SharePoint 2010 Workflows IN ACTION

Phil Wicklund

MEAP

MANNING

MEAP Edition
Manning Early Access Program

For more information on this and other Manning titles go to
www.manning.com

SharePoint 2010 Workflows in Action

Tentative Table of Contents

1

SharePoint Workflows for your Business Processes

Business processes surround us every day. The average employee is affected by them daily. Whether you like it or not, the company you work for is heavily dependent upon process to get stuff done and be profitable. Even someone who makes burgers at a fast food joint has to follow a specific process that will take that burger from nothing, into something that is delicious and nutritious.

Workflows are a system that manages the execution of a business process. They solve many of the most troubling problems that face a business process. The burger joint example is pretty simple, but there's no doubt that large companies have very complicated business processes and it can be quite difficult to know how where in the process they are, or who the process may be waiting on.

Also, consider how business processes often get bogged down because of poor communication. Does your business process live and die entirely in email? This is very common. Email has become the dumping ground for everything from conversations, decisions, tasks, documents, and so on. Consider a process that runs when a new person is hired into your company. That employee needs a new account, email, badge, phone number, benefits, direct deposit, contracts, etc. Getting all that accomplished in many cases involves many people, and they all go back and forth via email. Inevitably some things get lost. Email works for small companies, but what if you on-board 50 people per day? A system to manage all these activities is needed or confusion and inefficiencies will certainly rear their ugly heads. A workflow is needed.

This chapter takes you through more detail on what a workflow is and how it relates to you business processes. Moreover, we'll talk about how workflows function within the SharePoint platform, along with the architecture of a SharePoint workflow. After you know those basics we'll then take a good peek at all the tools and applications that go into building workflows in SharePoint, and you'll find there are quite a bit of options to choose from. Beyond this introductory chapter is a world where your business processes come to life. Essentially the rest of the book is about how to better build your company's workflows, but first let's make sure we're all speaking the same language.

1.1 What is a Workflow?

A workflow is primarily described as a process that manages the flow of work between individuals, offices, departments or entire companies. Everybody works when they go to the office, but some work depends on other people or systems before that work can get completed. As these recurring dependencies are identified in a company, a business process emerges. Business processes run all over a company. Most business processes are common place among even very different companies.

Take for instance a business process that manages expense reports. Most companies need a defined business process to manage the submission and approval of an employee's monthly expenses. The flow of work here (Figure 1.1) involves an employee tracking their expenses and then submitting those expenses electronically. The flow then rests on some sort of business logic that determines who needs to approve the expenses, and after it is approved, somehow that individual needs to get reimbursed. A workflow will help negotiate the execution of the steps in a process like this.

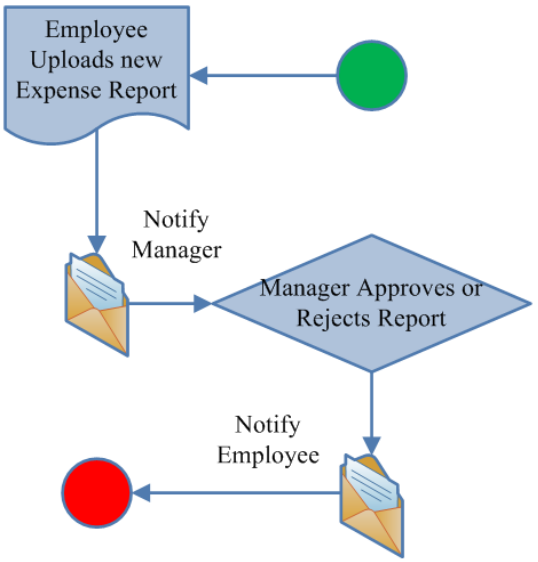


Figure 1.1 A workflow is most generically described as a business process. This example shows a common workflow that manages an expense reporting business process.

Business processes are running in a company regardless if there is a workflow that manages them. Some business processes are very ad hock, and easy for humans to manage. Others are much more complicated, and difficult for people to keep their heads around. It's with these more complicated business processes where workflows really show their value to a company.

Workflows can bring value to a company by providing insight into where in the business process the "flow of work" is currently executing. Workflows can also help a company automate their business processes. Consider the expense report example again. Maybe your business process allows for all food expenses fewer than one hundred dollars to be automatically approved and sent to accounts payable for reimbursement. A workflow could manage this business logic and automatically approve the expense, without needing to get your manager in the loop.

Workflows are also really good at managing parallel processes, or when multiple instances of work are all running at the same time. A manufacturing company would appreciate workflows because there are many processes that all run in parallel, and at the end all finally come together as the final product. A car manufacturer could have a workflow that keeps an eye on the engine construction, and another for the frame, and another for the interior, etc. Then a parent workflow could manage all of the "child" workflows and start another process as a dependant one finishes.

It is easy to see that there is a lot of value around investing in workflows to help manage and automate your company's business processes. Essentially doing something that can help minimize human dependencies as they pertain to business processes always saves a company money. Human

costs are always the most expensive investment a company makes, so let's make the people in our organizations work as efficiently and effectively as possible. That is the nature of why workflows are so important.

1.2 How Does SharePoint Help?

A SharePoint workflow is a piece of functionality you can attach to objects in SharePoint that have a business process surrounding them. An object in SharePoint is something like a document, or an item in a list like an announcement or task. For example, one of the workflows that you get when you install SharePoint is the Approval workflow. You can attach this workflow to a document in a document library, and specify individuals who need to approve the document for use before some other action can occur.

SharePoint Document Libraries

SharePoint, in addition to being a collaboration platform (teammates sharing information with each other), is also a document management system. A document library in SharePoint is the tool you can use to upload documents into SharePoint.

A common case for this in the business world is again the expense report system (Figure 1.1). Within SharePoint, users can upload their expense reports into a document library. The upload action will kick off the Approval workflow on the document, and a series of individuals will get an email stating they now need to approve the expense report. When all those individuals have approved the expense report, the document could be routed to the payroll team site where a payroll officer handles the actual processing of the expense report.

A SharePoint Workflow, like the document Approval workflow that was just described, could be setup to manage the business process from start to finish. The workflow will handle all user interaction within the system. It will also manage the point of execution in the workflow. Additionally, SharePoint will provide an out of box user interface that reports on the state of the workflow, or more specifically, who the workflow may be waiting on before it can continue, or if it has finished executing.

This “out of box experience” that SharePoint provides is a really compelling reason to manage your business processes within SharePoint itself. I say this because SharePoint provides a user interface and other workflow fundamentals like security, reporting, logging, and many more. These features make SharePoint workflows and your business processes a powerful combination.

Another great strength of SharePoint workflows is that individuals who are not technically savvy can configure their workflows directly through the browser window. Consider the expense report system again. If a company was to build this system from scratch it would cost them much more time and money because they would not have all the fundamental components that SharePoint provides out of box like what was just described. Rather, you can empower your end users to build these business processes, and at the most basic level, all they will need is a browser and possibly just a few minutes of their time. That’s what I call cost effective!

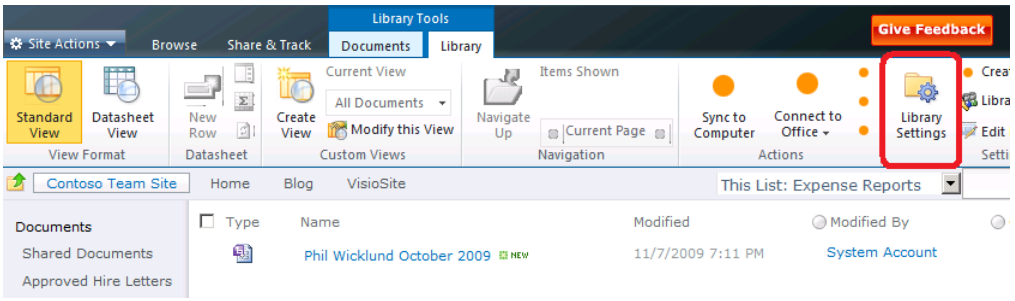


Figure 1.2 To manage workflows on a list or library, go to that list or library's Settings page

Notice how on a document library that contains some expense reports; you can manage the settings of that library (Figure 1.2). Once in settings, there's a Workflow Settings link under the Permissions and Management heading. This workflow settings (Figure 1.3) page is where you can add a workflow to a library. Simply select Add a Workflow and choose the one you want. It's that easy!

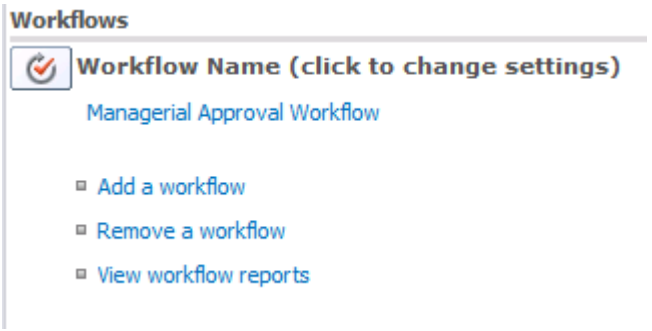


Figure 1.3 Once within the Settings Page of a list or library, you can then add a workflow onto that list or library

After adding the workflow to the library, and initiating the workflow process, a new column (Figure 1.4) will show on the document the workflow process is running on. This column will track the execution point on the workflow. The workflow may be halfway through the process and is waiting on some user interaction. Maybe a manager needs to approve the expense before it can continue. This status would show up in the new column as "Pending Manager Approval" or something else that accurately describes what the workflow is currently doing. Once the workflow has finished, "Completed" will show in this column.

Figure 1.4 After a workflow is started on a list item or document, an auto-generated column showing the workflow status is generated

1.3 SharePoint as a Technology Platform

So far we've been disusing what a workflow is from a business perspective, and how that workflow can execute within SharePoint. The truth is this only scratches the surface of the foundation that SharePoint workflows sits upon. SharePoint workflows actually leverage a separate platform called Windows Workflow Foundation (WF), which is a part of the .NET 3.5 application development framework. This foundation actually has many applications totally unrelated to SharePoint - in fact you can use WF to build all sorts of workflow enabled business applications that never touch or interact with SharePoint at all. However, SharePoint brings a very valuable benefit to an application developer, in that it provides a robust user interface, as well as implements the necessary persistence services required to run Workflow Foundation workflows. This means that SharePoint will manage the persistence of that workflow if it goes idle, as well as provides a user interface that end users can use to start and stop workflows and review the status of where the workflow is executing. For example, the expense report workflow needs to be persisted as it waits managerial approval.

SharePoint also has out of box workflows built on top of the programming layer (Figure 1.5), and with tools like SharePoint Designer you can customize those workflows if they don't meet your requirements. If you determine that a custom workflow is necessary, it is important to consider what type of workflow your custom workflow will need. There are two types of workflows that the WF architecture and SharePoint support. It can either be a sequential workflow or a state machine workflow. Each type serves a unique purpose in what kinds of business processes that it can support. However, before we dig into the differences between these two types, let's first take a look at the architecture of this foundation we're building upon…

1.3.1 Windows Workflow Foundation Architecture

The architecture of WF is built upon three main tiers of services, the hosting layer (1st tier), runtime layer (2nd tier), and programming layer (3rd tier). You could say that SharePoint workflows sit on top of the 3rd tier's services as its foundation. Things such as the out of box workflows, SharePoint Designer

workflows, Visual Studio workflows, and our expense report example all would build on top of this foundation. Figure 1.5 shows graphically how this looks.

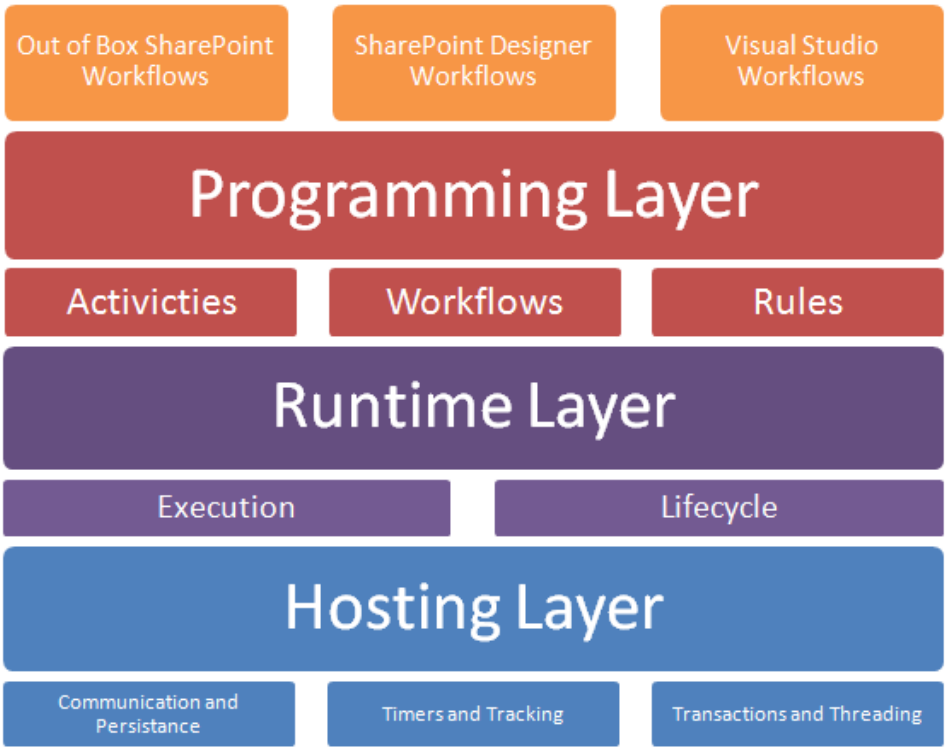


Figure 1.5 SharePoint workflows build upon the three layers of the Windows Workflow Foundation architecture. The programming layer is the interface between SharePoint and WF, as well reside on top of the core layers that manage the runtime and hosting.

HOSTING LAYER

Every workflow needs a host process to run in. A workflow in and of itself is not an application. Similarly, if you install the .NET framework on a server, you don't have a line of business application when you click finish on the install wizard. WF acts as a platform you build upon. However, there are a few things that WF requires the application to implement from a hosting perspective to "keep the lights on", you could say.

Part of what is required of the application is the host process, as was mentioned. Another aspect that the application must provide is persistence capabilities. Consider that a workflow is typically long running, meaning that it may start, and then go dormant for a while, possibly even many months. The "state" of the workflow needs to be persisted while the workflow is waiting for some action to occur, and when that action occurs, the workflow should resume where it left off.

WF has some "canned" persistence objects a developer can use. For instance, the SQLWorkflowPersistanceService class will help facilitate the serialization of the state of the workflow,

and store that state within a SQL database while the workflow goes dormant or idle. This is very similar to what SharePoint does, except with SharePoint the service is already setup and the developer doesn't need to concern himself with it. Notice in Figure 1.5 the Communication and Persistence bubble. This at a high level is what part of the hosting layer is responsible to do.

Another area of responsibility for the Hosting Layer is to provide timers and tracking capabilities. As was mentioned, a workflow may go dormant as it waits for an outside action to occur. In addition to an outside action, it may instead be simply time bound. For example, maybe the expense report workflow assigns a task to the manager to approve the report, but that task goes 7 days without being completed. At that point, the workflow "wakes up" and reassigns the task to someone in payroll. This also relates to tracking, in that you'll want to know, or track, where in the process the workflow is currently executing. In a nutshell, this is part of the responsibility of the timer and tracking aspect of the host process. Transactions are another important aspect of the hosting framework, in that you can leverage transactions to roll back a workflow to a previous state if an error occurs, for example.

RUNTIME LAYER

This layer represents all the core services that come with WF. For instance, the tracking, scheduling, and persistence services all at run time are performing WF critical activities that negotiate the workflow's execution and life cycle. This layer has interfaces that the hosting layer uses to connect the outside world to the WF engine.

PROGRAMMING LAYER

The programming layer is the Developer's favorite layer, and especially for SharePoint Developers, is all they'll typically need to worry themselves with. This layer has out of box activities that can perform various actions in the workflow, as well as allows for custom activities and rules that workflows interact with.

1.3.2 Types of Workflows

The way in which a workflow will execute from one step in the process to another, falls in one of two categories. A workflow is either sequential, in that the steps within the workflow execute sequentially, one after another, or a workflow is a state machine where it executes in no particular order. A sequential workflow always progresses forward, and never goes back to a previous step (Figure 1.6). A State Machine on the other hand has no such constraint, but rather moves from one state to another, until some logic concludes the workflow has completed. A good example of this is a bug tracking workflow that tracks bugs in a computer program (Figure 1.7). When the workflow first starts the bug may be placed into a "Pending" state, where it waits for a developer to be assigned to the bug and start working on the bug. Thereafter, the developer starts working on the bug and fixes it, putting the bug into a "Fixed" state. When a bug is fixed, a tester goes to confirm the resolution of the bug, in which time he finds that it was not fixed and places the bug back in a "Pending" state. This ability to go back in time, or to a previous state, is only available with State Machine workflows.

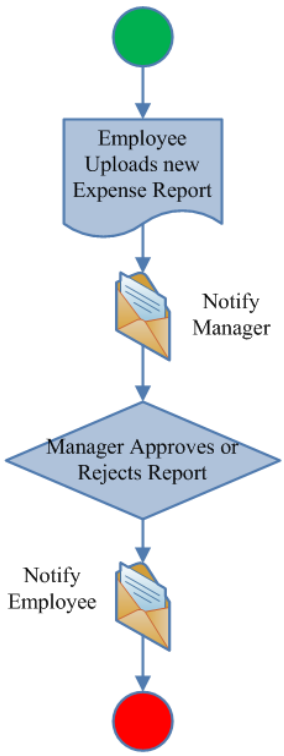


Figure 1.6 Sample "Sequential" Workflow that always advances forwards in the process, never backwards.

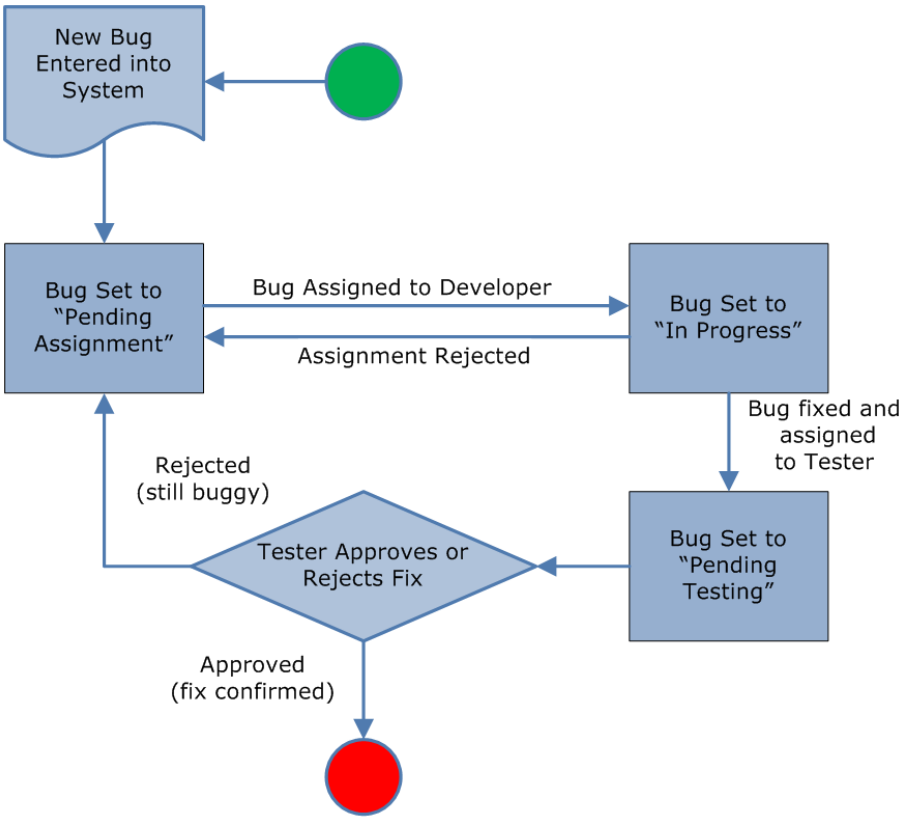


Figure 1.7 This sample workflow needs a Sate Machine because it in many different instances may need to go back a step, or repeat the whole process entirely.

Obviously some business processes will require a state machine and others won't. It is important to think through your business process's requirements before you start building a custom workflow, because it is difficult to change course after you have already started down a path. As you progress through this book you'll notice that some workflow tools can only do sequential workflows (as is the case with SharePoint Designer), whereas other tools can do both types (such as Visual Studio). If you start building a workflow with a tool and that tool doesn't support state machines you may find yourself starting over if later down the road you realize you need to change course. So the bottom line is, think carefully through which type you need before you start.

1.4 Workflow Enabled SharePoint Objects

There are many different types of objects within SharePoint. Objects come in many forms such as lists, list items, libraries, documents, forms, content types, site columns, views, web parts, sites, site

collections, and so on and so forth. Going back to our expense report example, the workflow runs on a document that was uploaded in the document library. In addition to documents, there are several other types of SharePoint objects that workflows can execute on.

LIST ITEMS

Just like documents, generic SharePoint list items can have a workflow running on them. For instance, you could setup an approval process on an Announcements list. With this setup, announcements won't show to end users until they're approved. Another great use of list items is on "task lists" and "issues list" (both are out of box SharePoint list types). When a task or issue is assigned to someone, and that individual resolves the task or issue, a workflow might forward the task to another individual who is responsible to verify its completion before it can be finalized or completed. If that individual finds an error, the workflow could reassign the list item back to the original user. Figure 1.8 shows the workflows menu item on a list item. Through this menu item you can start a new workflow instance on the item.

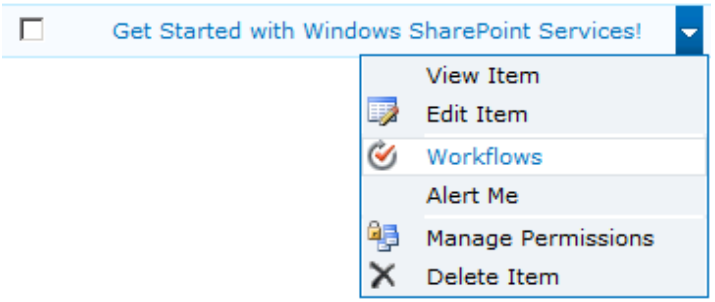


Figure 1.8 To start a workflow on a list item or document, click the drop down on that item, and then select the Workflows menu item to take you to a page where you can initiate a new workflow instance.

INFOPATH FORMS

Technically an InfoPath form (Figure 1.9) is a document, and would fall into the document library category of SharePoint objects that has already been discussed. However, there is also great value in calling them out separately. It is commonplace to attach a workflow to a form. Take for instance the expense report system again. Oftentimes, a company will want a standard template that their employees must use when filling out an expense report. This template usually has unique needs, or must look a certain way that is unique for that company. The Microsoft Office InfoPath 2010 client application is a great tool to build forms with that your users can fill out.

The strength of InfoPath is its ability to make form creation easy for even novice users. If you're familiar with Microsoft Word, you stand a good chance to catch on to InfoPath quite well. With InfoPath you have tons of flexibility to make a form that looks and feels a certain way that meet your needs. You have far more user interface flexibility with InfoPath, than say if you used an Excel Document. Also, just like Microsoft Office documents, workflows can be bound to an InfoPath form, and when the user fills out all the appropriate areas within the form the workflow can manage the business process behind it, and get that form approved or denied. There are other form options that are discussed in section 1.6.4 of this chapter, and in greater detail in Chapter 7 and 9.

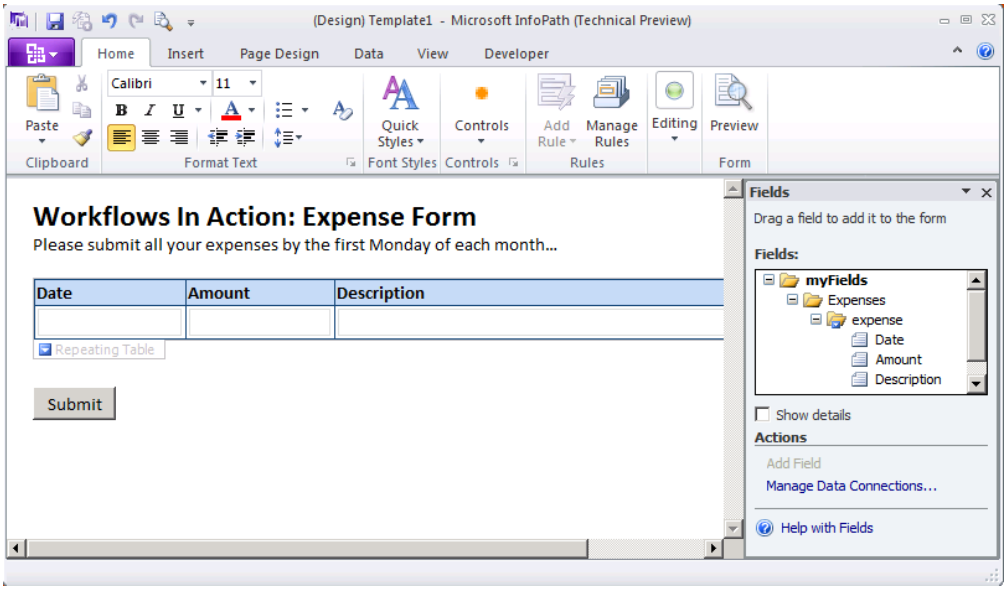


Figure 1.9 InfoPath Forms are a great avenue to take to develop custom forms for your workflows. This example shows the InfoPath Office client in Design mode

CONTENT TYPES

Content Types within SharePoint are an important concept, but are highly documented and won't be covered in great detail within this book. At a basic level, content types are a way to package up pieces of meta-data, and make those meta-data collections reusable. For instance, let's say you wanted three columns on every list or library in your site collection. Rather than go to each list, and add each column manually, you can create a content type that has those three columns within it, and then just add the content type to the lists or libraries. This can be a big time saver, and also allows for a lot of reuse benefits when changes to the content type need to be made later.

As far as workflows go, you can also add a workflow into a content type. Just like how adding columns into a content type can save you time, putting the workflow in that same content type will save time as well. Additionally, a content type can have one or many workflows assigned to it. So if you have a complex business process with many types of workflows that all need to execute simultaneously, deploying that workflow to a Content Type may be a good idea. When a workflow is deployed onto a content type, new instances of that workflows can be initiated wherever list items of that content type exist, no matter what SharePoint list they reside in. So in some sense, this introduces some reusability into your workflow across more than just one list.

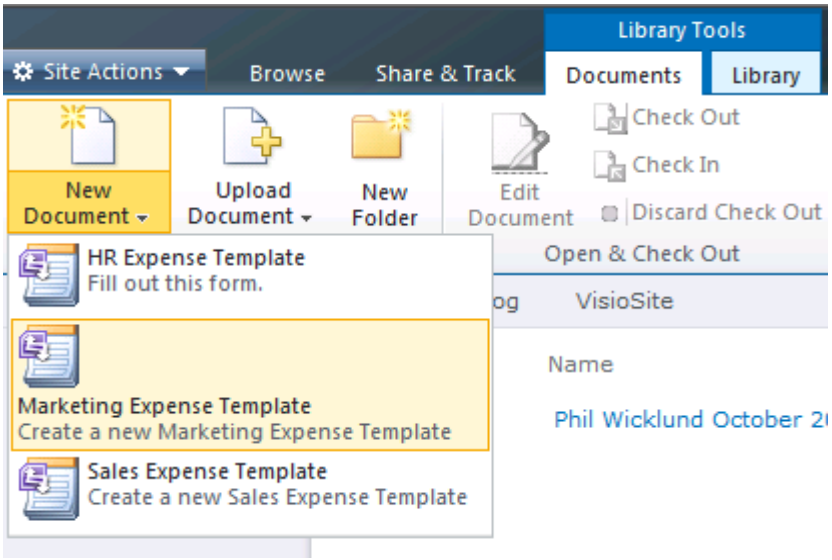


Figure 1.10 Workflows oftentimes run on a content type. As you add content types to a list or library, more options appear in the New drop down to select which content type to use

Notice the new expense report drop down (Figure 1.10). This demonstrates how you can use content types to allow for several different expense report templates to be available, possibly one for each department in the company. The form and/or workflow for the HR department's expense reports may be different than the workflow for the Sales department, in which case using two different content types would allow the user to choose which form and workflow is right for them.

SHAREPOINT SITES

One final area that you can bind a workflow to is on a SharePoint site itself. Whereas all the other SharePoint object examples boiled down to a list item of some sort (whether it be a document, form, or content type), a workflow deployed onto a site can run actions on and react to events across all the lists, document libraries, and items in that site. For example, take a site that has many document libraries and many documents in each. A workflow that would be well suited to running at the site level could check each document within that site and ensure that all the documents have been routed for approval and that none have been declined. Workflows can execute across an entire SharePoint site, and are initiated from the within View Site Content.

1.5 Out of the Box SharePoint Workflows

Now that we have an idea of all the types of objects in SharePoint that workflows execute on, we should explore further what workflows come out of box in SharePoint. In section 1.2 of this chapter we introduced the Approval workflow. This is just one of several workflows that are available in SharePoint. SharePoint provides 6 workflows out of box that even an end user can configure on their sites. What is great about these workflows is you don't need to be a low level programmer to introduce valuable workflows into your organization. These workflows are available to be added to various SharePoint

objects right through the user interface, and require little if any configuration. Figure 1.11 shows the Start a new workflow page on an Excel document in SharePoint. It is very easy to start a workflow which kicks off a wizard like experience for the end users to configure that workflow. Let's take a look at all the out of box workflows at a high level. Reference the last section in chapter 2 for a more detailed overview and a flow diagram of each.

THREE STATE WORKFLOW

The Three State Workflow by default is leveraged on issue tracking lists within SharePoint. The issue tracking list tracks an issue through three states, Active, Ready for Review, and Complete. This workflow can be customized and be used in other or custom SharePoint lists, and the names of the three states are configurable.

SharePoint Foundation versus SharePoint Server

It is important to note at this time that all the out of box workflows from here on out require SharePoint Server. SharePoint Foundation is what everybody has when they have "SharePoint" installed. SharePoint Server on the other hand is an add-on, on top of SharePoint Foundation. In the case of workflows you get a few extra out of box workflows when you purchase Server. Foundation only comes with one workflow, the Three State Workflow. With Server you get an addition 5 out of box workflows. However, with Foundation you still have most of the functionality available to you to build custom workflows.

APPROVAL WORKFLOW

While being one of the simplest workflows, the Approval workflow is certainly the most popular. It involves routing a piece of SharePoint content to all designated approvers to submit their approval or denial on that content. The submission process can either be serial, where the order of approvers is predetermined, or parallel, where any approval can approve at any time.

COLLECT FEEDBACK WORKFLOW

The Collect Feedback workflow gives submitter the ability to acquire feedback for his peers on the status of the document being submitted. This workflow routes the document across the specified team members, in which case each can way in and contribute their feedback on the document. Once it has gone around the team, the feedback is compiled and the submitter is notified.

COLLECT SIGNATURES WORKFLOWS

An alternative to the approval workflow where the approval process makes no changes to the actual document, the Collect Signatures Workflow will require a digital signature to be placed upon the document from each approver. After those signatures have been acquired via the workflow, the document can be taken offline and the "approval" will still be recognizable.

DISPOSITION APPROVAL WORKFLOW

This workflow allows you to manage document expirations and retention by integrating with the Records Center features that come with SharePoint Server. This gives you the ability to make decisions around what happens to documents when they expire where a possible option is they shouldn't be deleted, but rather are archived and a few email notification are sent out.

TRANSLATION MANAGEMENT WORKFLOW

Use this workflow to help facilitate the manual process of translating office documents from one language to another. This workflow works around two list types, a Translation Management Library and a Translators list. A document that needs to be translated is uploaded into the translation management library, and translators in the translators list receive translation tasks to start the work of translating the source document into their respective languages. When all the translation tasks are completed on that document, the translation management workflow is completed.

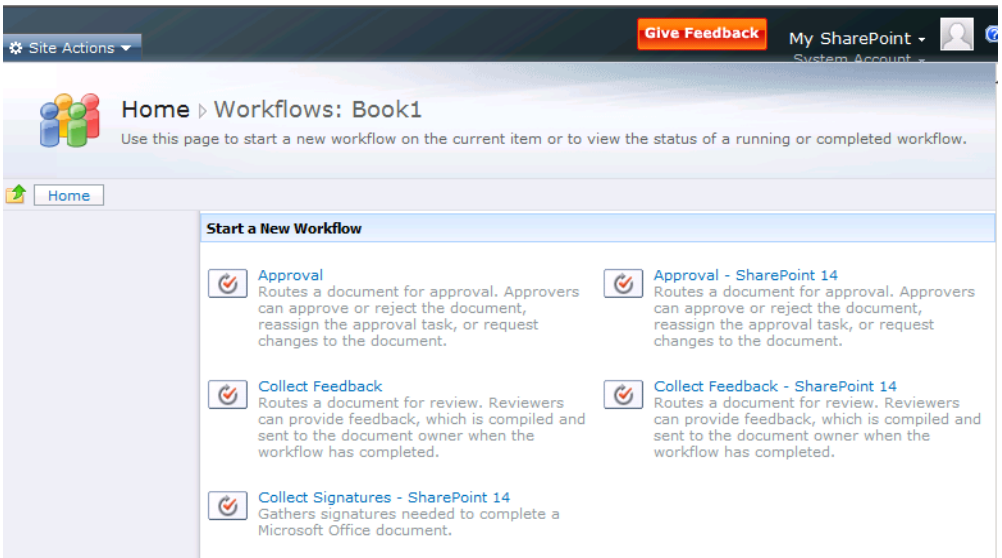


Figure 1.11 There are many Out-of-Box workflows that come in SharePoint. This is a list of out of box workflows that can be started on an Excel document

1.6 Tools used in the Building of Custom SharePoint Workflows

Even under the SharePoint umbrella, there are still many layers of tools that you will want to use when building your SharePoint workflows. Some of them are entirely optional, and others are not. Also, some tools are for different audiences. An end user may simply love digging into SharePoint Designer, but be entirely daunted by Visual Studio, yet Visual Studio may be necessary for what he's trying to do. It is critical then to have a high level understanding of all the tools available so you can know why they are helpful, and for what purpose each serves.

1.6.1 SharePoint Designer

The SharePoint Designer is a powerful tool that can be used to customize SharePoint sites. Many of its unique capabilities are used to change the look and feel (brand) of site, create/add/move web parts, and bring list data and external data onto SharePoint pages. There's a lot you can do in SharePoint Designer, and building workflows is one of them.

Building workflows with SharePoint Designer is one of the most popular and widely used approaches for SharePoint workflows. The tool is easy to use because it provides the user with a wizard like

experience (Figure 1.12) that they would be more familiar with, than say Visual Studio's code editor. However, SharePoint Designer isn't the tool for the average end user. Microsoft would categorize the tool into the Power Users group, people who are not programmers, but who are savvy enough to be proficient with other Microsoft Office tools like Excel and Access. This is unlike simply using the browser, where things are much more intuitive and less complex. Fortunately, this book will cover SharePoint Designer workflows in much greater depth in chapters 3, 4, and 5.

