**Lab 8**

**What is Workflow?**

A workflow is a series of activities involved in a task or set of activities collectively forms a work. Except for the first and last step, steps will have a preceding and succeeding step.  
  
Windows Workflow Foundation provides a very nice graphical interface to create the activities involved in a workflow. Windows Workflow Foundation is a great technology for creating workflows and can be used in combination with different technologies, like SharePoint, WCF, ASP.NET etc. Windows workflow foundation comes with a set of commonly used activities packaged along with it. They will be available in the Toolbox and you can just drag and drop them to your workflow. You can create your own custom activities and make them available in the Toolbox as well.   
  
A workflow model when compiled, it can be executed inside any Windows process including console applications, forms-based applications, Windows Services, ASP.NET Web sites or Web services. Since a workflow is hosted in a process, a workflow can easily communicate with its host application. Though we use graphical interface to desgin workflows, they are storedsas XAML markup. A sample workflow will look like this:

1. <Activity mc:Ignorable="sads sap sap2010" x:Class="BMIRuleEngine.Workflow1" xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:mca="clr-namespace:Microsoft.CSharp.Activities;assembly=System.Activities" xmlns:mva="clr-namespace:Microsoft.VisualBasic.Activities;assembly=System.Activities" .......................................... .......................................... .......................................... <x:Members>
2. <x:Property Name="BMIInput" Type="InArgument(x:Double)" />
3. <x:Property Name="Recommendation" Type="OutArgument(x:String)" /> </x:Members>
4. <sap2010:ExpressionActivityEditor.ExpressionActivityEditor>C#</sap2010:ExpressionActivityEditor.ExpressionActivityEditor>
5. <sap:VirtualizedContainerService.HintSize>1350,572</sap:VirtualizedContainerService.HintSize>
6. <mva:VisualBasic.Settings>Assembly references and imported namespaces **for** internal implementation</mva:VisualBasic.Settings>
7. <If sap:VirtualizedContainerService.HintSize="1310,532">
8. <If.Condition>
9. <InArgument x:TypeArguments="x:Boolean">
10. <mca:CSharpValue x:TypeArguments="x:Boolean">BMIInput < 18.5</mca:CSharpValue>
11. </InArgument>
12. </If.Condition>
13. <If.Then>
14. <Assign sap:VirtualizedContainerService.HintSize="242,60">
15. <Assign.To>
16. <OutArgument x:TypeArguments="x:String">[Recommendation]</OutArgument>
17. </Assign.To>
18. <Assign.Value>
19. <InArgument x:TypeArguments="x:String">["Your BMI is less than 18.5. You are Underweight. You should try adding some muscle mass through regular exercises."]</InArgument>
20. </Assign.Value>
21. </Assign>
22. </If.Then> .......................................... .......................................... .......................................... </If>
23. </Activity>

We will see with an example, how to create a workflow with a few activities and how to consume it using an ASP.NET MVC application.  
  
Windows Workflow Foundation can be used in following types of scenarios but not limited to this:

* Applications with User-interface page flows.
* Workflows that are document-centric.
* Human workflows.
* Service-oriented applications wtih composite workflows.
* Rule-driven workflows for business.
* Workflows for systems management.

workflow framework can be used or are useful in the following situations:

* For creating business rule engines. This may include simple rules, such as if-then-else decisions based on an input argument to more complex rules, such as the potentially large set that is a combination of several other rules.
* Case where maintaining state for the workflow's lifetime is needed. For example, if the employee who is in charge of approving a task is on leave for a week, the task should be bookmarked or stored for use at a later point of time and should be able to continue the workflow. The bookmarked or stored task should be able to re-activate when required.
* Interactive applications that connect with people. For example, if an HR must approve some policy documents, the workflow must be able to display a user interface or communicate to the HR through other software.
* Changing workflow behavior like changing a condition or editing an action which are common use cases.
* Multiple applications that work together. For example, a workflow rule engine gets input from an ASP.NET web application and does workflow activities and call another application for generating a travel request.
* Monitoring of a running workflow and examining its execution in real time.
* Support for graphical user interface tools for creating workflows easily. A workflow consists of different steps or activities and we should be able to re-use the activities if we create one.

