So You Want to Develop SharePoint Workflows…

Key Concepts for Understanding How to Develop SharePoint Workflows in Visual Studio

Authored by Eilene Hao

Published in December, 2006

Applies to:
Microsoft® Windows® SharePoint® Services 3.0 and
Microsoft Office SharePoint Server 2007

Table of Contents

Introduction – What Is This Paper?

So you want to develop SharePoint workflows… not sure where to start? Then perhaps this paper can help.

With so much information out there, it's sometimes hard to see the big picture amidst details and know what's really important or how to put all the pieces together. This document aims to be a cohesive, quick guide of key concepts, from high level workflow terminology to critical architectural concepts. We will go over the steps for developing a workflow, covering the most common questions and points of confusion, and noting pitfalls and tips for success along the way. Understanding the material in this document will hopefully equip you with the right tools to solve or work around most problems that you encounter.

Note that this is *not* a replacement for the SDK, and you should still look there for the finer details. But it will highlight the major concepts so that you have a sense of what to look for.:)

The Workflow Objects

Workflow Template vs. Association vs. Instance

There are a few important terms in workflow that often get confused: template, association, and instance. **If you don't read anything else from this document, make sure you understand this section.** Keeping these terms straight is critical to understanding other documentation and the pieces of a SharePoint workflow.

First we have a **template**. The template is the SharePoint workflow "Feature" that is deployed to a site collection (a Feature is a custom SharePoint part, such as web parts or content types). The template basically says, "Hey, this workflow feature uses these forms and this assembly." It's just a skeleton stored in the central repository for a site collection that any list or content type can look at.

But it's just for looking; a list can't use the template directly. Should the workflow be started automatically? Where should tasks be stored? We need an extra layer to customize a template with this information. Enter the **association**. To connect the list to the template, we create an association. An association stores customizations for the template and binds it to a list or content type. Such customizations can include a title for the association, when to start running the template (e.g. manual or automatic start), and custom workflow settings like default approvers. The important thing to understand here is that **a list or content type can have multiple associations of the same template**. For example, you can have associations Expense Approval and Publish Approval that that bind to the Approval workflow template, each customized with their own settings and default values; they should just be named differently.

When we actually start a workflow, we create a workflow **instance** of an association (not a template). Each item can run **one** instance of a particular association at a time. So there can only be one

ExpenseApproval instance running on each list item, and one Publish Approval instance. But both are based on the Approval template.

So the workflow that you develop in Visual Studio is a workflow template that defines the general feature. The **Association form** that it specifies collects customized association data to create the association. The **Initiation form** collects data to create a new instance from an association, and uses the association data to pre-populate initial values.

Tasks, History Items, and Modifications

The other important workflow-specific SharePoint objects are the workflow task, workflow history item, workflow modification.

Once a workflow is running, it can call any SharePoint object model code. Because SharePoint workflow focuses on human-based process rather than automating purely programmatic steps, an important action for workflows is creating tasks on a SharePoint task list specified in the association. **Workflow tasks** are like normal SharePoint task items, but they also link to the workflow that created them and can notify their "parent" workflow when they change or get deleted so that the workflow can take appropriate action. Users can edit a workflow task by using a **task form** (also specified in the template), and the workflow can process these edits and mark the task as complete. Since part of the benefit of having workflows is to gain visibility into a process, the workflow can log events to a **workflow history list**. **History items** are just list items that have special columns for workflow-specific data, such as instance id, association id, etc. These items appear on the workflow status page, which filters according to instance id.

But as with any process, sometimes a workflow needs to be adjusted while it in progress, for example, if a new person needs to be added, if a due date needs to be changed, or if a parameter, such as a "max limit" for an expense report that determines whether the report requires manager approval, should be reduced in-flight. To allow users to talk to or change workflows in progress, SharePoint has **workflow modifications**. The modification object represents a channel of communication with a workflow, and once registered with a workflow, can "wake" a workflow up when the modification is called. A **modification form** collects data required for the modification, and once the workflow is awake, it can perform whatever modification the developer programs. In general, use modifications if you need to talk to a running workflow and don't want to do it through tasks.

Understanding these fundamental concepts in SharePoint workflow will help you understand how all the pieces fit together when you create your workflows. To summarize, here is an application of these terms in the workflow life cycle:

1. Developer develops a **workflow template**.
2. Server box administrator deploys the template to a site collection.

3. List administrator **associates** a workflow template with a list or content type on that site collection to create a new **association.**

4. User starts an **instance** of a workflow association on an item (or workflow instance starts automatically on event).

5. Workflow executes, perhaps creating some **tasks** and **history items** on the Tasks and Workflow History lists specified in the association.

6. Workflow owner (the person who started the workflow) or list administrator can talk to a running workflow at any time by **modifying** it with a **modification**.

David Chappell has a great whitepaper on MSDN that describes the structure in more detail, so if you're interested, I'd recommend it☺.

