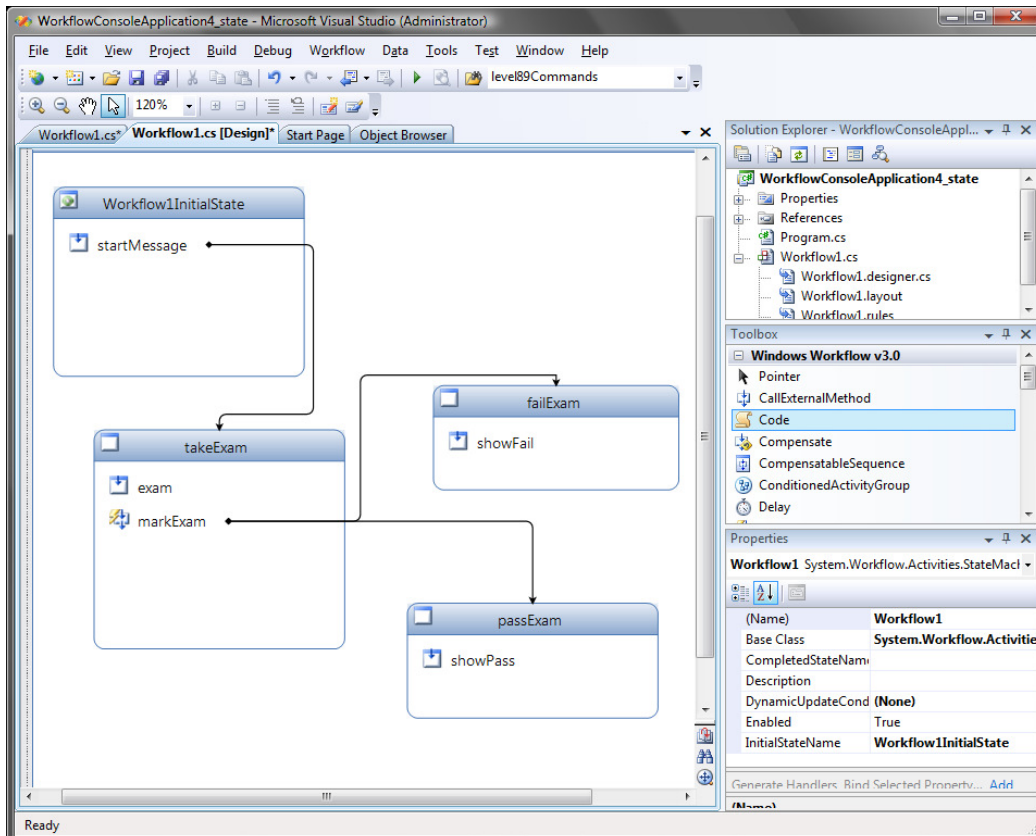


Figure 6:

## Windows Communications Foundation Tutorial

**PDF:** <http://ceres.napier.ac.uk/staff/bill/wwf.pdf>

**Lecture:** [http://ceres.napier.ac.uk/staff/bill/e\\_presentations/dotnet\\_wfc/dotnet\\_wfc.htm](http://ceres.napier.ac.uk/staff/bill/e_presentations/dotnet_wfc/dotnet_wfc.htm)

- 1a.** In this tutorial we will create a service which will determine the capital of a country. First create a WCF Service Library which will act as the server. An outline of the code is:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;

namespace Tut1.Server
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
    class ServiceImplementation : Tut1.Contract.IService
    {
        public string GetCapital(string Country)
        {
            Console.WriteLine("Contacting server...");
            if (Country == "scotland") return ("edinburgh");
            else if (Country == "england") return ("london");
            return "Not known";
        }
    }
}

public class Program
{
    private static System.Threading.AutoResetEvent endServer = new
System.Threading.AutoResetEvent(false);

    public static void Main()
    {
        ServiceHost svh = new ServiceHost(typeof(ServiceImplementation));
        svh.AddServiceEndpoint(
            typeof(Tut1.Contract.IService),
            new NetTcpBinding(), "net.tcp://localhost:8080");
        svh.Open();

        Console.WriteLine("Server running");
        endServer.WaitOne();