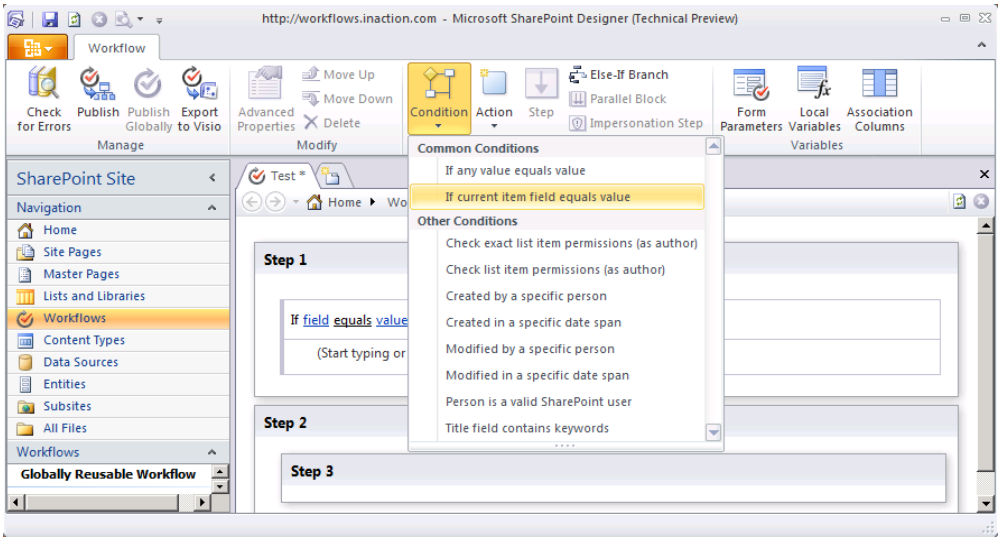


Figure 1.12 The SharePoint Designer workflow engine provides a rich suite of customizations to workflows allowing you to easily meet unique and sometimes complex workflow requirements

1.6.2 Visual Studio 2010

Despite SharePoint Designer being a highly usable and robust workflow tool, inevitably you will come to a point where your business requirements will ask for something that is more than SharePoint Designer can deliver on. This is where building custom workflows within Visual Studio comes into play. Visual Studio adds a lot of flexibility in the types of activities your workflow can perform. This is because Visual Studio provides a full fidelity development experience, whereas SharePoint Designer is wizard based.

By using Visual Studio you'll have a designer interface where you can drop activities. Also, each workflow and activity will have its own code behind that you can call into or extend. Workflows built in Visual Studio leverage the .NET 3.5 framework, and specifically the Windows Workflow Foundation platform. Windows Workflow Foundation workflows can be packaged up and deployed into SharePoint. This will allow you to meet even the most complicated business requirements imaginable.

Figure 1.13 shows what a workflow looks like within Visual Studio. It shows how the various activities are laid out on the workflow designer surface. Just by looking at the activity names and how they're associated with one another, you can see what the workflow is doing.

If you are interested in downloading the workflow (Site Creation and Approval Workflow), it is freeware available from my public website at:
http://philwicklund.com/freeware/siteapprovalworkflow

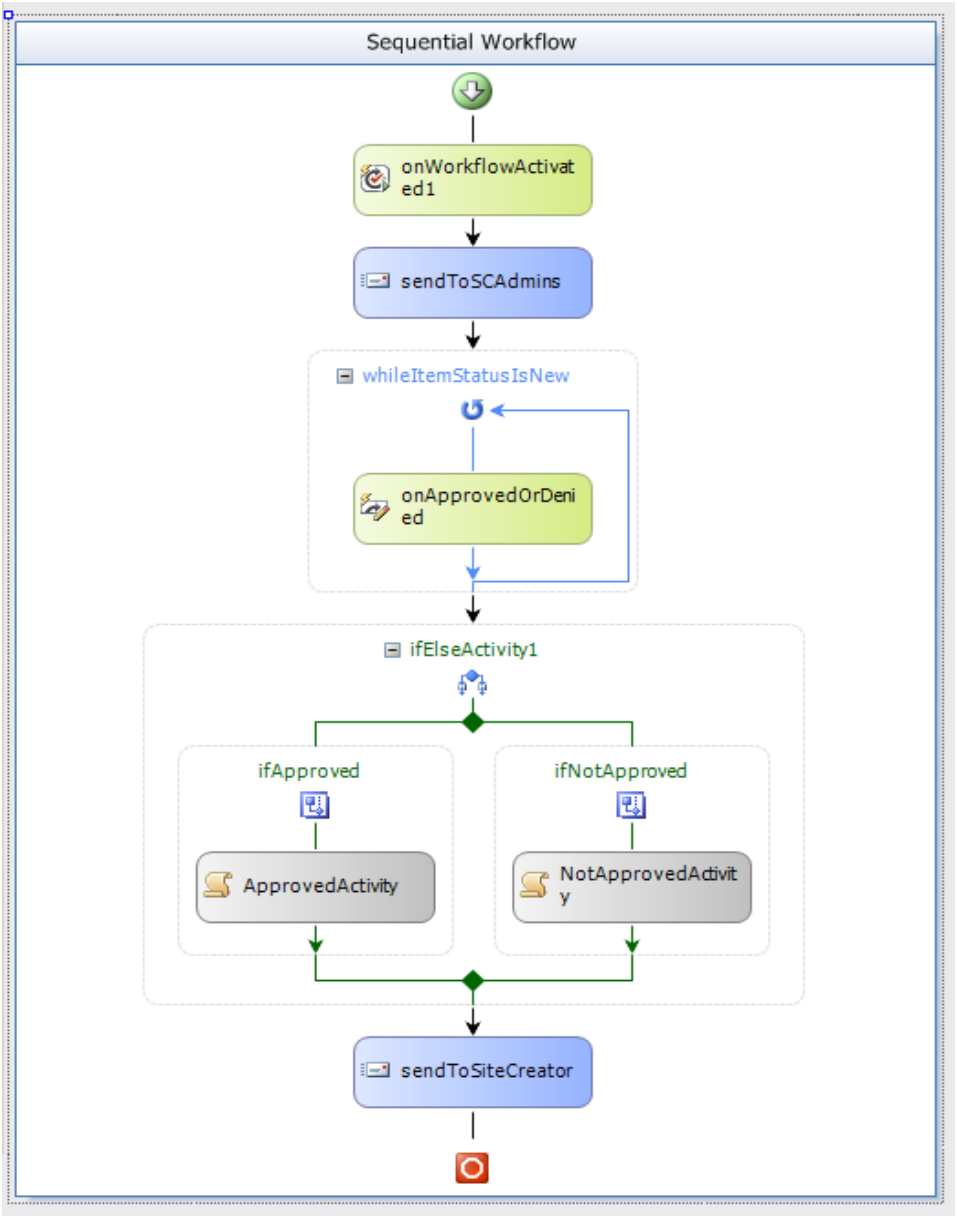


Figure 1.13 Visual Studio is the uttermost in workflow customization flexibility. This is a view of the workflow designer surface within Visual Studio

A detailed walkthrough of how to build a workflow in Visual Studio is given in Chapter 8.1.6.2 Office Visio 2010

Microsoft Office Visio 2010 comes with a great new feature for SharePoint workflow implementers. Visio 2010 has a new template called Microsoft SharePoint Workflow (Error! Reference source not found.) which allows Business Analysts to model custom workflows that need to be built out by power users (in SharePoint Designer) or developers (in Visual Studio). In earlier versions of Visio you could still do this by using a generic flowchart template by simply adding shapes and connectors onto the Visio designer surface to model out the basic flow of events. However, now with this new SharePoint Workflow template, your power end users or developers can actually take the work you have done in Visio and import it into SharePoint Designer, and subsequently Visual Studio. This re-use of work has some obvious time saving benefits, as well as benefits around requirements integrity because of the connectedness between tools. Details on how to implement a Visio 2010 SharePoint workflow will be discussed in chapter 6.

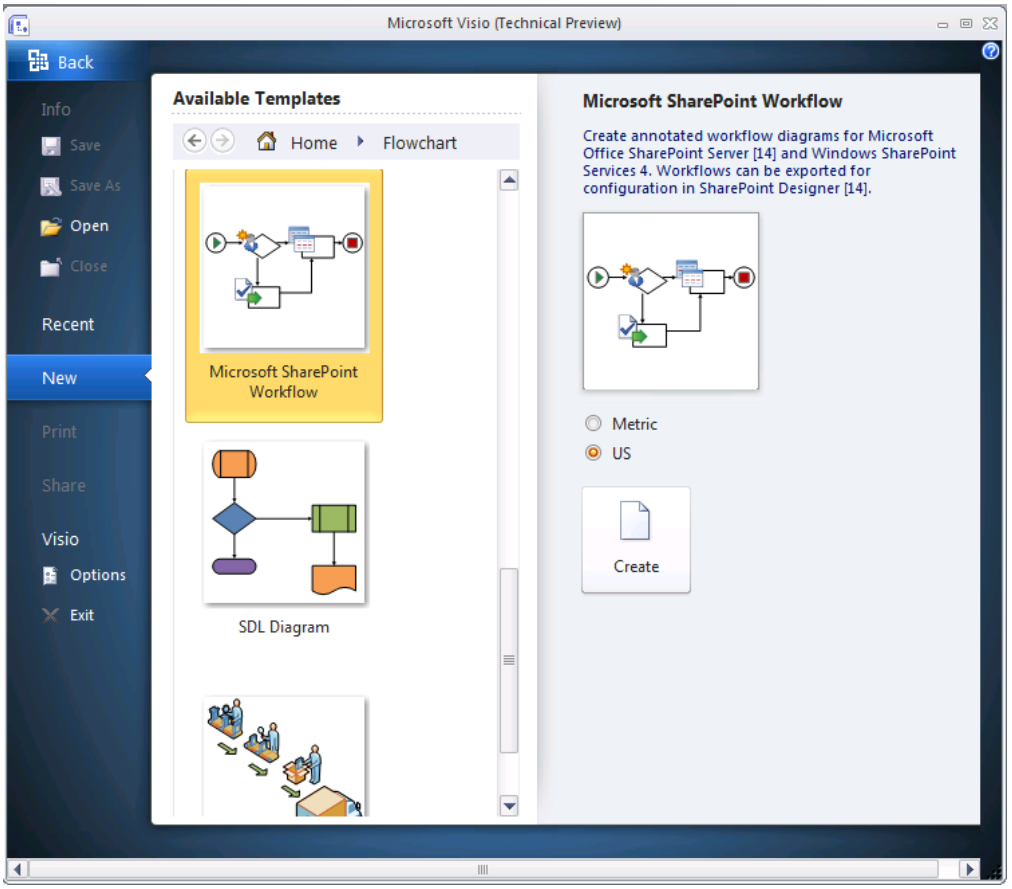


Figure 1.14 Visio 2010 comes with a SharePoint workflow template that can be used to model out your business process before you build

1.6.4 Forms

Forms and SharePoint workflows make a powerful combination. Much of the data in a business process is captured within a form of some sort, and thereafter the business process reacts to the values specified in the form. For an expense reporting workflow, an employee typically fills out a form and enters their expenses. Submitting the form will kick off a business process to retrieve approval for that expense. There are countless other form/workflow combinations that are scattered throughout the average business. Some common form examples are forms for gathering personal information (health insurance benefits), purchase order requests, invoices, PTO requests, SharePoint site requests, helpdesk tickets, and so on. It is because of this strong relationship between a business process and a form that makes SharePoint workflows and forms a powerful combination. The retention and digital availability of form data, paired with the automation of these business critical processes brings a ton of return on investment into an organization. Paper is the past, and workflows are the future!

In SharePoint there are three main tools used in the building of forms that workflows can execute on. You can use the out of box forms that SharePoint allows you to build by simply adding metadata onto lists and libraries. For more complicated forms there is Office InfoPath 2010 and ASP.NET. InfoPath is a great non-programmer form designing tool that end users can use, whereas ASP.NET is geared to developers when the most demanding requirements need to be met. Figure 1.15 shows a sample out of box form in SharePoint that can be used to gather information from a user and kick off a workflow.

Chapter 7 offers detailed instructions on form fundamentals such as building custom InfoPath forms, and using forms in SharePoint Designer workflows. For Visual Studio workflow form, check out chapter 9.

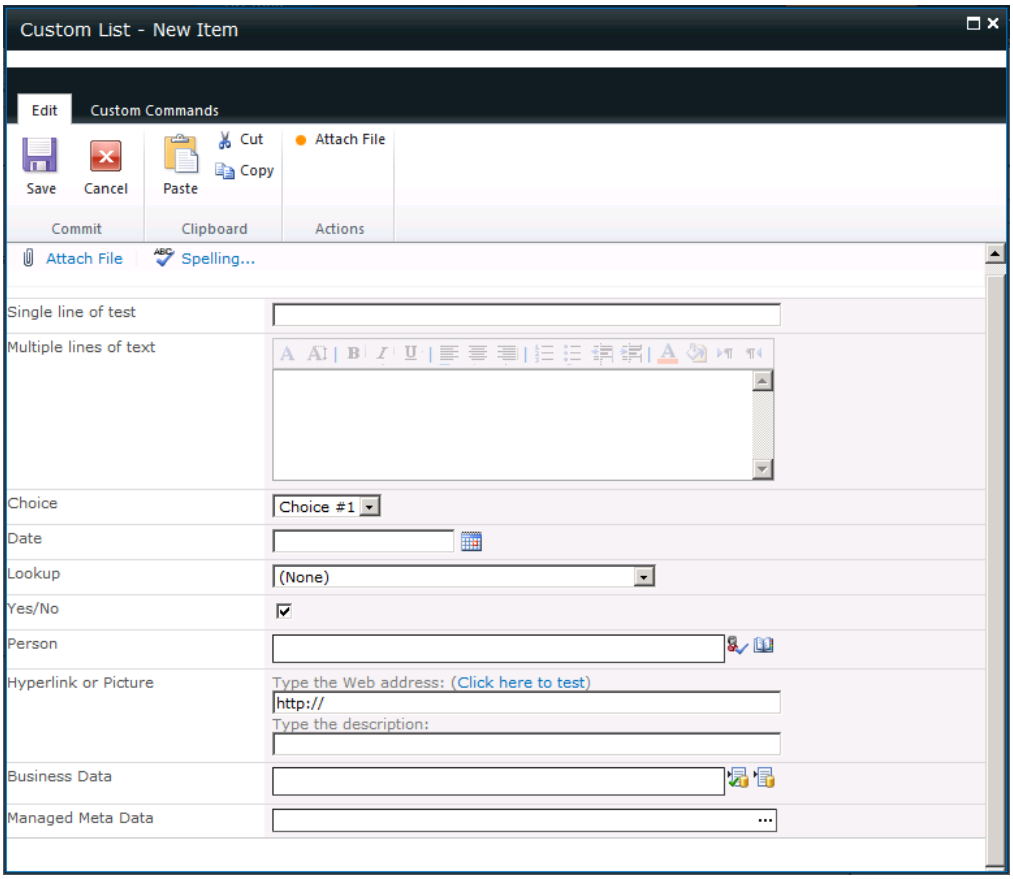


Figure 1.15 Forms and workflows are powerful combinations. A workflow oftentimes reacts to a user submitting some information.

1.6.5 Object Models

There are times when you may need to programmatically interact with workflows via custom code. An example is when there's a custom report that shows the statuses of various business critical workflows on a manager dashboard. Another possibility is there's a need to start a new workflow on a weekly basis, but you want the initiation of that workflow to be automatic and not dependent upon human interaction. Both these cases lend themselves to some sort of code deployment that programmatically interacts with the workflows via the respective object models. System.Workflow, Microsoft.SharePoint, and Microsoft.SharePoint.Workflow are three main object models that a developer will leverage when working with SharePoint workflows. A few more examples of what one can do with these object models are listed below. Object model techniques will also be covered in detail in chapter 12 of this book.

- Start or stop a workflow on a SharePoint object
- Get a list of the running workflows on an SharePoint object

- Detect and delete orphaned workflows
- Report off the state of a workflow

1.7 What's new for Workflows with SharePoint 2010

With the release of SharePoint 2010, there is a host of new functionalities available around workflow. This is especially true around custom workflows, where a few of the new features radically make developing workflows much easier. Take for example the introduction of the "Reusable Workflow". With 2010, workflows can now be deployed in a reusable fashion from within SharePoint Designer, which enables users of Designer to be much more efficient in their work. In the 2007 product, they used to have to literally recreate each workflow onto every list instance that they wanted it to be deployed onto. Now with 2010 a workflow only needs to be created and maintained in one place, and it can be installed onto many lists and receive updates if the original workflow is edited. In addition to reusable workflows, there are many more great enhancements that have been made to workflows in the 2010 release. Some of the key improvements are as follows:

OFFICE VISIO 2010 SHAREPOINT WORKFLOWS

Bar none, the new functionality in Office Visio 2010 will bring the most smiles for SharePoint Business Analysts. With Visio 2010, you can now model out your SharePoint workflows, and leverage that model to help elicit business approval. The best part is after you've solidified that high level flow, you can export the workflow as a template and import it into SharePoint Designer and start building out all the steps! This will greatly improve the efficiency of requirements gathering and translation to developers. See chapter 6 for more information.

CUSTOMIZING THE OUT OF BOX WORKFLOWS

Have you ever used an out of box workflow in SharePoint 2007, but it didn't quite do just what you needed it to do? If this statement is true for you, you'll be pleased to hear that these out of box workflows can be customized in SharePoint Designer 2010 (Figure 1.16). See chapter 5 for more information.

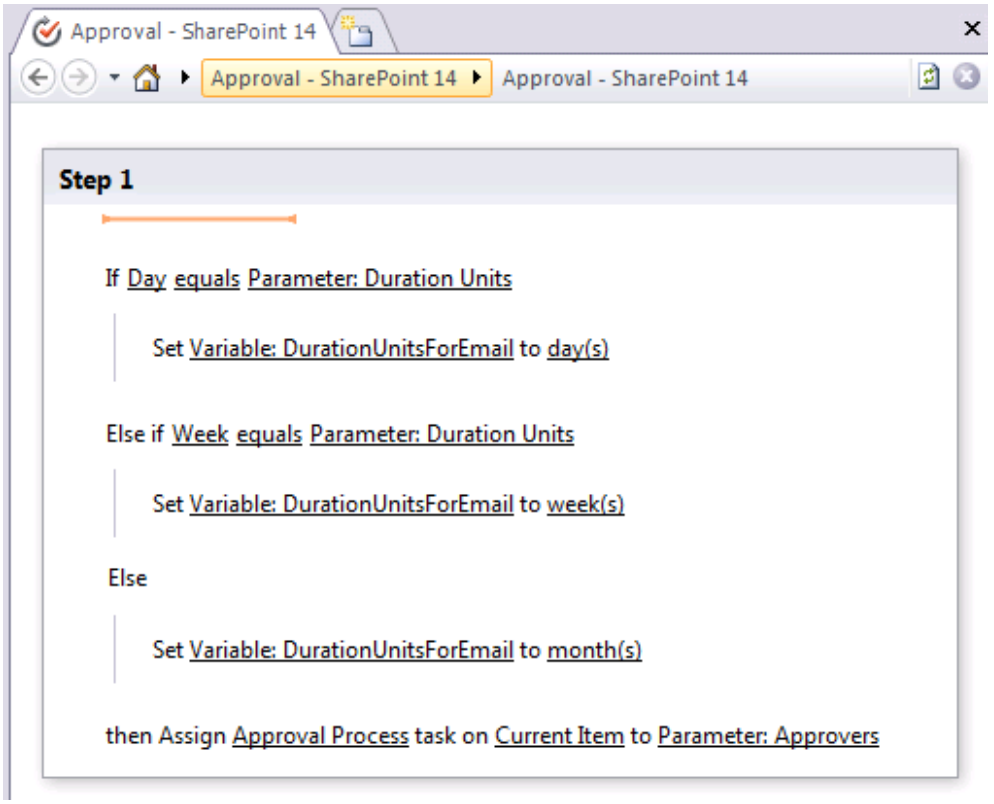


Figure 1.16 Editing the Out-of-Box Approval workflow in SharePoint Designer 2010

NEW ACTIONS AND CONDITIONS AVAILABLE IN SHAREPOINT DESIGNER

There's a host of new out of box conditions and actions available in SharePoint Designer with 2010. A couple of the highlights are you can now manage permissions in a workflow (just think of the possibilities here!), as well as interface with Records center to help accommodate retention policies. See chapter 3 for more information.

REUSABLE WORKFLOWS

As was mentioned in the introduction to this section, reusable workflows can now be created and deployed to the site or site collection level and be consumed by various objects within that scope. The major benefit here is that you no longer need to maintain more than one copy of the same workflow. The example in chapter 3 covers how to build a reusable workflow in SharePoint Designer.

SITE WORKFLOWS

Site Workflows takes the concept of reusable workflows a step further. In addition to a workflow being able to be run on a list, library, or list item within a site, a Site Workflow can run on the site itself. A

good business example of this is a workflow that needs to ensure that all the documents in the entire site, regardless of what list it resides in, have been approved. A site workflow could iterate through all the libraries and check each document for whether it has been approved or not. There is a host of other applications for deploying a workflow as this scope as well. See chapter 3 for more information.

TASK PROCESSING CUSTOMIZATION

In most workflows there will exist the delegation of tasks to certain individuals. When a task is assigned, the workflow will typically wait for an action to occur upon that task, in which case the workflow will resume processing. In SharePoint 2007, the nature of this task processing was pretty static, meaning you could not alter the way the out of box workflows handled tasks and events associated with them. However, in SharePoint 2010 you can now fully customize what actions you want to take place when various task events occur. A few of these events can react to when a task is assigned, expires, deleted, and completed. Now when each of these events occurs you can inject your own custom activities to change the way the task processing flows. Task customization for SharePoint Designer workflows is discussed in detail in chapter 4. For tasks in Visual Studio workflows, reference chapter 10.

WORKFLOW TEMPLATES IN SHAREPOINT DESIGNER

In SharePoint 2007 you couldn't move a SharePoint Designer workflow from one farm to another. So if you have a Development, Test, and Product series of farms, you couldn't prototype a workflow, in say Development, and then promote it to Production. Now in 2010 you have the ability to save a workflow as a template (WSP file), and export and import that template into another farm!

VIEWING WORKFLOW STATUSES WITH VISIO WEB ACCESS

A neat new reporting feature that is available in SharePoint Server Enterprise edition is the ability to view a workflow's status via a Visio diagram. If you first build your workflow in Office Visio 2010, and then import that workflow into SharePoint Designer, you can enable Visio web access on that workflow and throughout the workflow's lifecycle the Visio diagram will dynamically update to reflect where the workflow is currently executing. Notice the green check boxes in Figure 1.17. You can see the path that the workflow has taken and where it is executing. In this case it has finished executing. For more information on how to set this up, reference chapter 6.

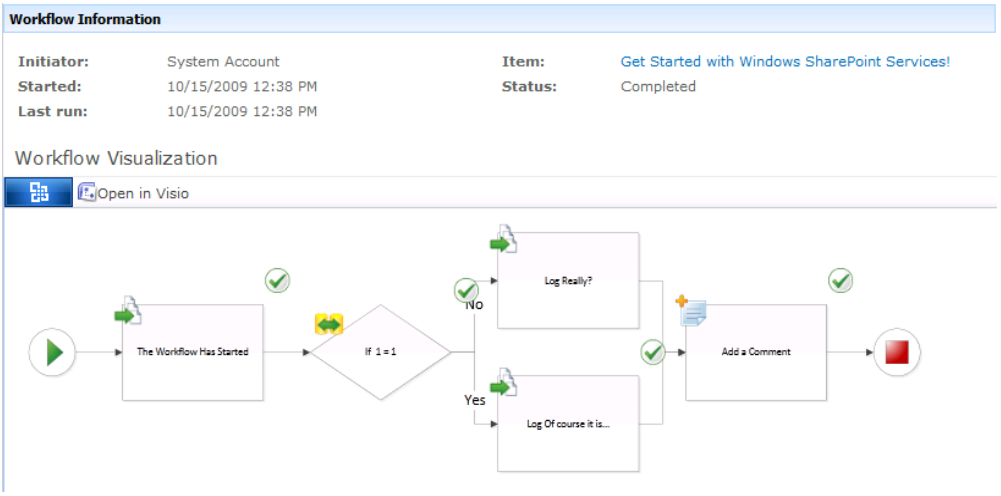


Figure 1.17 Workflow statues can now be consumed through a dynamic Visio 2010 diagram

IMPORTING SHAREPOINT DESIGNER WORKFLOWS INTO VISUAL STUDIO

A stunning new feature that comes with 2010 is the ability to export a SharePoint Designer workflow and thereafter import that workflow into Visual Studio (Figure 1.18). This is very powerful, because oftentimes you may first create a workflow in SharePoint Designer since it is such an easy tool to use. However, after a year or so goes by, you may realize that your business requirements have changed and become more complicated, necessitating a Visual Studio workflow. In SharePoint 2007, you would have had to recreate the SharePoint Designer workflow from scratch within Visual Studio, but now because of the new export/import functionality, you won't lose those valuable man hours that it took to build the Designer workflow in the first place.

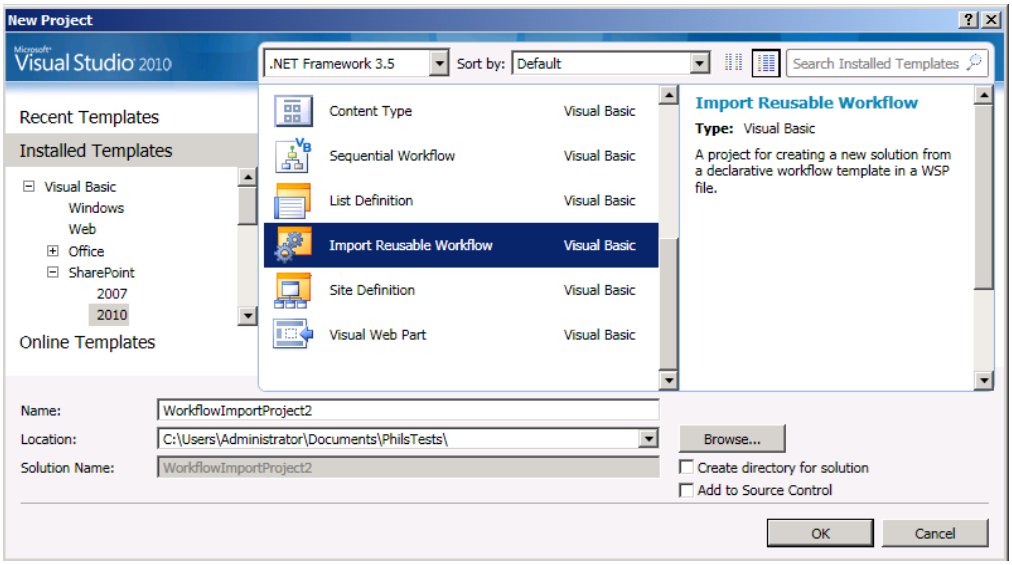


Figure 1.18 Visual Studio 2010 comes with a new ability to import a workflow created in SharePoint Designer

VISUAL STUDIO 2010 ENVIRONMENT IMPROVEMENTS

There's no doubt that building custom workflows within Visual Studio has gotten dramatically easier with the 2010 releases of SharePoint and Visual Studio. Many would consider that most of the effort to build Visual Studio 2008 workflows in SharePoint 2007 was around the packaging and deployment of the workflow itself. You can to build all the features, DDFs, manifests, keys, tokens, GUIDs, and everything else by hand! This is no more with Visual Studio 2010 - all the necessary features and solution packages necessary to deploy the workflow into SharePoint are generated for you manually. Just right click, and deploy! For the steps in greater detail, see chapter 8.

PLUGGABLE WORKFLOWS

In SharePoint 2007 there was no easy way for running workflows to receive updates from the outside world. With the new "pluggable" capabilities that come with SharePoint 2010, workflows can now execute up to a certain point, and then wait for information from an external process. The developer simply needs to implement an event handler or web service of some sort to handle the request of the external process, and thereafter respond by calling into a method within the workflow itself, informing it to continue processing. For more on this topic, reference chapter 12.

NEW EVENT HANDLERS

A couple new event handlers are present within SharePoint 2010, and that's the ability to react to a workflow being initialized or completed. These handlers may be external to the workflow, or embedded within the workflow itself. For instance, you may want to fire some code that logs in a centralized repository every time a workflow of a certain type has been started, and when it is completed. With these new event handlers, this is easily possible. Event handlers are covered in chapter 12.

1.8 Architecting your custom Workflow Solution

Before you jump into the following chapters and start building workflows, it's important to consider further how a workflow is born and progresses to completion. We already discussed the many tools you can use to build custom workflows, but a careful comparison of these tools and a discussion of how to diagram a workflow should be enlightening. In this next section, you'll diagram, design, scope, and choose authoring tools for a generic business process.

1.8.1 Diagramming your Business Process

When gathering requirements for any software application, you usually work from the top down. First, you determine the high-level requirements, and thereafter you get more and more detailed (low level) as necessary. As far as SharePoint workflows go, this is no exception. You first want to model the high level business process. In effect, you want to determine what you need to build. After you've gotten agreement from business stakeholders around *what* you're going to build, you can then consider *how* you're going to build it.

You may feel compelled to start with this SharePoint Workflow template right off the bat because, after all, you're building a SharePoint workflow. Well, the problem with this template is it is easy to get into the *how* prematurely. A lot of the shapes in the template assume a fair amount of SharePoint knowledge. Take the "Create List Item", "Send Document Set to Repository", and "Wait for field change" shapes. Now for a SharePoint person, this may not be confusing. However, you probably wouldn't want to use that diagram in front of non-technical stakeholders.

Instead of the SharePoint Workflow template, start with a standard flowchart template. With the flowchart template, focus on the *what*, specifically meeting the business need. The first diagram you should focus on identifying the various "states" the workflow may use. A "state" defines the time a workflow waits for something to happen before proceeding to the next step. Figure 1.19 shows a flowchart model of a shopping cart workflow for an internet business that sells golf equipment. At a high level, there are only four "states" in this business process:

1. **Pending Payment**: When an order is submitted, the workflow waits for the payment to be processed.
2. **Pending Manufacturing**: In this example, a buyer may order golf clubs fitted to a custom length which causes the workflow to wait upon the manufacturing department.
3. **Pending Shipping**: Before the order is fulfilled, the workflow must wait for it to be shipped. This may involve employees assembling the ordered equipment, packaging, and delivery to FedEx.
4. **Fulfilled**: after the order is shipped, the order is complete.

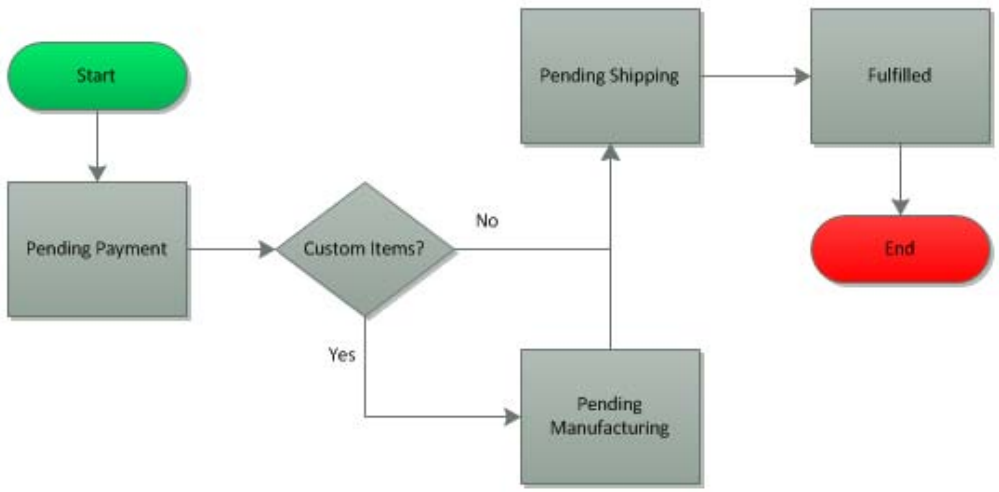


Figure 1.19 Start molding your business process by showing how the workflow will move from one "state" to another. Agree upon the high-level requirements before getting too technical.

This workflow will certainly be more complex than this diagram shows, but this is a great start. Quickly, someone viewing this diagram can see what the workflow will do. With this high level diagram agreed upon by business stakeholders, you can move to a lower level.

Taking it to a lower level doesn't mean we go to the SharePoint Workflow template in Visio, yet. First, let's expand upon each of the four "states". Figure 1.20 shows how you might expand upon the "Pending Payment" state:

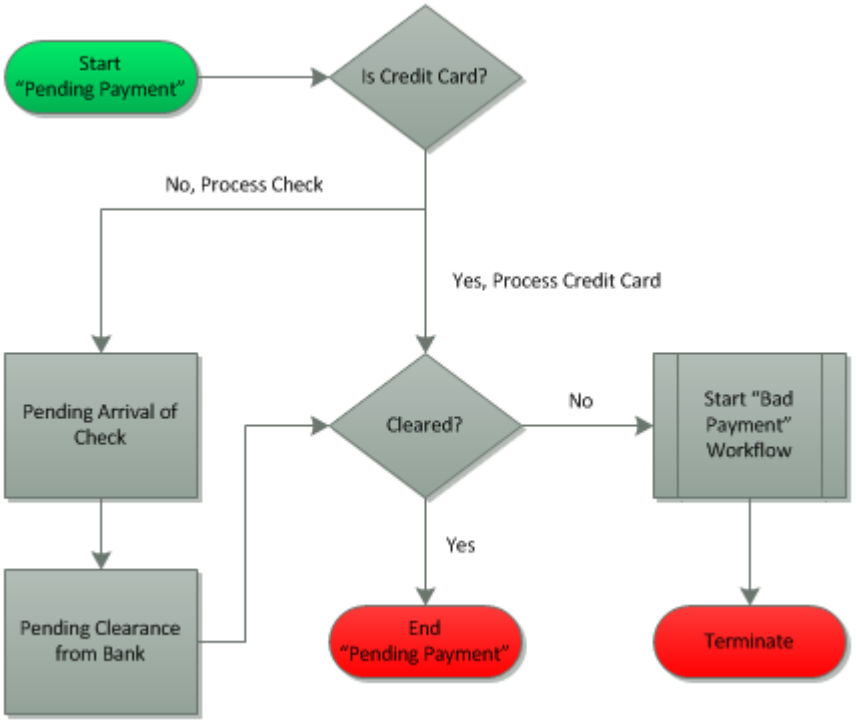


Figure 1.20 Take your high-level model and expound upon each of the "states" by creating a new diagram for each state. This will provide valuable detail, but yet remains non-technical for easy consumption.

The first thing the Pending Payment state does is check the payment type. If payment is by credit card, the card is processed. If it is a check, the workflow waits for the check to arrive in the mail, and clear the bank. In either the credit card or check method, if the payment clears the Pending Payment state, the workflow completes. Otherwise, a different workflow is started that handles bad payments and the current workflow terminates.

Figure 1.20 is still fairly high level. It would be easy to get into the weeds around *how* to do this, but again, this early in the process focusing on *what* is more important. Create diagrams for the other three states and run the flowchart past stakeholders. If at this point you get a green light to go forward, you're probably ready to start using the SharePoint workflow template.