Here is a quick reference of the concepts and related SharePoint objects:

Template –Base workflow feature stored in the central feature place. If activated on a site collection, it is visible and available for adding to a list or content type on a site collection in a process called "association". For VS workflow developers, the "workflow.xml" defines the template, including what assembly contains the workflow schedule (code), and what forms to use for different interactions with the workflow. Object: SPWorkflowTemplate Related: SPWorkflowAssociation.BaseTemplate, SPWorkflowManager.GetWorkflowTemplatesByCategory()
Association – Custom binding of a template to a list or content type that makes a template available to run. Workflow instances start from associations, not templates. You can have more than one association that binds to the same template to a list or content type. Object: SPWorkflowAssociation Related: SPWorkflowAssociation::CreateListAssociation(), SPWorkflowAssociation::CreateListContentTypeAssociation(), SPWorkflowAssociation::CreateSiteContentTypeAssociation() SPList.WorkflowAssociations, SPList.AddWorkflowAssociation(), list.RemoveWorkflowAssociation() SPContentType.WorkflowAssociations, SPContentType.AddWorkflowAssociation(),

SPContentType.RemoveWorkflowAssociation() SPWorkflow.ParentAssociation
Instance – a running workflow on an item or document, based on an association. Only one instance of an association can be run on an item at a time. Object: SPWorkflow Related: SPListItem.Workflows SPWorkflowManager.StartWorkflow(), SPWorkflowManager.RemoveWorkflowFromListItem() SPWorkflowManager::CancelWorkflow()
Workflow Task – special SharePoint task item that has a parent workflow Object: SPWorkflowTask Related: SPWorkflowTaskProperties – object used within workflow to set and receive task data SPWorkflow.Tasks
Workflow Modification – object that allows users to interact with or change workflows that are in progress Object: SPWorkflowModification Related: SPWorkflow.Modifications SPWorkflowManager.ModifyWorkflow() SPWorkflowModificationCollection.AddOrUpdate()

Crash Course on Workflow Mechanics

It’s not critical understand the low level details of how workflows operate internally to develop workflows, but there are a few the important high level concepts that will help:

1) When running workflows reach a point in the code, a.k.a. **schedule**, where they need to wait for something to happen (such as a task edit), they are **serialized** (i.e. turned into binary) and

dehydrated to the SharePoint database. This means that the workflow object is turned into a string and stored to the database. We can say that the workflow is **asleep**.

2) Dehydrated workflows are no longer in memory.

3) When the event the workflow is waiting for happens, the workflow is **deserialized** and **wakes up,** or **rehydrates**, then continues where it left off in the schedule.

A typical SharePoint workflow is a series of "execute custom code, dehydrate, and rehydrate" cycles. Dehydration and rehydration requires the use of workflow services.

A **local service** is a line of communication between SharePoint and the workflow. The important thing to know about them is that they can manipulate workflow items and **register workflows** with SharePoint events, **put workflows to sleep** to wait for the events, and **wake up** workflows when those events fire.

The SharePoint workflow service APIs provides both **methods** and **event handlers. Methods will perform actions, and handlers will put the workflow to sleep and wake up on events.** For example, using the CreateTask service method will create a task and register the workflow for events on that task. Using the OnTaskChanged service event handler mapped to that task will dehydrate the workflow while it's waiting and rehydrate the workflow when the task is changed.

To **map these functions to the same task**, the services **use correlators**, unique identifiers that the workflow engine can use to map events and actions to objects. These are specified in a workflow by using **correlation tokens** and GUIDs. (Both correlation token and GUID are required.)

Ok, so that was really dry. But basically, the key thing to know is that you **must use the service methods** to get the workflow **to respond to events** on items. The other thing to know about service methods is that they're transacted actions, batched actions that are **not committed until the workflow is dehydrated**. This means that if you use CreateTask, the task will not actually be created until the workflow is successfully dehydrated.

Pitfall: Operating on uncommitted items
Do not use SharePoint object model (OM) calls on tasks or items that have not been committed. Remember to dehydrate first, using an event handler or delay, before trying to fetch objects. For example, if you use CreateTask, the SPListItem for the task will not exist until you use an event handler like OnTaskCreated or Delay to put the workflow to sleep. Using other workflow service methods is ok, but don't use direct OM on the SPListItem object before putting the workflow to sleep.

Planning Your Workflow: Two Things to Keep in Mind

Before writing any code, a little planning to map your business process to SharePoint workflow can go a long way. The two most important things to keep in mind are that

a) SharePoint workflows are **document-centric**, meaning they run process around a document or list item, and

b) They are **human-based**, meaning that they can drive people process by assigning tasks, in addition to automated computational process.

How do these things help? Well, knowing the capabilities and strengths of SharePoint workflow can help you plan the structure of the workflow's surrounding SharePoint objects, what the workflow should do, and how it can get the information that it needs. For example, since a workflow instance runs on an item, you have access to the content and fields on that item; do you want the workflow to manipulate metadata, and if so, **what columns do you need**?

If you're designing a workflow that doesn't really "process" or route a single document, e.g. a workflow that involves reviewing a set of documents instead of just one, would it work to use items that link to those documents in a regular **list instead of a document library** as starting points for the workflow?

If you need to drive people to do different tasks, **where and when should tasks be assigned**, and **what data do you need to collect** from those people when they complete their tasks?

Also, **workflows run as System Account** (Administrator privileges) so that it can do things that ordinary people cannot, like creating audit entries. Can you take advantage of this for any part of the process?

Once you have a sense of what your workflow needs to do, you can start the authoring process.

Five Steps for Developing Your Workflow

Okay, the moment you've all been waiting for: how do you write a custom SharePoint workflow? In this section, we will break it down into five steps:

1) Model your workflow in Visual Studio
2) Create and bind your forms
3) Code your workflow
4) Deploy your workflow to the server
5) Debug

(Note that these are only suggested steps; you can do this any way that fits best for you.)

Step 1: Model Your Workflow

The first step is to model your workflow using the Visual Studio Extensions for Windows Workflow Foundation designer, i.e. create a new workflow project, drag and drop activities into your workflow surface, and set the properties that you need. Having a skeleton of your "flow" will help you see what needs to be done and where you will need to bind your form data in later steps. Also, initializing the properties on your activities up front may save you some time having to switch back and forth between designer and code-behind file when you start writing your code.