        Console.WriteLine("Server shutting down");
        svh.Close();

        Console.WriteLine("Server stopped");
    }
}
```

```

    public static void StopServer()
    {
        endServer.Set();
    }
}

```

**1b.** In the IService.cs file add code in the form of:

```

using System;
using System.Collections.Generic;
using System.Text;

using System.ServiceModel;

namespace Tut1.Contract
{
    [ServiceContract]
    public interface IService
    {
        [OperationContract]
        string GetCapital(string message);
    }
}

```

**1c.** Build the project, and note the DLL location created.

**1d.** Next create a Windows console application, and add the WCF DLL created in the first part of this tutorial. After this, add the code which can access the server, such as with:

```

using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;

namespace WCFSimple
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Calculate Capital - Simple WCF");
            // start server
            System.Threading.Thread myCountryServer = new
            System.Threading.Thread(Tut1.Server.Program.Main);
            myCountryServer.IsBackground = true;
            myCountryServer.Start();
        }
    }
}

```

```
System.Threading.Thread.Sleep(100); // give the server time to start
ChannelFactory<Tut1.Contract.IService> scf;
scf = new ChannelFactory<Tut1.Contract.IService>(
    new NetTcpBinding(),
    "net.tcp://localhost:8080");
Tut1.Contract.IService s;
s = scf.CreateChannel();
while (true)
{
    Console.Write("Enter a country: ");
    string country = Console.ReadLine();
    if (country == "") break;
    string capital = s.GetCapital(country);
    Console.WriteLine("The capital is " + capital);
}
(s as ICommunicationObject).Close();
// shutdown server
Tut1.Server.Program.StopServer();
}
}
```

- 1e.** Prove the operation of the program. Now add three services in the Service, which should either FindCapital(country), FindCountry(capital) and FindCurrency(country), where FindCapital() finds the capital city, FindCountry() finds the country of a certain capital, and FindCurrency() finds the currency of a certain country.
  
- 2a.** Create a Service which returns the number of prime numbers for a given range (FindNumberOfPrimes(start, end)) and also finds all the primes and returns as an ArrayList (FindPrimes(start,end)).

# Appendix A

## Tutorial 1

```
using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Collections;
using System.Drawing;
using System.Linq;
using System.Workflow.ComponentModel.Compiler;
using System.Workflow.ComponentModel.Serialization;
using System.Workflow.ComponentModel;
using System.Workflow.ComponentModel.Design;
using System.Workflow.Runtime;
using System.Workflow.Activities;
using System.Workflow.Activities.Rules;

namespace dotnet3
{
    public sealed partial class Workflow1 : SequentialWorkflowActivity
    {
        int i;

        public Workflow1()
        {
            InitializeComponent();
        }

        private void codeActivity1_ExecuteCode(object sender, EventArgs e)
        {
            Console.WriteLine("Value less than 5");
            Console.ReadLine();
        }

        private void codeActivity1_ExecuteCode_1(object sender, EventArgs e)
        {
            Console.WriteLine("Enter i>>");
            i = Convert.ToInt32(Console.ReadLine());
        }

        private void codeActivity3_ExecuteCode(object sender, EventArgs e)
        {
            Console.WriteLine("Value greater than 5");
            Console.ReadLine();
        }
    }
}
```

```

using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Collections;
using System.Drawing;
using System.Reflection;
using System.Workflow.ComponentModel.Compiler;
using System.Workflow.ComponentModel.Serialization;
using System.Workflow.ComponentModel;
using System.Workflow.ComponentModel.Design;
using System.Workflow.Runtime;
using System.Workflow.Activities;
using System.Workflow.Activities.Rules;

namespace dotnet3
{
    partial class Workflow1
    {
        #region Designer generated code