**What is Activity?**An activity defines a piece of work. The work an activity implements can be very simple or it can be quite complex. Activities are the fundamental building blocks or unit of a workflow. An activity represents an action in a workflow. They can be added to a workflow either programmatically or using graphical design interface where you can drag and drop activities. The workflow instance is completed only when all the activities in a given flow path are finished. An activity can perform a single action, such as writing a value to a database, or it can be a sequence of activities. Many of the activities in the base activity library, such as IfElse, Sequence, and While, are composite activities.  
  
An activity, like a workflow, can be sequential, which means that the order of its actions is specified at design time. Or the activity can be event-driven, which means that the order of its actions is determined at run time in response to external events.  
  
Windows Workflow Foundation contains a library of standard activities and provides the mechanisms for you to create your own. This enables extensibility and reusability between workflows.  
  
Windows Workflow Foundation (WF) includes default activities that provide functionality for communication with applications and services, control flow, state management, event handling, and conditions. However, we might need a custom activity in most scenarios to implement specific behavior of our application or workflow.  
  
Windows Workflow Foundation (WF) API's will help you to create custom activities. Custom activity can be created by using code or using graphical user interface.

**How to create an Activity?**

For creating a custom activity using code, you need to define a custom activity class that either derives directly from the activity, or from a default activity that derives from activity. Custom activities by default, display within the workflow designer as an empty rectangle with the activity’s name. To provide a custom visual representation of your activity in the workflow designer, you must add the required functionality to the activity. You can combine existing activities together to form a new *composite* activity.

Using the graphical Activity Designer tool is the easiest way to create an activity. As we have already mentioned, activities can also be created directly in code. Like a workflow, an activity is just a class. Each activity can have events that it responds to, and properties containing its state.

C# definition of the activity looks something like the following:

1. using System.Workflow.ComponentModel;
2. **public** **class** SampleActivity: Activity {
3. ...
4. ...
5. }

**Windows Workflow Foundation Rules Engine**

Windows Workflow Foundation (WF) introduces rule capabilities to the .NET Framework development platform. Rules creation ability extend from simple conditions to complex rulesets executed by a full-featured forward chaining rules engine.

The rules capability allows for the declarative modeling of application units which can be easily created and plugged into the overall business process. Rules capability allow us to model the application in a way which can be easily modified without affecting the other parts of the application. Sample scenarios for rules engine technology are lab result analysis system, promotion enforcement fare calculation, inventory management etc.  
  
**Rules and Conditions**

Conditions are logic that return true or false. A number of Work Flow activities work based on conditions by affecting their behavior. These activities include the While activity, the IfElseBranch activity, and the ConditionedActivityGroup. The IfElseBrach activity, for instance, executes the *Then* part, if condition evaluates to true otherwise executes the *Else* part. We can implement conditions in code, or in XML.

Rules contains various conditions with a set of actions to perform. Rules use a declarative if-then-else style, where the "if" is a condition to evaluate. There are substantial differences between rules and procedural code. Procedural code in most applications, changes the flow of the control based on the condition. In case of rules, execution engine evaluates the logic and invoke their actions.

The third term, "rule set" is a collection of one or more rules.

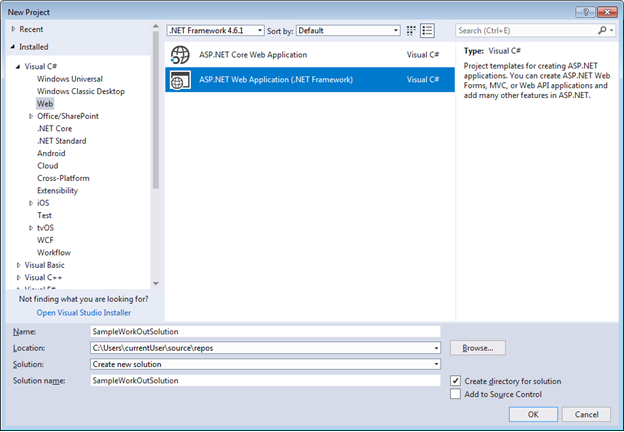
**Creating a Rule Activity and Consuming it using an MVC application**

Ok, we have learned some basics. Now, it is time for some action.

Let us create a simple rule using a workflow application and read the result of the rule from an ASP.Net MVC 5 application.

**Step 1 Create an ASP.Net MVC Application**

* Open Visual Studio2017
* Go to File->New->Project.
* Select Web ASP.NET Web Application.



