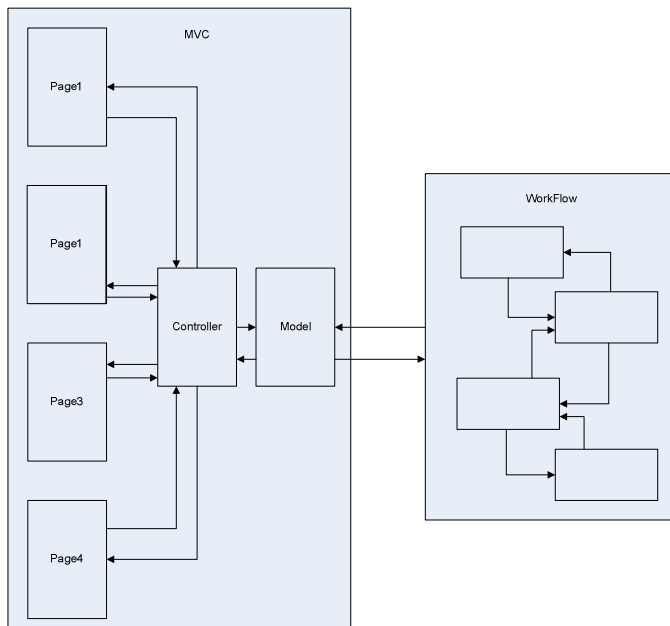


Using Windows Workflow and MVC to control page display order

This sample will run through the development of a wizard based user interface for displaying web pages. It is not going to explain how MVC or Windows Workflow work, but it will show you how to use a workflow state machine to control the order that pages are displayed on a web site.

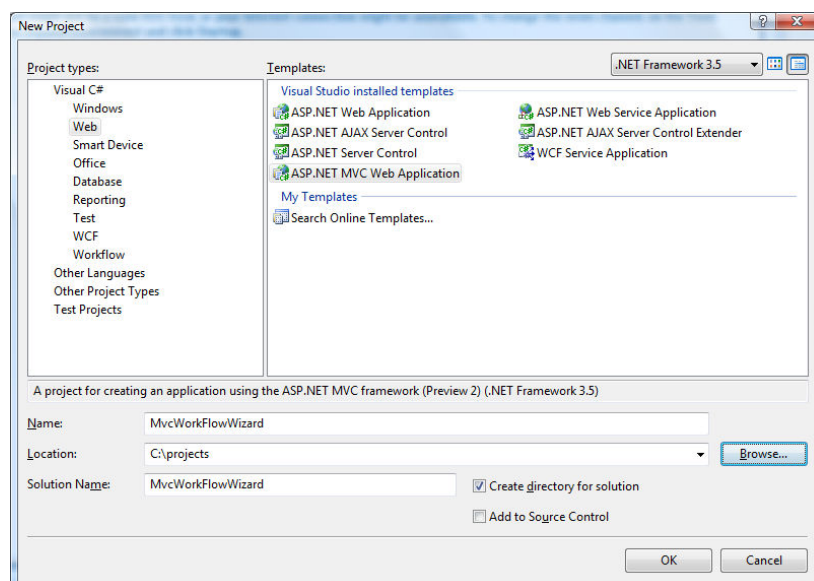
The samples are written using Visual studio 2008 with Microsoft ASP.NET MVC preview 2.

Using the MVC pattern allows the decoupling of the web pages and the order they are displayed. This separation can be seen in the following diagram.