FAQ: The workflow template can't open in Visual Studio
When using the SharePoint workflow VS templates, you may encounter an error opening your .csproj. This usually means that you do not have the Visual Studio Extensions for Windows Workflow Foundation installed. Check to see if you have anything under the "Workflow" node before you worry about the "SharePoint" node.

SharePoint Activities
SharePoint comes with a set of activities. Many of them wrap the service methods mentioned in the mechanics section above, or rather, they are the service APIs in activity form. The **method activities** are **blue**, and the **events handler activities** are **green**. The methods derive from the WF CallExternalMethod activity (see Item Activities sample in the ECM Starter Kit), and the functions you write for the Method Invoking property are executed right **before** the service method executes. The handlers derive from the HandleExternalEvent activity, and the functions you write for the Invoked property are executed **after** the service handler executes and wakes up the workflow.

Add an event handler activity whenever you want the workflow to commit the method actions or go to sleep, then wake up when an event fires.

To create and bind variables to your activity properties, use the activity property dialog by clicking on the blue icon next to the property. If instead of clicking on the blue button, you type in a value directly, for example, typing 1 for an int property, the property will be set to that constant value. (It's less expensive for performance to do this if you know a value won't change)

Pitfall: Using CreateTask or OnTaskChanged's SPWorkflowTaskProperties objects as a real time representations of the SharePoint task item
When using CreateTask, the object that you bind to the TaskProperties property to set initial data for the task is only initial data; it's not synchronized with the task, so it will not reflect any changes to the task, or vice versa. The same concept applies to OnTaskChanged's AfterProperties property.

The Importance of Correlation Tokens
A common mistake when initializing properties is not setting the correlation token correctly to map different activities to the same object. If not done correctly, the workflow will not wake up or respond to events.

Think of a correlation token as an identifier that WF uses to identify and respond to the corresponding objects in SharePoint. For example, if you have in a sequence the activities CreateTask, OnTaskChanged (to wait for the user to edit his task), then CompleteTask, you want these three activities to map to the same task in SharePoint; you do this by specifying the same correlation token in all three.

You will need a token for activities that relate to the workflow itself, a token for activities related to a particular task or item, and a token for each modification. Here is a **grouping of activities that need the same correlation token** (each group should have its own token):

"Workflow" token:	Task token:	Modification token:
• OnWorkflowActivated • OnWorkflowItemChanged • OhWorkflowItemDeleted • SetState • SendEmail • UpdateAllTasks	• CreateTask • CreateTaskWithContentType • UpdateTask • DeleteTask • CompleteTask • RollbackTask • OnTaskChanged • OnTaskDeleted • OnTaskCreated	• EnableWorkflowModification • OnWorkflowModified

Missing from this table are the list item service APIs, which don't have activities around them in the box. But if you use wrapper activities for them (such as those included in the ECM Starter Kit item activities sample), you would treat them similarly to tasks: every related item activity should use the same correlation token.

Pitfall: Using the workflow token as the task token
Sometimes developers use the workflow token as their task token; now all tasks map to the same thing, right? This isn't quite correct. Make sure each group and object has its own token. For example, don't use the overall workflow token for your tasks. Also, create a separate token for each distinct task. Note: if you have a task in a sequence inside the replicator activity, changing the "Owner Activity Name" to the sequence will automatically create a new token for every instance of the sequence.

Step 2: Design and Bind Your Forms

With your workflow skeleton in place, you should now be able to see where you will be collecting data and what data to collect. With this information, you can create your forms, set up a schema for the collected data, and write the code behind your workflow to utilize that data.

Behind the Scenes of the Four Types of Workflow Forms

There are four types of workflow forms that you can implement: **association, initiation, task edit, and modification**. To determine which ones you need, ask yourself the following questions:

Association: Do I want to input default values for initiation, or values that the workflow needs that only list administrators should set?

Initiation: Do I want people to start the workflow manually? If so, is there information that the workflow needs to start off with?

Tasks: Are people involved in my workflow? Will I be assigning tasks?

Modifications: Will the workflow need to be adjusted in-flight?

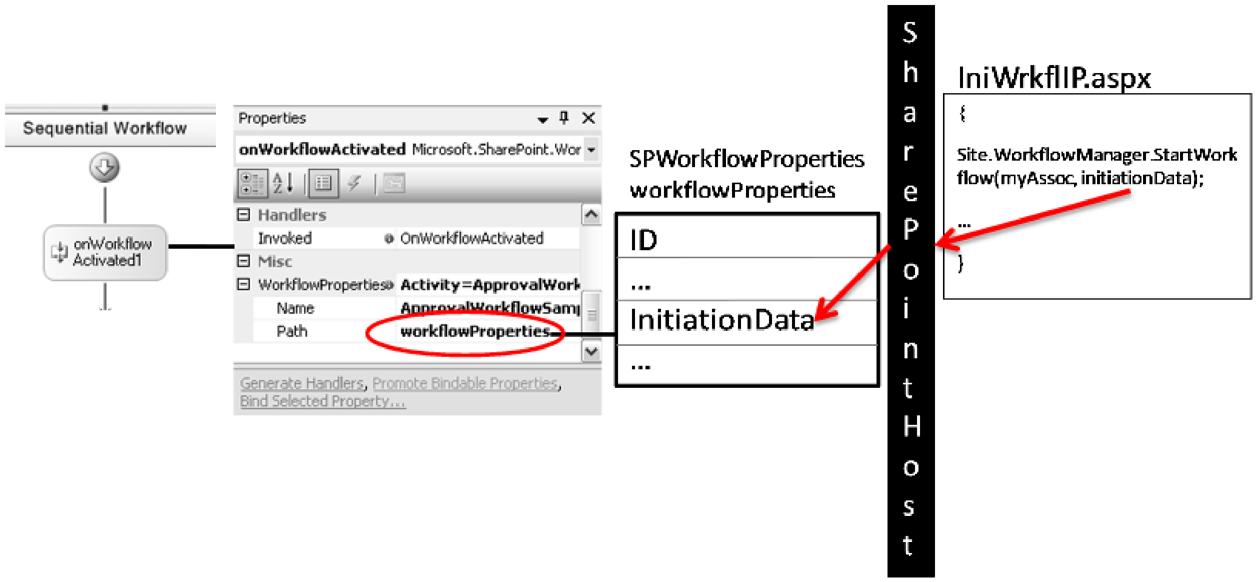
If you answer yes to these, then you'll probably need a form.