1.8.2 Identify Human Interaction and SharePoint Objects

Now that you know what the high level business process is, it's time to turn your attention to how you're going to implement it. The first step is to take the flow charts you already created and look to see where you'd expect there to be human interaction. Examples of human interaction could be someone uploading a document, or perhaps submitting a form with some data. You also must consider what SharePoint objects your workflow will touch. In the case of the Golf equipment e-business, the workflow starts by a buyer submitting their order. That order could be logged into a SharePoint list, and your workflow would automatically fire when a new item is added.

From a human interaction perspective, let's consider the Pending Payment state again. Notice that if the payment is in the form of a check, the workflow waits for the check to arrive in the mail and clear the bank. Both of these points are examples where human interaction can help the workflow progress. For instance, you might have a mail department that receives mail' when they get a check, they log into SharePoint and tell the workflow that the check the workflow is waiting on has arrived. Similarly, an Accounts Receivable employee may monitor bank deposits and notify the workflow when the check clears.

One way to accomplish this is through tasks. Before the workflow waits for the check to arrive, it could create a task assigned to an individual in the mail department. When the mail clerk processes the check, he/she would edit this task in SharePoint, thus informing the workflow that the task has been received. You could similarly assign a task to someone in the Accounts Receivable department. That person could edit the task when they see the check has cleared.

Both of these examples of human interaction illustrate a workflow waiting for some event to occur before proceeding. You'll need to think through these points of interaction, and how to best inform the workflow to proceed.

This is where the SharePoint diagram template in Office Visio Premium can help. This template definitely lends itself to designing the workflow. With the task example, there's a shape called "Assign a to-do item". By dropping this shape onto the diagram, you specify that the workflow will create a task assigned to someone. This is seen in Figure 1.21, where our original high-level flowchart has been created with the SharePoint Workflow template:

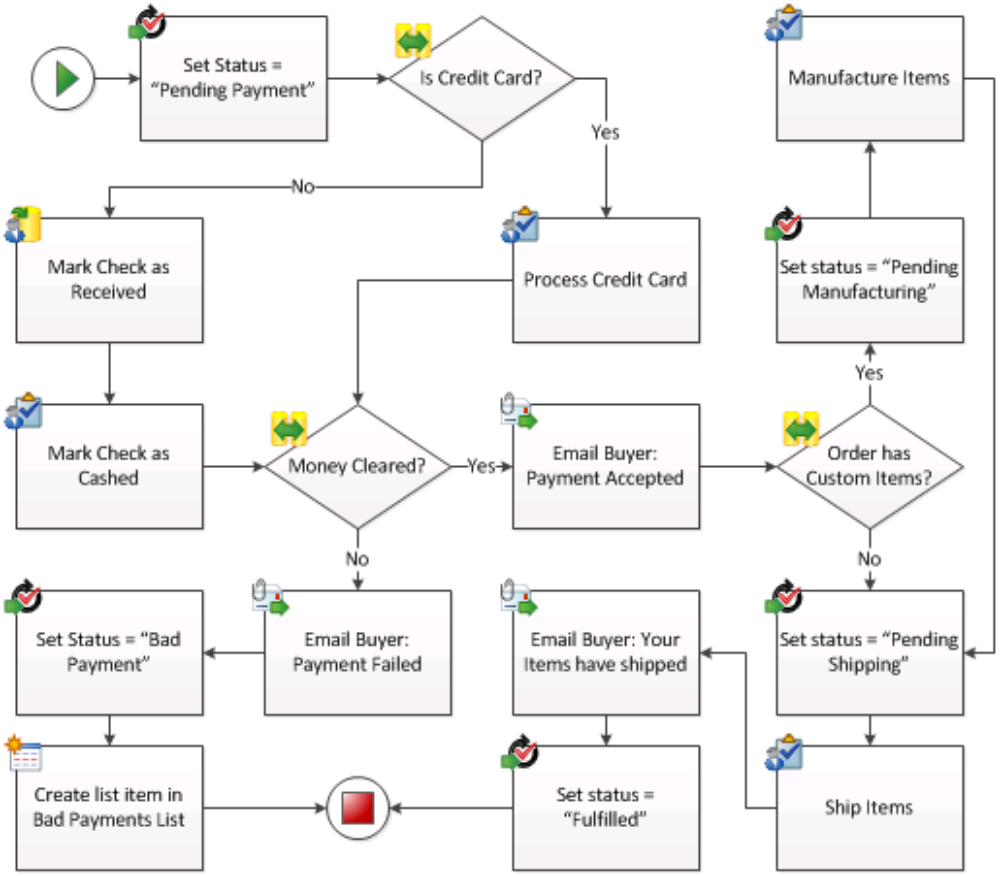


Figure 1.21 The SharePoint Workflow template in Visio produces detail to a much lower-level, than say the standard flowchart diagram. The advantage is the design is more clearly seen, but the disadvantage is it may be harder for non-technical people to read.

You'll notice the two boxes for check processing have been replaced with the Collect Data From a User shape and the Assign a To-Do Item shape. You'll also notice the diagram is considerably more granular. Notice the shapes signifying that an email is sent. You can see this if the payment was bad. First the workflow sends an email, presumably to the buyer notifying them that they submitted a bad payment. Then the "Create List Item" follows, signifying that the workflow will create a new list item in a separate this. This in effect is what starts the "Bad Payment" workflow which is defined elsewhere. After the secondary workflow is started, the current workflow sets its status to "Bad Payment" and then terminates. The other three states (Pending Manufacturing, Pending Shipment, and Fulfilled) have only slightly been expounded upon.

What type of person uses the SharePoint Workflow Visio template?

Each shape in the workflow Visio template corresponds to an action in a SharePoint Designer workflow. Because of this, the developer of the Visio diagram really needs to know quite a bit about SharePoint Designer workflows to model the workflow using the workflow Visio template. It is for this reason that I recommend non-technical people use the flowchart template, and those who are more technical use the SharePoint Workflow template. My opinion is to let the non-technical Business Analyst focus on the high-level business process, and to let someone with experience building workflows do the design work in the SharePoint Workflow template.

This section has been a brief introduction for chapter 6, Custom Visio SharePoint Workflows. For more detail on how to diagram workflows in Visio, reference chapter 6. In addition, one of the best features of workflows diagramed in the SharePoint Workflow template, is that the diagram can be imported into SharePoint Designer. This is a big time saver because all the actions and conditions are pre-created on behalf of the workflow developer.

1.8.3 Determining the Deployment Scope

With the design of the workflow now at a fairly low level, it's time to turn our attention to the scope of the workflow. Scope encompasses a few different things. First, what does the workflow run on (eg, a Site, a document, a list item, or a content type). Secondly, if the workflow needs to be reusable, where is it made available (site, site collection, or farm).

For our Golf Equipment example, it is easy to see that the workflow runs on top of a list item. This list itself may be called "Orders", and every order (list item) has its own workflow instance. Alternatively, a workflow could run on a document, a content type, or on the site itself. That latter is called a Site Workflow, where the workflow runs on a site, not an item.

"Reusable" means that you want users to be able to add the workflow onto multiple lists or sites. For example, all of the out-of-the-box workflows are reusable, in that they can be used more than once. Our Golf e-business example would not need to be reusable because it meets a one-time business need. If it did need to be reusable, you'd decide which scope the workflow should be deployed. This involves four possible options, site, site collection, web application, and farm.

Workflows targeted to either site or site collection can be created with either SharePoint Designer or Visual Studio. In SharePoint Designer, you would create a "Reusable Workflow" to make a workflow available across a single site. If you want that workflow to be available across an entire site-collection (multiple sites), you'd promote the workflow to be a "Globally Reusable Workflow". Alternatively, if you want the workflow to be available across an entire web application or the entire farm, you'll need a Visual Studio workflow.

1.8.4 Choosing the Appropriate Workflow Authoring Tool

There are a few tools you can use to build custom workflows, but SharePoint Designer stands out for many reasons. Just a couple of reasons that stand out is its ease of use and is speed. There are some downsides to SharePoint Designer where tools such as Visual Studio are needed to fill the gap, and it's important to know what tool is right for the job.

SHAREPOINT DESIGNER

SharePoint Designer is the first of the two authoring options. SharePoint Designer workflows are what's called declarative, meaning the workflow is XML based (XOML to be precise), rather than compiled code (as in the case of Visual Studio workflows). Check out these six compelling reasons to use SharePoint Designer for your custom workflows:

It's easy to learn. The language and terminology within the workflow components is easy to understand because it is written in plain English. Wizards and familiar interfaces are also used. For example, when composing a workflow-based email message, the dialog box is structured to look like an email message in an email client.

It's Fast. Writing software, which is essentially what a workflow is, typically requires a complex programming language such as .NET or C#. A workflow that might take all day to develop with .NET could be written in just a few minutes with SharePoint Designer (although there are many limitations when compared to .Net workflows). This speed is due in large part to SharePoint Designer's point and click interface.

It's More Powerful than previous versions. SharePoint Designer 2007 allowed users to create workflows, but the new version includes many more features and has refined the development process significantly. It is now easier to work with and move workflow steps. Also, there are many more actions available than before, such as the Utility workflows which allow you to manipulate text strings. It is also now possible to access a user's profile data, such as their phone number and manager's name via workflow. Also, some complex tools have been introduced, such as Parallel Blocks which allow multiple activities to happen at the same time. All of these concepts, and many others, make SharePoint Designer 2010 more powerful than ever before.

Its workflows can be moderately complex. It is easy to configure your workflows to make decisions using components called Conditions. Large selections of actions allow the workflows to modify data and send notifications to users.

It does not require Visual Studio .NET coding or knowledge. As powerful as Visual Studio .Net is, it takes a long time to become proficient with it. SharePoint Designer 2010 requires no code, but still allows for the user of common programming tools, such as variables, which store data temporarily, and functions (in this case called actions) which perform a variety of tasks. The learning curve for SharePoint Designer is much smaller than its Visual Studio counterpart.

It uses a Familiar Interface. SharePoint Designer uses the same interface as the rest of Office 2010, allowing business users to adopt it more quickly. In addition, the hyper-link based configuration process will be accessible to anyone familiar with web browsing.

Anyone who needs to create SharePoint workflows will find value in SharePoint Designer 2010. In the past, business people did not have a role in the development of new software functionality.

SharePoint and SharePoint Designer have changed that, allowing people who do not write code to create functionality that can be shared with other users.

.Net developers will be impressed with how quickly a complex workflow can be created compared to starting from scratch with a code-based solution. They may even find that creating a new workflow in Designer initially, then using Visual Studio .NET to add more functionality later yields faster results than using Visual Studio .NET alone. This process is aided by the fact that workflows created with SharePoint Designer can now be imported directly into Visual Studio.

Most businesses have a group of people who know the businesses processes very well, but are not developers by trade. These users will appreciate how easy it is to translate existing processes into SharePoint workflows using SharePoint Designer.

Finally, SharePoint administrators and power users will likely find the most value from SharePoint Designer 2010 workflows. This is because they are often asked to translate business requirements into SharePoint solutions in a limited amount of time and technical knowledge.

VISUAL STUDIO WORKFLOWS

When you get to the point where you know you need a workflow, and you're considering using SharePoint Designer because of its apparent ease of use, it's important to stop and do a deeper comparison Visual Studio workflows. There are some key pros and cons that ought to be considered between the two tools before moving forward with one or the other. Visual Studio workflows are covered in detail in chapter 8.

The foremost thing to consider is how technically skilled the developer of the workflow is. SharePoint Designer is primarily for Power Users and Site Administrators. Visual Studio is primarily for .NET developers with a programming background. If you don't have any resources available that have a .NET programming background, it is pretty easy to conclude that a SharePoint Designer workflow is your best choice. However, be aware that SharePoint Designer comes with some out of box functionality, and while it is rather extensive, it may not be able to accommodate the most complicated of business requirements. This may necessitate going outside of your organization for a .NET developer if you don't have one.

Beyond these "high level" deciding factors, there are several more comparisons that should be made between SharePoint Designer and Visual Studio workflows. Notice how Table 1.1 highlights the main differences between the two tools. The ordering of the table brings to attention the most common "deal breakers" that necessitate Visual Studio or SharePoint Designer at the top, followed by differences that are not quite as critical lower in the table.

Table 1.1 Comparison of the main differences between SharePoint Designer and Visual Studio workflows

SharePoint Designer	Visual Studio
Code Free development (limited to "safe", pre-deployed activities)	Code centric development (anything goes)
Sequential workflows only	Supports both Sequential and State Machine
Deployable across a site collection, but not beyond	Deployable across entire farm via a Feature

Automatically deployed into SharePoint	Must be packaged within a Feature and deployed by a farm administrator or in a sandbox
Office Visio can be used to model workflow logic	No support for Office Visio
No debugging available to "step through" a workflow at runtime	Full debugging experience available
Intuitive support for Forms customization	Less intuitive InfoPath integration. Availability for ASP.NET custom forms
Workflows cannot be modified at runtime	Workflows can be modified while running (see chapter 9)
Compiled Just-In-Time	Compiled at design time

The first differentiator was already mentioned, being that SharePoint Designer is code free environment, whereas Visual Studio is a code centric environment meant for programmers. This has obvious implications.

The second differentiator is that SharePoint Designer only supports sequential workflows. This can be a deal breaker, because there are many examples of business processes that don't take place sequentially, but rather go from state to state without any predetermined foreknowledge. For instance, consider a bug tracking system for a computer program. A bug may be placed into a "Pending" state, waiting for a developer to start working on the bug. That developer starts working on the bug and fixes it, putting the bug into a "Fixed" state. Thereafter, a tester goes to confirm that the bug was fixed, and finds that is was not and must place the bug back in a "Pending" state. This ability to go back in time, or to a previous state, is only available with State Machine workflows and not present with Sequential workflows, thus ruling out SharePoint Designer as a viable option.

While there are several more differentiators mentioned in the table, the third in the list is the last of the most common "deal breakers" necessitating Visual Studio. SharePoint Designer supports deploying workflows "Globally", however, this global deployment only means to deploy the workflow across one particular site collection. You may have a requirement that a workflow ought to be able to be instantiated from anywhere, on every site, across the entire farm. This true global deployment is only achievable with a Visual Studio workflow deployed via a feature. More on Visual Studio and "features" in chapter 8.

WHAT'S RIGHT FOR THE GOLF EXAMPLE?

The golf equipment online sales workflow example lends itself to a Visual Studio workflow, in my opinion. The most apparent reason is the "Bad Payment" requirement. If someone orders something online, and they type their credit card number wrong, you don't want the workflow to just stop. The example shows it starts a new workflow instance, but wouldn't it be better if the workflow just set its state back to Pending Payment, and requested from the buyer to resubmit their info? If you recall from earlier, this would require a State Machine to "go back in time" like this. Since only in Visual Studio can you build State Machines, it remains the best option. Of course if just starting a new workflow and terminating, as the diagram shows, is acceptable, SharePoint Designer would remain viable.

1.9 Real World Examples Found in this Book

The examples found throughout this book can easily be applied against real world business needs. Rather than make you dig through the pages to find all the neat examples herein, this section will serve as your table of contents for examples that you can take with you into your company. Table 1.2 shows the examples within this book, where it's found, and a brief description of what the example is and does:

Table 1.2 Examples found within this book

Chapter	Example Name	Description
1	Purchase Order and Fulfillment Workflow	At the end of Chapter 1, and complicated business process such as this workflow is conceptually "architected" from start to finish. You'll get a good idea of the decisions and techniques used to plan and design your own custom business process.
2	Requirements Document Workflow	This simple example uses the Three-State workflow to garner the compilation and approval of a requirements document for a software application.
3	PTO Request Workflow	This SharePoint Designer workflow allows for a user to submit a request for paid time off (PTO), and garner his/her manager's approval for their request.
4	Capital Expenditure Request Workflow	A form is built that allows users to submit a capital expenditure request when they need funds for a new company initiative. A custom task process is used to garner the approval of the request.
5	Document Sets, Security and Expense Reports, and Sales Order external data Workflow	This chapter has three examples. The first includes a walkthrough of how to send a Document Set containing sales presentation materials to a Records Center for retention. The second involves an expense report workflow that manages security. Users can upload their expense report, and the SharePoint Designer workflow alters the security on the report so only their manager can see and approve or reject the expense. The final example is a SharePoint Designer workflow that tracks sales orders store in an external SQL database. The workflow uses Business Connectivity Services (BCS) and external content types to connect to the external data.
6	Training Request Workflow	Office Visio 2010 is used to Model a Training Request workflow. After the workflow is built in Visio, it is imported into SharePoint Designer and thereafter published to SharePoint.
7	Expense Report InfoPath form	InfoPath is used to create a custom Expense Report form that users can enter and publish their expenses with. This is a great

		primer for anyone new to InfoPath. Advanced IntoPath topics like creating dynamic drop downs off SharePoint Data, customizing the out of box forms, as well as working with initiation forms in SharePoint Designer workflows are also covered in this chapter.
8	Maintenance Order Fulfillment Workflow	A workflow is created with Visual Studio that manages the submission and fulfillment of a maintenance request for a collage dormitory.
9	Service Request Workflow	A Visual Studio workflow is created that is used for the fulfillment of service requests for a technical helpdesk. The example relies heavily on custom InfoPath and ASP.NET forms for the submission and fulfillment of the requests.
10	Capital Expenditure Request Workflow	A similar workflow is created to the one seen in chapter 4, except this time it is created in Visual Studio rather than SharePoint Designer. This is a good comparison between the two authoring tools.
11	Create sub-site custom action and Sub Site Exists custom condition	A custom Visual Studio workflow activity and condition is created and later published to SharePoint Designer for power end user use. The action provisions sub sites from within SharePoint Designer workflows.
12	Pluggable workflow	A workflow local service is created that can send and receive messages from the "outside world". The example shows an event handle communication with a running workflow instance.

1.10 Summary

There are many powerful reasons why SharePoint workflows are a great tool to automate, track, and organize your company's business processes. Automating many of your most common business processes including expense reporting systems, paid time off requests, and capital expenditure requests is easily feasible within SharePoint and by doing so it is easy to see how you can add lots of value to your organization.

SharePoint workflows can execute on list items, documents, forms, content types, and even across an entire site or site collection within SharePoint. The boundaries are almost limitless in where you can take this technology and platform. Without even thinking too hard, you can put to use several compelling out of box workflows that come with SharePoint, and introduce immediate value into your SharePoint sites. If those workflows don't meet your business's unique needs, there's a host of workflow customization tools available like Visio diagramming, SharePoint Designer, InfoPath forms, and Visual Studio.

In that regard too, the 2010 release of SharePoint has introduced a slew of new functionality that greatly improves how workflows are architected on SharePoint. Tools like Visio will allow a non-technical Business Analyst to model out a workflow, and hand that diagram to a SharePoint designer or programmer to import and use, thus saving time, energy, and even miscommunication problems.

Another great improvement in 2010 is the ability to create reusable workflows. Before with SharePoint Designer 2007, you literally had to recreate a workflow on every list in your entire farm that you needed that workflow to execute. This was terribly inefficient and costly. Now with 2010 you can publish workflows globally which is a great time saver and drives a lot of consistency as well.

Next, we will take what you've seen at a high level in this chapter, and walk you through the details and specifics on how to setup your first out of box workflow. In later chapters you'll see a detailed walkthrough of all the out of box workflows and how to implement them at your company. Also much care will be given later in the book to describe specifically how to build custom workflows from scratch.

2

Your First SharePoint Workflow

Workflows that track features for in-development software are highly sought after in the workplace. Clients most likely have these software requirements in a document in SharePoint. A workflow could help this scenario, because it tracks the progress on that document from start to finish. Additionally, before completion, the workflow could enforce a final approval of that document before it is given to the team. The example demonstrated in this chapter will help facilitate the development of this requirements document by automatically assigning tasks, tracking versions and centralizing management of the document. This is accomplished by using one of the out of box workflows called the Three State workflow.

This chapter will take you through the planning process for developing your first workflow, as well as preparing a document library for a workflow in SharePoint. Once our foundation is laid, we'll walk through the steps to implementing and thereafter maintaining the Three State workflow. This will serve as the foundation that other chapters build upon, in that a lot of the basic nomenclature and concepts around SharePoint workflows will be demonstrated. The Three State Workflow used in the requirements document example will serve as a great starting point for a new workflow developer.

The Three State Workflow is only one of a suit of out of box workflows that are available to you. This includes prevalent workflows such as the Approval Workflow, Disposition Approval workflow, Collect Feedback workflow, and more. All of the rest of the out of box workflows in SharePoint will be covered at a high level towards the end of this chapter.

2.1 Planning and Preparing for your Workflow

The first workflow we're going to look at is the Three State workflow that comes out-of-the-box with SharePoint. This workflow is one of the most straight-forward to implement, and it is available across all versions of SharePoint 2010, which makes it a great place to start for beginners and experienced users alike. The first step to setting up any workflow is to get an idea of what business process you're trying to solve, and mapping that process out. Most of the out of box workflows, including the Three State workflow, have a very defined set of steps and parameters, so it's important to make sure it will appropriately meet your needs before starting. Additionally, we'll need to configure some columns before we can use the workflow.

2.1.1 Identifying your Business Process

The requirements document example demands a certain number of steps. A new requirements document would be created by an analyst. Afterwards, their manager would be required to review the document for approval. In the outline below you will notice additional detail around tracking the tasks, identifying the need for some sort of task list as well as a document library to store the documents themselves:

Step	Detail
Create Document	For this example you are creating a requirements document. A document is added to a library, and the workflow is started on that document.
Assign Task to Analyst	Assign the document to the appropriate analyst so it can be worked on. The task needs to be tracked so the manager can see the status while it is being worked on and knows when it is completed.
Update Document	The analyst will work on the document in a SharePoint library where versioning, check-in, and check-out features can be used to manage the creation of the document. When the analyst marks the task as complete it will trigger the workflow to move to the review stage where it will go to the manager.
Assign Task to Reviewer	The document will now be assigned to the reviewer who will receive a task informing them to review the document. The reviewer can track the review process in the task by setting the percent complete field.
Update Document	When the review process is done the reviewer will mark the task as complete and the workflow will move into the final state. Once the workflow moves into the final state the workflow will end it the document status will now be “Approved”.

Tasks Lists

Task lists is a feature in SharePoint that allows you to assign tasks to members in your team. Those tasks are stored in a SharePoint lists, and the assignee can navigate into that list and edit or complete the task. Examples of how tasks lists can be used are perhaps tracking bugs in software, or possibly handling maintenance requests for a facilities management department.

To get this set up, you will need to enlist the help of one of the out of box workflows called the Three-state workflow. There are a number of workflows that can be used when managing documents and tasks, each have their own advantages. Since you are going to be assigning tasks and tracking status' for the development of the requirements document, the Three-State workflow is the best fit for this job. Other options would include the Approval Workflow or even just turning on the Content

Approver option for the library. But, since you want to track issues you will benefit from the Three-State workflow.

2.1.2 Introducing the Three-State Workflow

By default, the Three-state workflow is designed to track the status of a list or library item through three states. This workflow is typically used for managing business processes that require your organization to track multiple tasks assigned to document or other type of content. It is also used to tack items, such as customer support issues, sales leads, or other project tasks.

The three different states mentioned occur sequentially. They can be named whatever you like, but there's always an initial state, a middle state, and a final state. Each state represents a different phase in the workflow and in between each phase is a transition state. With each transition between states, the workflow assigns a customized task to a person and sends that person a customized e-mail that refers back to the task they need to complete. Figure 2.1 shows how this may look:

Figure 2.1 The Three State Workflow has three states, with transitions between each state.

When this task is completed, the workflow updates the status of the item (in this scenario a requirements document) and progresses to the next state. You will use the Three-state workflow with a document library, but it can be used with any list that is set up with a “Choice” column with three or more values. The values in this “Choice” column serve as the states that the workflow tracks.

> **Issue Tracking and the Three State Workflow**
>
> The Three-state workflow is designed to work primarily with the Issue Tracking list. You can also use it with any custom list that has been configured to contain a Choice column with three or more values.

Here is a more formalized visualization of how the workflow works when using the Three-State workflow (Figure 2.2). The workflow automatically manages the creation of each task as well as the final state, and the tasks are updated and completed by the individual they are assigned to.

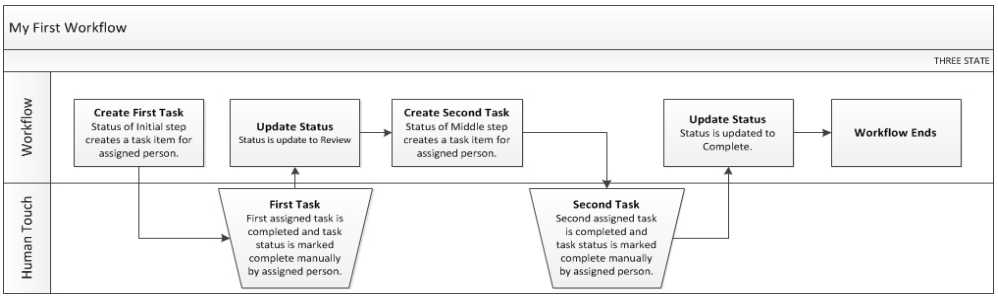


Figure 2.2 This diagram shows when human interaction occurs in the lifecycle of the workflow.

2.1.3 Preparing a Document Library for the Three State Workflow

Before you can use a Three-state workflow, you must set up a list or a library to use in conjunction with the workflow. This library must contain the items that you plan to track or manage by using the workflow. For our requirements document example, we need to setup a document library that will hold our requirements documents, as well as a couple columns on that library that the Three State workflow requires.

When you create a custom library in a team site to use with the Three-state workflow, you must make sure the library contains at least one column of type “Choice”. This column in turn must include three or more choice values that the workflow needs to track.

To get started, you first need to create a document library where documents can be created, uploaded, shared and tracked. This library is where the workflow will be assigned, and it can automatically start when a new document is created or added. In this scenario, you should be using the *Shared Documents* folder that is provided by default with a new team site.

The very first thing you need to know how to do is to get to the Workflow Settings screen. As with many features of SharePoint there are a number of ways to get here. As you have probably noticed by now the menus in SharePoint 2010 have adopted the look, feel and functionality of the ribbon style (Figure 2.3) that most Microsoft Office products have been using for the last few years.

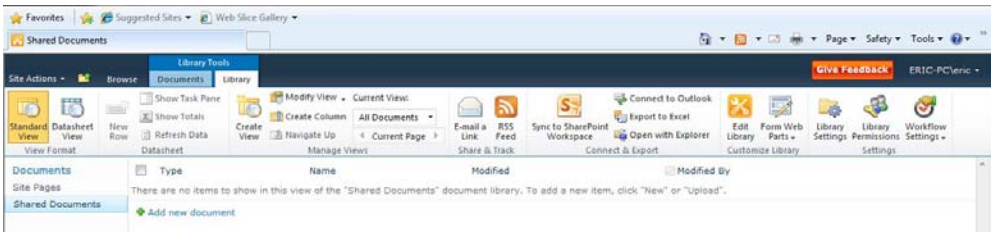


Figure 2.3 The new ribbon look and feel in SharePoint 2010 make it very familiar to Microsoft Office users. Use the Workflow Settings button on the far right to add and configure workflows.

Figure 2.3 shows the ribbon style layout displaying the Library Tools menu. Specifically, it is on the sub menu item of Library. This menu is where you will find most of your library settings and features like Views, Create Column, RSS Feed, E-Mail a Link, Edit Library, etc. With SharePoint 2010 the introduction of the Ribbon has dramatically reduced the number of clicks it takes to operate many of the administrative features.

The feature you are looking for on the ribbon is the *Workflow Settings* button on the far right. Once you have located the button you can click directly on the image and it will take you to the Workflow Settings screen (Figure 2.4), or if you click on the context arrow for this button you will see all the options that are available. The first option is *Workflow Settings*, since this is the first option displayed on the menu this is the default setting for the button, which is why when you click on the arrow it takes you to the Workflow Settings screen.

The next option is *Add a Workflow*; you can use this as a way of skipping the extra screen if you want to quickly get straight to adding a workflow. The last two options *Create a Workflow in SharePoint Designe*r and *Create a Reusable Workflow in SharePoint Designer* are also present but you will not be using these in this chapter. Refer to Chapter 3 of this book to learn how to create custom workflows in SharePoint Designer.

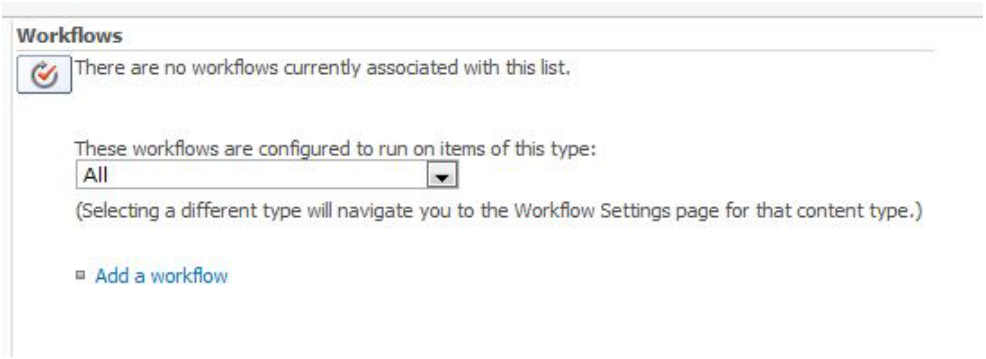
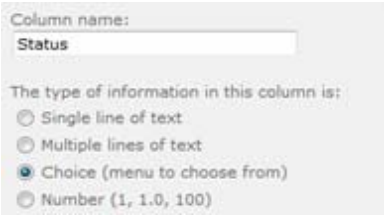


Figure 2.4 Workflow settings screen shows what workflows are available to add to a list or library. You can also remove a workflow from this screen as well.

The workflows screen for this library should now be displayed; this library should not have any workflows associated with it for this example. As you can see in Figure 2.4 the message “There are no workflows currently associated with this list” should be present. Otherwise, the names of workflows associated with this library would be displayed.

Now you need to add two new columns to this library, the first one will be a column type of choice and you will call it Status. The second will be a column type of Person or Group and you call it Assigned To. These two columns will be used by the workflow to track status and to whom the documents are going to be assigned to. You will use the column type of Person or Group, also known as the People Picker, because the information that is pulled from SharePoint will allow you to send this person e-mails or assign tasks. If you used a text field, you would not be able to assign tasks or send e-mails. Here are the steps for creating these columns:

Adding two custom columns to the Document Library used by the Workflow

	Action	Result
1	Create a Status column: A) From the Library Tools menu in the Ribbon, click on the *Create Column* button. B) Add a column called *Status*, this field should be a field type of *Choice* and should have three options: Create, Review, and Approved. Make sure you type each choice on a separate line.	 ... 
2	Create an Assigned To column: A) Create a new Person or Group column called *Assigned To*, this field will be used to assign the work of the workflow to a specific individual. B) Accept the rest of the settings as default. Click *Ok*.	A new column is created that tracks who the requirements document is assigned to.

The default view of this Shared Document library should look like the document library in Figure 2.5. You should see the default columns of Type, Name, Modified and Modified By and the new columns you just added Status and Assigned To. It is possible your default library template is different than ones used out-of-the-box; some organizations change the default settings to include customized default configurations.

Figure 2.5 After the columns have been setup, you should see an empty document library with the new columns.

You now have a customized document library that has two columns that will be used by the workflow that we will create in the next section. The order of the columns does not matter to the workflow, so you can order them however you want.

2.2 Implementing a Workflow

There are three main steps to implementing an out the out of box workflow in SharePoint. First, you must add the workflow to the list or library followed by starting the workflow on one of the list items or documents in that list or library. Lastly, you must test the workflow to ensure everything is behaving as you expect.

Adding the workflow to a list is done through the list's settings page or through the ribbon when you're on that list in the browser. When you select to add a workflow to that list, you'll see a list of all the currently installed workflows for you to choose from. Some workflows have some other dependencies. This was the case with our Three State workflow where it necessitated the two custom columns we added in the previous section.

After the workflow is added, it can either start automatically or manually. In either case, the workflow should be tested from start to finish. For our requirements document example, we'll add the Three State workflow onto the library we worked with earlier, and to test the workflow we'll create a new word document that will start the workflow after it is saved into the library. This will generate tasks and as those tasks are completed, the workflow will progress from start to finish.

2.2.1 Adding the Three State Workflow to a Document Library

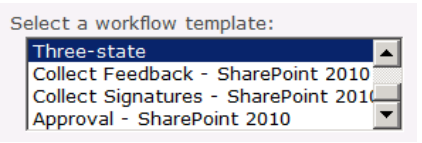
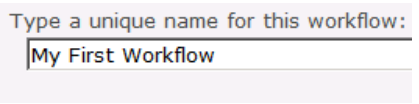
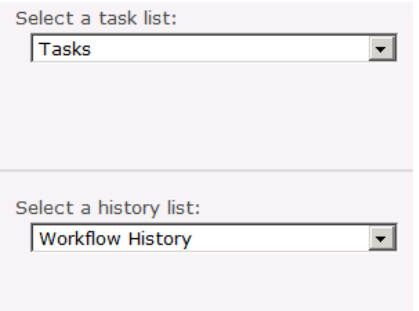
Now that the document library is setup, it is time to setup the workflow. There are a number of out-of-the-box workflows that come with SharePoint Server 2010. For our requirements document example we'll be using the Three-State workflow for this scenario. Many people tend to start with the Approval Workflow because the name of it sounds like exactly what they want to do but end up discovering that it is not robust enough for what they really want to do. The approval workflow only provides basic approvals and does not assign or manage tasks, so for a scenario like the requirements document example we're discussing, a workflow that is a little more robust like the Three-State is needed. This workflow also happens to be the only out-of-the-box workflow that comes with SharePoint Foundations so even someone with the free version of SharePoint can follow along with this part of the book.

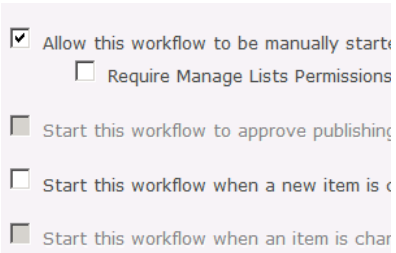
> **Out of Box Workflows that come with Foundations versus Server**
>
> SharePoint Foundations only includes one out-of-the-box workflow and that is the Three-State Workflow. With SharePoint Server you also get the Approval, Collect Signatures, Collect Feedback, Publishing, Disposition and Translation workflows - Please check out chapter 3 for more information on these out-of-the-box workflows.

When you add a Three-state workflow to a library or a list, you must specify which column in the list contains the state values that you want the workflow to track. In this scenario, you will be using the *Status* column that you added to your document library in the previous section. Since there is only one column with the type of *Choice*, the workflow template will automatically choose it by default.