        /// <summary>
        /// Required method for Designer support – do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        [System.Diagnostics.DebuggerNonUserCode]
        private void InitializeComponent()
        {
            this.CanModifyActivities = true;
            System.Workflow.Activities.Rules.RuleConditionReference ruleconditionreference1 =
new System.Workflow.Activities.Rules.RuleConditionReference();
            System.Workflow.Activities.Rules.RuleConditionReference ruleconditionreference2 =
new System.Workflow.Activities.Rules.RuleConditionReference();
            this.codeActivity3 = new System.Workflow.Activities.CodeActivity();
            this.codeActivity1 = new System.Workflow.Activities.CodeActivity();
            this.cancellationHandlerActivity1 = new
System.Workflow.ComponentModel.CancellationHandlerActivity();
            this.ifElseBranchActivity2 = new System.Workflow.Activities.IfElseBranchActivity();
            this.ifElseBranchActivity1 = new System.Workflow.Activities.IfElseBranchActivity();
            this.ifElseActivity1 = new System.Workflow.Activities.IfElseActivity();
            this.codeActivity2 = new System.Workflow.Activities.CodeActivity();
            //
            // codeActivity3
            //
            this.codeActivity3.Name = "codeActivity3";
            this.codeActivity3.ExecuteCode += new
System.EventHandler(this.codeActivity3_ExecuteCode);

```

```

//
// codeActivity1
//
this.codeActivity1.Name = "codeActivity1";
this.codeActivity1.ExecuteCode += new
System.EventHandler(this.codeActivity1_ExecuteCode);
//
// cancellationHandlerActivity1
//
this.cancellationHandlerActivity1.Name = "cancellationHandlerActivity1";
//
// ifElseBranchActivity2
//
this.ifElseBranchActivity2.Activities.Add(this.codeActivity3);
ruleconditionreference1.ConditionName = "ival";
this.ifElseBranchActivity2.Condition = ruleconditionreference1;
this.ifElseBranchActivity2.Name = "ifElseBranchActivity2";
//
// ifElseBranchActivity1
//
this.ifElseBranchActivity1.Activities.Add(this.codeActivity1);
ruleconditionreference2.ConditionName = "ival";
this.ifElseBranchActivity1.Condition = ruleconditionreference2;
this.ifElseBranchActivity1.Name = "ifElseBranchActivity1";
//
// ifElseActivity1
//
this.ifElseActivity1.Activities.Add(this.ifElseBranchActivity1);
this.ifElseActivity1.Activities.Add(this.ifElseBranchActivity2);
this.ifElseActivity1.Activities.Add(this.cancellationHandlerActivity1);
this.ifElseActivity1.Name = "ifElseActivity1";
//
// codeActivity2
//
this.codeActivity2.Name = "codeActivity2";
this.codeActivity2.ExecuteCode += new
System.EventHandler(this.codeActivity1_ExecuteCode_1);
//
// Workflow1
//
this.Activities.Add(this.codeActivity2);
this.Activities.Add(this.ifElseActivity1);
this.Name = "Workflow1";
this.CanModifyActivities = false;

    }

    #endregion

private IfElseBranchActivity ifElseBranchActivity2;

```

```

private IfElseBranchActivity ifElseBranchActivity1;
private IfElseActivity ifElseActivity1;
private CodeActivity codeActivity2;
private CancellationHandlerActivity cancellationHandlerActivity1;
private CodeActivity codeActivity3;
private CodeActivity codeActivity1;

}
}

```

```

<RuleDefinitions xmlns="http://schemas.microsoft.com/winfx/2006/xaml/workflow">
  <RuleDefinitions.Conditions>
    <RuleExpressionCondition Name="ivalue">
      <RuleExpressionCondition.Expression>
        <ns0:CodeBinaryOperatorExpression Operator="LessThan"
xmlns:ns0="clr-namespace:System.CodeDom;Assembly=System, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089">
          <ns0:CodeBinaryOperatorExpression.Left>
            <ns0:CodeFieldReferenceExpression
FieldName="i">
              <ns0:CodeFieldReferenceExpression.TargetObject>
                <ns0:CodeThisReferenceExpression />
              </ns0:CodeFieldReferenceExpression.TargetObject>
            </ns0:CodeFieldReferenceExpression>
          </ns0:CodeBinaryOperatorExpression.Left>
          <ns0:CodeBinaryOperatorExpression.Right>
            <ns0:CodePrimitiveExpression>
              <ns0:CodePrimitiveExpression.Value>
                <ns1:Int32 xmlns:ns1="clr-
namespace:System;Assembly=microsoftlib, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">5</ns1:Int32>
              </ns0:CodePrimitiveExpression.Value>
            </ns0:CodePrimitiveExpression>
          </ns0:CodeBinaryOperatorExpression.Right>
        </ns0:CodeBinaryOperatorExpression>
      </RuleExpressionCondition.Expression>
    </RuleExpressionCondition>
    <RuleExpressionCondition Name="ival">
      <RuleExpressionCondition.Expression>
        <ns0:CodeBinaryOperatorExpression Operator="GreaterThan"
xmlns:ns0="clr-namespace:System.CodeDom;Assembly=System, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089">
          <ns0:CodeBinaryOperatorExpression.Left>

```