All your workflow forms will need to do the following things:

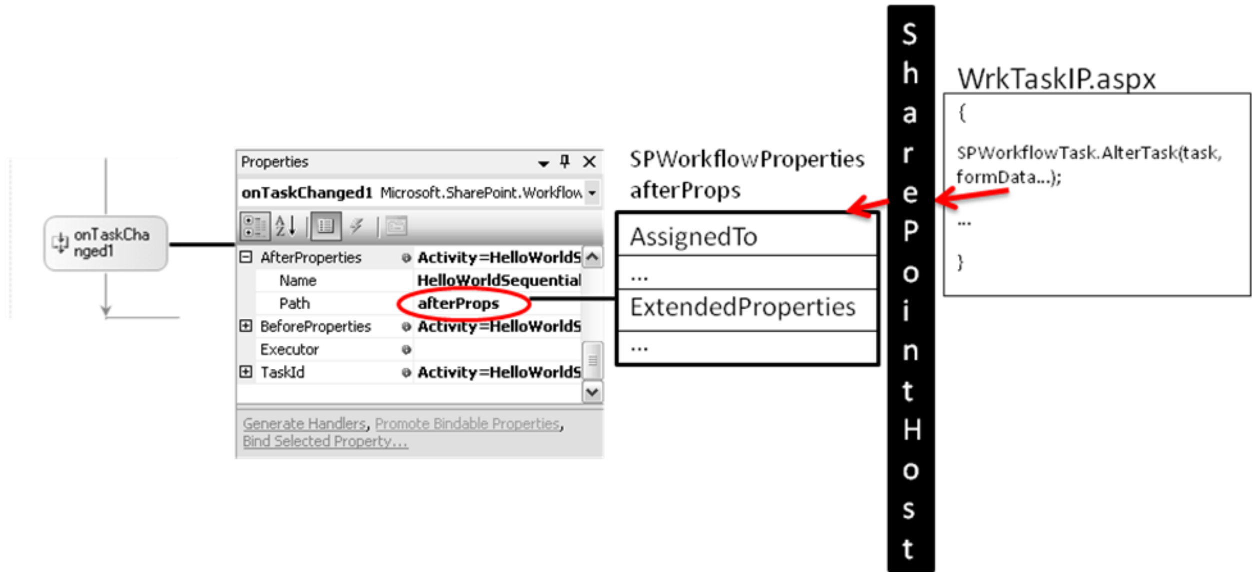
1) **Collect data** from the user
2) Call an object model function to **perform its action** (e.g. create an association, start the workflow, edit a task, etc). This function takes the data as a parameter and **passes it into the workflow**. (MOSS provides a shortcut for IP forms that we'll talk about in a bit)

Once this function is called, the SharePoint will **trigger an event** in the workflow schedule and pass in data (excluding associations, which happen outside of the schedule). Thus, the workflow **schedule must have a corresponding event handler activity** to respond to the event. The **data will be stored into a variable bound to a receiving property on that activity**.

For example as seen in the following diagram, to start a workflow, the initiation form calls a function called StartWorkflow, which takes the initiation data string as an argument. StartWorkflow triggers the workflow activated event, which triggers the OnWorkflowActivated activity (Hence, OnWorkflowActivated needs to be the first activity in your workflow). It stores the initiation data into the variable bound to the WorkflowProperties property on that activity. The following diagram shows this process: 1) aspx page calls StartWorkflow and passes in form data, 2) SharePoint stuffs the data into an event handler activity in the workflow (OnWorkflowActivated).



Similar model for editing tasks: 1) aspx page calls AlterTask and passes in form data, 2) SharePoint stuffs the data into an event handler activity in the workflow (this time, OnTaskChanged)



And it works the same way for modifications, which we won't show here. Below is a chart of what functions each form type calls, which activity you need to receive the event, and which property on that activity gets stuffed with the data:

Form Type	**Required Actions**	**Event Handler activity and Data Receiving Property**
Association (MOSS host page:	Create SPWorkflowAssociation object (SPWorkflowAssociation::CreateListAssoci	None (associations exist outside of the workflow schedule)

CstWrkflIP.aspx)	ation(… associationData)) Add association to list or content type (SPList.AddWorkflowAssociation(association))	
Initiation (IniWrkflIP.aspx)	Start a new instance of a workflow (SPWorkflowManager.StartWorkflow(association, web, list)	OnWorkflowActivated.WorkflowProperties Stored into these SPWorkflowActivationProperty fields: InitiationData, AssociationData
Task edit (uses form specified in OffWFCommon task content type, WrkTaskIP.aspx)	Change the task (SPWorkflowTask.AlterTask())	OnTaskChanged.AfterProperties
Modification	Modify the workflow (SPWorkflowManager.ModifyWorkflow)	OnWorkflowModified.ContextData