Click OK and choose MVC from the list of options given in the next screen.

**Step 2 Create a Model**

* Right click on Models folder
* Choose -> Add -> Class
* Give name as BMI and add the below code to the class
  1. using System;
  2. using System.Collections.Generic;
  3. using System.ComponentModel.DataAnnotations;
  4. using System.Linq;
  5. using System.Web;
  6. namespace SampleWorkOutSolution.Models {
  7. **public** **class** BMI {

        [Display(Name = "Heigh in metre")]

**public** **double** Height {

            get;

            set;

        }

        [Display(Name = "Weight in Kilogram")]

**public** **double** Weight {

            get;

            set;

        }

        [Display(Name = "BMI")]

**public** **double** BMIValue {

            get {

                // Avoiding DivideByZero exception

**if** (Height > 0) {

                    // BMI = Weight (in Kg) / Height (in metre) Square

**return** Weight / (Height \* Height);

                } **else** {

**return** 0;

                }

            }

        }

**public** string Recommendation {

            get;

            set;

        }

    }

* 1. }

**Step 3 Create a controller, an action and a view**

I am going to use here the default HomeController which is created by default. I am also going to use the default Index() method to get collect the BMI input data.

So, my Controller now look like this:

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Web.Mvc;
7. namespace SampleWorkOutSolution.Controllers {
8. **public** **class** HomeController: Controller {
9. **public** ActionResult Index() {
10. **return** View();
11. }
12. }
13. }

Now let’s remove the default text and add a few text boxes to the view Index.cshtml.

I have made the view strongly typed by binding the BMI model.

@model SampleWorkOutSolution.Models.BMI

@ {

    ViewBag.Title = "Home Page";

}

< div **class** = "row" >

      < div **class** = "col-md-12" >

           < h1 style = "text-align:center" >

                 Weight Status

           < /h1>

      < /div>

< /div>

< div >

     @using(@Html.BeginForm("FindBMI", "Home"))

     {

           < div **class** = "row" >

                   < div **class** = "col-md-4" >< /div>

                   < div **class** = "col-md-2" >

                           @Html.LabelFor(model => model.Height);

                    < /div>

                    < div **class** = "col-md-6" >

                           @Html.TextBoxFor(model => model.Height, **new** {  @**class** = "form-control"  })

                    < /div>

           < /div>

           < div **class** = "row" >

                    < div **class** = "col-md-4" > < /div>

                    < div **class** = "col-md-2" >

                            @Html.LabelFor(model => model.Weight);

                    < /div>

                    < div **class** = "col-md-6" >

                            @Html.TextBoxFor(model => model.Weight, **new** {  @**class** = "form-control"  })

                   < /div>

           < /div>

           < div **class** = "row" >

                   < div **class** = "col-md-6" > < /div>

                   < div **class** = "col-md-6" >

                            < input type = "submit"  name = "Submit"  value = "Submit" / >

                   < /div>

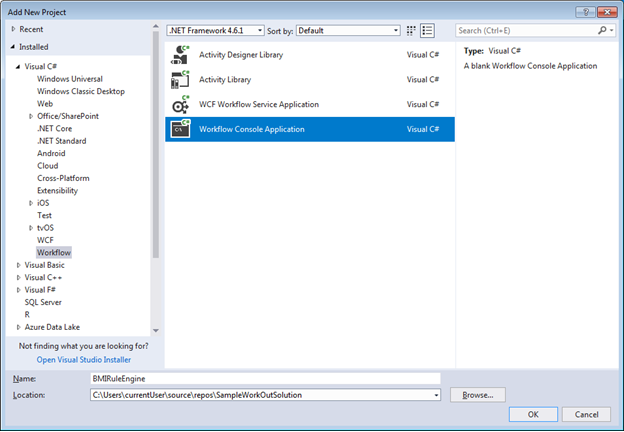
            < /div>

      }

< /div>

**Step 4 Create a Workflow Application and a rule activity**

* Right click on the solution and Add -> New Project
* Choose Workflow -> Workflow Console Application
* Give name BMIRuleEngine and click OK



A default workflow named Workflow1.xaml is created.

We can just use it for our purpose here.

**Step 5 Create Arguments**

To compare the BMI value, we need to pass it as an argument to the activity.

