**[SharePoint 2007 Workflow with Visual Studio 2005 + InfoPath 2007 (RTM VERSION!)](http://weblog.vb-tech.com/nick/archive/2007/02/25/2207.aspx)**

I have been amazed at how many hits and questions the 'how-to' article I posted on creating a [SharePoint 2007 workflow with Visual Studio 2005](http://weblog.vb-tech.com/nick/archive/2006/09/04/1753.aspx) and InfoPath 2007 got. That article was based on Beta2 of SharePoint 2007, and although most of the steps are still the same, I decided to rewrite the article now that RTM is out. There were so many questions on the previous post that it was difficult for me to answer them all, so I've got to thank other people who tried out the workflow and got it to work for coming along and helping others. At the end of this article I've a link to forum post where I think we can put questions about this guide. That way people can sign up to alerts and we can hopefully get peoples questions answered in a much more organised way. So here it is, buckle down and good luck...

[after thought - this post and capturing all the images has taken almost 5 hours. If you are reposting it somewhere else (ie stealing it!), please at least give a link back! And if you like the post also please give me a link ;-) ]

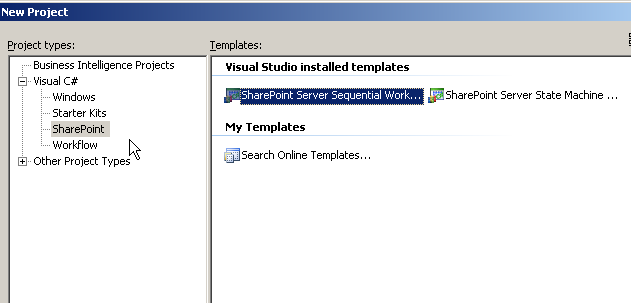
**Getting Started**

The first thing you need to get right is your development environment. I'm working on Virtual PC 2007 which has Windows 2003 server and MOSS installed on the image. This image is also my development environment so I've installed Office 2007 Pro Plus, and Visual Studio 2005 directly onto Windows Server 2003.

Once Visual Studio 2005 is on we need to install the necessary components to be able to work with Windows Workflow Foundation. Download and install the [Visual Studio 2005 Extensions for .NET Framework 3.0](http://www.microsoft.com/downloads/details.aspx?familyid=5D61409E-1FA3-48CF-8023-E8F38E709BA6&displaylang=en).

Once you have downloaded and installed those components you'll be able to create Workflow type projects but we need the SharePoint workflow templates. To get these, download and install the [Microsoft Office SharePoint Server 2007 SDK](http://www.microsoft.com/downloads/details.aspx?familyid=6d94e307-67d9-41ac-b2d6-0074d6286fa9&displaylang=en) which also includes the Enterprise Starter Kit.

Now that we have these two things installed in the Create New Project box in VS 2005 you should be able to create a SharePoint Sequential Workflow and a SharePoint State Machine Workflow.



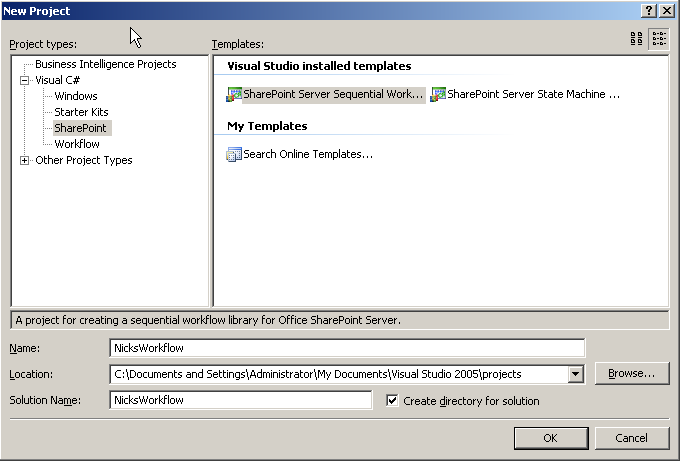
So that's the setup done, now lets just recap what it is we'd like our workflow to do. This is copied from the original workflow article:

*We want to allow our workflow to be associated with a list or document library. When a document is created or changed (this is a setting chosen upon binding the workflow to the list/library), a person specified will have a task created for them, with some pre-defined instructions and comments. When clicking the task the user will have a designed form where they will have instructions present, and can decide whether to complete the task with a checkbox.*

*From the above we’re going to need 2 InfoPath forms. Number 1 is used when binding our workflow to a list/library where we’ll enter the user who’ll get the task, and a few instructions and a comment. The second form is when the user clicks on the task. In this form they can add comments and click a checkbox to complete the workflow.*

This project is basically the HelloWorldSequential workflow from the ECM starter kit, but it's a nice walk through of how to put it together yourself!

Lets go and create our workflow project in Visual Studio 2005. We're going to be using a sequential workflow template, and lets give it a name of NicksWorkflow

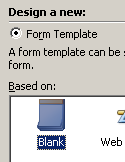


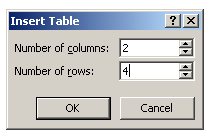
Once the project is created you'll see in the Solution Explorer that there's a new folder called Deployment Files. This is where you'll now find feature.xml and workflow.xml. We'll dig into how to deploy our workflow in the next blog post so don't worry about these for now.

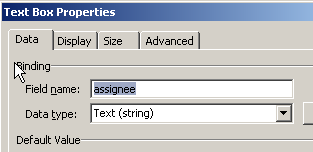
As with the previous article, before we start writing any code we want to create our two InfoPath 2007 forms. We want to do it this way as we're going to generate a C# class from one of our forms to help with the passing of data to and from it. The first form we are going to create is the initiation form. This captures information such as who we want the task to be assigned to, a field for instructions to them and a comments textbox.

**Initiation Form**

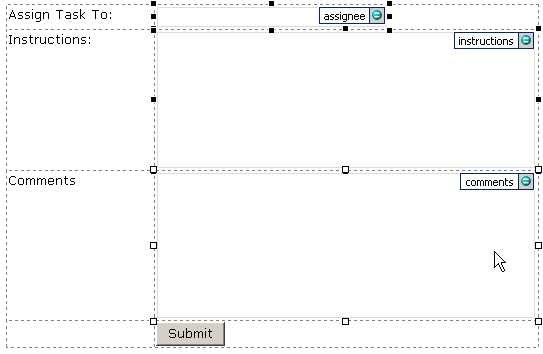
1, Open up InfoPath 2007 and from the first form select 'Design a form template...'  
http://weblog.vb-tech.com/nick/workflow/workflow3.png

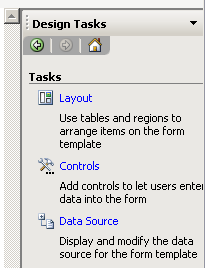
2, Click OK to create a blank form template...  


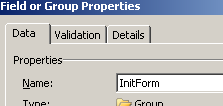
3, From the top menu bar click Insert->Layout Table... and then select your table to have 2 columns and 4 rows...  


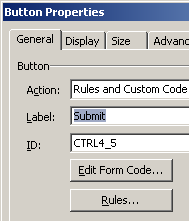
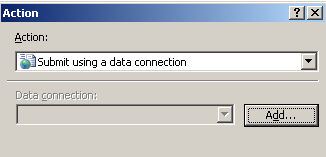
4, Add three textboxes and a button to the columns in the right, and a description in the cells to the left of each textbox. We need to also give each textbox a proper name, do this by double clicking on it, and entering the new name into Field Name  