Specifying Which Forms To Use

When you have your forms, you'll probably want to specify which form to use for what stage of the workflow, i.e. which form to use for association, initiation, etc. This is defined in the workflow template xml file (workflow.xml), which we'll see in the deployment step. There is a one-to-one mapping between form and type, for example:

```
<Workflow

    Name=…

    TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F980F93160"

    AssociationUrl="_layouts/CstWrkflIP.aspx"

    InstantiationUrl="_layouts/IniWrkflIP.aspx"

    ModificationUrl="_layouts/ModWrkflIP.aspx">
```

In this snippet, we're using the out-of-box IP host pages for that come with MOSS (we'll explain what we mean by IP host pages in the next section). In short, you would use the pages specified above if you're using IP forms, and you'd want to specify your own aspx pages in these fields if you're using aspx forms.

For Tasks, instead of specifying a page, you specify a content type that specifies the appropriate view and edit aspx page. The OffWFCommon content type that comes with MOSS is for use with IP forms.

FAQ: How to specify multiple task or modification forms
You can only specify one aspx page for each form type, and one default task content type. However, your workflow might need more than one form for tasks and modifications. **If you are creating your own custom aspx pages** and need more than one task edit form, you will need to create a content type for each type of task and use the CreateTaskWithContentType activity to create a task that uses that type (instead of the CreateTask activity). If you need more than one modification form, either create an aspx page that redirects to multiple forms, or create a page with multiple views based on page parameters and workflow template fields. **If you are using InfoPath forms**, the MOSS out-of-box aspx pages, which automatically load the appropriate IP form based on "FormURN" tags in Metadata section of the workflow template xml.

InfoPath vs. ASPX Forms
We now know what needs to happen in the forms behind the scenes, so we just need to choose the technology: InfoPath (IP) or ASPX forms.

In reality, all (browser-based) workflow forms that you see are aspx pages. What makes InfoPath forms different is that they are **hosted in an XmlFormView control in out-of-the-box aspx pages in MOSS.** These are the pages we specified in our workflow.xml snippet in the last section.

There are several advantages to using InfoPath forms that you should consider:

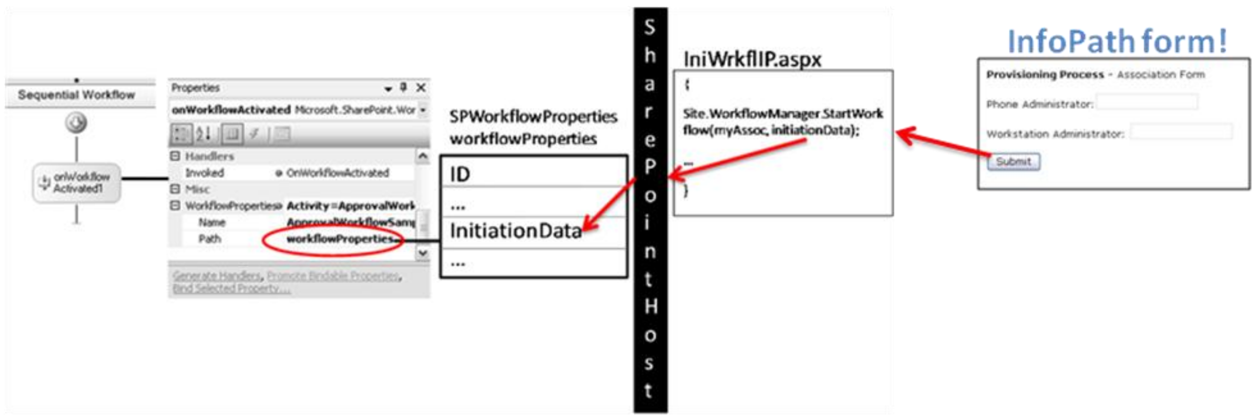
1) **They can be used directly in the core Office 2007 client applications.**
   The client applications (Word, Excel, PowerPoint, Outlook, InfoPath), like aspx pages, can host InfoPath forms for workflow. So when you click on that "Edit this task" button that appears in Outlook or Word, the IP task form will appear as a dialog directly in the application to maintain context. This form is exactly the same form as you'd see in the browser, so you have a **symmetric** experience. If you're using ASPX forms, this button will redirect you to a browser.

2) **They are easy to author with InfoPath designer.**
   Drag and drop, IP rules wizard… if you enjoy the authoring experience in InfoPath, this is a plus.

3) **No code required**
   No code required? But what about calling the functions to perform the specified action? Well, the **MOSS aspx host pages make the object model calls for you**, so that all you need to focus on is designing your IP forms in the IP designer. More specifically, **these forms 1) load the IP form, 2) wait for the IP form to send its xml back to it, and 3) make the OM calls.** If the out-of-box

functionality is not what you desire, you can code your own ASPX page to host the IP form and make the appropriate calls. But remember that you might also be able to write code behind your IP form to add that functionality.

So with IP forms, our data "flow" diagram now looks like this (hosted IP form submits its data as xml to host page, host page calls function, etc.):



```
{
Site.WorkflowManager.StartWork
flow(myAssoc, initiationData);
...
}
```

Creating InfoPath Forms
For every IP form you create for workflow, you'll need to do the following things:

1) **Name your fields**
Define your schema according to how you want to reference your data in the workflow. So if you have a control to collect comments and call it "comments", and in your workflow code, you'll be able to reference the data in that field as "comments".