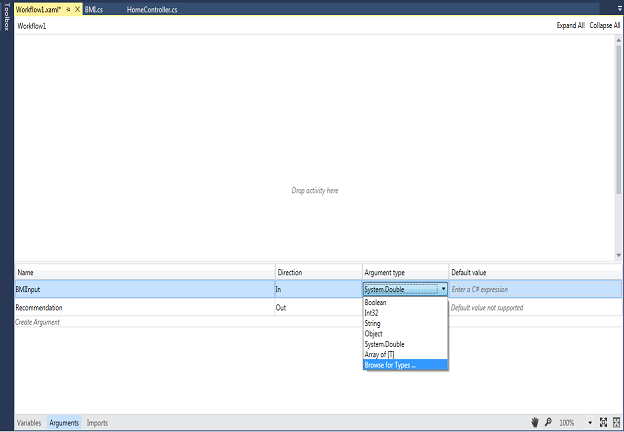
Let’s create an input argument called BMIInput for this.

Click on the Arguments tab on the bottom of the page, give the name as BMIInput, choose the Direction as In and argument type as Double.

If Double is not already there in the type list, choose Browse for Types.

For storing the recommendation and passing to our MVC application, we need an output argument.

Let’s create an Out variable with name Recommendation with type as string.



Type “Double” in the Type Name input field

Choose Double from the items listed below and click OK.

**Step 6 Add a few activities**

 A workflow contains a set of activities (actions).

Let’s create a few of the activities available in the Toolbox.

Open Workflow1.xaml and drag and drop an If activity from the toolbox to the area which reads “Drop activity here”.

Now add our condition in the IfEsle branch activity we added.

Add the below condition in the Condition text box:

BMIInput < 18.5

Drag and drop an assigned activity from the primitives group in the ToolboxToolbox to Then area of the IfElse activity.

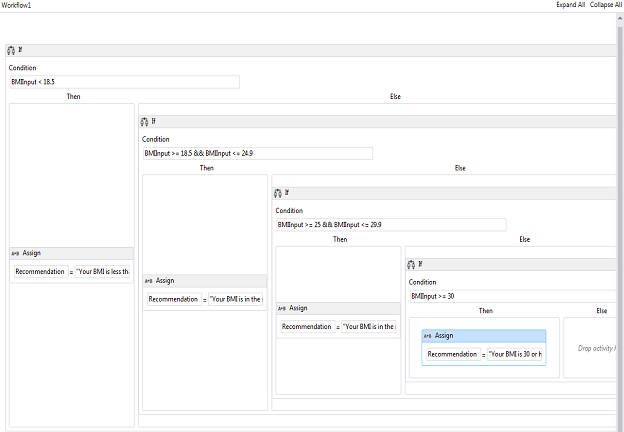
To the left-hand side text box of the Assign activity, enter Recommendation and to the right hand side of the activity, enter "Your BMI is less than 18.5. You are Underweight. You should try adding some muscle mass through regular exercises.". Remember to give the value of the Recommendation in quotes as it is a string variable.

In the Else part of If condition we can again Drag and Drop another If activity.

We can continue this process until we have finished adding all our conditions.

It is as simple as that.

Finally, our Rule should look like below



**Step 7 Add Actions in MVC application to consume the Rule and show result.**

First, add a reference to the Workflow project BMIRuleEngine in the MVC application.

Add a POST action to call the Workflow rule engine on Submit button click.

We use WorkflowInvoker class and its Invoke method to invoke the workflow activity. Pass the object of the activity and the input arguments as the parameters. Input arguments is passed as a dictionary object.

Workflow activity will return the output as a dictionary object. We just need to get the value of the output argument using the name of the argument.

[HttpPost]

**public** ActionResult FindBMI(BMI model) {

**var** result = WorkflowInvoker.Invoke(**new** BMIRuleEngine.Workflow1(), **new** Dictionary < string, object > {

        {

            "BMIInput",

            model.BMIValue

        }

    });

    model.Recommendation = result["Recommendation"] as String;

    TempData["wfResult"] = model;

**return** RedirectToAction("ShowResult");

}

Let’s create another GET action to show the result.

[HttpGet]

**public** ActionResult ShowResult() {

    BMI model = TempData["wfResult"] as BMI;

**return** View(model);

}

Right click on the ShowResult action method and choose Add View and add a strongly typed view with BMI model.