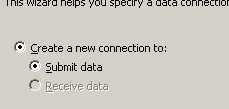
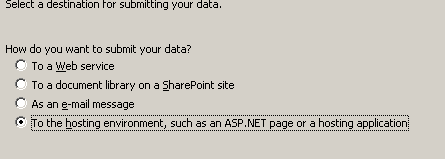
Once you've renamed your textboxes and given the button a Label value of submit, your InfoPath form should look something like this:

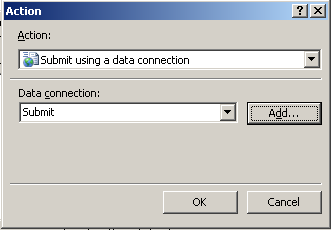


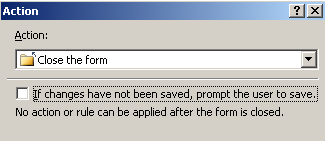
5, Data from InfoPath forms is represented by XML. To make it easier for us to get our hands on the XML data we require we can give the XmlElement that holds these control values a better name. From the Design Tasks toolbar click on Data Source...  


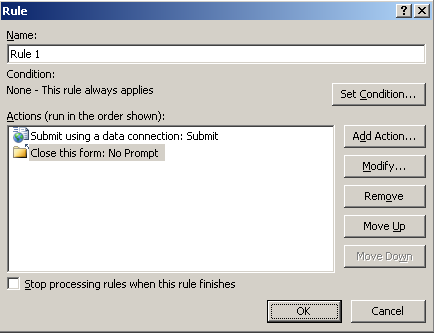
Double click on myFields  
  
And in the following form that opens up enter InitForm as the Name value...  
  
Click OK to close that.

6, Now we need to configure what happens when people click on the submit button. Double click on the submit button to bring up it's properties, from the General tab click on the Rules button  
  
In the following form, click Add to create a new rule. Click 'Add Action', and from the following form chose 'Submit using a data connection' from the drop down, then click the Add button just below:  


In the next wizzard chose the following options to submit your data:  
  
and then the hosting environment as the destination  


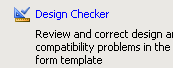
Leave the name as Submit and click Finish. And then OK on the 'Add Action Form'  


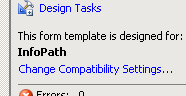
Now we want to add another action, so back in the Rule form click 'Add Action...' This time the action 'Close this form' and make sure the checkbox is not selected...  


Clicking OK takes us back to our Rules form. That's the two actions we want to add for our Submit button. With them both together they should look like this:  


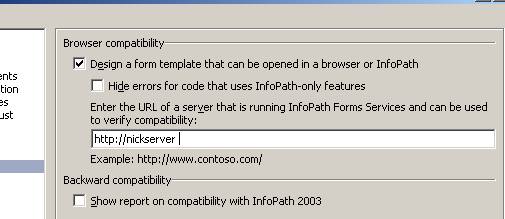
Click OK to close this form.

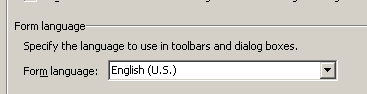
7, Now we need to ensure our form can be viewed in a browser as this is how InitForm will be used when setting up a workflow.

Go back to our Design Tasks toolbar, and click 'Design Checker'  


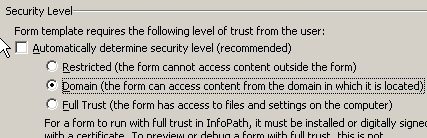
From the Design Checker pane chose 'Change Compatibility Settings...'  


Click the checkbox to allow our form to be opened in a browser, also enter the url for MOSS 2007 to help verify compatibility...



While you are in this form, click on the Browser category, and ensure the language selected is infact a language pack that you have installed on your MOSS server...  


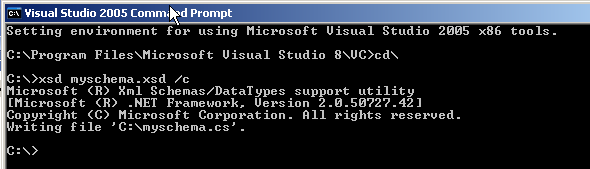
One final step is to go to the 'Security and Trust' category. In here untick to automatically determine the level of security, and select Domain...