2) **Add a submit button** with 2 rules
We know that IP forms are hosted in an aspx page or client application, so when the user wants to submit the form, have the submit button a) create a **connection to submit the xml data/string to the host environment** (the host will handle the rest), then b) **close the form** so that it doesn't just submit and stick around.

3) **Add domain level trust**
Yeah, you need this for workflow forms. In IP, it's in Tools->Form Options->Security and Trust

4) **Publish to your project folder**
Workflow forms will be deployed when you deploy your workflow, so keep them handy with your project so that the deployment tool picks them up and you can tweak as necessary. Don't forget to **leave the alternate access path blank,** or else it won't install!

Pitfall: Forgetting to clear the alternate access path
It's easy to forget to clear the alternate access path in the Publish wizard, since it's filled in by default after you specify the publish location; be sure to actively delete the default value that they provide.

Pitfall: Confusing the IP document with an IP workflow form
A trap that people run into is thinking that a workflow IP form is the same as an IP document that is saved as an xml file (such as a forms library document). If you are designing a workflow for a forms library, think of the library's form template as an ordinary document, and the workflow forms as process UI. Workflow forms are not files that are saved; they are merely UI to collect data, and the results are passed into the workflow, not a file.

A Fifth Step for Tasks: ItemMetadata.xml
If you need to load task data in your task forms without any code, you need an extra step. Unlike the other workflow objects, tasks are list items that can be tweaked in ways that the form and workflow don't expect, for example, adding a column and hence changing the item schema. So to compensate for the possibility of schema differences between the task data and the form, we let the developer define the fields in the task that are needed to populate the form controls, through a file called **ItemMetadata.xml**.

SharePoint passes the task as xml into this file, which **should be a secondary data source on the form.** It's **case sensitive**, and it uses the exact schema of the task. So step 5 (do this before step 4) for task forms is:

5) **Add an ItemMetadata.xml file as a secondary data source to your form**

The file contents would look something like this:

```
<z:row xmlns:z="#RowsetSchema"

        ows_instructions=""/>
```

where "instructions" is the name of a field or column on the SharePoint task itself. You can select ows_instructions as a default value for your fields.

Binding Form Data into Your Workflow
As we saw earlier, the form pages call functions that trigger events in the workflow and **stuff the form data into a property** on the corresponding activity. So all you need to do when the event is triggered is **write a handler that parses the data** and uses the data to perform actions in the workflow.

Initiation form data is passed into the OnWorkflowActivated activity as a **string**, so whether it's via a serializable class or by xml parsing helper functions, you just need to get the string into data you can use. **Modifications** also pass a **string** into the OnWorkflowModified activity. Parse this similarly to Initiation form data. Again, if you're using IP forms, the "Submit to host" passes the form data as an xml string to the host that is then passed directly into the function call, so for initiation and modifications, follow the form schema to parse it (e.g. by using the xsd generator tool to create a class for it).

Tasks are slightly different in that when a form is submitted, data is **saved directly to the task item** on the list. SharePoint does a little magic on the task and copies the task info into the workflow as an

SPWorkflowTaskProperties object that contains the **properties that changed** on the task item itself (and null for properties that didn't). For IP forms, the host parses the xml and saves each field to a column on the task item with the same name. So if you have a form field named "AssignedTo", it would write the value to the "AssignedTo" property on the item. For a field that does not have a matching column on the list, e.g. "foo", that field will be saved as a hidden field on the task object. Hidden fields are exposed in the workflow as a hashtable called ExtendedProperties on an SPWorkflowTaskProperties object, and can be referenced in the workflow by using the field name as a key, e.g. afterProps.ExtendedProperties["foo"].

Binding Object Data into Your Forms
Pre-populating data into the forms does not come from the workflow itself. Rather, these forms get their data from SharePoint objects related to the workflow.

For example, both association and initiation forms load initial data from the SPWorkflowAssociation object on the list or content type.

Task forms pull their data from ItemMetadata.xml, which is the SharePoint task object as xml.

Modifications pull data from the ContextData string in the SPWorkflowModification object. Note: this object needs to be reinitialized by the workflow if the data changes.

Step 3: Code Your Workflow

With your data in place, write your workflow logic. It's pretty straightforward, but here are a couple pitfalls that you should watch out for:

Pitfall: Re-using non-serializable SharePoint objects after rehydration
Many non-workflow specific SharePoint objects, like SPListItem, are not serializable, so when the workflow dehydrates, these items become invalid. So if you try to use them when the workflow wakes up, you will get an error. To avoid this, refetch items if you're using them after a workflow rehydrates.

Pitfall: Forgetting to make custom classes serializable
When creating your own class in a workflow, don't forget to add the "Serializable" attribute to the class. If you don't do this and declare a variable of that class in the workflow's scope, the workflow will fail when it tries to go to sleep and serialize the class. If you notice that the workflow "completes" when it isn't supposed to, this may be the culprit.

Pitfall: Updating locked items
If a document is open or locked, updating the item will throw an exception. If you need a workflow to write to a column on the item, you can try to: a) Have the workflow poll for the lock and executing the update only when the lock is released, or b) use item.SystemUpdate() to "burn through" the lock. Note that for option (b), the user who has the document open will blow away the workflow's changes if he saves the document.

FAQ: How do I set a custom status for my workflow?
The SharePoint SetState activity does this. See WSS SDK for details.