You also will specify information about what happens at each stage of the workflow. For example, you will be specifying the individuals to whom tasks should be assigned and the details of the e-mail alerts that task recipients receive. Follow these steps to add the Three State workflow to a document library:

Adding two custom columns to the Document Library used by the Workflow

	Action	Result
1	Navigate to the Document Library's workflow settings page: A) Open the Shared Documents document library from your Team Site where you'll add the workflow. B) On the Library Tools menu in the ribbon, click *Workflow Settings*.	You're on the workflow settings page and ready to add a workflow to the library.
2	Add a workflow and choose a workflow template: A) On the Workflow Settings drop down, click *Add a Workflow*. B) On the Add a Workflow page, in the Workflow section, select *Three-state* under *Select a workflow template.* *C)* In the Name section, type a unique name for the workflow. In this scenario you will call this workflow "*My First Workflow*".	 *Note:* If you're running just SharePoint Foundations you will have fewer options in the list of workflows than if you're running SharePoint Server. You may also see custom workflows, if applicable. 
3	Associate the workflow with a task and history list: A) In the Task List section, specify a task list to use with the workflow. You will not create a new task list; you will use the default of *Tasks.* B) In the History List section, select a history list to use with this workflow. The history list displays all of the events that occur during each instance of the workflow. Use the default of *Workflow History* for this scenario. Note: You can use the default History list or you can create a new one. Much like tasks above, if you plan on having numerous workflows, you might want to create a separate history list for each workflow.	

4 Specify start options: A) In the Start Options section, *Allow this workflow to be manually started by an authenticated user with Edit Permissions* should be checked. All other check boxes should be blank or unchecked.	
5 Specify Workflow States: A) Click next to continue to the second stage of the workflow's configuration. B) In the Workflow states section, under Select a 'Choice' field, you should see the Status column you created earlier selected by default, if not, and then select the Status column manually. C) Select the column values that you want for the Initial state, Middle state, and Final state of the workflow. This is where you will refer back to the "Status" column you created earlier that had the different choices of Create, Review and Approved.	The states should be configured similarly to Figure 2.6: Initial = Create Middle = Review Final = Approved

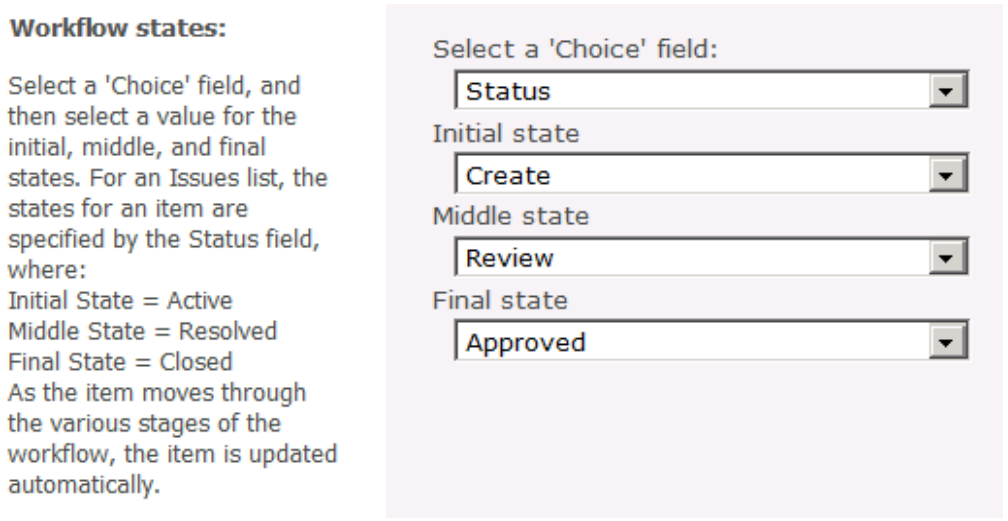


Figure 2.6 When setting up the Three State workflow, you must specify the "three states" to be used from the custom column that was created earlier.

Most of the time you'll leave the defaults for the next two sections in this second configuration screen. The basic premise is how do you want your tasks to function. If you remember, the Three State

workflow issues tasks between states. After the task that is issued is completed, the workflow moves to the next state. If you look at Figure 2.7 you'll notice our Assigned To column is already set in the Task Assigned to field. This is enough to continue in our example, but note there are other neat settings. For instance, you can specify unique text to be added into the email the task's assignee receives, rather than something generic.

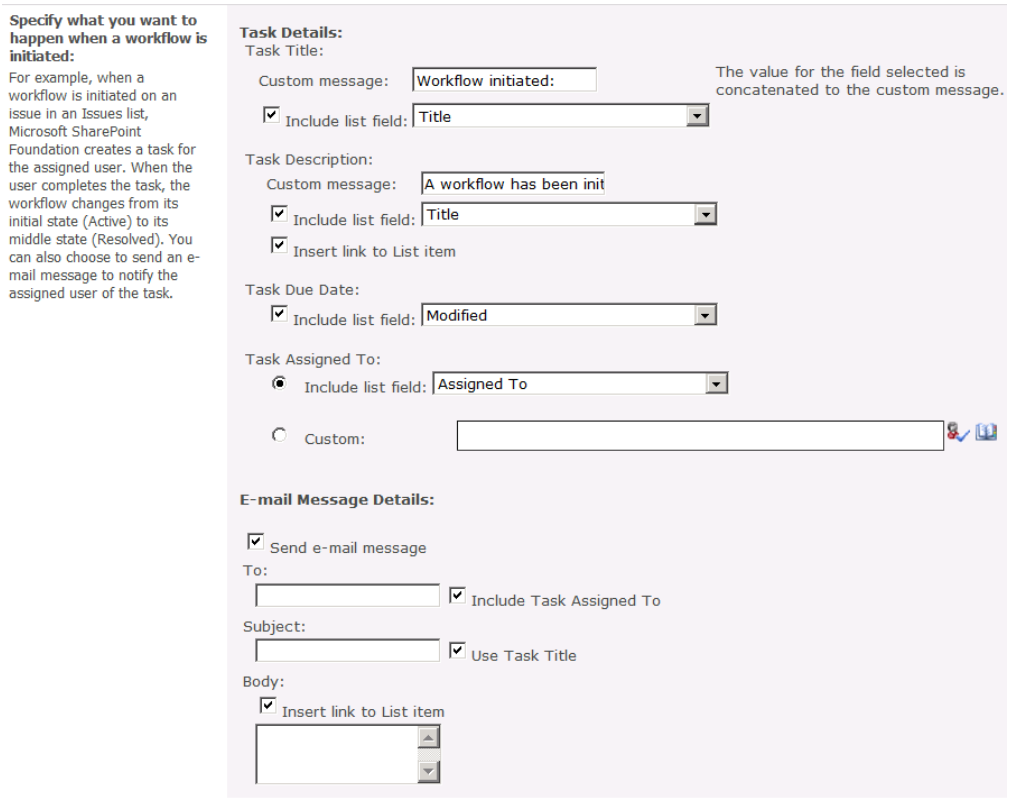


Figure 2.7 You can configure the look and behavior of the tasks that are created by the three state workflow. For example, you can specify who should be assigned the tasks, as well as provide a custom email message the assignee should receive when they're assigned.

After you have configured these final two sections to your liking, click the OK button. This will complete the adding of the workflow to the library. Below is a more detailed description of the settings on Figure 2.7 if you feel more customization is needed:

Task Title:

Type any information that you want to include in the in the task title. If you select the Include list field check box, the Title information for the list item is added to the custom message.

Task Description:
Type any information that you want to include in the description of the task. If you select the Include list field check box, the Title information for the list item is added to the custom message. If you select the Insert link to List item check box, a link to the list item is included in the description.

Task Due Date:
If you want to specify a due date for the task, select the Include list field check box, and then select the date column from the list that contains the due date information that you want to use. If you wanted to, you could make this a calculated date, for example the due date would be 30 days after the created date. You would create a date column in the document library that has a calculated value and you would use that as your choice in this field.

Task Assigned To:
To assign the task to a person who is specified in the list, click Include list field, and then select the column from the list that contains the user information that you want to use. When this workflow is started, the first task is assigned to the person whose name appears in this column for the workflow item. To assign this task in all instances of this workflow to a person or group you specify, click Custom, and then type or select the name of the person or group to whom you want to assign the task.

If you want to send an e-mail message to notify the assigned user that they have a task assigned to them, you can check the "Send E-Mail Message" box. Or, if you do not want to send E-Mail alerts, simply uncheck the "Send E-Mail Message" box.

To:
Type the name of the person to whom you want an e-mail alert about the workflow task to be sent. Select the Include Task Assigned To check box if you want to send the e-mail alert to the task owner.

Subject:
Type the subject line that you want to use for the e-mail alert. Select the Use Task Title check box if you want to add the Task title to the subject line of the e-mail message.

Body:
Type the information that you want to appear in the message body of the e-mail alert. Select the Insert link to List item check box if you want to include a link to the list item in the message.

Emails not working in SharePoint?

If you or other workflow participants are not receiving e-mail messages you need to check with your system administrator to make sure that e-mail messaging is configured for your environment.

Sometimes e-mail messages are also referred to as e-mail alerts.

2.2.2 Starting a Workflow

With our workflow added to our library, it's time to start the workflow on a document. To start the Three State workflow in this library you will need to upload a document to your library and initiate the workflow on that document. You can either upload a document manually, or you can create a new document from the Microsoft Office Word client and save it back up into the library. Since we'll be able to set our column data created in the previous section from within the Word client, we'll create a new document from Word and start the workflow manually off that new document.

First navigate to your Shared Documents document library that you added the workflow to in the beginning of section 2.1.3. The first thing we need to do is add a new document in this library, and set some metadata on that document. Follow these steps to create and assign metadata onto a new document:

Adding a new Document into a Document Library

	Action	Result
1	Create a new document: A) From within Shared Documents, click on the *New Document* icon.	A new Microsoft Office Word document will open. You will notice that the columns you added earlier are present at the top of the document in the Document Properties section of the page.
2	Set Assigned to and Status columns: A) Enter Create in our Status column, add a user into the Assigned To column, and lastly enter some text in the body of the document. B) Enter some text in the body of the document.	The columns should be set similarly to Figure 2.8.
3	Save the document and provide a unique document file name: A) In the Save dialog popup, change the filename to “My First Workflow.” B) Lastly, click the *Save* button. Note: When you save the document, you will see a save window that has been updated for SharePoint and Office 2010. This window shows the location the file will be saved to. This should be the same library you initiated the creation of the document from when you click on the New Document link. There are a bunch of options here that you are not going to get into at this point, like Adding Tags, Thumbnails, Map a Network Drive, etc…	After saving the document you should be taken back to your SharePoint library, where you will see your new document (Figure 2.9).

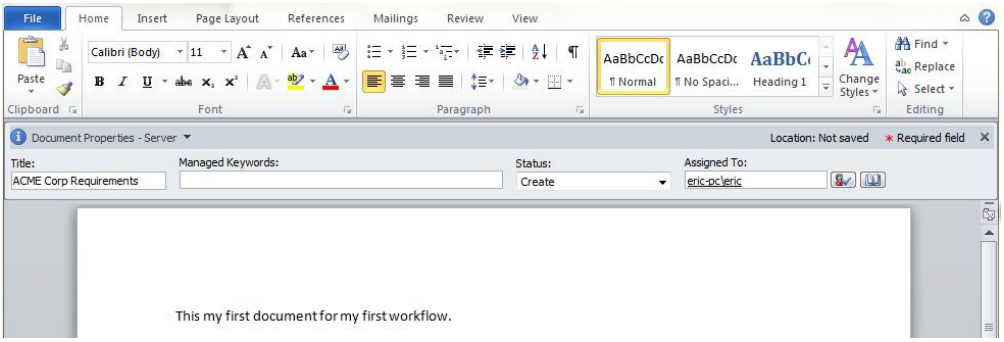


Figure 2.8 A workflow can be started on a document in a document library. From the Office Word client, you can publish a document into a library along with meta data that is needed by the workflow (Assigned To column).

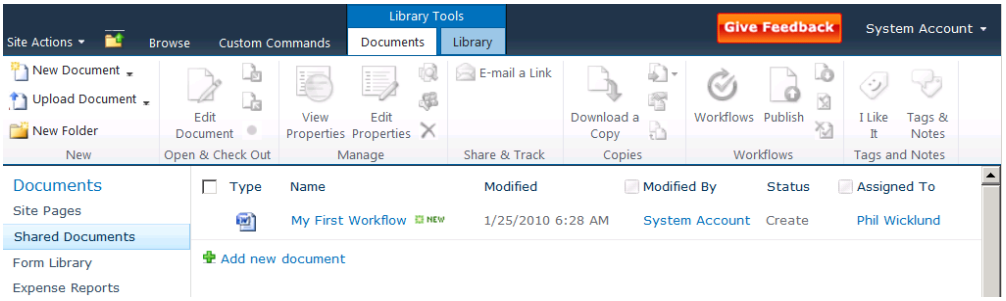


Figure 2.9 After the document is saved, it will appear in the browser showing the columns specified from the Word client.

Notice that the status of the document is set to "Create", and the value of the *Assigned To* column is set as well. Now that you have your document you need to initiate the workflow because we told this workflow to be started manually. If we had set it to automatically start it would have started as soon as we saved the document to this library.

Click on the arrow next to the Name of the file and a context menu will drop down (Figure 2.10). These are all the actions you have available for this document; obviously, you are only interested in the Workflows option for this scenario, so click *Workflows*.

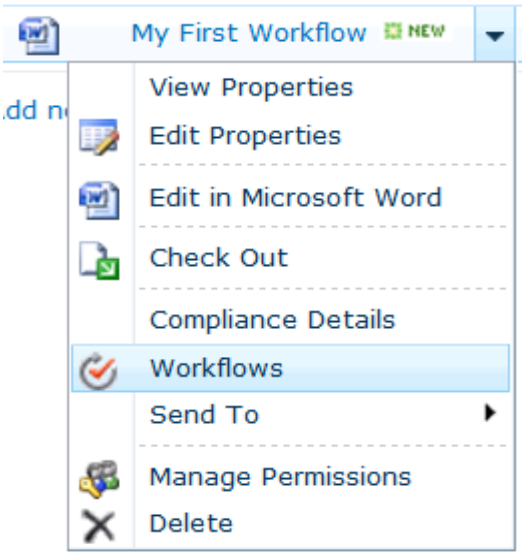


Figure 2.10 Context Options - The drop down context menu gives the user quick access to all the actions that can be performed on this document including starting a workflow on that document.

After you click the Workflows button, you will see the following screen with all available workflows that can be chosen for this library. Figure 2.11 shows the three defaults that come with SharePoint as well as the one you created earlier called My First Workflow.

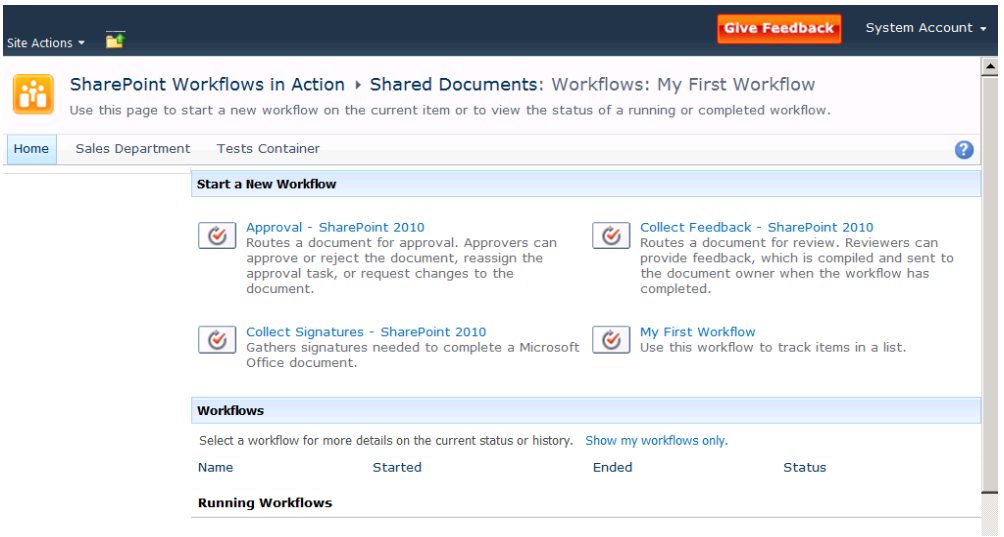


Figure 2.11 After you click "Workflows" on a document, you'll notice you can start a workflow or view any running or completed workflows.

Click on the *My First Workflow* link. You will briefly see a processing screen while the workflow starts. Then, SharePoint will return to your document library where you will see a new column called *My First Workflow* (Figure 2.12). This column will hold the status of the workflow, right now since it just started it is "In Progress". You will also notice that the Status field in Figure 2.12 just been grayed out indicating a workflow has updated this field.

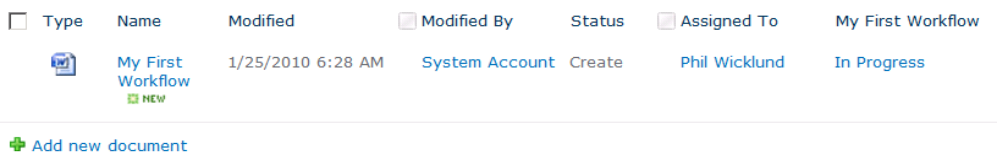


Figure 2.12 My First Workflow column shows the status of this workflow

2.2.3 Testing the Workflow

Now that you have created a document, saved it and kicked off the workflow successfully you need to test the moving parts that create the tasks and move the document through the workflow. First, you will look at your first assigned task and make sure the task looks right and that it actually works. You will then close the task and verify that the next phase of the workflow is kicked off properly. Then of course you will test the second task in the same manner and afterward the workflow should have completed successfully.

To get started, navigate to your tasks list that you associated the workflow with. Since this task list is only being used for this scenario you only see one task in your screen, see Figure 2.13. This is the task that was created and assigned to you based on the parameters you set in the workflow earlier. You will see the name of a task that has been assigned to you and that it is due today.

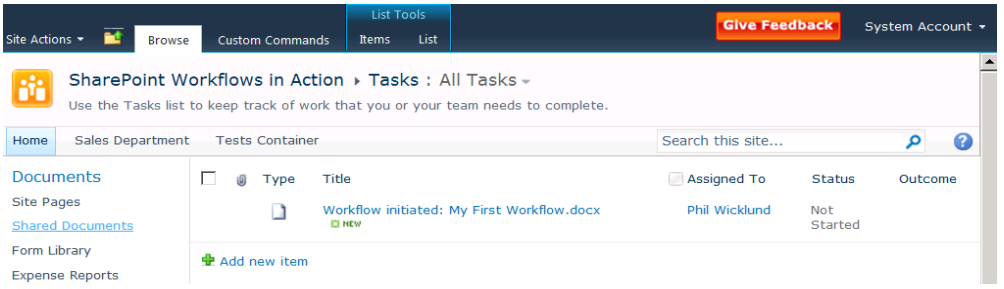


Figure 2.13 The Three State workflow created a task in the Task list that the analyst needs to complete before the workflow can proceed.

Click on the Title of the task to get to the next screen. In Figure 2.14 you'll see the task details. This is where you can read about or edit the task. There is also a link to the document this task is associated with.

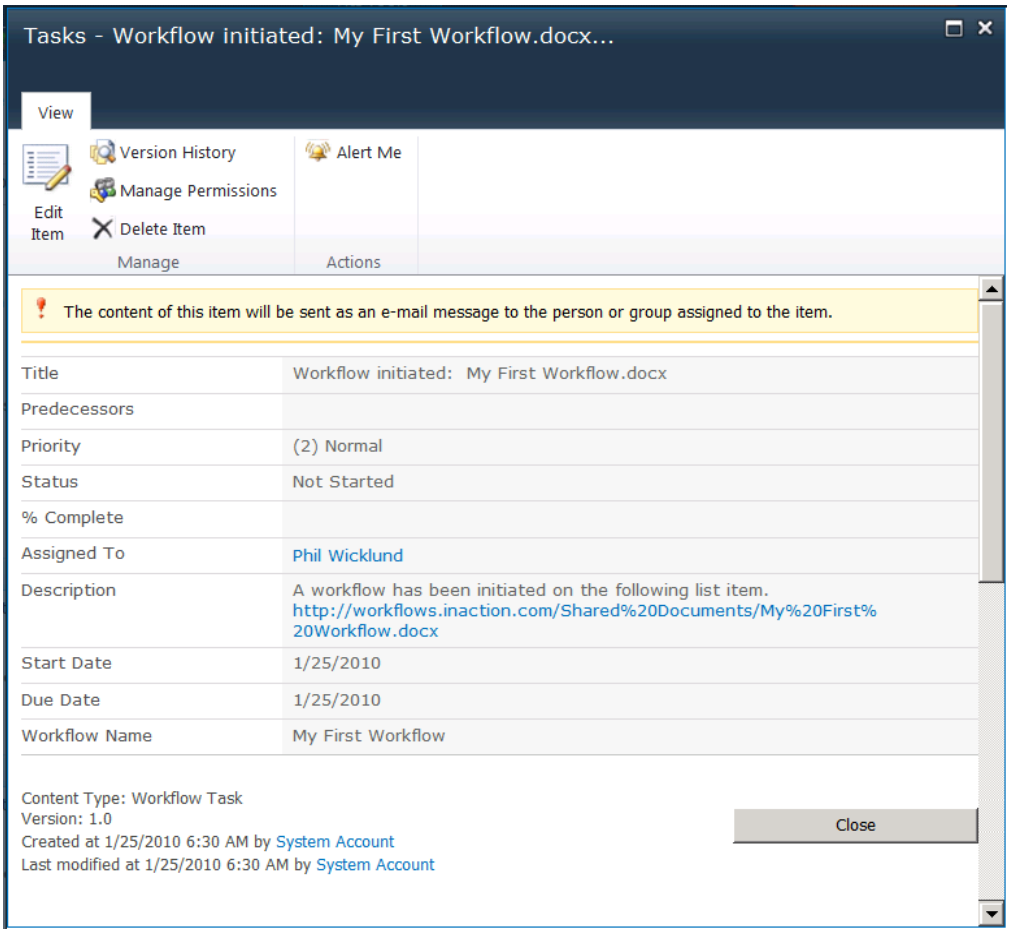


Figure 2.14 On the task details screen you will find all the details of the task and associated content. When the task is completed, the workflow will move to the next state.

Here you will find the Title, Status, Due Date, Assigned To, Description and some other details about this task. This is where the assigned task owner would update his or her status as they worked on this task. Click on the *Edit Item* button on the task so we can change the status of the task. You are only going to change the Status field. Change the Status to Completed, and click on the save button.

Afterwards, you will be taken back to your task list where you should now see two tasks, one completed and one Not Started (Figure 2.15). Following the requirements document example, this second task would be assigned to your manager. If you don't see two, refresh your screen.

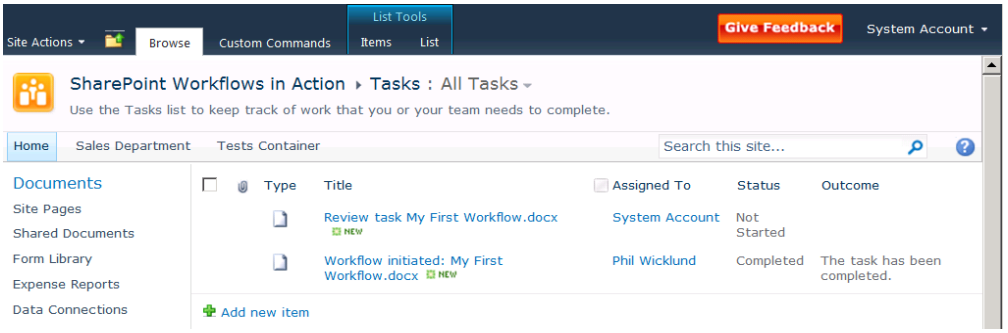


Figure 2.15 After the first task is complete, a second task will be created assigned to the review of the document.

Before you edit this second task and marked it complete like the first task, let's take a look at the document library again. Go to the document library you initiated the workflow from. You will notice in Figure 2.16 that the Status for this document is now Review, this means the workflow is working so far because the completion of the first task moved the workflow from the first state, to the second state.

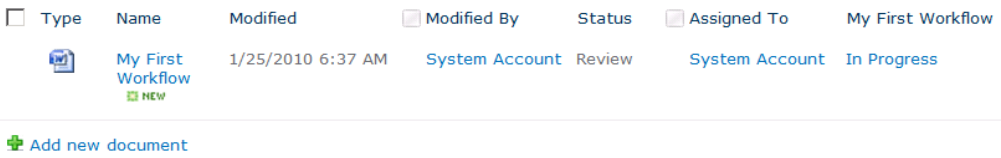


Figure 2.16 Notice when the task is complete, the workflow's status column is updated to "Review".

Now go back to your task list and complete the second task. After completing the second task you will see the workflow will be completed and the Status field will be in the third state, *Approved*. The workflow's column in, in this case "My First Workflow", will now show "Completed".

2.3 Maintaining Workflow Instances

Beyond simply creating and testing workflows comes also some management and clean up that has to happen from time to time. It is important to have a strong understanding of all the information that is on the workflow status screen, as well as to understand how to terminate workflows and delete workflows. The workflow status screen provides a view into where in the process the workflow is currently executing. Terminating a workflow may be necessary in case a workflow fails on starting, or if an error occurs and the workflow gets hung up. If a workflow is no longer useful, that workflow may need to be deleted off the list or library so no new instances get created. Beyond specific workflow instances, there are also settings that can be applied across all workflows in a given web application. A few of these settings that are managed in Central Administration are around notifications, disabling SharePoint Designer, and the auto deletion of a workflow's history.

2.3.1 Working with a Workflow's Status Screen

The workflow status screen contains workflow information such as started date, last run date, a link to the document and the status. This is where you get your high level overview of the workflow. The screen also contains a list of tasks associated with the workflow and links to navigate users to those tasks.

This screen is also very helpful especially when dealing with multiple documents and many different workflows. Most people use it to track down where the document is in the workflow lifecycle or to get more details on where it has been so far. This screen will allow certain users to terminate workflows if needed.

Most importantly, the last section on the workflow status page contains a workflow history. This history log provides your end users with a view into where in the process the workflow is currently executing.

To get some additional details on a workflow, click on the workflow's status column. For our example, click Completed or In Progress link for this item in the My First Workflow column. You will now see the screen in Figure 2.17 with all associated information for this workflow, including tasks, basic information and the history. This screen can be used to help track the workflow and all the events associated with it. Many times people will look here to see where in the process the workflow is currently executing.

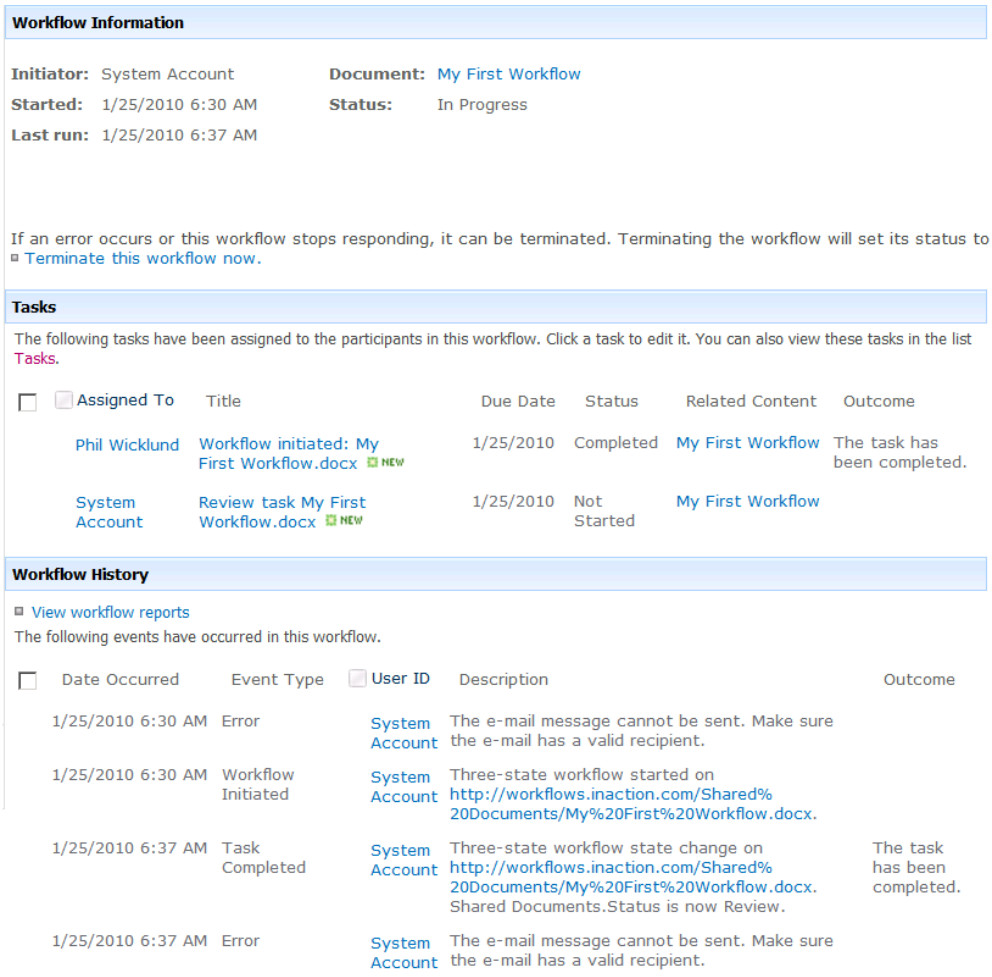


Figure 2.17 On the workflow status page you will find all the associated details and history for a particular instance of a workflow. This included tasks associated with the workflow, and a history log of where the workflow is currently executing.

2.3.2 Terminating Workflows

Sometimes it is necessary to terminate or cancel a workflow. Maybe it was started on accident or maybe some data changed in the initial library or list item. Another possibly is when if the workflow failed for some reason, as in the case with Figure 2.18. Whatever the reason, it is not difficult to cancel one if you have the right permission level in SharePoint.

Figure 2.18 Failed On Start - The workflow status shows that this document has a workflow attached to it that failed on start. It may be good to terminate this workflow to clean up orphaned tasks, etc.

Locate the item you want to cancel a workflow for. The example in Figure 2.18 shows an item with a workflow that Failed on Start (note the status of the workflow). If you click on the status of the workflow for the item you will be taken to the Workflow Status screen. Here you will find the link for "Terminate this workflow now." You will find it at the bottom of the Workflow Information section as shown in Figure 2.19.

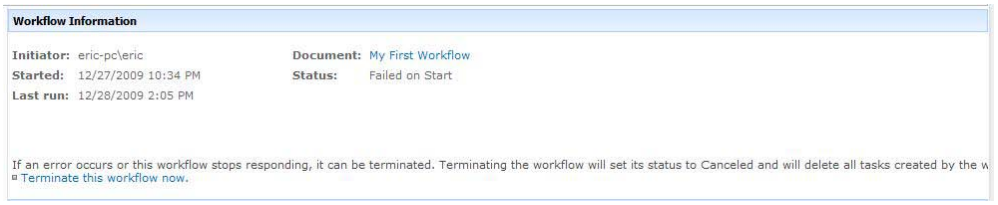


Figure 2.19 The Workflow History screen will have the Terminate this workflow now link you can use to cancel the workflow.

Terminating deleted associated tasks…

Terminating a workflow will also delete any related tasks to that workflow, be sure this is really what you want to do because you will not be able to get them back.

When you terminate your workflow the Workflow Status screen will update accordingly, as shown in Figure 2.20. All tasks will disappear and basically only the events will still be present on the screen.

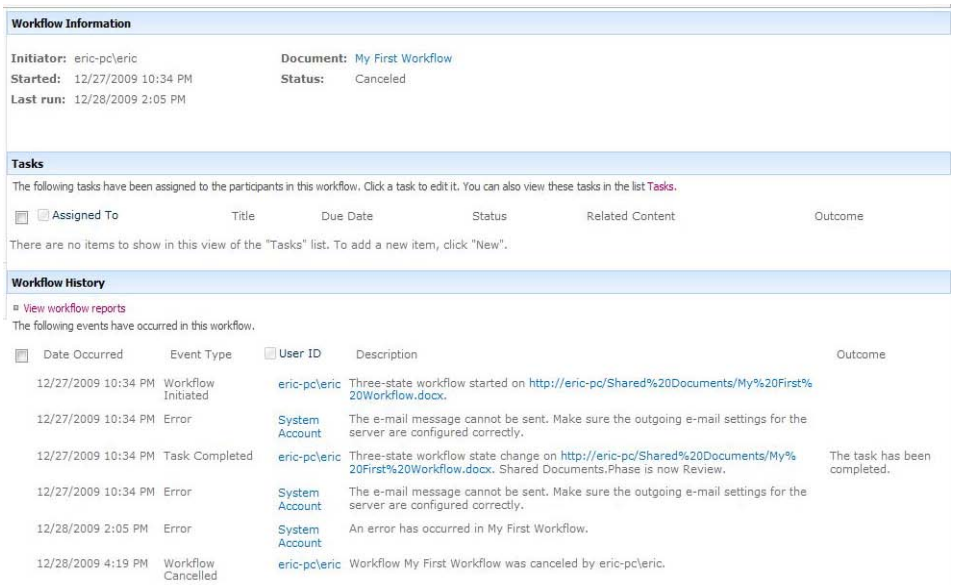


Figure 2.20 After a workflow has been terminated, the workflow status page will now show that the status of this workflow instance is cancelled.

When terminating a workflow you terminate only one instance. Note that you can have the same workflow running multiple times simultaneously on the same document. If you want to remove all instances across all documents, you will need to delete the workflow off the library.

2.3.3 Deleting Workflows

You may want to delete a workflow because it isn't in use anymore or you have created a bigger and better one that has replaced it. Be very careful when making the decision to delete a workflow, there is no going back and this can cause some pretty big issues if other workflows or instances rely on this workflow being there.

Deleting a workflow is done from the same Workflow Settings screen where you manage and add the workflows. When you click on the Remove Workflow link you will be prompted to select which workflow you want to remove (Figure 2.21).

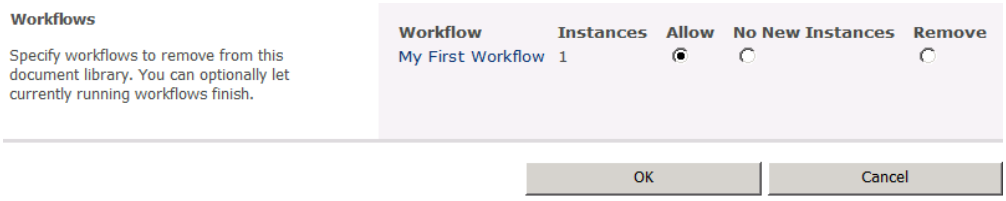


Figure 2.21 When you go to remove a workflow, you can either remove it outright, or you can just specify to not allow new instances of that workflow to start. The latter protects you from potential data loss so it is recommended, especially if you have a lot of instances running.

The options are fairly straight forward. By default the Allow option is checked, which allows new instances of the workflow to be initiated. Instead of removing the workflow outright it is highly recommended that you choose the *No New Instances* option instead of just outright removing or deleting the workflow. Deleting (removing) workflows instead of just not allowing new instances to be created can cause data loss, as well some interesting issues with other workflows that may also rely on this particular workflow to get stuck in weird states that can be very difficult to recover from. This option will save you a ton of headaches down the road, so use it.