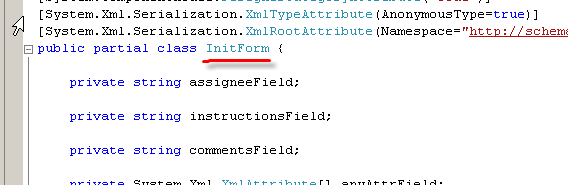
Click OK to close the 'Form Options' form. Now we need to save our form. Save it directly to the C:\ as DemoInitiation.xsn. Once saved we can publish it by going File->Publish, chose to publish it 'To a network location' and then chose the location of your VS 2005 project, and the "Deployment Files/Feature Files" folder.  
  
Click Next, remove the path as the alternate access location. You will get a warning when clicking next on that form but that is ok. If you do not remove the alternate access path you will have problems when publishing your form to work with workflow. Finally Publish and Close.

When we submit our forum to SharexPoint we need to be able to get the data from the submitted form to be able to use in our workflow. To help do this we can generate a class using xsd.exe based off the form schema file. First we need to save our InfoPath form as Source Files. File -> Save as Source Files. Browse to the location you want to save the file (c: is a nice easy place), and click OK. By default the source files are saved as filename myschema.xsd. Once saved close InfoPath.

Now open up a Visual Studio 2005 command prompt and navigate to where you saved myschema.xsd. From the prompt type xsd myschema.xsd /c



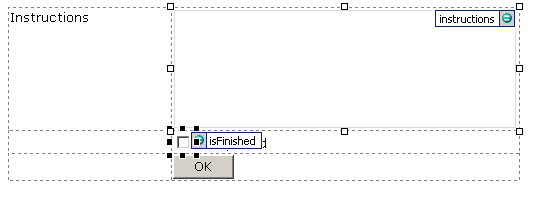
this generates a c# class file called myschema.cs. Rename the file to InitForm.cs, and add it to you VS 2005 workflow project. If you take a look inside the classes code you’ll see the name of the class is the same that we gave to the forms field collection.



**Edit Task Form**

Now we need to jump back into InfoPath 2007 so we can create our form that enables users to edit the task they get assigned to complete the workflow.

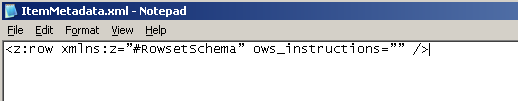
1, Open InfoPath 2007 again and create a blank form as we did before. On this form we’re going to want an instructions textbox where we’ll place the instructions entered when we bound the workflow to a list/library, a checkbox to check to say the workflow is complete, and an ok button. Lay it all out and rename the fields to something like below...

  
Underneath the isFinished tooltip is simply some text of 'Completed'.

2, Add the same rules to the OK button as you did to the submit button for the DemoInitiation form (ie submit to a hosted environment, and then close).

3, When this Edit form opens we’re going to want to pass some data to it (ie the instructions), we do this by creating a task schema and adding it as a secondary data source.

Open up notepad and add the following:  
<z:row xmlns:z="#RowsetSchema" />  
In this file we need to define every property that we are going to pass to the Task Edit form. To do this, add an attribute comprised of the prefix ows\_ and the name of the task property. Set the attribute equal to an empty string. So to pass the instructions to the Task Edit form that were added when we bound the workflow to the list/library we add the following attribute:  
ows\_instruction=””

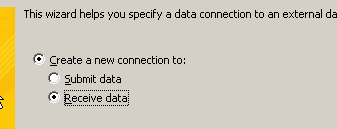
So your finished notepad text should be:  
<z:row xmlns:z="#RowsetSchema" ows\_instructions="" />  


It doesn't matter where you save the file, just make sure it's called ItemMetadata.xml (yes it does matter about the case with this).