1. @model SampleWorkOutSolution.Models.BMI

@{

   ViewBag.Title = "BMI Report";

}

<div **class**="row">

    <div **class**="col-md-12">

        <h1 style="text-align:center">BMI Report</h1>

    </div>

</div>

<hr />

<div **class**="row">

    <div **class**="col-md-4"></div>

    <div **class**="col-md-2"> @Html.LabelFor(model => model.Height); </div>

    <div **class**="col-md-6"> @Html.TextBoxFor(model => model.Height, **new** { @**class** = "form-control" }) </div>

</div>

<div **class**="row">

    <div **class**="col-md-4"></div>

    <div **class**="col-md-2"> @Html.LabelFor(model => model.Weight); </div>

    <div **class**="col-md-6"> @Html.TextBoxFor(model => model.Weight, **new** { @**class** = "form-control" }) </div>

</div>

<div **class**="row">

    <div **class**="col-md-4"></div>

    <div **class**="col-md-2"> @Html.LabelFor(model => model.BMIValue); </div>

    <div **class**="col-md-6"> @Html.TextBoxFor(model => model.BMIValue, **new** { @**class** = "form-control" }) </div>

</div>

<div **class**="row">

    <div **class**="col-md-4"></div>

    <div **class**="col-md-2"> @Html.LabelFor(model => model.Recommendation); </div>

    <div **class**="col-md-6"> @Html.TextAreaFor(model => model.Recommendation, **new** { @**class** = "form-control" }) </div>

</div>

<div **class**="row">

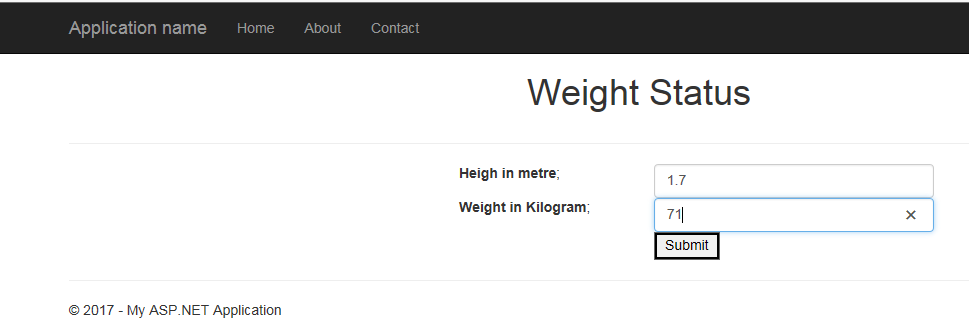
    <div **class**="col-md-6"></div>

    <div **class**="col-md-6"> @Html.ActionLink("Back", "Index") </div>

</div>

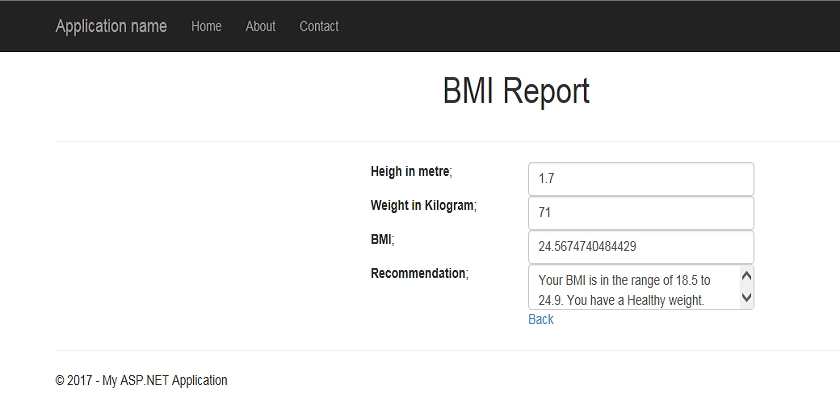
**Step 8 Run the application**

Build your application and run it to see the input screen.

You should get a similar screen  
  


Enter the height and weight and click Submit.

If we get a result from the rule workflow, we should have a value in the Recommendation field.



**Tasks**:

1. Create a workflow for simple calculator
2. Create a workflow for any real world problem and test the asp.net mvc application using that workflow