2.3.4 Non-Authorized access to Workflows

Up until this point we've assumed that the people participating in the workflow are users that have been granted access to the site the workflow is running in. Well, if a workflow in the current site assigns a task to a user that doesn't have access, we may or may not want to send an email to that person. For internal users, the default is to send the email letting them know they have a task. For external users, the default is to not send email. To toggle this behavior, navigate to SharePoint Central administration and under Application Management click Manage web application. Choose the application you want to configure, click the drop down below General Settings in the ribbon and select Workflow. Figure 2.22 below shows the defaults set in the Workflow Task Notifications section:

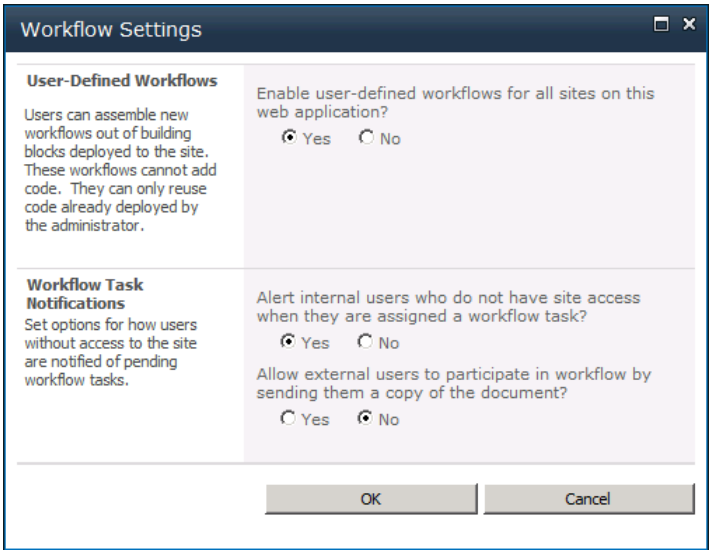


Figure 2.22 In Central Administration, you can disable SharePoint Designer workflows as well as specify notification rules for workflows.

Global Workflow Settings

Note: the settings found in sections 2.3.4, 2.3.5, and 2.3.6 are all global settings. This means the setting applies to all workflows in the specified site collection. Up until this point, the settings we've discussed are for a given workflow instance. Rather, these settings apply across the entire web application.

2.3.5 Enable or Disable SharePoint Designer Workflows

Now that we have our first workflow under our belt, we'll be transitioning to custom workflows. Not all companies want end users to create custom workflows with SharePoint Designer. Similar to where we managed anonymous access and history, we can also disable SharePoint Designer workflows.

This functionality is configured in the same dialog menu as seen in Figure 2.22. Navigate to SharePoint Central administration and under Application Management click Manage web application. Choose the application you want to configure, and click the drop down below General Settings in the ribbon. In the workflow settings dialog box, select No next to "Enable user-defined workflows for all sites on this web application".

2.3.6 Preservation of Workflow History

Earlier we discussed the workflow history log associated with each workflow. What is not well known about this log is that the events in the log are deleted 60 days after the workflow completes and you might not want that to happen. To change this setting for a web application, within Central Administration click *Monitoring* on the left navigation and then click *Review Job Definitions*. Click the Workflow Auto Cleanup job for the web application you want to configure. Note, there are several hundred jobs and only the first hundred are shown. You may need to toggle a few pages before you find the right job. After you've found the job, click the Disable button to disable the auto clean up feature.

> **Caveat with disabling the auto cleanup**
>
> If you disable the cleanup, you may experience performance issues with the Workflow History lists used by your workflows. Browsing workflow history may slow down when there are thousands of items in the list. If you want audit functionality for your workflows, it may be better to have a history list for each workflow, or write to an external database such as SQL that can scale better.

2.4 Additional Out-of-the-box Workflows

The common misperception surrounding anything labeled "Out-of-the-box" conjures up thoughts of inflexibility, too simple, irrelevant to my needs and simply not very usable. SharePoint has changed these misperceptions by developing six very powerful out of box workflows to help you be more efficient, organized and productive with your daily tasks. You can look at these workflows as templates, whereas they can quickly become the foundation or starting blocks to get quite creative. The great thing about having these readily available is their ability to be used by individuals without a high level of technology acumen. This being said, regular users are now equipped with highly powerful tools to make them more successful in their daily activities.

In the previous sections we walked through the first of these six out of box workflows in great detail. Fortunately for you, the other 5 workflows are configured in a very similar way. For the most part you'll use the same pages, configurations, and settings for all of the six out of box workflows. Because of this, and since the walkthrough provided in the previous sections was quite extensive, this section simply focuses in on "business" behind the workflow. What's the purpose of the workflow, and what business problems does it solve will be discussed for the other 5 out of box workflows in this section.

SharePoint Server License Required

All of the workflows in this section require a SharePoint Server license to use (Standard or Enterprise). The Three-State workflow is the only out-of-the-box workflow that comes with SharePoint Foundation.

2.4.1 Approval Workflow

The approval workflow (Figure 2.23) is very similar to the Three State workflow just described, except that a name of Two State Workflow would be more fitting. Essentially this workflow is used to approve or reject documents. When the workflow is started, the state of the workflow is "Pending". A task is assigned to the approver(s), and after the task is approved or rejected, the workflow goes into its second state of "Completed". The Three State workflow has a beginning, middle, and an end state, whereas the Approval workflow only has a beginning and an end.

Another difference though is the with the Three State workflow you can define the states however you wish. With the Approval workflow, the states are not configurable ("Pending, etc). The Approval workflow is really a trimmed down version of the Three State workflow, used only in the approval of documents or items.

You may be wondering, why not simply use the out-of-the-box content approval features that are available? The nice thing with the Approval workflow is it creates tasks assigned to people to manage the approval, whereas with content approval, you update the document or list item directly. With content approval, the document may be waiting to be approved, but no one is notified that it's waiting. With the approval workflow, the approver will get the notification via email.

Another important thing that the approval workflow brings to the table is the use of Assignment stages. Assignment stages are discussed in greater detail in chapter 10, but essentially they give your users the ability to specify more than one individual that needs to approve the document. You can also specify in what order the approvals need to take place, or conversely if the approvals can happen in parallel. Assignment stages also let the person starting the workflow specify all the approvers and settings at run time. This is nice because you don't have to know who the approvers are ahead of time, as you would in the Three State workflow.

Finally, the Approval workflow has an advantage of allowing the tasks assigned to the approvers to be reassigned. This may happen if the designated approver doesn't feel qualified to make the approval. They can then reassign the task to approve the document to someone else, in which case the workflow will no longer be waiting on the original individual, but rather on the one whom the document was just reassigned to.

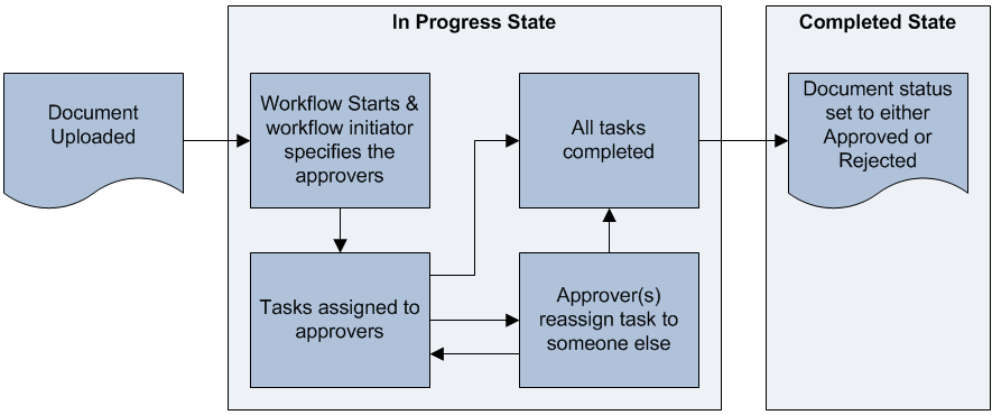


Figure 2.23 The approval workflow uses tasks to track the approval of a document or list item.

2.4.2 Collect Feedback Workflow

The Collect Feedback workflow (Figure 2.24) is a highly efficient and manageable way for individuals to have their items reviewed by their team members. When the workflow starts, the initiator of the workflow first needs to specify who they want to provide feedback. They enter the names of those individuals, and if they want the feedback to be gathered all at the same time (in parallel) or one after another (sequentially). If they choose parallel, for example, a task will then be created for each reviewer and assigned to that reviewer. The reviewer can than edit the task, and from within the task, they can type in their feedback they want to give on the document. As feedback comes in, it gets logged into the workflow's history log, and the requestor of the feedback can then go to the log to see the thoughts the reviewer's left for them.

Similarly to the Approval workflow, a reviewer can reassign the review to another person if they don't want to provide the feedback. The task will then be reassigned to that person, and the workflow will no longer be waiting for the original reviewer to leave their feedback.

As the reviewer notices things in the document that ought to be changed or updated, they can "Request a Change" to be made on that document. When the reviewer starts the request for the change, they need to specify the individual who will actually to the work and make the update. This individual will get a separate task stating the work that needs to be done, and after it's completed the requestor then gets a new task to review the change that was made. If they are agreeable to the work that was done, they can then complete that task and leave their feedback into the original workflow. Figure 2.24 shows the full feedback request process, along with these alternate paths. After all the feedback has been received across all the reviewers, the workflow will be completed.

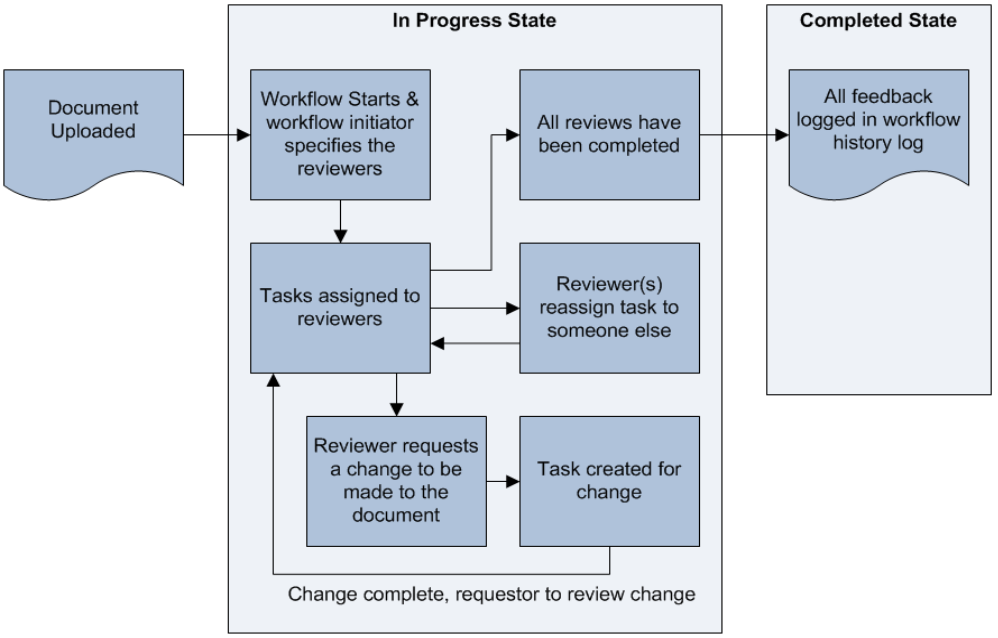


Figure 2.24 The Collect Feedback workflow assigns tasks to people who are requested to provide feedback on a document or list item. The person leaves their feedback in the task, which the workflow then compiles for the requestor to read over.

2.4.3 Collect Signatures Workflow

The Collect Signatures workflow (Figure 2.25) helps facilitate the gathering of digital signatures on Office Word documents. Consider a contract document that needs to be signed by say 4 individuals. Rather than emailing the document around and having someone add their signature manually, this workflow will add the signature to the document through a task.

When the workflow is started, the initiator will specify what individuals need to sign the document. Each signer then gets a task and email informing that they need to sign the document. When they open the task, at the bottom of the task there is a button titled "Sign". After they click the Sign button, the workflow adds their digital signature to the document. After all signatures have been placed on the document, the workflow completes.

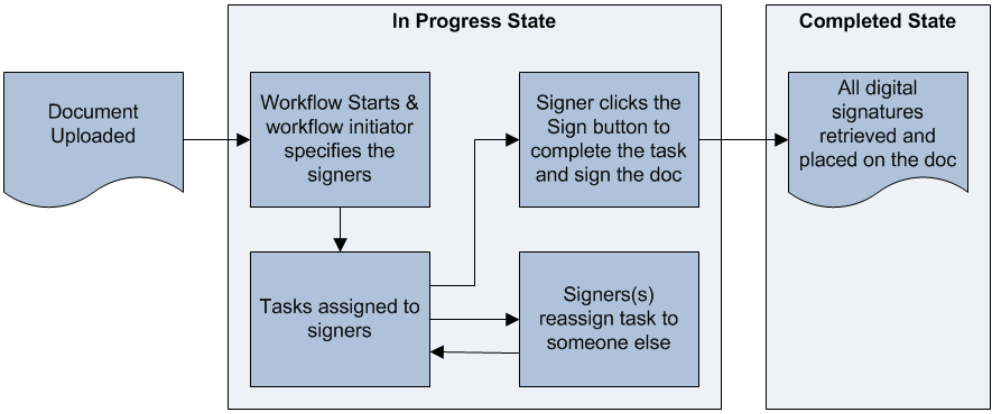


Figure 2.25 The Collect Signatures workflow helps facilitate the gathering of digital signatures to place on a document. The workflow assigns tasks to the signers, who then sign the document simply by clicking the Sign button within the task.

2.4.4 Disposition Approval Workflow

The disposition approval workflow (Figure 2.26) helps facilitate the deletion of expiring documents. Oftentimes this workflow works in conjunction with Records Center, a SharePoint add-in that manages company records and compliance. Depending on the requirements, a document may need to expire after a specified amount of time, and when that document expires this disposition workflow can be used to verify the document's deletion.

After the workflow starts, it will add a task into a task list. A user can accept or reject the request to delete the document by clicking either the *Delete this Document* or *Do not Delete this Document* link, respectively. If the document is deleted, the approver can optionally retain the document's metadata in the workflow's history log.

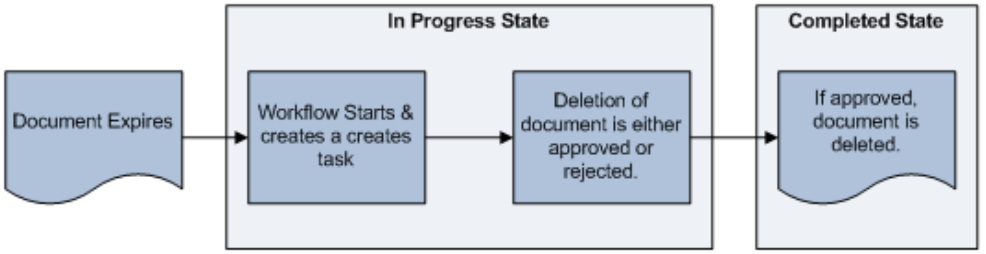


Figure 2.26 Documents can be set to automatically expire, and the Disposition Approval workflow helps facilitate the approval to delete these expiring documents.

2.4.5 Translation Management Workflow

One of the more unknown and less used out of box workflows is the Translation Management workflow (Figure 2.27). Organizations, in particular those with a global presence, can immediately see the

benefit of instituting this type of workflow to help facilitate the translation of documents from one language to another.

In this workflow, there are two major components that need to be created prior to the workflow being enabled. Documents needing translation must be stored in a Translation Management Library. Also, in complement to the translation management library, a Translation List is needed to store the people that actually do the translation of the documents from, say English to German. These two items are always present with the translation management workflow.

When a document needing translation enters the document library, the workflow begins by creating a copy of this source document. It will create as many copies as needed, based upon the number of translators associated to the document (ie. each translator will receive their own copy of the source document that must be translated into their corresponding language). The workflow then sets the appropriate language type to the document (based upon the specific translator) and creates a unique relationship between the source document and each of the corresponding copies. Each translator is sent an email with their corresponding tasks. The workflow is completed when ALL translators have marked their tasks complete (signifying they have finished translating the document), or if the source document was changed in any way.

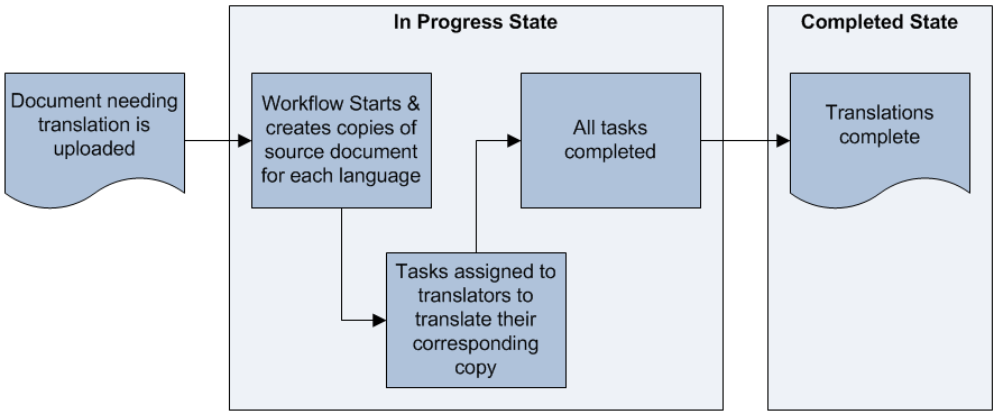


Figure 2.27 The Translation Management workflow helps with the translation of a document from one language to another. Tasks are assigned to the translators, and after all the translators translate their copy of the document, the workflow completes.

2.5 Summary

With SharePoint you can configure and setup some pretty robust workflows without involving anyone from the technology or support group. SharePoint has done an excellent job of putting the power of the platform into the end users hands allowing the IT department to focus on other areas instead of having to be involved with day to day processes like simple workflows.

This chapter introduced the concept of using a workflow to automate an everyday business process. The workflow was setup through the out-of-the-box features provided within SharePoint. The idea of a business user being able to take a basic napkin sketch and turn it into a full blown workflow within SharePoint is much easier to do than ever before.

The Three State workflow is one of the easiest out of box workflows to setup. Setting this workflow up involves creating a few columns on a list or library, adding the Three State workflow to that library, and starting the workflow on a list item or document. After the workflow has be started you can then do interesting things like viewing the workflow's status page which contains neat information such as tasks that are associated with the workflow, as well as the workflow's history. You can also terminate the workflow from this page as well.

The Three State workflow is just the first of many out of box workflows available in SharePoint 2010. There are 5 other out-of-the-box workflows available to those with a SharePoint Server license.

3

Custom SharePoint Designer Workflows

SharePoint Designer (SPD) is a powerful tool that you can use to help extend the out of box features in SharePoint. You may want to use SPD because you have some unique business needs such as implementing a custom branded look and feel with your company's logos, colors/fonts, etc on your SharePoint sites. Branding is a common use for SPD. SPD does a lot of other things too, such as building custom web parts, modifying page layouts, and you guessed it, building custom workflows.

SPD has a strong capacity to build custom workflows that can even meet very complex business needs and processes. SPD leverages the workflow architecture that SharePoint is built on by generating XOML, declarative workflow markup language. SPD publishes this XOML into SharePoint which then is interpreted by the Windows Workflow Foundation architecture, and your business processes start coming to life! SharePoint Designer workflows are often called declarative workflows, whereas Visual Studio workflows are called compiled workflows.

This all sounds complex, but it's actually the greatest benefit SPD brings to the table - namely, that non-technical people can "design" workflows that are then translated into what the platform needs to run them. Ease of use and speed are just two of the great benefits of SPD workflows.

SPD workflows are made up of several types of components, including Steps, Actions, Conditions, and others. These components are the building blocks of all SPD workflows. Steps organize the flow of the workflow from one "step" to another. Within steps you can have Conditions that check values before proceeding, for example. And Actions actually do "work" like sending an email, or creating a list item. This chapter will feature an example that puts all of these components to use by managing the approval process of Paid-Time-Off (PTO) requests from employees.

3.1 Introduction to SharePoint Designer Workflows

As you've heard in the brief intro, SPD is a great tool to use for customizing your SharePoint sites. In particular, SPD has a powerful workflow engine that you can use to model, add logic, and publish workflows into SharePoint that your users can interact with.

With SharePoint Designer 2007, we really only had one type of workflow available to us, a list workflow. In SharePoint Designer 2010, we have a lot more options, and a lot more flexibility. This chapter will only cover the List Workflow in detail, regardless of what type of workflow you are creating, the process for creating them and the tool set remains the same. All of the different types of workflows that are available in SharePoint Designer 2010 are summarized here.

LIST WORKFLOWS

List workflows are similar to the workflows that were available in SharePoint Designer 2007. They are created to work with the data in a specific SharePoint list and are commonly used for one-off scenarios where you need to run a specific set of actions against a single list. List workflows can also access data in other lists, as you'll see in the example at the end of the chapter. List workflows cannot be easily copied and used with other SharePoint lists, but they are easy to create.

SITE WORKFLOWS

Site workflows are not tied to a single list, instead they are created at the site level and can perform operations on data from any list within the site. Site workflows are useful when you need to run processes that involve multiple lists. For example, a site workflow could analyze all of the data created in a site within a date range, regardless of which lists the data was created on. Since they are not tied to a specific list, site workflows cannot be started automatically when a list item is created or modified the way that way other workflows can be. Instead, they must me started manually by a site administrator from within View all Content.

REUSABLE WORKFLOWS

Reusable workflows are not associated with a specific list, but are instead associated with a site. Once created, they can be added to any type of list by using the List Settings screen, which is the same way that the default Approval and Collect signatures workflows are added to a list. Because they are re-usable, they can be added to as many lists as necessary within the site. If the default workflow, such as Approval and Collect Signatures, do not meet your needs, you could create your own versions of these are reusable workflows and tailor them to your needs.

GLOBALLY RE-USABLE WORKFLOWS

Globally re-usable workflows are similar to Reusable Workflows, except that they are available to the entire site collection, not just a single site. They can be added to any list within the site collection, while a regular reusable workflow is only available to lists within the site it was created for. Globally reusable workflows would be handy if you need to implement a corporate-wide process and needed all sites in your site collection to use the same workflow to complete this process.

WORKFLOW TEMPLATES

After creating a workflow you can save it as a template, which is actually a SharePoint WSP file. The workflow can then be deployed to another SharePoint site or Farm. Previously this was only possible when working with Visual Studio workflows. For a walkthrough on how to work with templates, reference chapter 5 section 1.

Anyone who has worked with the previous version of SPD will notice changes to the interface as soon as they open SharePoint Designer 2010. Microsoft has now adopted the same interface across all of its Office 2010 products. This interface includes the new File tab, which replaces the Office Orb and is used to connect to and open sites. It also includes the Ribbon, which is used to access most of the tools commands. Finally, tabs are used to access open files and display their contents.

The first thing you see when you open SharePoint Designer 2010 is the File tab, which replaces the File menu and Office Orb familiar to users of older Microsoft Office products. The File tab, shown in Figure 3.1, includes tools for opening SharePoint sites and creating new sites.

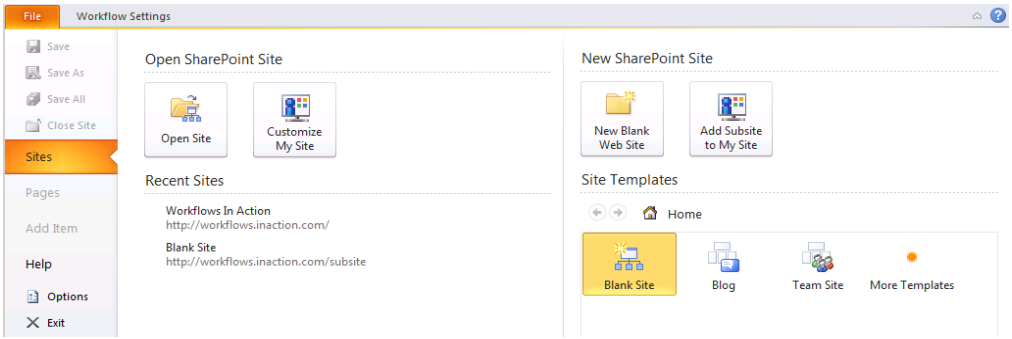


Figure 3.28 - The same interface is used in all Office 2010 products, including SharePoint Designer 2010. The new interface is designed to make it easier to find the available tools and options.

The editing interface, which is displayed after connecting to a SharePoint site, includes three major areas, the Office Ribbon, Site Objects on the Left, and the Home Tab. See Figure 3.2 for an example of the editing interface. The Office Ribbon provides access to the functions needed to work with SharePoint sites, and for the purposes of this book, SharePoint workflows. The Site Objects area is used to access the different areas of a SharePoint site. Finally, the Home Tab is where you will create and edit lists and workflows.

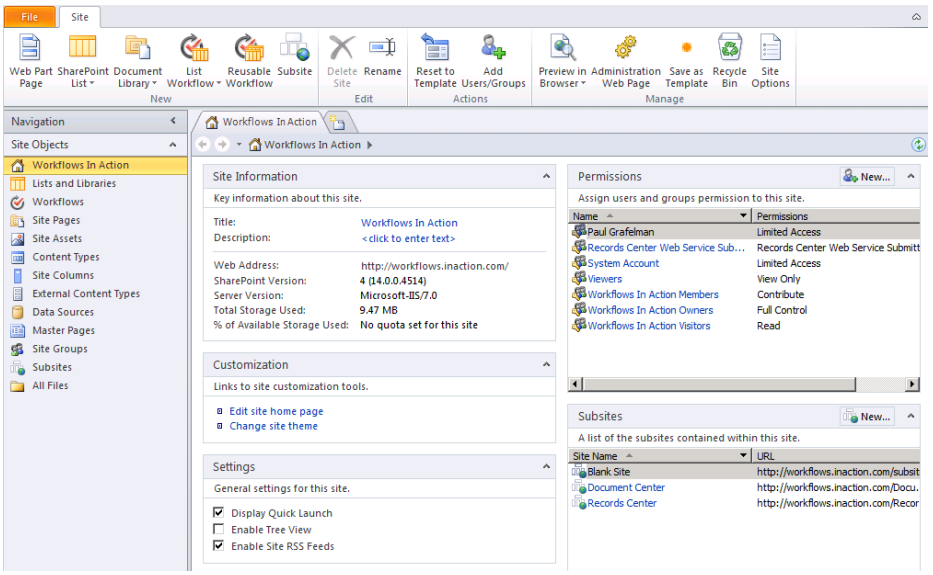


Figure 3.29 The editing interface changes based on what area of Designer you are working in. When

creating workflows, for example, the interface will show you information relative only to workflows.

The Ribbon (Figure 3.3) is used to access tools and functions that make up SharePoint sites and workflows. The options shown on the ribbon will change depending on what you are working on. For example, if you are editing a list, options specific to lists will be show. If you are instead editing a workflow, only options specific to workflows will be shown. The dynamic nature of the Ribbon means that you will spend less time looking for the options and tools you need and more time creating functionality.

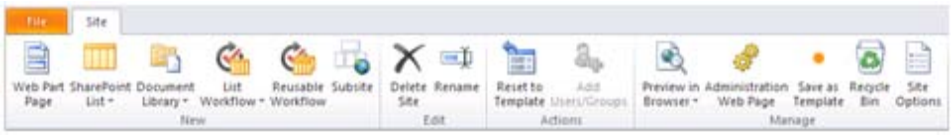


Figure 3.30 The Office Ribbon also changes depending on what you are editing. The buttons, and in some cases the tabs, will adjust to show you only the relevant options.

The Site Objects pane (Figure 3.4) allows you to access all of the different components of a SharePoint 2010 site. This is where you will go to create new pages, lists, and content types. It is also how you access existing workflows and create new ones. Clicking the different object types under Site Objects will allow you to work with the different object types. As you click the different object types, notice how the Ribbon and Home Tab changes to show you relevant options.

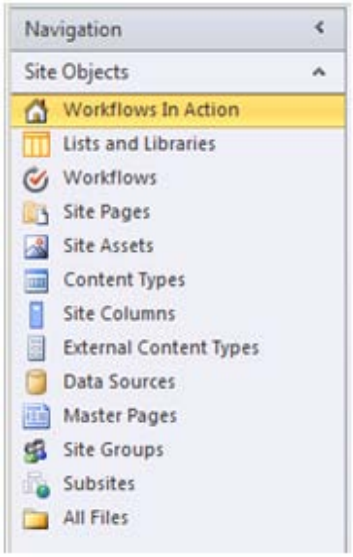


Figure 3.31 The Site Objects panel lists all components that make up a SharePoint 2010 site. For this book we are primarily interested in the Workflows object, but will also look at the Lists and Libraries object.

The Home tab, and later other tabs, is where you make settings changes and edit SharePoint objects. Figure 3.5 shows the Home tab which is displayed when you first open a SharePoint site. The content of this tab changes depending on what Site Objects section you have opened and what type of objects you are editing. For example, when you first connect to a site you can change the name of the site and modify security on the Home tab. When editing a workflow you can adjust how the workflow is started and develop the workflow, all within the same tab. As you open and work with objects, they will appear as additional tabs at the top of the content area, allowing you to quickly switch between them.

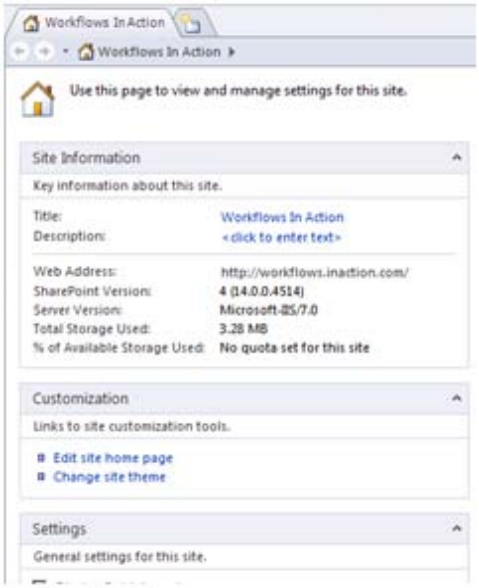


Figure 3.32 The Home Tab is displayed when you first connect to a site with SharePoint Designer. It allows you to view and adjust settings that affect the entire SharePoint site.

3.2 Components of a SharePoint Designer Workflow

SPD workflows are comprised of many components, including Steps, Conditions, and Actions. Similar to writing code in a traditional programming language, these components are used to control the workflow, the actions it takes, and its results. Fortunately for non-developers, all of the components are easy to add and configure with the built in wizards and the point and click interface. In addition, all of these components are written using plain English statements like "Calculate 100 plus 50", which make them very easy to configure. You will see some at work in the examples at the end of the chapter.

The components of a workflow fall into 6 functional categories:

- **Steps**: help you organize the workflow process
- **Conditions**: add decision making logic into the workflow
- **Actions**: make the workflow do something
- **Variables**: store data temporarily

- **Else-If Conditions**: add additional decision making
- **Forms**: allow the user to provide data to the workflow

3.2.1 Steps

Steps are the foundation of any workflow and allow you to organize the workflow into logical sections. The first step typically includes the preparation of data that will be needed later on in the workflow. This could include preparing variables, which are used to store data temporarily while the workflow is running, or collecting data from a user. Subsequent steps are used to work with the data and process commands. This is where the bulk of the work is done in the workflow, such as performing calculations and waiting for users to make changes to data. The last step is usually used to record the results of the workflow and send notifications to user or administrators if required. This might include sending an e-mail to indicate the conclusion of the workflow or logging error message for troubleshooting. Although it is possible to create a large workflow with only one step, breaking it into multiple steps can make it easier to understand and modify later. This is especially important when someone other than the original author is making modifications.

It is also important to name your steps to improve readability of the workflow. Steps can be named and renamed by simply clicking the existing titles. In addition steps can be moved around within the workflow using the Move Up and Move Down buttons, shown in Figure 3.6. In Figure 3.7 the Finish Step has been placed before the Start Step using this method. Although illogical, it demonstrates how you can re-arrange the steps in your workflows at any time in the development process.

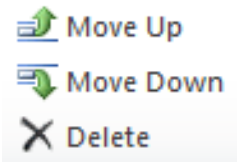


Figure 3.33 The Move Up and Down Ribbon buttons are used to move workflow steps, actions, and other components, while the Delete button will delete the currently selected item. Note that there is no 'Undo' when working with Designer workflows, so use Delete carefully.

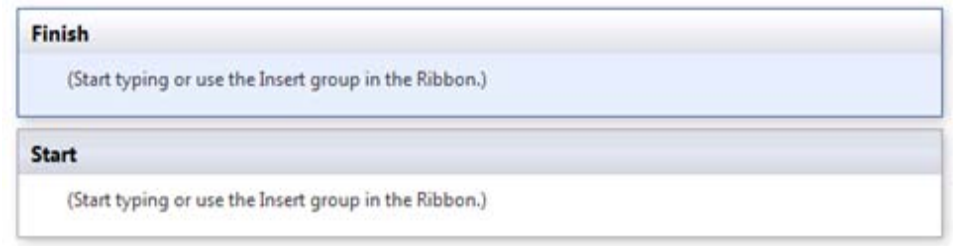


Figure 3.34 Steps can be moved into any order using the Move Up and Move Down Ribbon buttons. In this image the Finish step has been illogically moved before the Start step.

Similarly, if you need to delete a step, click the header of the step to select it and click the delete button on the ribbon or hit the Delete key on your keyboard. Figure 3.6 shows the Delete button. Keep in mind that there is no Undo option, so be sure that you want to delete the step.

Nested steps, like those in Figure 3.8 are also possible. These allow you to organize your workflows into major steps and minor sub steps. As your workflows become more complex, nested steps will allow you to move entire sets of commands at one time, making development and modification of your workflows easier.

Figure 3.35 Steps can be nested, or placed within other steps, if required. in this example you can see that Steps 4, 5, and 6 are all nested within Step 1. This will allow you to organize complex workflows into logical sections.

Another layer on top of steps is the concept of Parallel blocks. Parallel Blocks are similar to Steps, except that they allow you to run multiple actions consecutively. For a complex workflow with many steps that are not necessarily dependent on each other, parallel blocks will allow you to complete the workflow, or at least progress to the next non-parallel actions, much faster. Note that Parallel blocks can only be created within Workflow Steps; they cannot be placed outside of the workflow steps. You can place steps within the Parallel Block, allowing you to run entire sets of actions in parallel (Figure 3.9).

Figure 3.36 With Parallel blocks, your steps can run in parallel to each other, rather than in sequence.

3.2.2 Conditions

Conditions allow the workflow to make decisions based on the data that is provided by SharePoint lists or end users. They will allow you to write workflows that can respond differently based on the data that end users are entering into the system. The example at the end of this chapter shows you how to perform certain process if a manager approves an item and take an entirely different set of actions if the item is rejected. Multiple steps can be included within the condition when you need to perform a series of commands.

The following conditions are available in SPD 2010. Some are only available on certain types of workflows. Site Workflows, for example, are missing some conditions because they are not tied to a specific list.

COMMON CONDITIONS

These conditions will likely be the ones you use most.

- **If any value equals value:** Allows you to compare a value from any data source to a value from any other data source, as in Figure 3.10. These data sources can include data from SharePoint lists, variables, or static values.

Figure 3.37 - This condition is checking to see if a item was modified 'Today'. Today always indicates the current date. If the item was modified Today, the actions below the If will be executed.

- **If field equals value:** Allows you to compare the value of a field on the current item with the value from any other data source. This condition is not as flexible as the previous condition, since it must compare data from the item the workflow was started on. Because it must be used with a

list item, this condition is not available when creating a site workflow.