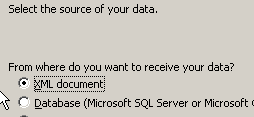
4, Now we need to add the task schema to our Edit Task form as a secondary data source. Back in InfoPath, in the Design Tasks pane select Data Source, and then click 'Manage Data Connection'.

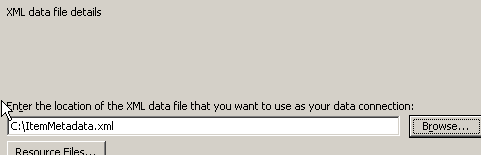


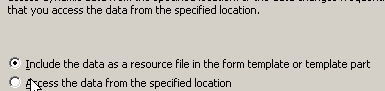
In the form that opens you'll see there is a Submit data connection already. On the form Click Add to create a new data connection. On the wizzard form that opens click so the new connection receives data...



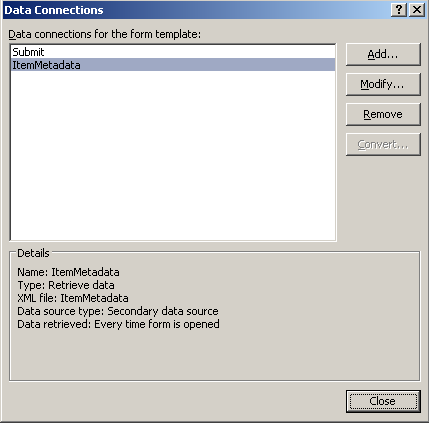
Select XML Document as the data source to receive data from...



Then browse and select the ItemMetadata.xml file you created in Notepad in step 3...  


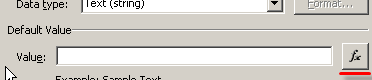
Leave the option selected to include this file as a data source...  


Click Next, and then Finish. The data connections form should now look as below:

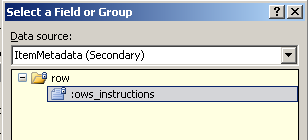


Click Close to get that job done!

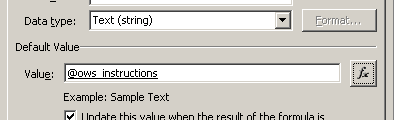
5, Now we need to bind the data that we are receiving from our new data connection to the necessary fields. Double click the instructions textbox and on the Data tab, under Default Value, click the formula button (underlined in red)...



On the Insert Formula dialog box, click Insert a Field or Group. In the Select a Field or Group dialog box, select your ItemMetadata data connection from the drop down menu. Select the ows\_MetaInfo\_instructions element.



Click OK. On the Insert Formula dialog box, click OK. On the Properties dialog box, click OK.



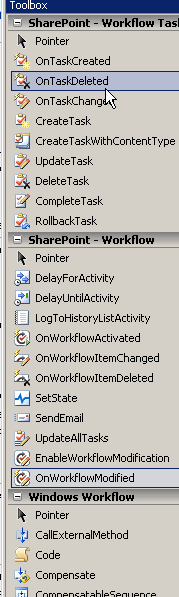
6, Now we need to do the same things as we did with the initiation form, set the form as browser enabled also entering the url of our MOSS server, check the language setting of the form, and also set the Trust Level as Domain.

Save the form to c:\ again (or whever you saved it), and publish it to your VS workflow project, "Deployment Files\Feature Files" directory again.

And that's it for InfoPath, our forms are created and ready to use!!!

**Back to Visual Studio 2005**

Now we can get down to some coding! In the solution explorer double click on Workflow1.cs and up will open the workflow designer view. If you open up the toolbox you'll see three new groups of components, SharePoint - Workflow Tasks, SharePoint - Workflow, and Windows Workflow.