Step 4: Deploy Your Workflow

Once you have your forms and code ready to go, it's time to compile and deploy your workflow. To prepare for deployment:

There are **two required files** for workflow features:

1) **Feature.xml** – just like every SharePoint feature, you need one of these for high level feature information.

2) **Workflow template xml file** (workflow.xml) – we've talked about workflow templates a lot, and this is where it comes from. Specify the aspx pages to use for your form types (for IP forms, specify our out-of-box host pages) and details about your dll, and put any extra information you need in the Metadata section.

You'll also need the **compiled assembly** and your **forms.**

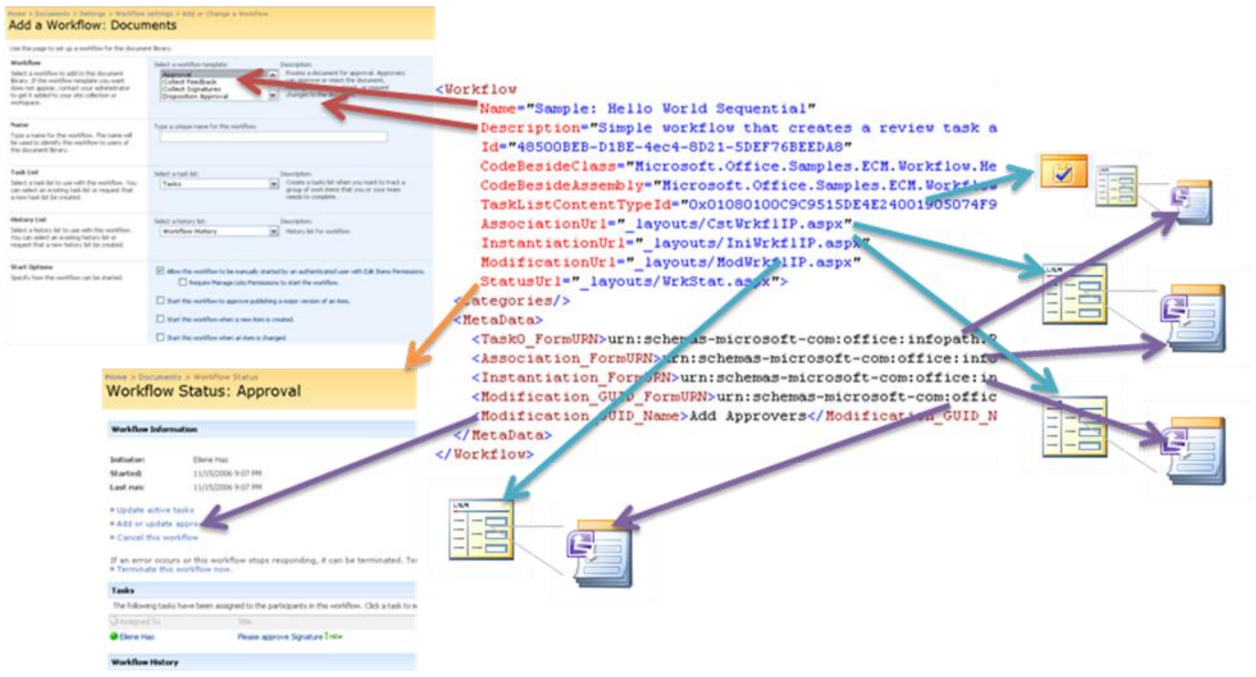
Once you have those, you can do the actual deployment. You will need to do the following things (if you're using the SharePoint workflow templates, this is what the PostBuildActions batch file does):

1) Copy the assembly to the **Global Assembly Cache** so that SharePoint can run it
2) Copy your xml files and forms to the SharePoint **FEATURES folder**, where SharePoint features live and are installed from. Custom aspx pages should go into the LAYOUTS folder.
3) Call **stsadm**, the magic SharePoint deployment tool, with parameters installfeature and activatefeature, to make the workflow available on a site collection.
4) Do an iisreset to refresh the assemblies and templates.

A couple more things to keep in mind:

- Deployment for workflow goes to an entire site collection at a time
- InfoPath forms will be **installed to the server automatically** during deployment if you have the <Property Key="RegisterForms" Value="*.xsn" /> tag in the feature.xml. ASPX forms need to be copied manually to the LAYOUTS directory.

The following diagram shows how the parts of the workflow.xml map to what you'd see in the SharePoint UI and workflow forms (= an aspx page and a hosted IP form;)):



```
<Workflow
    Name="Sample: Hello World Sequential"
    Description="Simple workflow that creates a review task a
    Id="48500BEB-D1BE-4ec4-8D21-5DEF76BEEDA8"
    CodeBesideClass="Microsoft.Office.Samples.ECM.Workflow.He
    CodeBesideAssembly="Microsoft.Office.Samples.ECM.Workflo
    TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F9
    AssociationUrl="_layouts/CstWrkflIP.aspx"
    InstantiationUrl="_layouts/IniWrkflIP.aspx"
    ModificationUrl="_layouts/ModWrkflIP.aspx"
    StatusUrl="_layouts/WrkStat.aspx">
  <Categories/>
  <MetaData>
    <Task0_FormURN>urn:schemas-microsoft-com:office:infopath:P
    <Association_FormURN>urn:schemas-microsoft-com:office:info
    <Instantiation_FormURN>urn:schemas-microsoft-com:office:in
    <Modification_GUID_FormURN>urn:schemas-microsoft-com:offic
    <Modification_GUID_Name>Add Approvers</Modification_GUID_N
  </MetaData>
</Workflow>
```

Step 5: Debug Your Workflow

With the feature ready to run on your server, you can now debug it live by adding break points, attaching your Visual Studio project to the IIS (w3wp.exe) processes and clicking through the workflow. Associate your workflow to a list and try running it. When you find a bug, recompile, redeploy, and reset. Repeat until satisfied☺.