OTHER CONDITIONS

Many of the comparisons allowed by the following conditions can be created using the two previous conditions. Because these are configured for a specific purpose, they are generally faster and easier to configure. But because they are fairly specific, you will likely not use them as often.

- **Created by a specific person:** Check to see if the current item was created by a specific person
- **Created in a specific date span:** Check to see if the current item was created within a specific data range
- **Modified by a specific person:** Check to see if the current item was modified by a specific person
- **Modified in a specific data span:** Check to see if the current item was modified within a specific data range
- **Person is a valid SharePoint user:** Checks to see if a specified user is recognized by SharePoint
- **Title field contains keyword:** Checks for a specified string within the Title field of the current item

Conditions can be configured to examine workflow data, which is any type of data that can be used by the workflow, including data from list items, variables, or other data sources. This workflow data can be compared to static information entered when the workflow is written or other pieces of workflow data.

For some conditions the type of comparison, also known as the operator (Figure 3.10), can be changed to allow for other types of comparisons. By default the operator is set to 'equals', but you can change this to a variety of options by clicking on the operator currently in use and selecting a different operator.

The list of available operators changes based on the type of data you are comparing. For example when comparing a value to a Date/Time field, the 'begins with' and 'ends with' operators are not available. The following operators are available in SPD Workflows:

- **Equals:** Check if two values are exactly equal, works all data types. The condition is case sensitive.
- 'Summer' equals 'Summer = True
- 'Summer' equals 'summer = False (Note the lowercase 'S')
- **Not equals:** Check if two values are not equal, works with all data types.
- 'Summer not equals 'Winter' = True
- **Is empty:** Check if the referenced field is blank
- **Is not empty:** Check if the referenced field is not blank
- **Begins with:** Check if the referenced field starts with a value.
- 'Summer' begins with 'Sum' = True

- **Does not begin with:** Checks if the first characters of a string match another string.
- 'Summer' begins with 'Wint' = False
- **Ends with:** Similar to Begins with, but checks the end of the value instead of the beginning.
- 'Summer' ends with 'er' = True
- **Does not end with:** Checks that the string does not end with the characters from another string.
- 'Summer' does not end with 'ter' = True
- **Contains:** Checks to see if a string contains a second string.
- 'Summer' contains 'int' = False
- **Does not contain: Checks if a string does not exist within another string.**
- 'Summer' does not contain 'int' = True
- **Matches regular expression:** Checks to see if a string matches a defined format. The format to match is defined using character sequences to match letters and numbers.
- **Equals (ignoring case):** Same as Equals, but does not compare the case of the characters in the strings.
- 'Summer' equals (ignoring case) 'summer' = True
- **Contains (ignoring case):** Same as Contains, but does not compare the case of the characters in the strings.
- 'Summer' contains (ignoring case) 'MME' = True

When in doubt, use the Any Value equals Value comparison, as it provides the most flexibility and allows you to compare the largest variety of data.

3.2.3 Actions

So far, we have only looked at telling a workflow how to make decisions. A workflow that only makes decisions and doesn't perform any actions is not very useful. To make our workflows useful, we need to add Actions. There are many workflow actions available in SharePoint Designer 2010, and each is designed to perform a different function. The available actions are categorized into six groups, plus a seventh group that includes recently used actions. To add actions to a workflow use the Action Ribbon button, shown in Figure 3.11.

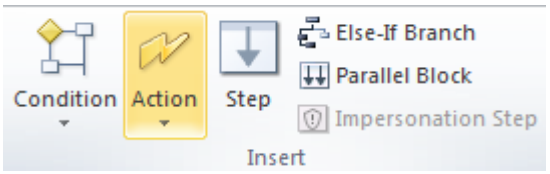


Figure 3.38 The Insert Action Ribbon button is actually a dropdown menu. After clicking it you can select from all of the available actions.

Most of these actions are self explanatory, simply by the Action's title. A few of the more complex actions are covered in the examples later on in this chapter. Here is a summarized list of the most popular actions:

- **Core Actions:**
- Add a Comment
- Do Calculation
- Log to History List
- Pause for Duration
- Pause until Date
- Send an Email
- Set Workflow Variable
- Stop Workflow
- **List Actions**
- Check In Item
- Check Out Item
- Copy List Item
- Create List Item
- Delete Item
- Discard Check Out Item
- Set Field in Current Item
- Update List Item
- **Task Actions**
- Assign a To-Do Item
- Collect Data From a User
- Start Approval Process
- **Utility Actions**
- Extract Substring from End of String
- Extract Substring from Index of String
- Extract Substring from Start of String
- Extract Substring from Index of String with Length
- Find Interval Between Dates

You can see that by default, SharePoint Designer 2010 allows you to perform many different types of actions. This is one area where it greatly exceeds what was available with SharePoint Designer 2007. The Core and List actions will be used more often than most since they allow you interact with

SharePoint data similarly to how an end user interacts with the same data. The most important action, is probably the Set Workflow Variable actions, since it musts be used to capture data from external sources before you can perform more complex actions, such as Do Calculation.

3.2.4 Variables

As mentioned earlier, variables are used to store and manipulate data while the workflow is running. Each time a new workflow instance runs, a new set of variables is used, even if multiple copies of the workflow have been initiated by multiple users. Also, once a workflow is finished, the data in the variables is lost, unless you first store that data to a field in a SharePoint List item or preserve the data in some other way, such as including it in an e-mail message.

When you first create a workflow, it does not have any variables. In some cases adding actions to your workflow will automatically create new a variable or variables. For example, adding a 'Do Calculation' action will automatically create a 'calc' variable. Adding a second 'Do Calculation' action will create a 'calc1' variable. All variables must have unique names, so SharePoint Designer appends a number to the end of automatically created variables to keep them unique. You can rename any variable whether it was manually or automatically created, to suit your needs.

Variables are set to only allow certain types of data, called data types. Some actions require certain data types to function correctly. The following data types are available when creating workflow variables:

- **String:** A text string of any length, 'ABC'
- **Boolean:** 'Yes' of 'No'
- **Date/Time:** '1/1/2010 12:00:00 AM'
- **Integer:** Number without any decimal value, '100'
- **List Item Id:** Unique Reference to a list item, '1'
- **Number:** Number with or without a decimal value, '1.5'

These are mostly self explanatory, with the exception of List Item Id. This data type allows you to store the unique identifier of a list item. This can be useful if you need to refer to a specific list item later in the workflow.

The actions in your workflow are used to assign values to variables. Some actions, as described above, do this automatically. In other cases, you will need to assign the values yourself using the Set Workflow Variable action. When the action itself assigns the value, you only need to choose which variable to assign the output to. Only compatible variables are shown, preventing you from selecting a variable with an incompatible data type.

When using the Set Workflow Value action, all available variables are shown, even if they are not directly compatible. In some cases, it is possible to assign an incompatible value to a variable by converting it, also called Coercion. To use coercion when assigning a value to a variable, adjust the Return String As dropdown on the Lookup dialog, shown in Figure 3.12.

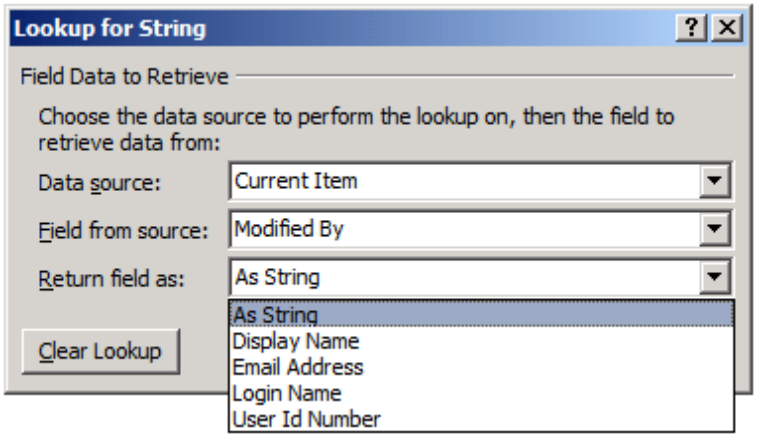


Figure 3.39 Using Coercion allows you to return the value of a field as a different type of data. In this example the Modified By field is being returned as a String data type. By default it would be returned as a Person data type.

The options available in the Return Field As dropdown are dependent on the data type of the variable and the data type of the field you have selected in the Field from source dropdown. For some field from source/variable data type combinations coercion is not possible and the return field as dropdown will be set to the only possible type and disabled. When coercion is possible, the dropdown will be populated with possible coercion types. In Figure 3.12 the variable has a String data type, and the 'Field From Source' has a User ID variable type. You can see that it is possible to apply several different user profile properties to the variable. Due to the number of possible combinations, the best way to see what coercion options are available in your workflows is to experiment.

Not all data types can be coerced into all other data types, even when those data types are listed in the Return Field As dropdown. You cannot coerce a String into a List Item Id, for example. The Set Workflow Value action does not prevent you from attempting an invalid coercion, but when you run the workflow you will receive an error in the workflow history, as in Figure 3.13. If this occurs, you will need to adjust either the type of coercion you are performing or the data type of the input being assigned to the variable.

Figure 3.40 A Coercion error logged in the workflow history indicates an invalid coercion selection was attempted. If this happens you'll need to adjust to coercion to return a valid type.

3.2.5 Else-If Branches

Else-If Branches work directly with Conditions to allow for more advanced decision making. They can be set to execute whenever the preceding condition is false (similar to an 'Else' statement). To do this, all you need to do is place the cursor inside the existing Condition and click the 'Else-If Branch' button on

the Ribbon. No additional configuration is necessary to use the Else-If in its simplest form. Figure 3.14 shows an Else-If added to a condition.

Figure 3.41 An Else-If branch added to a condition allows the workflow to perform specific processes whether the result of the condition is True or False. An Else without an additional condition will always be executed if the preceding If is false.

If an additional comparison is needed within the Else-If Branch, you can add another condition, again using the condition button on the Ribbon. You can choose any type of available conditions, it does not need to be the same as the initial condition. Figure 3.15 shows an Else-If with an additional condition.

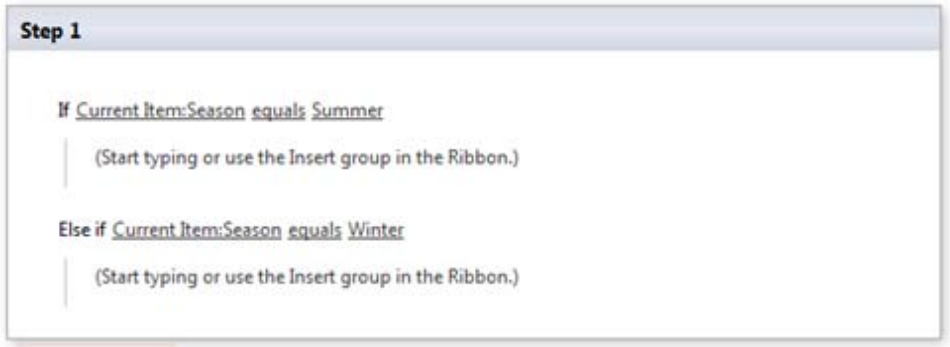


Figure 3.42 An Else-If branch with a second condition allows multiple values to be compared or the same value to be compared in multiple ways. In the example above a Season value of Spring would mean that neither the If actions nor the Else-If actions would execute.

Finally, you can compare multiple pieces of data within a single condition using 'And' conditions and 'Or' operators. This is done by adding another condition immediately after the existing conditions and changing the operator as necessary. SharePoint Designer 2010 will combine these into a single statement automatically. And operators and Or operators can be added to the primary If statement or to any Else-If branches. An example of this is shown in Figure 3.16.

Figure 3.43 And and Or statements allow for multiple comparisons within one condition. This will allow you to create a single complex If statement instead of several simple If statements.

3.2.6 Workflow Forms

One way in which users can interact with workflows is through forms. Forms are pages viewed through the browser and allow the user to supply additional data to a workflow. The easiest way to use forms is to add initiation parameters to your workflow. Initiation parameters are similar to variables, except that they are displayed to the user when the workflow is started manually. The user can then choose what data to enter into the parameters. Variables, on the other hand, use data from the lists and do not allow users to directly interact with them. Forms are covered in detail in chapters 8 and 9 of this book.

3.3 Creating your first SharePoint Designer Workflow

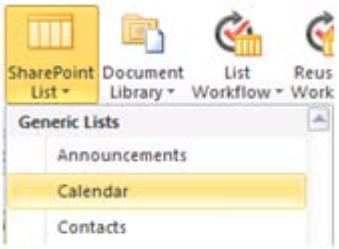
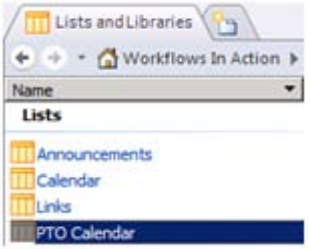
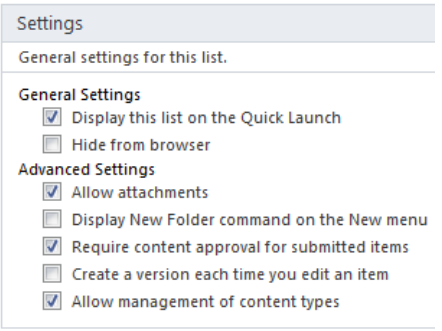
Now that you have learned about the core components of a SPD workflow, it's time to get some hands on experience. We are going to create a Paid Time-Off (PTO) request system using SharePoint lists and workflows. The example is divided into several sections, with each section focusing on different workflow components of the workflow. Also, each section builds on the previous sections, so make sure to go through the example in order.

The first section will concentrate on building the lists necessary to support the PTO workflow. In the second section we'll create a simple workflow that does some logging. Thereafter we will add some basic functionality that handles notifications when a PTO request is submitted, and lastly we will add advanced functionality that tracks the available hours of PTO.

3.3.1 Configuring a PTO Calendar for the Example Workflow

The example of a PTO Request calendar will allow employees to go to a calendar and create a new calendar item that spans the length of their requested PTO. The workflow that will be built in the next section will route that request to the employee's manager. But before we build that workflow we first need to create the calendar, and setup content approval. Follow these steps to create this PTO calendar list:

Setting up your PTO Request Calendar List

	Action	Result
1	Create a new Calendar list from within SharePoint Designer: A) Open SharePoint Designer from the Desktop or Start menu and connect to your SharePoint site using the Open Site button. B) Click the Lists and Libraries button from within Site Objects and thereafter Click the SharePoint List (figure below) ribbon button and select the Calendar List  C) Name the new calendar 'PTO Calendar' and click OK	A new Calendar will be created.
2	Setup content approval on the PTO request calendar: A) Click the 'PTO Calendar' link under Lists  Note: After creating a new list, it will appear on the Lists and Libraries tab. Opening the list from here does not give you access to the data within the list, but allows you to adjust the lists	 After opening the list you can adjust a variety of list aspects. The Settings section allows you to enable Content Approval, which is needed for the PTO workflow

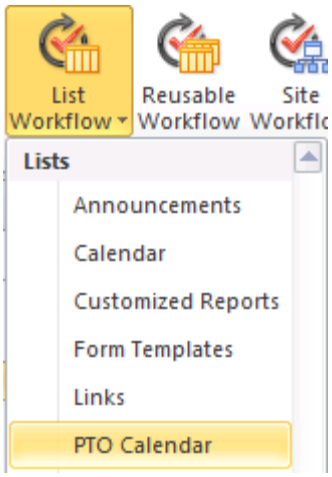
settings. B) Under Settings, check the box for 'Require content approval for submitted items' and then save your work by clicking the save button.	

Now with our PTO Calendar setup, out employees can start submitting their PTO requests. The only thing now is the build the workflow that helps facilitate their approval!

3.3.2 Creating a new Custom Workflow that logs to the History List

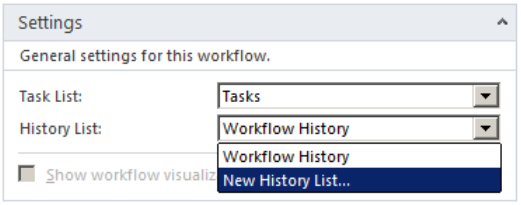
With our Calendar ready to go, we can now build the PTO request workflow. Before we add a lot of logic into the workflow, let's first build a simple workflow that does some logging, and publish that workflow to the calendar to make sure everything is deploying properly. Follow these steps to create a basic SPD workflow, and publish that workflow to the PTO calendar:

Create a new workflow that logs to the workflow history list

	Action	Result
1	Create a new List Workflow off the PTO Calendar: A) Connect to your SharePoint site with SPD, and click the "Workflows" Site Objects button on the left B) Click the List Workflow button on the Ribbon and choose the PTO Calendar:  C) Name the new workflow "PTO Workflow" and click OK.	A new List Workflow will be bound to the PTO calendar, after it's published. At this point the workflow will be empty.
2	Setup a dedicated history list that you can use to log workflow information to:	A new list will be created that

A) In the breadcrumb (Figure 3.17), click the 'PTO Workflow' link to access the workflow settings.

B) Select New History List for History List. Click OK on any pop up dialogs. After you click OK, a new history list called PTO History List is created automatically.



Note: Instead of creating a new History List we could use the pre-defined history list. In some cases this is OK, but in other cases it is helpful to log each workflows history to a dedicated list. This is especially true for complex workflows that are generating a large number of log messages. Having the history logged to a dedicated history list can make troubleshooting easier since you will not be sifting through the log message of multiple workflows. For simple workflows, or workflows that do not have a lot of log messages, a shared history list is fine.

the workflow can log to.

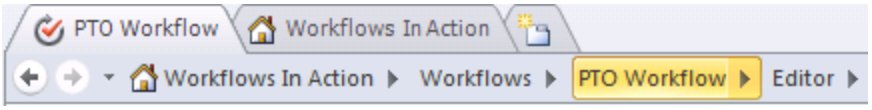
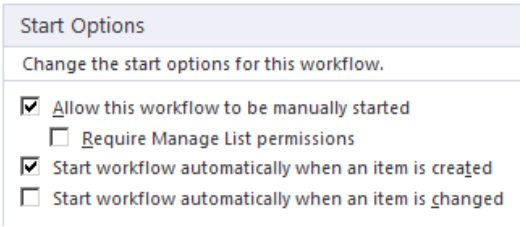
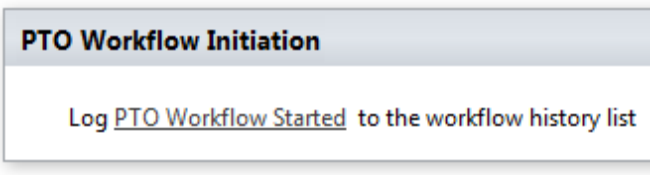
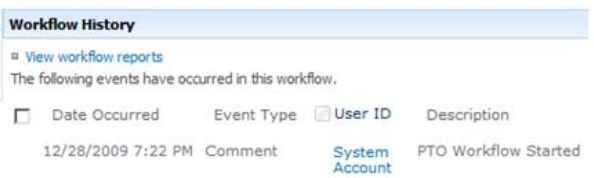


Figure 3.44 As you dig deeper into the workflow settings and editor you will need to use the Breadcrumbs displayed beneath the tabs. The breadcrumbs allow to not only see exactly where you are, but they allow you to move to different levels of the workflow editing process.

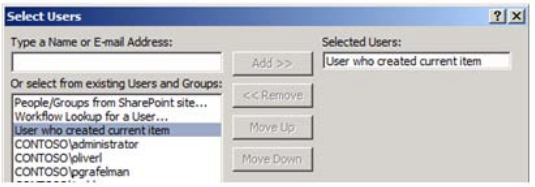
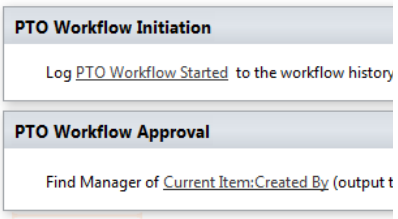
	Action	Result
3	Specify that the workflow should start automatically when a new PTO request is created: A) Still in the PTO Workflow tab via the breadcrumb (Figure 3.17), under Start Options, check 'start workflow automatically when a new item is created.	
4	Add a logging Action to log that the workflow has started: A) Click the Edit Workflow link under Customization. B) Click 'Step 1''s title and rename the step to 'PTO Workflow Initiation'. C) Click inside the step, and then click the action drop down (in the ribbon) to show the list of actions. Click Log to History List action to and change the message 'PTO Workflow Started.	A single action will be in the workflow that logs to the history list. After published, the workflow is bound to the PTO Calendar and will fire when new requests are created. 
5	Click the Publish button in the Ribbon and test the workflow: A) Click the Publish button and then open the SharePoint site using Internet Explorer and browse to the PTO Calendar. B) Create a new calendar item. Thereafter, verify the workflow ran successfully. C) Navigate to the workflow settings page, and verify the workflow's history was populated from our logging Action.	 If the workflow successfully ran, and the 'PTO Workflow Started' message was logged to the History List, you are ready to move on to the second example. If not, re-check your steps and ensure that everything was followed exactly

We now have a very simple workflow deployed onto our PTO Calendar. The workflow isn't too terribly powerful yet. In fact, all it does is logs to the history log. Now it's time to start sending emails when PTO requests are submitted.

3.3.3 Adding Notifications into the custom PTO Request Workflow

Now that the workflow is running automatically and logging successfully, it's time to add some more advanced functionality to the workflow. We'll add some conditions and actions to notify the user's manager that a PTO request was submitted. We'll also add some logic and actions to notify the user whether or not their request was approved or rejected. Finally, we'll add some error handling in case something goes wrong with the approval. The actual approval of the PTO request will be done through the out of box content approval features. Follow these steps to add this managerial approval to the PTO workflow.

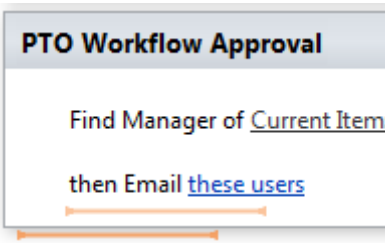
Add managerial approval notifications to the PTO Workflow

	Action	Result
1	Add a manager lookup action to the workflow: A) Add a step after the 'PTO Workflow Initiation Step' and name it 'PTO Workflow Approval' B) Within this new step, add a 'Lookup Manager of a User' action using the Actions button on the Ribbon. Note: the lookup manager action is under the "Relational Actions" category. C) Set the action to lookup the manager of the user who created the 'PTO Calendar' item by clicking the 'this user' link, and then click OK.  The Select Users dialog box allows you to select a dynamic user, such as the User who created the current item, or a specific user. You can also perform a looked based on workflow data.	

2 Add a Send Email Action:

Note: If your environment does not have outgoing e-mail configured or you cannot use e-mail for any reason, use a Log to History action instead of the Send an Email Action. Be sure to configure the Log message to include references to the same data, such as Manager and Created By, to ensure that these components are working.

A) Add a 'Send an email' action to the step after the Manager Lookup action.



3 Specify to send the email to the manager:

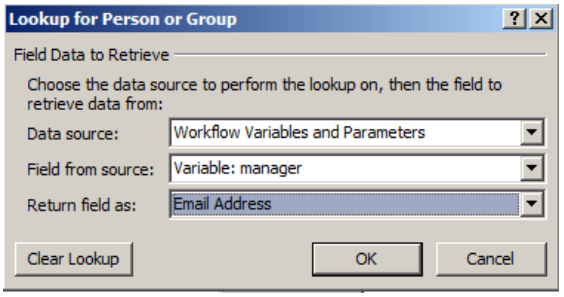
A) Click the 'these users' link on the email action to open the email configuration dialog.

B) Click the address book icon next to the 'To:' field to select a user to send the email to.

C) Select 'Workflow Lookup for a User…' and click the Add button in the center of the dialog.

D) Change the Data source to 'Workflow variables and parameters' in order to select the 'manager' variable.

E) Select 'variable:manager' for the *Field from Source* then change the 'Return field as' drop down to be Email Address, and click OK twice.



The email will now be configured to send to the requestor's manager.

4 Configure the contents of the email message, including the subject and body of the email:

A) Still within the email configuration dialog box, click the

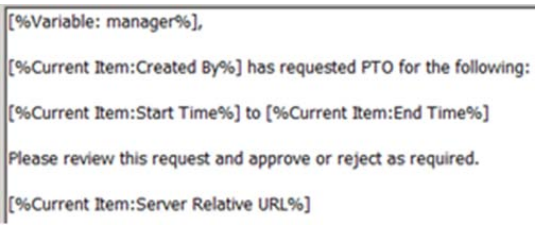
The email will now have a subject and body defined.

ellipsis for the Subject field to add subject line text.

B) Add text and a lookup for the 'Created By' field to the subject line using the Add or Change Lookup button:

Note on Subject: The Subject line of the email action can include workflow data. In this case the Created By field has been added to the subject line along with some static text.

C) Add text and a lookup to the body, including a link to the PTO Calendar item, again using the Add of Change lookup button:



Note on Body: Just line the Subject line, the email body can include static text and workflow data. Including all relevant data in the email will allow the end user to take action or make a decision without necessarily having to open the SharePoint item in question.

D) Click OK to close the email action editor and save your changes.

At this point, we want to pause the workflow until the user's manager either approves or rejects the PTO request. We will also need to check the Approval Status field on the item to determine how the manager responded to the request. Note that the values in the Approval Status field contain special characters along with the text, these are required when doing an 'equals' comparison. You could use a 'contains' comparison instead, but the equals comparison is more efficient because it does not need to examine the string characters one by one.

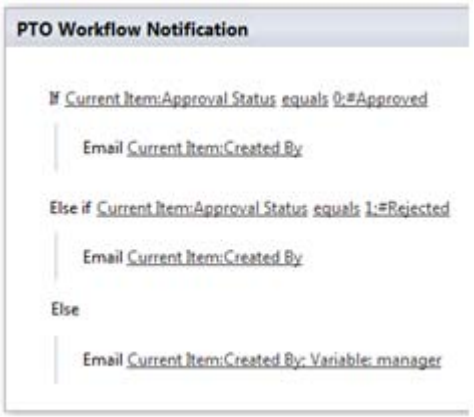
	Action	Result
5	Add a pause action to wait for the request to be approved or rejected: A) Add a 'Wait for field change in current item' to the PTO Workflow Approval step. B) Set the field to Approval Status, the operator to 'not equal' and the value to '2#;Pending'.	then Wait for Approval Status to not equal 2;#Pending Pausing a workflow: The Wait for field change in current item action allows you to suspend the operation of the workflow until certain criteria are met. In this case, the workflow will pause until the PTO request is either Approved or Rejected.
6	Add an If condition to check if the PTO request was approved: A) Add another step to the workflow and name it 'PTO Workflow Notification'. B) Add an 'If current item field equals value' condition to the new step. C) Set the field parameter to 'Approval Status' and the value to '0;#Approved'. D) Add a 'send an email action' to the If condition and configure it to notify the creating user that their request is approved.	PTO Workflow Notification If Current Item:Approval Status equals 0;#Approved Inside the if condition will be a send email action notifying the requestor that their request has been approved.

Again, if email is not available, use a Log to History instead of the Send an email action. Be sure to include all of the referenced fields in the log message to ensure that everything is working as designed. We now need to add an email notification that will go to the user that created the PTO request when the request is approved. We also need to create an Else-If condition to check for a rejected request and send an email accordingly.

	Action	Result
7	Add and Else-If branch to check if the PTO request was rejected: A) Add an Else-If branch to the condition using the "Else-If" button on the Ribbon. B) Add an 'If current item field equals	Else if Current Item:Approval Status equals 1;#Rejected Also, inside the else-if condition will be a send email action notifying the requestor that their request has been rejected.

value' condition to the Else-If branch. C) Set the field parameter to 'Approval Status' and the value to '1;#Rejected'. D) Add a 'send an email action' to the condition and configure it to notify the creating user that their request was rejected.	

If the Approval Status field is not equal to Approved or Rejected at this point, something went wrong with the workflow. Instead of just stopping the workflow, it is important to let the users know that something went wrong. In the real world, it would also be beneficial to notify whoever is responsible for maintaining the workflow that an error occurred. These notifications can be done by adding another Else-If with no conditions.

	Action	Result
8	Add another Else-If to notify both the manage and requestor that an error had occurred: A) Add another Else-If Branch to the end of the step. B) Add an 'Send an email' action to the final else-if branch. C) Configure it to notify both the user and the manager that an error occurred.	

Publish the workflow and create two test items on the PTO Calendar as your test user. The Manager of the test user will receive two e-mails, one for each PTO Calendar item. As the manager, approve one request and reject the other. The test user will receive two email notifications, one for each PTO request.

More options for managing Approvals:

There are multiple approaches to getting approval within workflows. This example uses the built in content approval functionality, but we instead could have used the out of box Approval workflow as well. In this scenario, the Manager would be assigned a new task in the task list instead of being sent an e-mail. The manager could then use this task to approve or reject the PTO request.

3.3.4 Adding Calculation Logic into the Workflow

The last step will be to configure a second list to track how many hours of PTO each employee has remaining. This list will be used to help the manager decide to approve or reject the request because it will keep track of cumulative and remaining PTO hours for each user. If the PTO request is approved, this new list will be updated to reflect the number of remaining and used PTO hours for the requestor.

To get started, let's first setup this PTO tracker list. In SharePoint Designer, click the Lists and Libraries site object menu. Click the Custom List Ribbon button to create a new custom list named "PTO Tracker". After the list is created, add a column of type Person or Group, and name the column "Employee".

Some list and column settings are not available when editing a list in SharePoint Designer and require you to return to SharePoint to make the changes. This is why the Administration Web Page button was added to the Ribbon. Click the 'Administration Web Page' Ribbon button to open the column settings in the SharePoint interface (Figure 3.18). Select the following options for the Employee column:

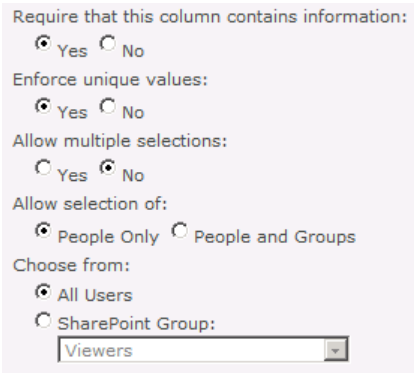


Figure 3.45 Because some list settings are not exposed in SharePoint Designer it is sometimes necessary to adjust the list settings using the SharePoint List Settings page.

By selecting the Enforce Unique Values option we are guaranteed to only find one record for each employee. This is important when doing lookups against this list because we need to ensure that we are finding the right employee's information. If there were two entries for the same person, the workflow would not know which item to update and your results would be unpredictable. The warning you will see later about ensuring unique values is indicative of this. Checking the *Ensure Unique Values* option for the column will prevent this problem (although the warning will still appear).

Click OK to save the changes to the Employee column. Switch back to SharePoint Designer and add two Number columns to the PTO Tracker list. One list should be named Available Hours, and the other column should be named Used Hours (Figure 3.19).

Column Name	Type
Title	Single line of text
Employee	Person or Group
Available Hours	Number (1, 1.0, 100)
Used Hours	Number (1, 1.0, 100)

Figure 3.46 List columns can be added to a list within SharePoint Designer. If adding a large number or columns to a list it is much faster to do so with Designer because you do not need to wait for page refreshes that occur when adding columns using the browser based settings page.

Changes to lists made in SharePoint Designer do not take effect until you click the Save button. The result is that any workflows you are editing will not be aware of newly added columns until the changes to the list have been saved. When making changes to lists directly in SharePoint, those changes take effect immediately and clicking a save button is not required. These different behaviors can be confusing when switching back and forth from SharePoint Designer to SharePoint, so make it a habit to always save list changes when working in SharePoint Designer.

Click the Save button to save the changes to the list. Open the PTO Tracker list in SharePoint and add an item for your test user. For the test user, set the available hours to 80 and the used hours to 0 and save the item (Figure 3.20).

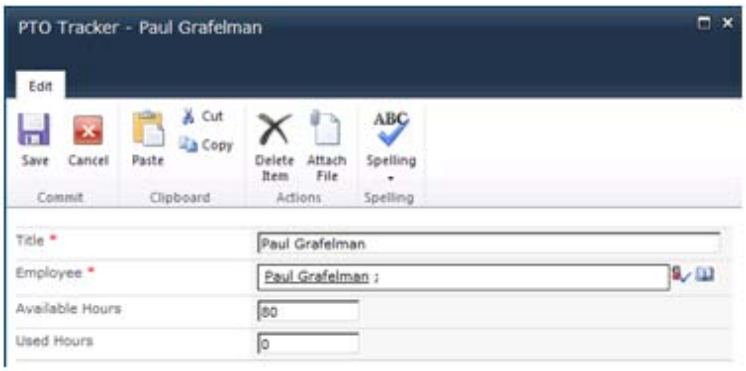


Figure 3.47 Add a new item to the list using the SharePoint interface to mimic how the end users will interact with the system. This test item will allow us to test the calculation of Available and Used hours.

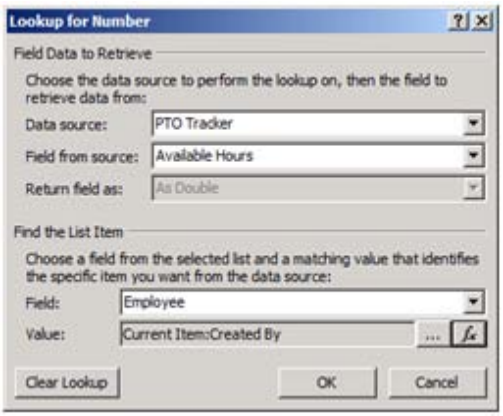
In order to use and modify the data from the PTO Tracker list in our workflow, we need to capture the data into variables. These variables can then be modified as required and eventually applied back to the PTO Tracker list. We can also add these variables into the email message body, so the manager will actually be able to see in the email if the PTO requestor has a positive PTO balance. Follow these steps to setup these variables pointing to the PTO Tracker list, as well as reference them in the email body:

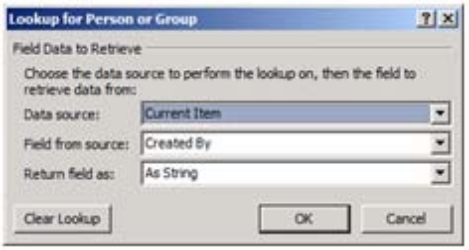
Updating Request Notification to include Available Hours and Used Hours

	Action	Result
1	Add a Set Workflow Variable action and reference a new variable titled Available Hours: A) Switch back to SharePoint Designer and add a 'Set Workflow Variable action' as the first action in the PTO Workflow Approval step. B) Click the 'Workflow Variable' link in the action and choose Create New Variable at the bottom of the dropdown. C) Name this variable 'Available Hours' and give it a type of Number.	The Set Workflow variable action will be added to the workflow, and will be configured to assign data to a new variable titled Available Hours.

To find the correct item on the PTO Tracker list we need to tell the workflow which item we are looking for. Setting the Ensure Unique Values option earlier will prevent duplicate entries, which would confuse the workflow.