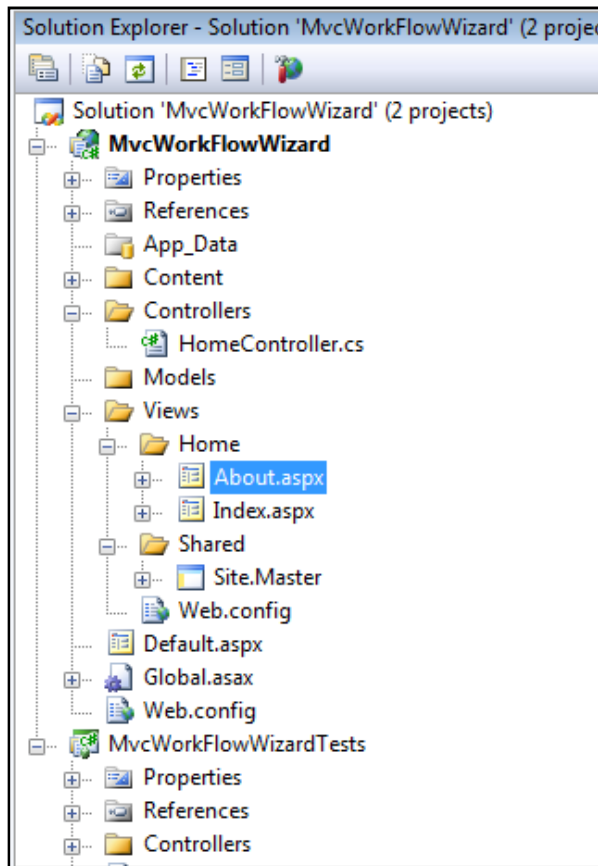


Getting Started

Create a new project and select ASP.NET MVC Web application



The solution creates a sample MVC application for you as follows:



We are not interested in the two pages that are created (About.aspx and Index.aspx). We are interested in the controller and we will be creating a model. The model does not exist as standard so we shall create one. Right click on the models folder and select Add, New Item, and then pick code file and name the file WorkflowModel.cs.

We are going to create a simple controller that will simply execute the previous and next functions of our wizard so that we can test that all our pages work.

```
namespace MvcWorkFlowWizard.Models
{
    public class WorkflowModel
    {
        public WorkflowModel()
        {
        }

        public delegate void NextPage(string page);
        public event NextPage nextPage;
        private void OnNextPage(string page)
        {
            if (nextPage != null)
            {
                nextPage(page);
            }
        }
    }
}
```

```
    }

    public void DoNext ()
    {
        count++;
        if (count > MaxPages)
        {
            count = 0;
        }

        NewPage (count);
    }
    public void DoPrev ()
    {
        count--;
        if (count < 0)
        {
            count = MaxPages;
        }

        NewPage (count);
    }

    private int count = 0;
    private readonly int MaxPages = 3;
    private void NewPage (int id)
    {
        id++; //Make the page index 1 based not zero based
        string page = string.Format ("Page{0}", id);

        OnNextPage (page);
    }
}
}
```

For this example we are creating a model that fires an event when the next page has been identified and simply loops around the four pages in either direction. The methods `DoPrev()` and `DoNext()` will be called by the controller to determine the next page that is to be shown. The `nextPage` event is used to allow the controller to be notified when the page is ready to be shown. In `NewPage` `string.Format("Page{0}", id)` creates the page name we are using. For simplicity I have used the names Page1 to Page4. Now we need to define four pages in the wizard. We will use the master page provided and add in a previous and next button.

```
<div class="rightColumn">
    <asp:ContentPlaceHolder ID="MainContentPlaceHolder" runat="server">
        </asp:ContentPlaceHolder>

        <input id="Button1" type="button" value="Previous"
onclick="javascript:window.location='<%=Url.Action("Prev", "Home") %>' />

        <input id="Button2" type="button" value="Next"
onclick="javascript:window.location='<%=Url.Action("Next", "Home") %>' />

</div><!--/rightColumn-->
```

Add the two inputs as defined above to the master page after the content place holder. This will allow the previous and next buttons to appear on all of our pages. Note the onclick action. This tells the Home controller to call the Prev or Next method. We now need to add these methods to the controller.