```

                                <ns0:CodeFieldReferenceExpression
FieldName="i">
    <ns0:CodeFieldReferenceExpression.TargetObject>
    <ns0:CodeThisReferenceExpression />
    </ns0:CodeFieldReferenceExpression.TargetObject>
                                </ns0:CodeFieldReferenceExpression>
                                </ns0:CodeBinaryOperatorExpression.Left>
                                <ns0:CodeBinaryOperatorExpression.Right>
                                <ns0:CodePrimitiveExpression>
    <ns0:CodePrimitiveExpression.Value>
                                <ns1:Int32 xmlns:ns1="clr-
namespace:System;Assembly=microsoftlib, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">5</ns1:Int32>
    </ns0:CodePrimitiveExpression.Value>
                                </ns0:CodePrimitiveExpression>
                                </ns0:CodeBinaryOperatorExpression.Right>
                                </ns0:CodeBinaryOperatorExpression>
                                </RuleExpressionCondition.Expression>
                                </RuleExpressionCondition>
                                </RuleDefinitions.Conditions>
</RuleDefinitions>

```

## Tutorial 2

```

using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Collections;
using System.Drawing;
using System.Reflection;
using System.Workflow.ComponentModel.Compiler;
using System.Workflow.ComponentModel.Serialization;
using System.Workflow.ComponentModel;
using System.Workflow.ComponentModel.Design;
using System.Workflow.Runtime;
using System.Workflow.Activities;
using System.Workflow.Activities.Rules;

namespace WorkflowConsoleApplication1
{
    partial class Workflow1
    {
        #region Designer generated code

```

```

        /// <summary>
        /// Required method for Designer support – do not modify
        /// the contents of this method with the code editor.
        /// </summary>
[System.Diagnostics.DebuggerNonUserCode]
    private void InitializeComponent()
    {
        this.CanModifyActivities = true;
        System.Workflow.Activities.Rules.RuleConditionReference ruleconditionreference1 =
new System.Workflow.Activities.Rules.RuleConditionReference();
        this.codeActivity2 = new System.Workflow.Activities.CodeActivity();
        this.codeActivity1 = new System.Workflow.Activities.CodeActivity();
        this.codeActivity3 = new System.Workflow.Activities.CodeActivity();
        this.sequenceActivity2 = new System.Workflow.Activities.SequenceActivity();
        this.sequenceActivity1 = new System.Workflow.Activities.SequenceActivity();
        this.whileActivity1 = new System.Workflow.Activities.WhileActivity();
        this.parallelActivity1 = new System.Workflow.Activities.ParallelActivity();
        //
        // codeActivity2
        //
        this.codeActivity2.Name = "codeActivity2";
        this.codeActivity2.ExecuteCode += new
System.EventHandler(this.codeActivity2_ExecuteCode);
        //
        // codeActivity1
        //
        this.codeActivity1.Name = "codeActivity1";
        this.codeActivity1.ExecuteCode += new
System.EventHandler(this.codeActivity1_ExecuteCode);
        //
        // codeActivity3
        //
        this.codeActivity3.Name = "codeActivity3";
        this.codeActivity3.ExecuteCode += new
System.EventHandler(this.codeActivity3_ExecuteCode);
        //
        // sequenceActivity2
        //
        this.sequenceActivity2.Activities.Add(this.codeActivity2);
        this.sequenceActivity2.Name = "sequenceActivity2";
        //
        // sequenceActivity1
        //
        this.sequenceActivity1.Activities.Add(this.codeActivity1);
        this.sequenceActivity1.Name = "sequenceActivity1";
        //
        // whileActivity1
        //
        this.whileActivity1.Activities.Add(this.codeActivity3);
        ruleconditionreference1.ConditionName = "flags";
    }

```