	Action	Result
2	Set the value of the Available Hours variable with the requestor's available hours from the PTO Tracker list:: A) Click the Value link on the action. Click the Fx button and in the pop-up, set the Data source to the PTO Tracker list. Note: The Lookup dialog allows you to use the data from one list to find a specific record in another list. In this case we are looking for a specific row in the PTO Tracker list, identified by the data in the Created By field of the current list B) Set the Field from source to 'Available Hours' to capture the PTO hours remaining for the specified user. C) Set the Field to Employee and the value to the Current Item:Created By (fx button), then click OK:	Afterwards you'll have two new variables, on called AvailableHours and one called UsedHours. These variables are referenced in the manager's email so that person can make a more well informed decision.





Note: When you click OK, you will see a warning about ensuring unique lookups, click OK on the warning.

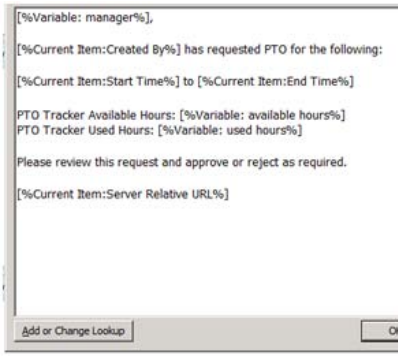
D) Repeat step 1 and 2 to store the used hours from the PTO Tracker in a variable named "Used Hours".

3 Use the Find Interval action to calculate the amount of PTO requested. Store the number of hours in a variable called "Requested Hours":

A) Add a 'Find interval between dates' action. Set it to find the number of hours between the current item 'Start Time' and current item 'End Time'.

B) Configure the find interval action to store the hours in a new variable called 'Requested hours'. This will be used to calculate Available and Used hours later.

The Find Interval action (Figure 3.21 is used to calculate the number of hours that were requested as PTO. The action is looking at the start and end times on the PTO Calendar item.

4	Update email message body to include how many hours of PTO the employee is requesting, how many they have available, and how many they have used: A) Modify the 'Send an Email action' to include the available hours, used hours, and requested hours in the message body.	

then Find hours between Current Item:Start Time and Current Item:End Time (Output to Variable: Requested Hours)

Figure 3.48 The Find Interval action is used to find the Days, Hours, Minutes, etc between two Date/Time fields. In this example we are calculating the Hours between the PTO requests start and end time in order to determine how many hours to subtract from the employees available hours.

With the email message going to the manager now updated and showing the data, we now need to handle if the manager approves the request. If the manager approves the request, the hours of PTO that were requested need to be deducted from the PTO Tracker list. Follow these steps to accomplish this:

Updated the employee's available hours if the request is approved

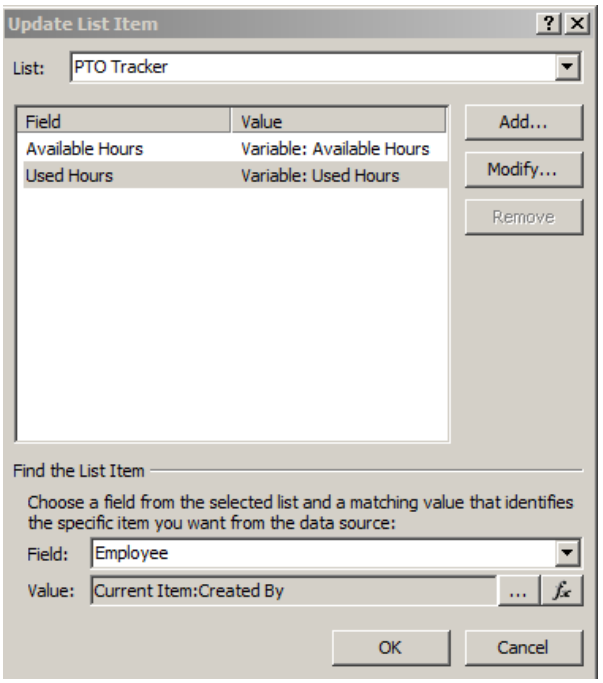
	Action	Result
1	Add a Do Calculation action that subtracts the Requested Hours variable from the Available Hours variable: A) Add a 'Do Calculation' action to the PTO Workflow Notification step. B) Click the first value link and set it to the variable 'Available Hours'. Set the operator to minus in order to subtract the available hours from the requested hours. Note: to select a workflow variable, change the Data Source to be Workflow Variables and Parameters. C) Set the second value link to the variable 'Requested Hours'.	The Available Hours variable will be updated by subtracting the Requested Hours from itself.

	D) Set the Output to field to the variable 'Available Hours'.	
2	Add another Do Calculation that adds the Requested Hours variable and the Used Hours variable: A) Add another 'Do Calculation' action to the PTO Workflow Notification step. B) Click the first value link and set it to the variable 'Used Hours'. Set the operator to plus in order to add the requested hours to the used hours. C) Set the second value link to the variable 'Requested Hours'. D) Set the 'Output to' field to the variable 'Used Hours'.	The Used Hours variable will be updated by adding itself to the Requested Hours.

Now that we have updated the data within the variables, we need to write this data back to the PTO Tracker list to store it permanently. Remember that once the workflow finishes, the data in the variables is lost. Using the Update List Item action we can set the values of multiple fields of an item with a single action. We only want to save the data back to the PTO Tracker list if the PTO request was approved. If it was rejected, we don't need to save the data. Perform the following steps to save the data back into the PTO Tracker list:

	Action	**Result**
3	Within the Notification Step, and inside the "Approved" if-else condition, add an Update List Item action to update the PTO hours data: A) Add an 'Update List Item' action. Click the 'this list' link. Change the list dropdown to 'PTO Tracker'. B) Click the Add button. Set the dropdown to 'Available Hours'. Set the value to the variable 'Available Hours'. Click OK. C) Click the Add button again. Set the dropdown to 'Used Hours'. Set the value to the variable 'Used Hours'. D) Click Ok.	The Update List Item action will write back to the PTO Tracker list, updating the requestor's available PTO.
4	Still within the Update List Item action 'this list' pop-up, use a lookup to find the correct item in the PTO Tracker list: A) In the *Find the List Item* section, Set the *Field* dropdown to Employee. B) Set the value field to 'Current Item:Created By' using the fx button. C) Click OK on the 'this list' pop-up.	The Update List Item action will be configured, and when PTO requests are approved, the data stored in the variables will be written back to the PTO Tracker list.

Note: You will again see the message about ensuring unique values. Click Yes because we are enforcing unique items on the list.



1.

Publish the workflow and create a new item on the PTO Calendar as a test user. Approve this item as the test user's manager. On the PTO Tracker list, you should notice that the available and used hours were updated correctly (Figure 3.22).

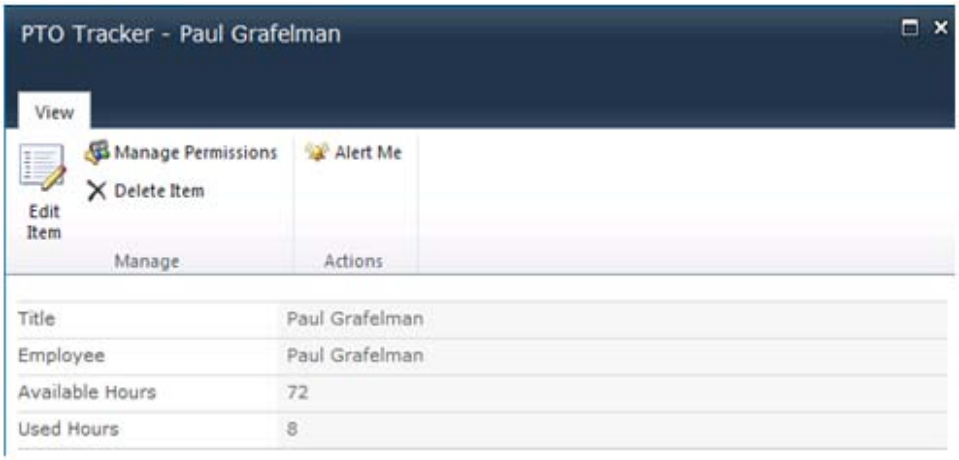


Figure 3.49 After running the workflow with the advanced functionality the PTO Tracker list will be updated to show how many PTO hours each employee has used and how many are still available.

3.4 Summary

This chapter has given you an introduction to SharePoint Designer 2010 workflows. You have learned that there are several reasons why Designer workflows are beneficial, including the ease with which they are created and that they do not require knowledge of complex programming languages like Visual Studio .Net. Because of this, SharePoint Designer workflows are intended to be created and used by both information workers and traditional developers.

You have also learned about the components that make up a Designer workflow and how these components are used in workflow development. Steps and Comments can be used to organize your workflows and make them easier to read and maintain later. Actions allow you to build workflows that do a variety of tasks including creating and updating list items and sending notifications to users. In addition, If and Else-if statements are available to add logic to your workflows, allowing them to make decisions. All these components when they come together can build very powerful workflows, including the example described in this chapter, a PTO request system.

Hopefully this chapter has gotten you to the point where you are comfortable thinking about and discussing how workflows can benefit your organization. We go from here back to a higher level of workflow planning and diagramming, by discussing workflows with Office Visio 2010.

4

Task Processing in SharePoint Designer Workflows

By introducing task processes into your workflows you can greatly increase the efficiency of the human interaction between your users and your workflows. This is because task processes can issue tasks assigned to a user, and the workflow can go idle while it waits for that task to be completed. It can be easy for a user to forget they have a workflow waiting on them. By using tasks in tasks lists, you can provide an easily referenced location where users can find their outstanding tasks.

SharePoint Designer has three main task related actions that help facilitate this process. The Assign a To-do Item action does exactly this, in that it assigns a task to a user and sits and waits for that task to be completed. If the workflow needs to gather information from the user, rather than simply wait for a task to be completed, the Assign a Form to a Group and Collect Data form a User actions can help take this task processing to the next level.

For more complicated workflow requirements, SharePoint Designer provides another action called the Start a Task Process action. This action creates what Designer calls a task processes, in that instead of just basic task processing, you can add actions that respond to all sorts of task related events like when a task expires, is deleted, and is completed.

This chapter's focus is to explore task processing for SharePoint Designer workflows. We'll cover the core task related actions. Also, we'll cover how to customize the overall task process through the Start a Task Process action.

4.1 SharePoint Designer Task Actions

SharePoint Designer workflows have several actions available that help add task processing functionality. You can use these actions to create tasks and assign them to users, wait for those tasks to be completed, and thereafter respond to those tasks when they have been completed. The Assign a To-do action gives you this functionality. There are other actions that can facilitate the gathering of information from users, like the Collect Data from a Use action. Most of these actions only take a few minutes to configure. Let's take a look at the three main out of box SharePoint Designer workflow task actions: Assign a To-Do Item, Assign a Form to a Group, and Collect Data from a User (Figure 4.1).

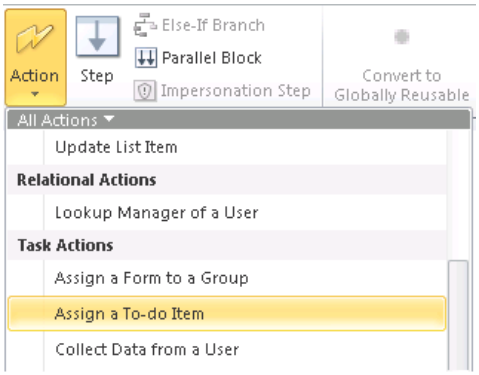


Figure 4.50 By default there are three main Task Actions that your workflows will interact with.

4.1.1 Adding to-do Items

The Assign a To-Do action is the most straight forward task action in SharePoint Designer. Essentially all you can do is create a task that's assigned to one or more people, and the workflow will go idle and wait for the tasks for each person to be completed before it continues. This action allows for two parameters, the task itself and the person or group to assign the task to (Figure 4.2). Although "these users" would typically indicate that it is used to assign the task to a group, it can also be used to assign the task to a single person if required.

Figure 4.51 The Assign a to do action is the simplest of task related actions. The workflow creates a task and waits for that task to be completed.

When you click on a to-do item a wizard box will appear and will prompt you for the task title and description values (Figure 4.3). You can set the Title and Description of the task, but no other fields. Also, the title and description can't be dynamic based on other workflow data, so this action is really only for the simplest of tasks since these values must be entered statically at design time. After you publish the workflow with this action in it, the workflow will create a task in the tasks list assigned to the user. At the point, the assignee won't be able to edit the task. All they can do is complete the tasks. In the tasks list, when the user clicks the task item, they will get a form in a pop up where with a button to complete the task (Figure 4.4). After the Complete Task button is clicked for each task, the workflow will continue.

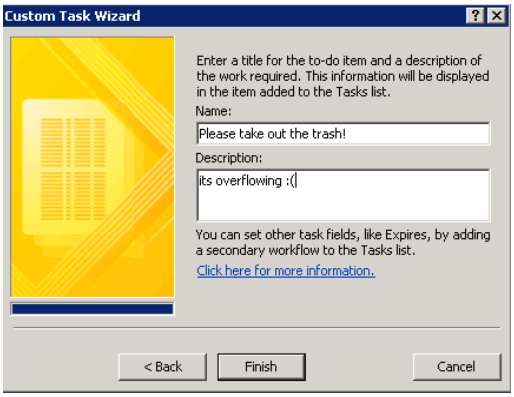


Figure 4.52 when assigning a to-do action, a wizard will appear prompting you for the task name and description.

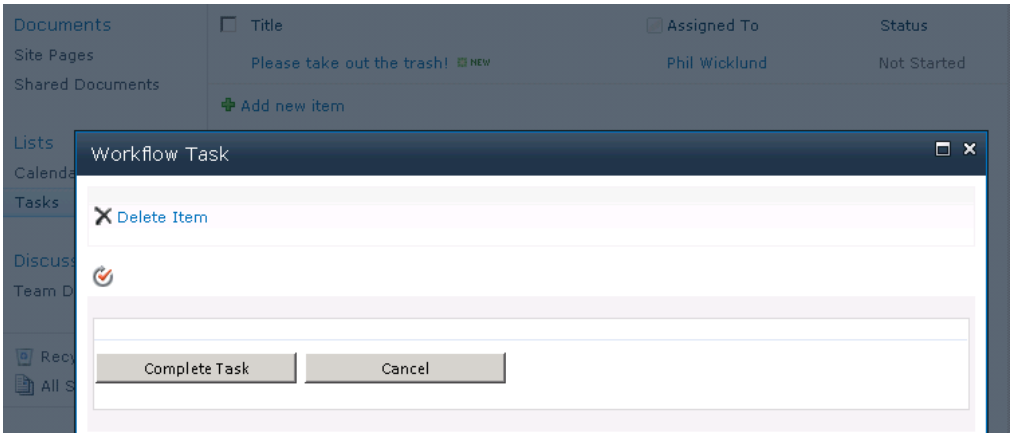


Figure 4.53 when the person that has been assigned the tasks clicks the task, they will see two buttons. The first completes the task, and the second cancels the operation.

4.1.2 Using the Assign a Form to a Group Action to Facilitate a Survey

The Assign a Form to a Group action is very similar to the Assign a to-do item action in that is assigns tasks one or more people. The big differences with the Assign a Form to a Group action is that when the user clicks on the task they are asked a defined list of questions. This was different with the previous action where all they saw was a complete button.

Essentially Assign a Form to a Group action behaves similar to how people would respond to a survey. The difference with using this action, rather than the default SharePoint Survey functionality, is that the workflow will wait for all the survey responses to come in before it continues. The answers to

these questions are stored in the task list with each answer stored in a separate column as part of a new content type. The action accepts two parameters, the first defines what questions to ask and the second specifies which users to assign the task to (Figure 4.5), and the second allows you to specify the users who are required to respond to the survey.

Figure 4.54 Assign a form to a group will use tasks to facilitate the gathering of responses to a survey.

When you click on a custom form another wizard popup will appear similar to the assign a to-do item actions. This time the big difference is you'll be prompted to enter the values in the form that the task assignee needs to fill out (Figure 4.6). After you publish, and the user opens their tasks, they will see these fields that they can fill out before they complete the task (Figure 4.7).

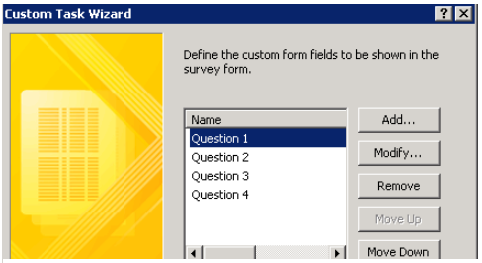


Figure 4.55 The Assign a Form to a User action will prompt the user to specify data, which will be saved in the task itself.

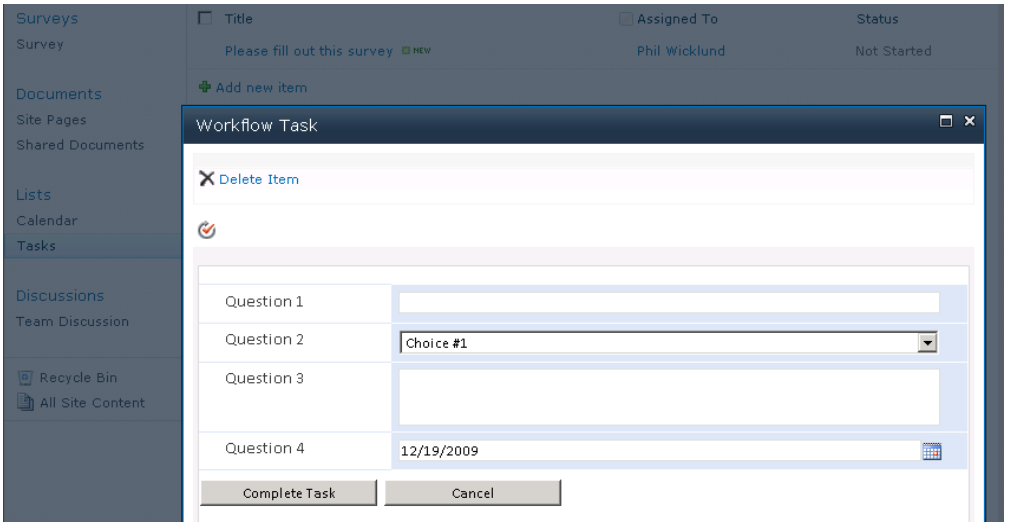


Figure 4.56 When the user clicks the task, they'll see all the data elements and questions that where specified in the action popup.

4.1.3 Using Tasks to Collect Data from a User

The Collect Data from a User action is very similar to the Assign a Form to a Group action with the big exception in that it allows the data entered in the form to be accessed by the workflow. The data entered in the form is still saved in columns in the task through a new content type; the data is also stored in a workflow variable. (Figure 4.8). The workflow can then react to the submission of the form by looking that the values entered in this variable.

Figure 4.57 Unlike the Assign a Form to a Group action, the Collect Data From a User action can save the values entered by the user in the task into a variable in the workflow.

4.2 Custom Task Processes in SharePoint Designer Workflows

We just got done discussing very simple tasks, where a task is assigned to a user and the workflow waits for the task to be completed. The trouble with these tasks, is that they don't support a lot of complex requirements. For example, what if your workflow needs to allow a task to be reassigned to someone else? Or, what if you need to set an expiration date on a task, and then react to when the task expires? It's with cases like these, and many others, where custom task processes are valuable. A custom task process still works off a single task assigned to one or more people. The main difference is

the dozen or so events you can respond to during that task's lifecycle. Events such as if the task is updated, expires, assigned, reassigned, and so on.

A custom task process can be created from within a SharePoint Designer workflow through the Start Custom Task Process action. The action requires three parameters, the task process instance, the item to start the task on, and the users to assign the task to (Figure 4.9).

Figure 4.58: A custom task process is used when requirements necessitate responding to events beyond simply the completion of a task.

When that task instance is clicked (first parameter), a new custom task process is created. You'll also notice when you click the instance you'll be navigated to the new custom task processes' settings tab (Figure 4.10, where the task process is named "Task (X)"). It's through this settings tab that you can add your functionality into the task process. There are three main boxes on this settings tab that we'll cover: the Customizations box, the Task Form Fields box, and the Task Outcomes box.

Notice the Customizations box in this tab. In this box you can click into three areas where you can drop conditions and actions into the task process. You can modify the overall task process, where you can respond to events like the process being cancelled or completed. Secondly, you can respond to task events like the task itself being assigned, expiring, deleted, or completed. And lastly you can setup conditions for the task being completed that must be met. All three of these sections contain the shell for all the events you can add actions into incorporate your unique business requirements.

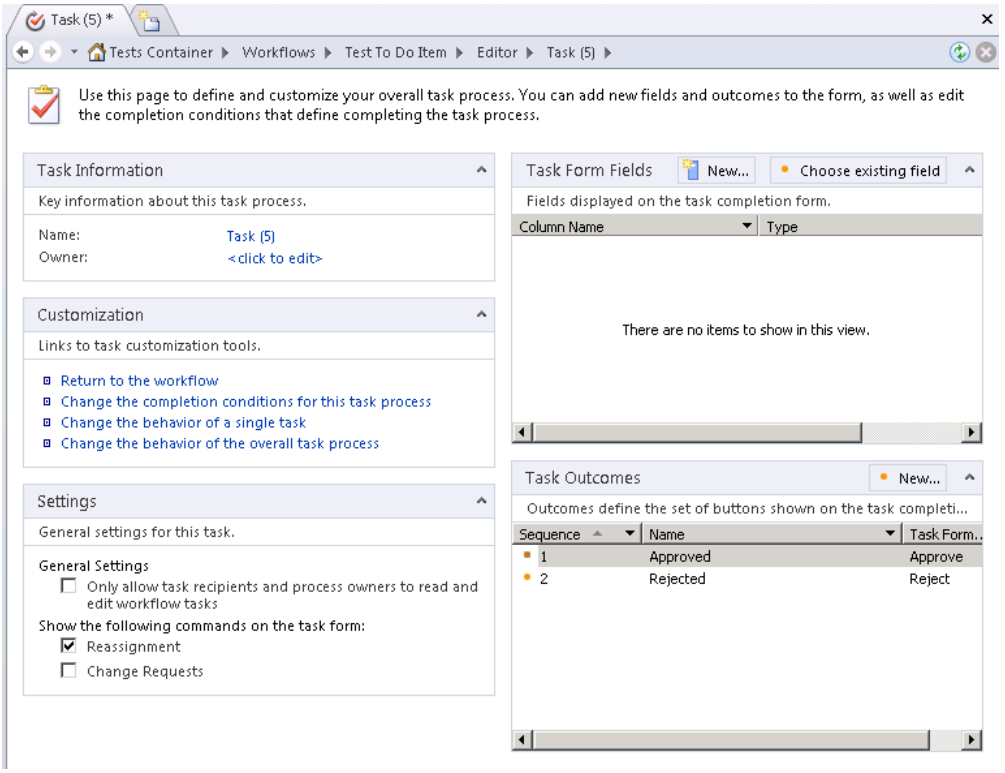


Figure 4.59 When you add a Start Custom Task Process action, you'll be navigated to the custom task process's settings tab where you can then start to build the custom task process.

Another interesting box on the tab is the Task Form Fields box. You can use this to add custom fields on the task edit form. The task edit form is presented to the task assignee when they click the task. By default they'll see the normal task fields, but perhaps you want to add some custom fields to the form that isn't on the task list itself. By adding task form fields, you can gather some new information from the user that your workflow can then use.

The Task Outcomes section is another area of particular interest. You can use this section to specify custom tasks outcomes for your tasks. A task outcome is the state of a task when it is completed. For example, Approved and Rejected are outcomes that come by default. You may want a custom task outcome called "Deferred". By adding a Deferred task outcome the user will see a third button on the task edit form that they can click. Your workflow will then flag the task with that outcome when the task completes.

The rest of this section in this chapter will take you through working with each of these 3 boxes of the custom task processes settings tab. We'll also discuss a subject called assignment stages where we will give the end user the ability to choose who should be assigned the tasks via an association form when the workflow first starts. This is helpful if while you're building the workflow you don't know who the appropriate assignees should be, and would rather let the user decide.

The example we're going to use to help explain the settings for a custom task process will be an example that deals with capital expenditure requests. A capital expenditure request is used by a company when a department or person needs to secure some of the company's capital to help fund a project or initiative. For example, a capital expenditure request may be submitted to purchase and setup a new branch office. A capital expenditure request is usually never as simple as a yes or a no. This is where our custom task process can help, because we'll be able to add our unique business logic into the approval of the expenditure request that will guide the approval from start to finish.

To setup the example, first create a new generic list called "Capital Expenditure Requests". Add two new columns to the list, "Request Description" as a multiline text field, and "Dollar Amount" as a currency type. Then, within SharePoint Designer, create a new List Workflow called "Expenditure Request Approval" and select the Capital Expenditure Requests list. You only need to add one activity onto the workflow, the Start Custom Task Process activity. Leave the second parameter at "Current Item", assign a user as the approver for the expenditure requests. Note, when you assign a user you'll be prompted to fill out an email template. The user will receive an email when the task process starts. Feel free to fill out this email template. At this stage you can also setup a due date for the task process if you wish. When completed, your workflow surface should look something like Figure 4.11. After you setup your workflow to this point, click the first parameter of the action, the task process instance. This will load the task process's settings tab, where the following sections will walk you through how to use.

Figure 4.60 The Start Custom Task Process action has three parameters, the process instance, the item to start the task on, and the user(s) to assign the task to.

The example will cover the three boxes on the custom task process setting tab, but in an alternating order. Firstly, we discuss how to change the overall task process from the Customizations box. We'll then discuss how to customize a task process's task edit form, and thereafter we'll return to the Customizations box on the settings tab to discuss how to change the behavior of a sing task. We'll lastly complete the example by setting up a custom task outcome and assignment stages.

4.2.1 Customizations Box: Changing the Overall Task Process

With our Capital Expenditure Requests list created, our Expenditure Requests Approval workflow setup, and the Start Custom Task Process activity added, it's time to add our business logic into our custom task process. The first thing we want to look at is the events surrounding overall task process. In the Customizations section in the custom task process's settings page, click the link titled "Change the behavior of the overall task process."

When you click this link you'll be taken to a tab that contains four events that we can add actions into (Figure 4.12). With these events we'll be able to respond when the task process first starts, when it is running, when it's been cancelled, and after it completes.

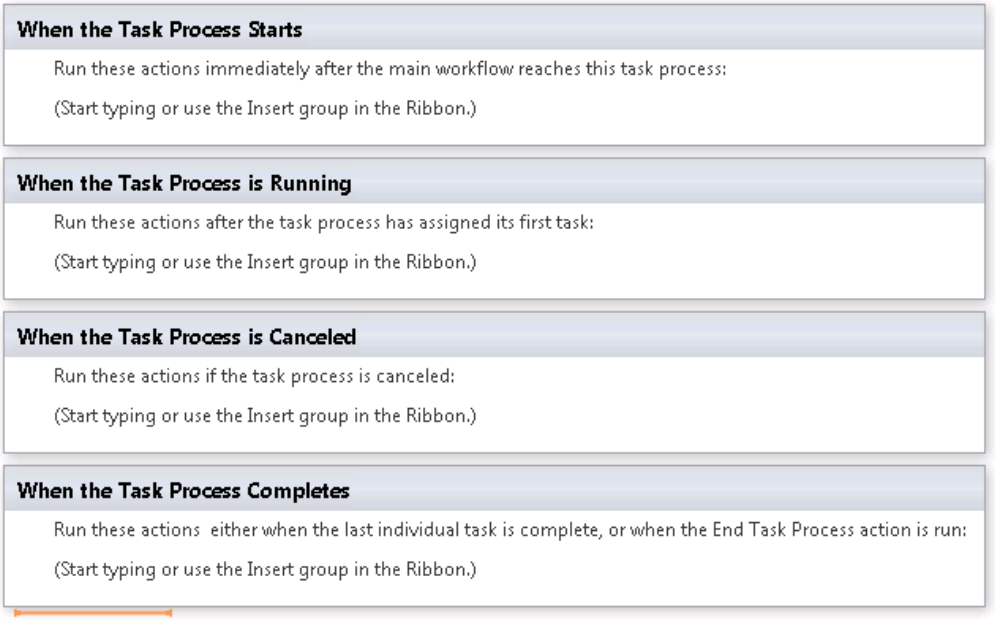


Figure 4.61 By editing the overall task process, you can add actions that respond to events like when the task process first starts, is running, is canceled, and when it is completed.

Let's first focus on when the task process complete. For our capital expenditure request system, we want the approver to be able to Approve, Defer, or Reject the request. In this When the Task Process Completes event, we want to identify the approver's action and email the requestor to let them know how things turned out. Our email should be context sensitive, in that the language is specific to whether it was approved or not. Also, we want to set the workflow's status to be either Approved, Deferred, or Rejected. Otherwise the status default will be set to Completed, which isn't too helpful if someone wanted to report on all the rejected requests, for example. Follow these steps to setup this event:

Email Requestor and Set the Workflow's Status when the Task Process Completes

	Action	Result
1	Create a new variable to hold the status of the approval: A) Click 'Local Variables' in the ribbon and create a new workflow variable called ApprovalStatus and select the *String* option as the variable's data type. Note: the actual value of the variable will be set later in section4.2.3's example.	A new variable is added that will store the approval status of the capital expenditure request.
2	Add an If any value equals any value, condition:	An If > Else if > Else series of conditions will

	A) Add the If any value equal any value condition into the When the Task Process Completes event. B) Add an Else-if condition C) Add an Else condition	be added to the event.
3	Complete the “if” conditions, checking to see if the ApprovalStatus variable is either Approved or Deferred: A) For the first if statement, check to see if the ApprovalStatus variable is equal to "Approved" B) For the second if statement, check to see if the ApprovalStatus variable is equal to "Deferred" Note: for the last "Else" block, we're at this point assuming since the request was neither approved nor deferred, it has been rejected.	The first If condition will meet the condition of the status is Approved, and the Else-if condition will likewise meet if the status is Deferred.
4	In each if/else block, add an email action to email the requestor of the status and add the Set Workflow Status action to set the workflow's status: A) drop the Send an Email action into each If and Else-If conditions. B) Fill out the email properties appropriately, like specifying a body for the email with language specific to the approver's response. C) Use the Set Workflow Status activity to set the status of the workflow for each approval response.	Your When the Task Process Completes event should now look like Figure 4.13.

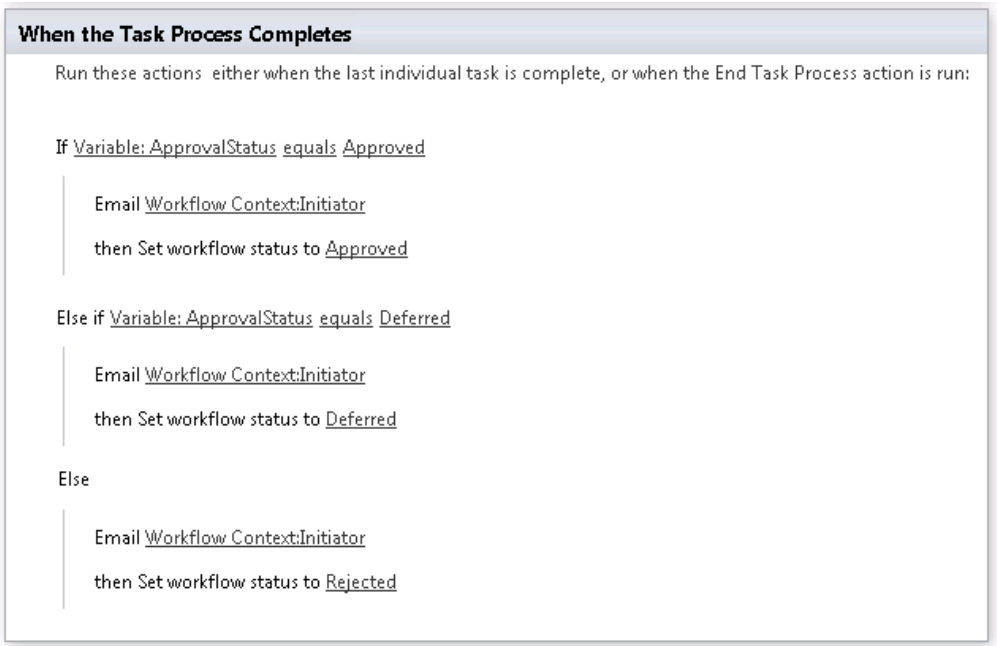


Figure 4.62 In the When the Task Process Completes event we added actions that demines if the approver approved the request or not, and sends an email to the requestor accordingly.

One more thing we want to do in this section is in the approval notification email, we want to let the requestor know what date the funds they are requesting will become available to them. We'll set this up more later, but for now, create a new variable called "DateFundsAvailable" that is of type Date/Time. In the approval email, make reference to this variable. For instance, your approval notification email may look something like Figure 4.14.

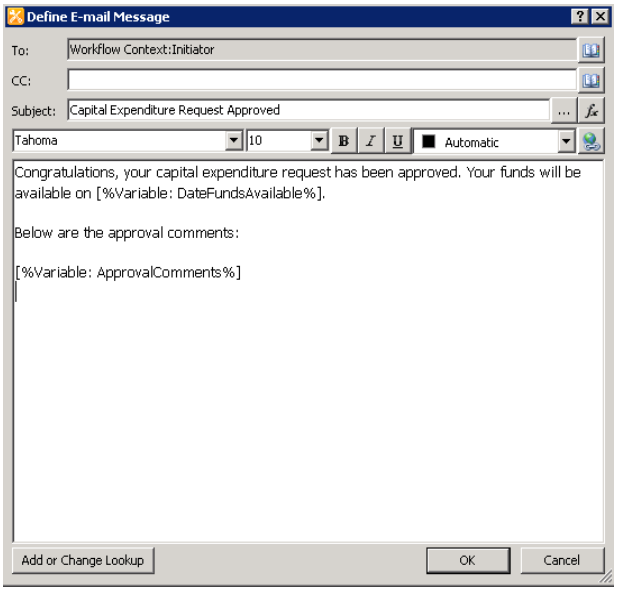


Figure 4.63 In the approval notification email, add a reference to the Date Funds Available variable, informing the requestor when their funds will be available.

The next thing we want to do in this overall process section is react to when the expenditure request list item changes. We don't want the user submitting the request to be able to change the amount after the request has been approved. What we want to have happen is if the request is changed or deleted, the requestor and approver need to be notified that the request has been canceled.

To set this up, drop a parallel block (click Parallel Block in the ribbon) inside the When the Task Process in Running event. What we want to do next is add two steps into this parallel block, and an event into each step. We'll add the Wait for Change in Task Process Item action and the Wait for Deletion of Task Process Item action into the two steps. The When the Task Process is Running event is helpful because if the request is edited anywhere in the task process, our workflow will react and execute our wait for change and delete actions. The parallel block is handy because we can listen for multiple events simultaneously. Since we want to listen for two events (edit and change), we need this parallel block.

After you add the parallel block, the steps, and the wait actions, follow them up with an email to the requestor and approver. Also, end the task process so the workflow will complete, and for the changed action, set the workflow's status to canceled. Afterwards, your When the Task Process is Running action should look like Figure 4.15.