As you can see on the Workflow1.cs design surface it already creates the first workflow step for us with a onWorkFlowActivated action. This will always be the first Workflow action of any workflow. Below this action is an arrow and a kind of stop sign. We can drag and drop any new workflow actions from the tool box and place them on the arrow. Before we do that though there are a few properties we can check before we get going. If you go into the code view of Workflow1.cs you’ll see

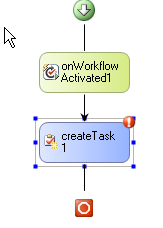
public sealed partial class Workflow1: SharePointSequentialWorkflowActivity  
{  
    public Workflow1()  
    {  
       InitializeComponent();  
    }

    public Guid workflowId = default(System.Guid);  
    public Microsoft.SharePoint.Workflow.SPWorkflowActivationProperties workflowProperties = new Microsoft.SharePoint.Workflow.SPWorkflowActivationProperties();  
}

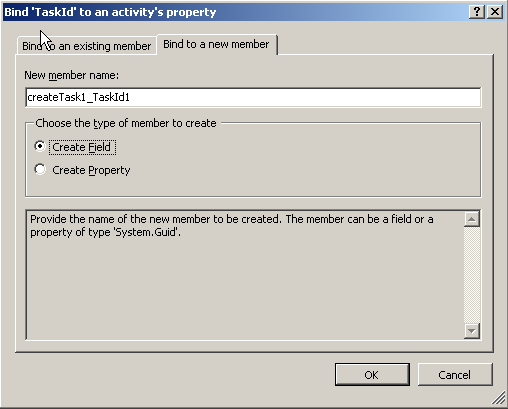
This is all created for you because you used a template project. If you view the properties of work onWorkflowActivated1 you’ll see the following important properties set for you:  
CorrelationToken : workflowToken  
    OwnerActivityName : Workflow1  
WorkflowProperties – expand this  
    Name : Workflow1  
    Path : workflowProperties – again this is a variable that was created for us and set from the code above.

We also get an event for this control. When this fires we’ll want to setup any initial variable values that are required for the workflow. Type into the Invoked space onWorkflowActivated and press enter. You’ll see it goes to the code view and creates our event handler for us with the correct interface. We’ll come back to this later.

Now to add our first workflow control. Now that the workflow has actually started the first thing we want to do is create a task for the person who’s been set to complete it. Remember we defined this when attaching the workflow to an actual list in SharePoint. To do this drag and drop a CreateTask control just below our onWorkflowActivated1 control.

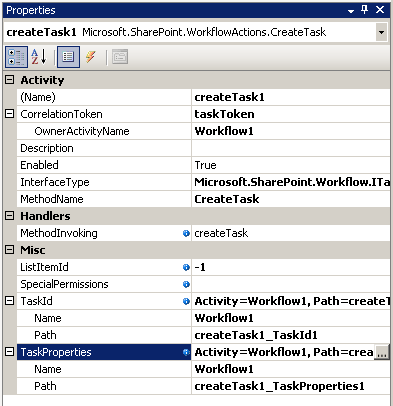


We need to set a few properties for this control. First type in the Correlation Token as taskToken. Upon pressing enter you’ll see that you can expand this property to reveal OwnerActivityName which once again should be set as Workflow1.   
Next we’ll set the TaskId and TaskProperties which can be accomplished in a couple of ways. First method is to click on the default value of TaskId (0000–0000…. or something) and you’ll see three ellipses on a button appear. Click on this and it’ll open a dialogue box for you. Click on the ‘Bind to a new member’ tab and click the 'Create Field' radio button.



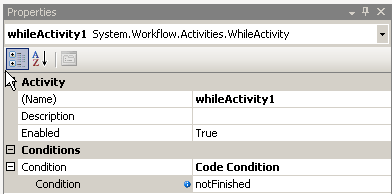
Click OK. In the propeties window of createTask1 you’ll see now that not only have 2 pairs of name and values gone into the TaskId property, but you can expand them out to set them seperately. Also if you switch to code view, you’ll see that a variable called createTask1\_TaskId1 (the name that was entered in the 'new member name' textbox) has been added to our code for us.