```

this.whileActivity1.Condition = ruleconditionreference1;
this.whileActivity1.Name = "whileActivity1";
//
// parallelActivity1
//
this.parallelActivity1.Activities.Add(this.sequenceActivity1);
this.parallelActivity1.Activities.Add(this.sequenceActivity2);
this.parallelActivity1.Name = "parallelActivity1";
//
// Workflow1
//
this.Activities.Add(this.parallelActivity1);
this.Activities.Add(this.whileActivity1);
this.Name = "Workflow1";
this.CanModifyActivities = false;

    }

    #endregion

private CodeActivity codeActivity2;
private CodeActivity codeActivity1;
private CodeActivity codeActivity3;
private SequenceActivity sequenceActivity2;
private SequenceActivity sequenceActivity1;
private ParallelActivity parallelActivity1;
private WhileActivity whileActivity1;

}
}

```

```

<RuleDefinitions xmlns="http://schemas.microsoft.com/winfx/2006/xaml/workflow">
  <RuleDefinitions.Conditions>
    <RuleExpressionCondition Name="flags">
      <RuleExpressionCondition.Expression>
        <ns0:CodeBinaryOperatorExpression Operator="BooleanAnd"
xmlns:ns0="clr-namespace:System.CodeDom;Assembly=System, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089">
          <ns0:CodeBinaryOperatorExpression.Left>
            <ns0:CodeBinaryOperatorExpression
Operator="ValueEquality">
              <ns0:CodeBinaryOperatorExpression.Left>
                <ns0:CodeFieldReferenceExpression FieldName="flag1">
                  <ns0:CodeFieldReferenceExpression.TargetObject>
                    <ns0:CodeThisReferenceExpression />

```

```

</ns0:CodeFieldReferenceExpression.TargetObject>

</ns0:CodeFieldReferenceExpression>

</ns0:CodeBinaryOperatorExpression.Left>

<ns0:CodeBinaryOperatorExpression.Right>

<ns0:CodePrimitiveExpression>

<ns0:CodePrimitiveExpression.Value>

  <ns1:Boolean xmlns:ns1="clr-namespace:System;Assembly=mscorlib,
Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">true</ns1:Boolean>

</ns0:CodePrimitiveExpression.Value>

</ns0:CodePrimitiveExpression>

</ns0:CodeBinaryOperatorExpression.Right>
  </ns0:CodeBinaryOperatorExpression>
</ns0:CodeBinaryOperatorExpression.Left>
<ns0:CodeBinaryOperatorExpression.Right>
  <ns0:CodeBinaryOperatorExpression
Operator="ValueEquality">

  <ns0:CodeBinaryOperatorExpression.Left>

  <ns0:CodeFieldReferenceExpression FieldName="flag2">

  <ns0:CodeFieldReferenceExpression.TargetObject>

  <ns0:CodeThisReferenceExpression />

  </ns0:CodeFieldReferenceExpression.TargetObject>

  </ns0:CodeFieldReferenceExpression>

  </ns0:CodeBinaryOperatorExpression.Left>

  <ns0:CodeBinaryOperatorExpression.Right>

  <ns0:CodePrimitiveExpression>

  <ns0:CodePrimitiveExpression.Value>

  <ns1:Boolean xmlns:ns1="clr-namespace:System;Assembly=mscorlib,

```

```
Version=2.0.0.0, Culture=neutral,  
PublicKeyToken=b77a5c561934e089">>true</ns1:Boolean>
```

```
</ns0:CodePrimitiveExpression.Value>
```

```
</ns0:CodePrimitiveExpression>
```

```
</ns0:CodeBinaryOperatorExpression.Right>
```

```
</ns0:CodeBinaryOperatorExpression>
```

```
</ns0:CodeBinaryOperatorExpression.Right>
```

```
</ns0:CodeBinaryOperatorExpression>
```

```
</RuleExpressionCondition.Expression>
```

```
</RuleExpressionCondition>
```

```
</RuleDefinitions.Conditions>
```

```
</RuleDefinitions>
```