FAQ: Debugger setup woes

- **Visual Studio crashes when I "attach to process"** – in the attachment dialog, only attach to Workflow or Managed Code.

- **My breakpoints aren't activating** – be sure the dll in the GAC is the exact same one in your project. If you change a file in your project, even without recompiling, Visual Studio will think the dll is out of date and won't link it to the project. Also, don't forget to iisreset to reload a new assembly.

FAQ: Tedious debugging

Debugging can be somewhat tedious because of the deployment steps, so here is some **advice to speed up the process**:

- The VS templates for SharePoint Workflow can automatically run the stsadm deployment commands for you when you build. Just prep your xml files, sign the assembly, and **set the DEPLOY parameter in the project Post Build Actions command line** in your project to turn it on.

- **If you've only changed the assembly**, just reinstall it to the GAC and call iisreset. (no need to reinstall the feature). The PostBuildActions.bat file in the SharePoint Workflow VS templates has a QUICK parameter that does exactly that.

- **If you've changed your forms or template**, you'll need to reinstall the feature. Don't forget to deactivate/uninstall before reactivating.

- The **<AssociateOnActivate> tag** in the metadata section of the workflow.xml will automatically associate a workflow with the Document content type. Setting this to "true" can save time setting up or reactivating an association. Just be sure you test on a doc lib that uses the Document content type.

- **Upload a bunch of documents** so that don't have to keep cancelling workflows that error out to run them again.

Other debugging tips:

- **Failed on Start** usually means an error is happening in SharePoint before your workflow code spins up, such as not being able to find the workflow dll. **Error Occurred** usually means the workflow started and the error is somewhere in the code.

- **Check the SharePoint ULS logs! –** fatal workflow errors might happen outside of your workflow, so they aren't always visible in the debugger. Consult the logs to see if this is the case by opening the latest log and searching from the bottom up for "Workflow". The logs are awesome resources that often get overlooked. Logs can be found in %Program Files%\Common Files\Microsoft Shared\web server extensions\12\LOGS.

Final Thoughts

You might feel a little intimidated; there's a lot to think about and understand. But don't worry! Like I said before, there are just a few key things to really get your head around, and once you can see how they all fit together, the world of workflow will unfold before your eyes☺ (or at least make a little more sense;)). The key takeaways are this:

1) Understand the difference between Template, Association, and Instance. The docs will make a lot more sense☺. Instances come from associations, which come from templates defined by a workflow.xml.

2) A little high-level planning of not just the workflow, but its surrounding lists and items, can go a long way.

3) A workflow can only respond to events that are registered through workflow services. You must use the SharePoint activities (i.e. the workflow service APIs) to create your items if you want to register your workflow for events on those items.

4) Data binding with forms is one of the most confusing parts of workflow development. Just remember:

   a. Your form pages call an OM function that wakes up a workflow via the workflow service. Each function calls the host, which takes your form data parameter and passes it into the corresponding workflow event receiver activity.

   b. Forms can be either aspx or InfoPath (hosted in an aspx page). But InfoPath provides benefits, such as client integration, easy authoring, and no code.

   c. If you use IP forms, use the names of your controls to refer to those values inside your workflow (i.e. the data uses the same schema as your form)

   d. Follow the checklist of steps for InfoPath forms to set them up for data submission.

   e. Tasks are different from the other workflow objects, in how form data is saved (directly to the task instead of into the workflow), how data is received inside the workflow (an SPWorkflowTaskProperties object instead of a string), and how it is defined in the workflow.xml (a content type rather than a page).

5) When you're ready to deploy a workflow, fill out a feature.xml and workflow.xml, then use stsadm to install it (the VS templates help automate this).

6) Debug your workflow by attaching to a live server. Use breakpoints and the ULS logs to figure out what's wrong.

Use this guide with the samples in the ECM Starter Kit, which are meant to build on concepts. Start with HelloWorld to understand InfoPath forms and correlation basics, then move on to Modifications or Replicator. Developing workflows just takes a little practice, so take it one step at a time. Anyway, thanks for reading, and good luck!

Appendix: Miscellaneous FAQ

What's the difference between Visual Studio and SharePoint Designer workflows?

	SPD	**VS**
Audience	Web Designer, Business Administrator, "Power users"	IT Professional, Developer, Box Administrator
Forms	ASPX	InfoPath, ASPX

Deployment	Direct deployment to a specific list, including association Can be done remotely	Deploy to all lists in a site collection, can pre-associate with Document content type. Must be done on the server box.
Learning Curve	Easy (Declarative wizard)	Advanced (Code required)

SharePoint Designer is an easy tool to create and deploy SharePoint workflows. You don't have to write any code, and you can implement a lot of workflows with it. But if you need to extend what's provided in the box, you might need VS. You will want to use VS if…

- You need to create a workflow that can be deployed to all lists in a site collection.
- You want to use InfoPath forms for your workflow
- You need more actions than the ones available by default in SPD
- You want to use state machine workflows

How can SharePoint workflow work with external systems?
You can use web services to interact with external systems or other sites. MOSS comes with web services to start workflows, edit tasks, and check status, so for example, if you want a system to trigger a workflow in SharePoint, you can create tasks for the external system and have that system call the web service to edit the task. See the Intersystem Purchase Order sample in the ECM Starter Kit for ideas.