Follow the same procedure for TaskProperties. Switch back to the code view again and you’ll see the variable created for us. If you go back to the createTask1 properties view and click the ellipses for TaskProperties again, you can see in the ‘Bind to an existing member’ that this property is bound to taskProps in our Workflow1 class. So as well as using this dialog box and the ‘Bind to a new member’ tab, we could have gone into our code view, created the createTask1\_TaskId1 and createTask2\_TaskProperties1 variables ourselves, and then used the Bind to an existing member view to set the values of the properties. Final thing to do is create the event that fires when this activity executes. In the Method Invoking field enter createTask and click enter. Again is creates our event handler and interface for us. Again we’ll come back to this is a little while. Finally we need to set the CorrelationToken as taskToken (just type it in). The same token name will be used in other task activities that we drag onto our workflow. Using the same token ensures we are working with the same task. Here is how the properties window for createTask1 should be looking:



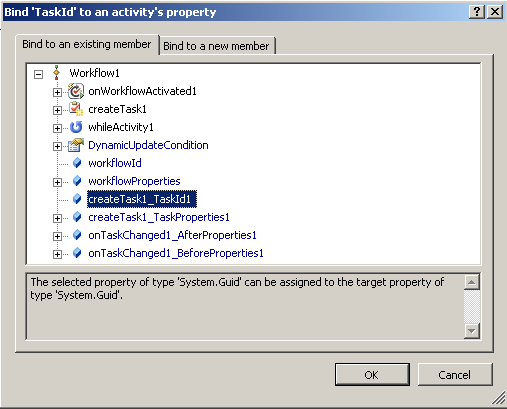
Now that our task has been created for the assigned user, we need to add some waiting functionality to enable the workflow to wait for the task to be completed by the user. We do this using the While workflow component.

Drag and drop a While component from the Windows Workflow section in the toolbox between the createTask1 and the end of the workflow. In the properties window select Code Condition as the Condition property's value and then expand the field. Then type notFinished into the extra field presented and press enter. This will create an event handler for you in code where you check whether the While condition has been met (ie has the user completed their task).

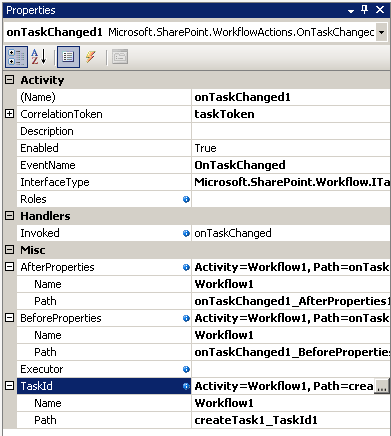


Now we want to add an activity to the centre of our While loop. Here we’ll place an onTaskChanged. This basically means that the While loop will execute and check our code condition, every time our task is edited. It won’t be able to exit out of the While loop until our method notFinished returns false (which means it is finished!). Drag an onTaskChanged component and drop it in the middle of our While loop.

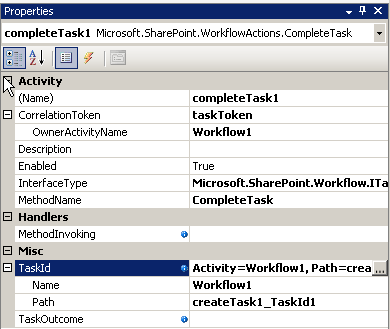
You’ll see in the properties window there are quite a few things we need to set:  
AfterProperties = click the ellipses and bind to a new Field member  
BeforeProperties = click the ellipses and bind to a new Field member  
CorrelationToken = select taskToken from the drop down  
Invoked = onTaskChanged. When you’ve typed it in press enter to create the event handler for you.  
TaskId = click the ellipses and bind to the existing member called createTask1\_TaskId1