The controller needs to contact the model and then execute the previous and next actions to return the correct next page. As we are running in web environment we need to persist the model somewhere so that we can keep track of the current page. To make this sample simpler to understand we will just use a static member variable on the controller to hold the model. This does however limit this application to a single user. In a real application the model can be written so that its information can be persisted for example in a db or in the session. The controller has 2 public methods to carry out the previous and next functions.

```
using System.Web.Mvc;

namespace MvcWorkflowWizard.Controllers
{
    public class HomeController : Controller
    {
        #region Model
        private static MvcWorkflowWizard.Models.WorkFlowModel model = null;
        #endregion

        public HomeController()
        {
            if (model == null)
            {
                // first time through so create the model
                model = new MvcWorkflowWizard.Models.WorkFlowModel();

                // wire up the new page event
                model.nextPage += new
MvcWorkflowWizard.Models.WorkFlowModel.NextPage(model_nextPage);
            }
        }

        void model_nextPage(string page)
        {
            RenderView(page);
        }

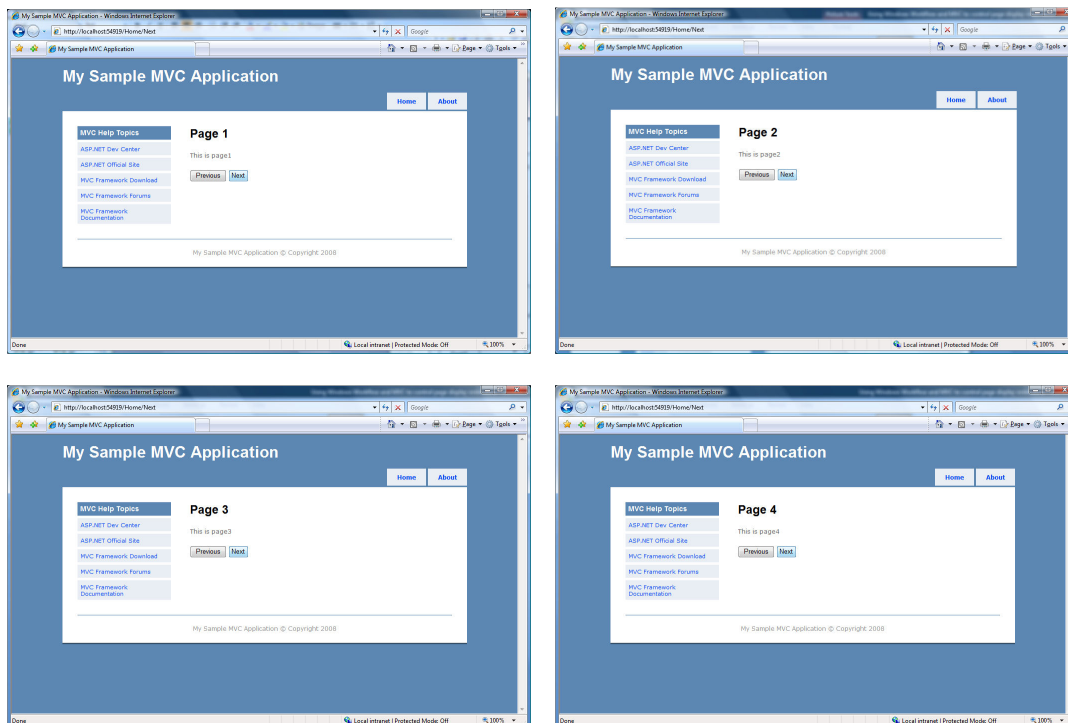
        public void Index()
        {
            RenderView("Index");
        }

        public void Next()
        {
            model.DoNext();
        }
        public void Prev()
        {
            model.DoPrev();
        }
    }
}
```

}

In the event handler for nextPage the method RenderView is an MVC command to display the page identified by the page name. We are using the pages Page1.aspx, Page2.aspx, Page3.aspx and Page4.aspx. We need to create these pages in Views\Home. Currently there are Index.aspx and About.aspx in here. All the pages need to inherit from the master page to get the previous and next buttons. Using About.aspx and Index.aspx as examples create the 4 new pages and make sure that they have different content so that you can see the difference later.

Part 1 is now complete. You have an MVC application where the ordering of the pages is derived from the model. This means that we can change the way the model works to change the way the pages are displayed. Run the application and see what happens.



We have hard coded the page order in this model and it is fairly simple. A lot of the time the page transitions are decided upon some sort of criteria and coding this can lead to spaghetti code. Putting the page decision into a state machine will allow the page order to be far more complex and allow it to be easily changed without rewriting the user interface. Also putting the page order decision into a model allows us to write unit tests against the model. I will discuss these issues in Part 2.