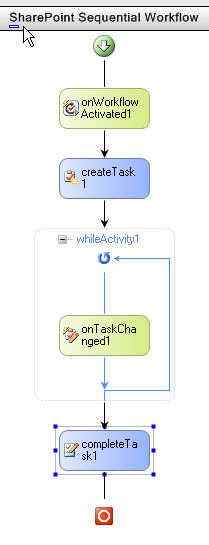
The properties window for onTaskChanged should now look as below, make sure you set the Correlation Token:



The final component we want to add to our workflow is the CompleteTask activity. Drag and drop a CompleteTask component between the While component and the stop workflow activity. In the properties set the Correlation Token to taskToken from the drop down. Again with the TaskId click the elipses and bind it to createdTask1\_TaskId1.



Now we’ve added all the components to our workflow. Our workflow designer should look as below:



Now we’ve got all our components added, and event handlers and methods created, we need to add some code to Workflow1.cs.  
Right click on Workflow1.cs and click View Code. The first bit of code we need to add declares a few variables we are going to need in our workflow. Just above the onWorkflowActivated method add:

private String assignee = default(String);  
private String instructions = default(String);  
private String comments = default(String);

These three values are ones that we are going to get from our DemoInitiation form. These properties are passed to the workflow as an XML string represented by the InitiationData property of the SPWorkflowActivationProperties object. To access these properties we need to parse this XML string. This is where we make use of the generated class based on the schema of our initiation form that we did in step 4 and added to our project as InitForm.cs. To get these values add the following code to the onWorkflowActivated method:

workflowId = workflowProperties.WorkflowId;  
XmlSerializer serializer = new         XmlSerializer(typeof(InitForm));  
XmlTextReader reader = new XmlTextReader(new         System.IO.StringReader(workflowProperties.InitiationData));  
InitForm initform = (InitForm) serializer.Deserialize(reader);  
assignee = initform.assignee;  
instructions = initform.instructions;  
comments = initform.comments;

The next method we need to add some code to is createTask Method. Here we want to set some properties of a task as this method fires just before the task does actually get created. Here’s the code:

createTask1\_TaskId1 = Guid.NewGuid();  
createTask1\_TaskProperties1.Title = "Demo Task";  
createTask1\_TaskProperties1.AssignedTo = assignee;  
createTask1\_TaskProperties1.Description = instructions;  
createTask1\_TaskProperties1.ExtendedProperties["comments"] = comments;  
createTask1\_TaskProperties1.ExtendedProperties["instructions"] = instructions;

as you can see there are a predefined properties such as Title and AssignedTo to use, and if there’s any other properties you want to name and create yourself you can use the ExtendedProperties. This is a hash table so you can name things as you like.

The final bit of code we need to add is probably the most complex to get our heads around. We need to add a private Boolean variable called isFinished. Place this just above the notFinished method:

private bool isFinished;

Now our While loop calls the notFinished method. One of the objects passed in is ConditionalEventArgs which has a property called result. If result is set to false, the while loop will end, if result is set to true, the while loop continues. As you should be able to work out, the variable above we created isFinished will be true once the task has been finished. Therefore we have to negate it to set it’s value to the ConditionalEventArgs result property. In the notFinished method type the following code:

e.Result = !isFinished;

The last thing we need to do is actually set isFinished when our task gets edited ie when onTaskChanged event files. In here we just need to parse out the isFinished value from our onTaskChanged1\_AfterProperties1 object. At the code below to the onTaskChanged method:

isFinished = bool.Parse(onTaskChanged1\_AfterProperties1.ExtendedProperties["isFinished"].ToString());

And that’s it. Build your solutions, and hopefully everything compiles succesfully. Now I was also going to include instructions on how to deploy this workflow, but I’m going to save that for a few days time.

If you find any errors in the post please let me know...