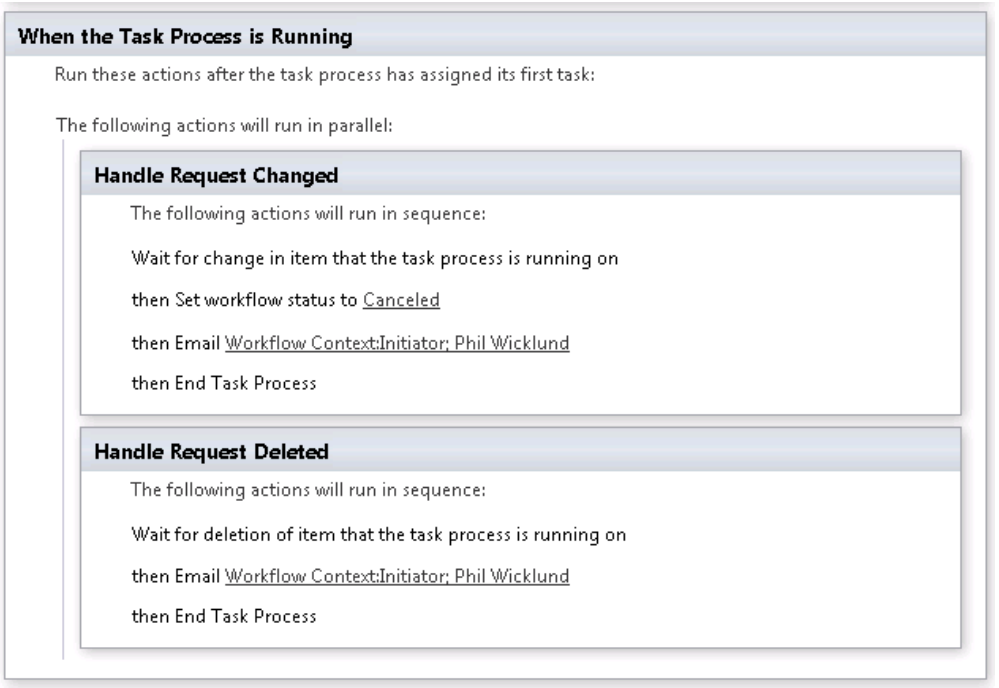


Figure 4.64 The When the Task Process in Running event is a great place to listen for changes made to the workflow's item, like if it is edited or deleted.

4.2.2 Task Form Fields Box: Customizing the Task Edit Form

Before we continue modifying the task process through the Customizations box, we need to first configure our task edit form. We need this form to prompt the user to enter the Date Funds Available field that the approval notification email is using. In the next section we'll take this date and assign it to the DateFundsAvailable variable that was setup in the previous section. But before we can assign the variable, we first need to prompt the approver to enter it as they are approving the task. On the custom task process settings tab, there's a section called Task Form Fields (Figure 4.16) where we can add new fields onto the task edit form.

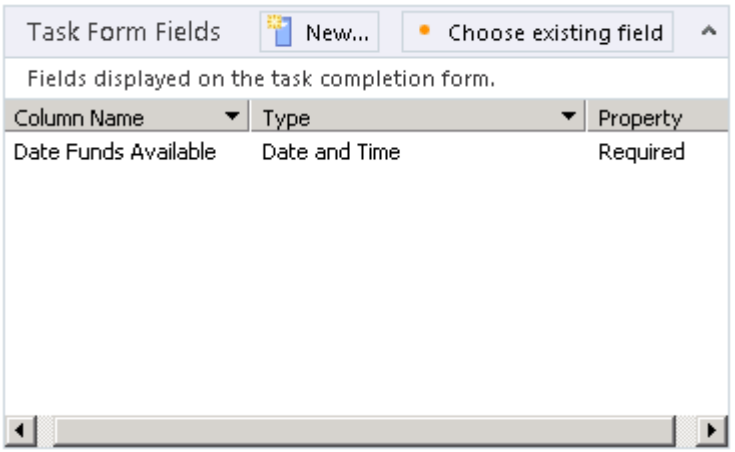


Figure 4.65 You can add more fields onto the task edit form be entering them into the Task Form Fields section on the custom task process's settings tab.

To do this, click the New button. A popup will appear asking you to specify the name of the field, its description, and its data type. Add a new field called Date Funds Available and set it as a Date and Time type. Don't publish just yet, but after you do our approvers will notice a new field at the top of the task edit form (Figure 4.17).

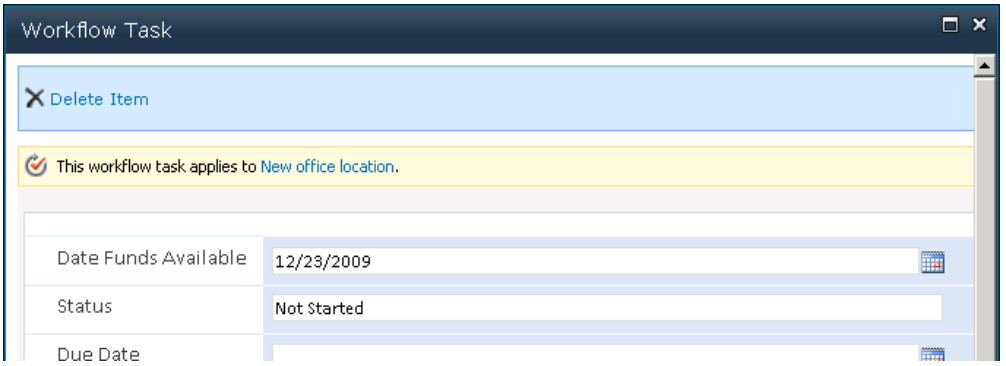


Figure 4.66 After the workflow is published, the fields entered in the Task Form Fields section will now display on the task edit form.

4.2.3 Customizations Box: Changing the Behavior of a Single Task

With our task form fields in place, we can now resume configuring our task process through the Customizations box. We now need to add actions that respond to the approver approving the task. To do this we want to add actions into the Change the Behavior of a Single Task, tab. To get to this tab, click the Change the Behavior of a Single Task link in the Customization box. In this tab, you'll see 5

events we can add business logic into. We can react to when the task is first assigned, goes into a pending state, expires, is deleted, and when the task is completed (Figure 4.18).

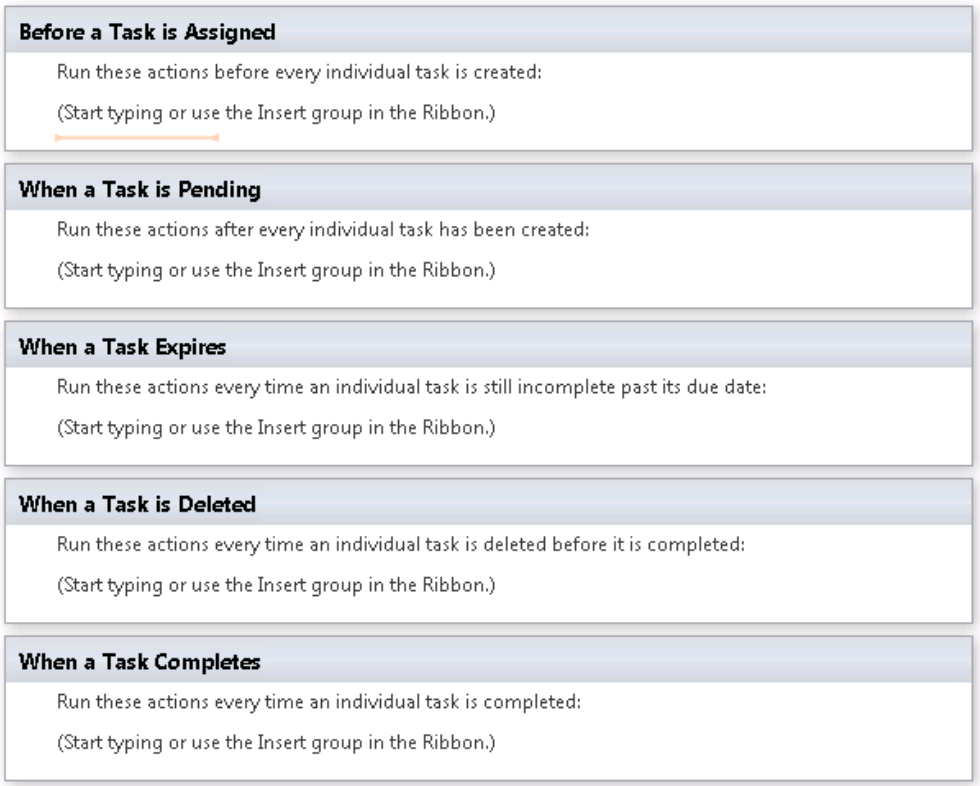


Figure 4.67 By changing the behavior of a single task, you can respond to events such as when the task is first assigned, when it is set to pending, expires, deleted, and when it is completed.

For our capital expenditure request workflow, let's drop actions into just three of these events. For the first event, we want to react to when the request expires. If the approver doesn't approve or reject the request in an allotted amount of time, we want to escalate the request to the approver's manager. Simply drop the Escalate Task action into the When a Task Expires event (Figure 4.19). This action automatically figures out who the approver reports to, and reassigns the task to that person.

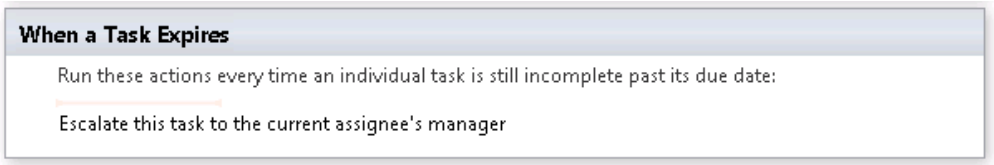


Figure 4.68 The When a Task Expires event is a great place to escalate the task to the assignee's manager.

Another event that is important to respond to is if the task itself is deleted. There's nothing stopping a user with full control to the tasks list to delete tasks, and our workflow should have logic in it that appropriately responds to a deletion. Inside the When a Task is Deleted event, add the Set Workflow Variable action and choose the ApprovalStatus variable to assign a new status to. Set the variable to "Canceled". Lastly, add the End Task Process action to cancel the task process and stop the workflow (Figure 4.20).

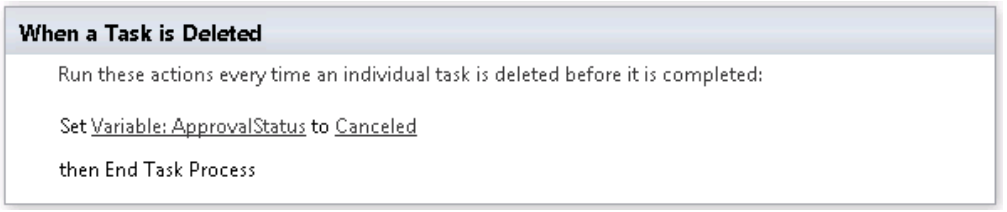


Figure 4.69 If a task is deleted that a workflow is dependant upon, make sure to accommodate a clean exit for the workflow. In this case, we're setting the workflow's status and ending the task process.

The bulk of our logic lives in the When a Task Completes action. If you remember in our overall task process set of events, we customized the When the Task Process Completes event. In that event we looked at a variable called ApprovalStatus, and depending on what the status is set to, we sent out different emails. Well, in the When a Task Completes event, we need to assign the ApprovalStatus variable to the based on the outcome of the task. If the task was approved, our capital expenditure request should also be approved, and so on. Follow these steps to set this up:

Setting the ApprovalStatus variable in the When a Task Completes event.

	Action	Result
1	Add another If/Else-if/Else condition like we did in the When the Task Process Completes example.	An If > Else if > Else series of conditions will be added to the event.
2	Set the ApprovalStatus and the DateFundsAvailable variables in the first If condition: A) In the first if condition, check if the current task's outcome is Approved, and If so, assign the ApprovalStatus variable to Approved by using the Set a Workflow Variable action. Note: to get the current task's outcome, click the Fx box in the If condition and change the Data Source to be "Current Task: Task" and then change the Field from source to be "Outcome". B) Add a second Set a Workflow variable action in the first If condition and assign the DateFundsAvailable variable to the current task's Date Funds Available field we created when we edited the task's edit form.	Both the ApprovalStatus and DateFundsAvailable variables will be set to values when the status is approved.

3	Configure the Else-If condition to check if the task was deferred: A) In the Else-if statement, check if the current task's outcome is set to Deferred by using another 'If any value equals any value' condition. B) Assign the ApprovalStatus variable to Deferred	The Else-If condition will now be set to check if the task was deferred. If so, the ApprovalStatus variable is set to deferred.
4	In the Else block, set the ApprovalStatus variable to Rejected.	The ApprovalStatus variable is set to rejected in the Else block.
5	Lastly, end the task process by adding the End Task Process action below the IfElse action. .	After completing these steps, your When a Task Completes event should look something like Figure 4.21.

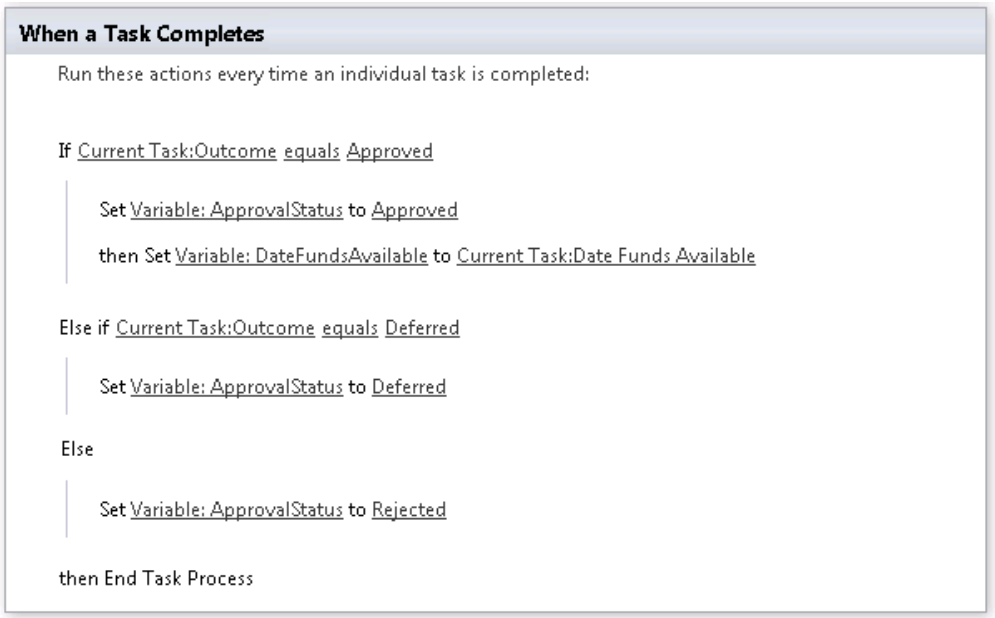


Figure 4.70 In the When a Task Completes event, we're assigning the ApprovalStatus variable to it's corresponding task outcome.

4.2.4 Task Outcomes Box: Defining Custom Task Outcomes

The last step in our capital expenditure request's journey is to setup a custom task outcome for request deferrals. As was mentioned, there are two outcomes by default, Approved and Rejected. Essentially a task outcome is the state at which a task is set after it is completed. In the case of our capital expenditure request, after the approver completes the task its outcome will either be Approved, Deferred, or Rejected.

Notice the Task Outcomes section in the custom task process's settings tab (Figure 4.22). By default the first two outcomes are present. To add the third, click the 'New' button and give the outcome a name of Deferred and a button title of Defer.

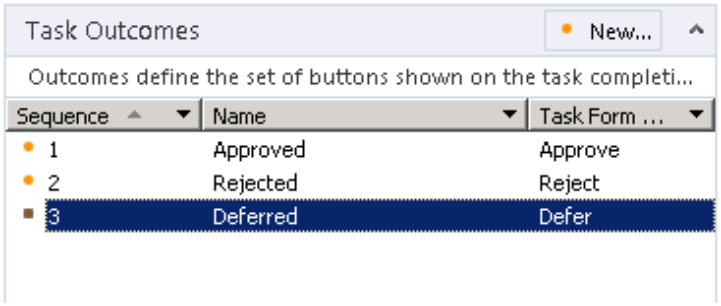


Figure 4.71 Beyond the default Approved and Rejected outcomes, you can add custom outcomes, such as Deferred.

When this third outcome is selected by the approver, the name "Deferred" will be stored in the task's Outcome variable. "Defer" refers to the name of the button on the task edit form. Notice how when the user goes to approve the task, there's a new option available to them on the task edit form (Figure 4.23). They can now defer the request until a later date when the workflow will be re-initiated.

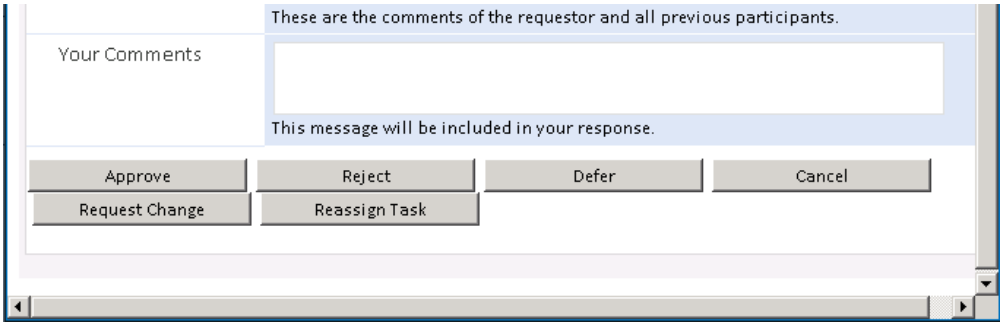


Figure 4.72 After the workflow is published, you notice your custom task outcomes will appear as buttons on the task edit form.

When the user clicks the Defer button, the task's Status will be set to completed, and the task's Outcome will be set to "Deferred" (Figure 4.24). Click the Publish button and test your custom task process for yourself.

Figure 4.73 After a task is completed, which ever task outcome is selected on the task edit form will be populated in the Outcome column of the task itself.

Since our custom task process is waiting for the task to be completed, it will respond to this button click as well. In the When Task is Completed event, our If/Else-if/Else branch will determine that the approver deferred the request, and it will set the ApprovalStatus to Deferred. Then, back in the overall task process events, the When the Task Process Completes event will look at the ApprovalStatus variable and will send an email to the requestor notifying them that their request has been deferred. After which, the workflow will complete and the workflow's status will be set to Deferred (Figure 4.25).

Figure 4.74 For our expenditure request workflow, after the task is completed, the outcome is stored in the workflow's status column.

4.2.5 Assignment Stages

If you remember, when we first setup our custom task process we assigned the task process to a specific person. Well, what if more than one person needs to approve the expenditure request? What's more, what if you don't know who the approvers should be at design time, and you want your user to specify the approver? Assignment stages are a great way to give the user the ability to determine who should receive the tasks that are being assigned. Rather than entering someone's name directly into the workflow, you can just assign custom task process to an assignment stage. Then, that assignment stage will appear on the workflow's initiation form, and the user can edit the stage and specify all the users he thinks should be assigned the tasks. Figure 4.26 shows what this association form may look like when assignment stages are enabled. The figure shows two stages configured, where the first stage has two users being assigned the task in tandem, and the second stage having a single user.

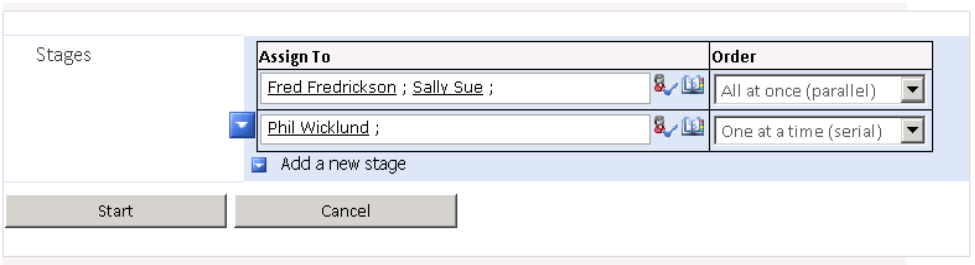


Figure 4.75 By enabling assignment stages, you can give your end users to ability to choose who the tasks should be assigned to. The Assignment Stage appears on the workflow's initiation form when it is first started.

With this assignment stage, the capital expenditure request will actually be executed three times. The first two times will be executed in tandem. Notice in Figure 4.26 that two users are assigned in the first stage. After these two users have both approved their request, the request will go into the section stage, directed toward a third user for approval. Notice, the user can specify an order to make the requests occur in parallel or sequentially. Now back in the Start Custom Task Process action, instead of assigning the task process to a specify individual, you can assign it to an assignment stage instead (Figure 4.27 and Figure 4.28).

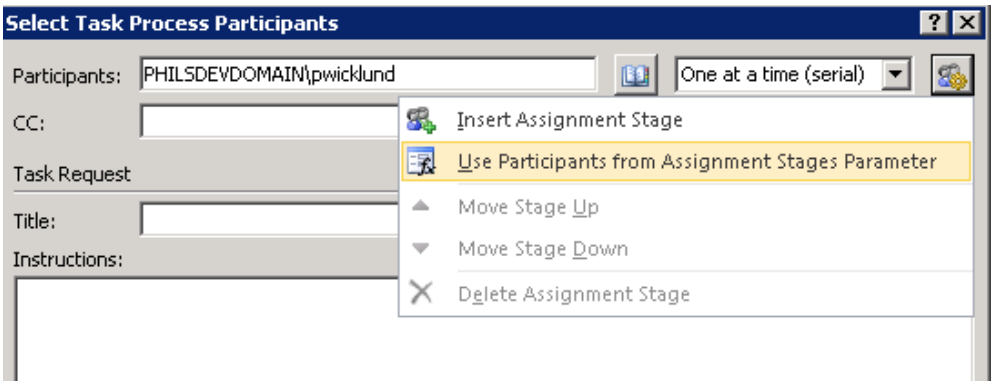


Figure 4.76 Instead of typing an individual, you can set the task to be assigned to an Assignment Stage.

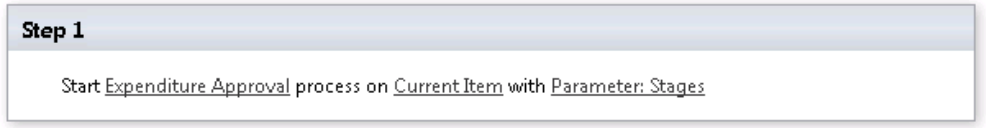


Figure 4.77 Notice the third parameter is set to an assignment stage rather than a specific user.

To make an Assignment Stage appear on the workflow's association form, simply click the Initiation Form Parameters button on the ribbon within SharePoint Designer. Add a new parameter by clicking the

Add button on the popup window that appears. Give the parameter a name, and change the Information Type to be Assignment Stage (Figure 4.29). Click OK, and after you publish the workflow the assignment stages will now be appearing on the workflow's association form.

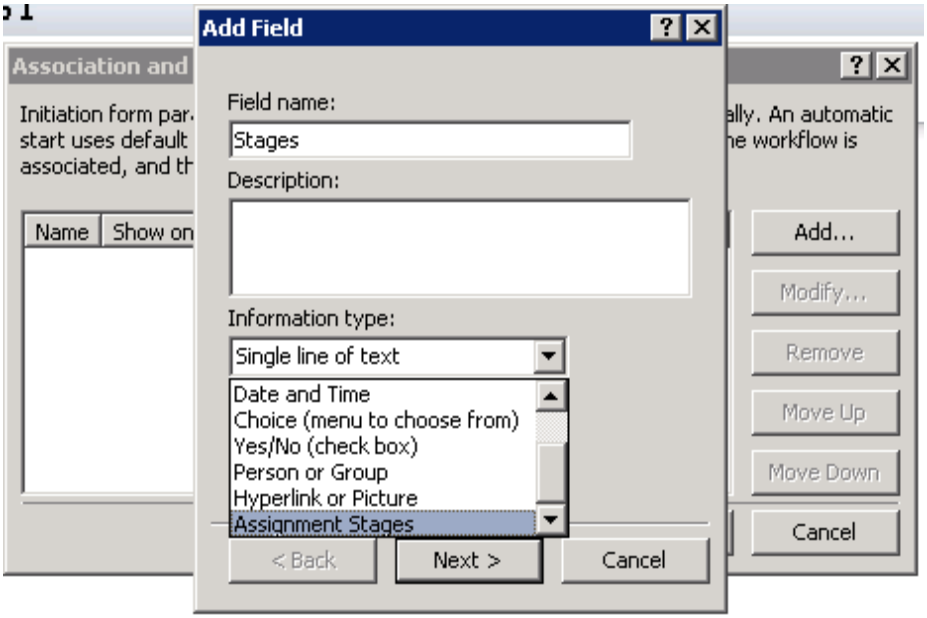


Figure 4.78 To enable Assignment Stages, simply add a new initiation parameter through the Initiation Parameters button in the ribbon.

4.3 Summary

Workflows and Task Processes are like peas and carrots - meaning, they complement one another very well. Since most workflows require a lot of human interaction, they oftentimes are heavily dependent upon tasks to help manage that human interaction. Tasks can be used to help a workflow facilitate the gathering of information that the workflow needs to proceed. An approval workflow, for example, issues tasks to the approvers, and the approver can approve the item directly from that task. After the task is completed, the workflow can then proceed.

There are several ways to develop a task process for a SharePoint workflow. SharePoint Designer has several actions right out of box that can be used for task management. You can create one-off tasks, like a generic task the workflow simply waits for it to be complete. Also, there's actions that help the workflow gather data if there's something the workflow needs before it can proceed. For the most complex task processing needs, you can create a custom task process in SharePoint designer using the Start Custom Task Process action. With a custom task process you can change the overall behavior of the process like dropping actions that fire when the process first starts, or is completed. You can also modify the behavior of a single task by adding actions that respond to a task being assigned,

completed, or perhaps expiring. Even custom task outcomes and custom task edit forms with InfoPath are possible through SharePoint Designer workflows.

5

Advanced SharePoint Designer Workflows

In the previous SharePoint Designer (SPD) chapter we looked at the basics of creating workflows, including how to use the fundamental components of a workflow and how to use the workflow editing interface. While that was a good start, there's definitely a lot more valuable information around SPD workflows that can be covered.

One of those areas is around workflow templates. In SPD 2010 you can now save a workflow as a template. This template can be deployed across the entire farm, rather than just the site or site collection. Even more valuable is the ability to deploy this workflow template to an entirely separate farm. This gives you the ability to test your SPD workflows in a non-production environment without risk of causing unforeseeable production issues.

Another important technique that didn't fit into the previous SPD chapter was around customizing the out of box workflows. SharePoint 2010 offers several powerful out of box workflows, but they oftentimes are too generic to meet specific business needs. Rather than starting from scratch, you can extend the out of box workflows with custom functionality.

In addition to those two techniques, SPD 2010 has a host of new actions and conditions of particular interest. The two most exciting groups include actions and conditions that manage document sets and security permissions. Document sets are groups of documents. Workflows can run on top of these sets and do interesting things such as route them for retention within Records Center. This may be desirable for compliance or legal reasons. The second group includes actions and conditions around SharePoint security. Now with SPD 2010 your workflows can manage security, such as add and remove users from documents.

Lastly, SPD workflows can now interact easily with external data through Business Connectivity Services (BCS). By creating an External Content Type and an External List our workflow's can generate the BCS pluming on our behalf to connect and work with external data such as SQL data. Each of these topics will allow your workflows to provide additional value to your organization, either by further improving and automating processes or by leveraging existing business data to help make decisions.

5.1 SharePoint Designer Workflow Templates

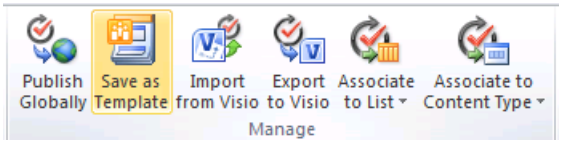
You can save a SPD workflow as a template. The workflow can then be deployed to another SharePoint Web Application or even another Farm. Previously this was only possible when working with Visual Studio workflows, which usually require significant amounts of time and experienced .NET developers. Using SPD 2010 it can be done quickly and without any .NET experience. The main benefit to these templates is the ability to test your workflows in a non-production environment, such as a development or test environment. After you've thoroughly tested the workflow and import it into production and make it available to your users. This reduces the risk of users experiencing issues with your workflow because it's currently being worked on.

To save a workflow as a template, open the workflow in SharePoint Designer and click the Save as Template Ribbon button. The template is saved as a file with a .wsp file extension in the Site Assets library within that site. It can then be downloaded and deployed to the entire farm or to a different farm, allowing you to develop workflows one time and deploy them to multiple SharePoint environments. Follow these steps to save a workflow as a template and deploy to the entire farm:

Templates for Globally Reusable Workflows

Note that globally reusable workflows cannot be saved as workflow templates.

Deploying an SPD workflow across the entire farm

	Action	Result
1	Open SPD and save a workflow as a template: A) Open SPD and connect to your SharePoint Site. B) Open a workflow or create a new one and save the workflow as a template by clicking the Save as Template ribbon button: 	The WSP (which is the solution file) file will be saved to the Site Assets library. You will see a prompt to indicate that it was successfully saved and the WSP file will appear under Site Assets, as in Figure 5.1.

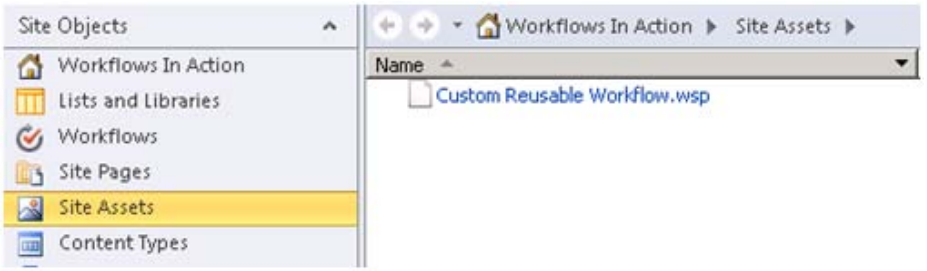
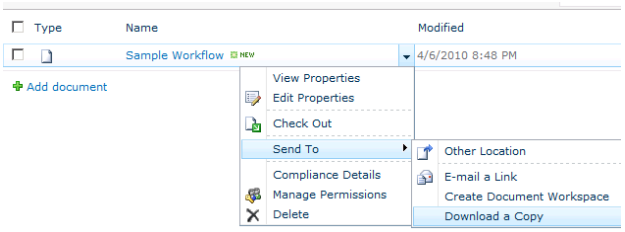


Figure 5.79 After you save a SPD workflow as a template, the workflow is packaged up and saved into the Site Assets library within that site.

	Action	Result
2	Download the WSP file to the server's file system: A) Browse to the Site Assets library and click Send To -> Download a Copy:  B) Save the file to the root of the C drive on one of the servers in the farm.	The WSP template file will be downloaded to the root of the C drive on one of the servers in the farm.

The next step is to add this template into the SharePoint farm. Some operations in SharePoint cannot be done using the Central Administration or Site Settings interfaces and can only be done using the command line prompt. SharePoint 2010 includes a utility called "stsadm" that allows these otherwise hidden operations to be performed. In this case, we are trying to add code to the server in the form of a WSP file, also called a Solution. The only way to add a solution to a SharePoint farm is via the stsadm tool. Because it is a command line tool and must be run directly on the server you may not have access to it within your organization. If you do not have access to the server or permissions to run a command line operation you will need to coordinate with your SharePoint server administrator to issue the following command:

	Action	Result
3	Log onto the server where you downloaded the WSP and execute the stsadm command to add the solution: A) Log onto the server and open command prompt by click Start -> Run, typing "cmd" and clicking OK. B) Run the following command: `C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\BIN>stsadm -o addsolution -filename c:\Custom_Reusable_Workflow.wsp`	The WSP template solution will be added into the SharePoint farm via stsadm.

Note: The command will be similar to this, but be sure to adjust the name and path of the WSP file as required	

Although a WSP (also called solution) can only be added to the farm using stsadm, they can be deployed using the Central Administration web interface. Continue with the steps to deploy the solution:

	Action	Result
4	Deploy the solution (WSP template) via Central Administration: A) In central administration of the target farm, click System Settings on the left of the screen and then click Farm Solutions under Farm Management. B) Find the WSP you just added to the farm which should be listed with a status of 'Not Deployed', and click the name of the solution to open the Solution Properties screen. C) Click Deploy Solution at the top of the screen and then leave all settings at their default and click OK.	You will be returned to the Solution Management screen where you should see that your solution is now deployed to the farm, seen in Figure 5.3.

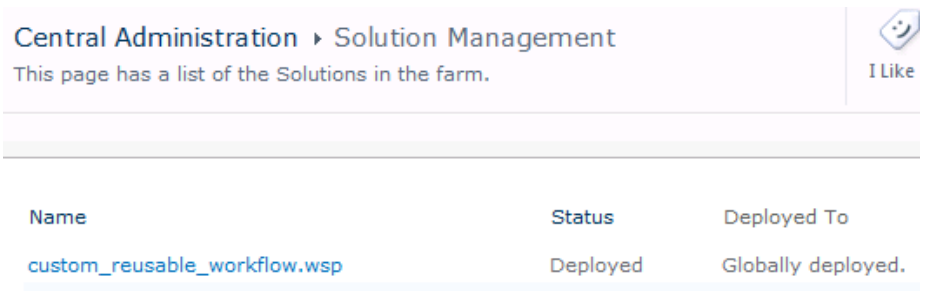
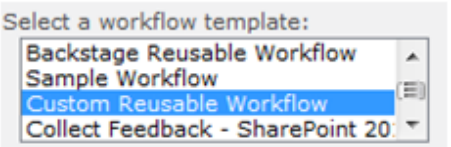


Figure 5.80 The Solution Management screen allows you to easily deploy solutions to the farm after they have been added using the stsadm command line tool.

After the Solution is deployed to the farm it must be activated in order to take effect or become visible to end users. This allows you to only activate it in the specific areas where it is needed. In the case of workflow templates, they are activated at the Site level. Continue on to activate the template in a Site:

	Action	Result
5	Activate the workflow's feature on a SharePoint site:	The workflow's

A) In a different site or site collection than where the workflow was created, activate the feature using the Site Features screen under Site Settings:  B) Add the workflow to a list using the Add Workflow option: 	feature will be activated on a site, and the workflow is now ready for use on that site.

5.2 Customizing the Out of Box Workflows

SharePoint 2010 includes several workflows out-of-the-box including list workflows and site workflows such as the Approval workflow, the Collect Signatures workflow, and the Collect Feedback workflow. The default workflows are very powerful and include a fair amount of complexity. As powerful as they are, Microsoft had to make them fairly generic in order to apply to a larger number of situations. For example the Approval workflow can be applied to any list in SharePoint 2010. In some cases the default workflows will come very close to meeting your needs, but may fall just a bit short. For instance, you may want the Approval workflow to log all approvals and rejections to a common list which is not done with the default Approval workflow. By modifying the default workflows you can add some (or a lot) of additional functionality to the built in processes. This provides a lot of value to you as a SPD workflow developer, because rather than start from scratch, you can extend the default workflows thus saving you a lot of time.

SharePoint 2007 Workflows cannot be modified

Unlike SharePoint 2010, the out-of-the-box 2007 workflows cannot be modified. This means if you upgrade from 2007 onto 2010, and you were using the 2007 Approval workflow, for instance, you won't be able to take that workflow instance and modify it. You'll need to start over with the new 2010 Approval workflow, and modify that one.

If you've notice " - SharePoint 2010" is added to the workflow names to differentiate the workflows from the 2007 versions, which are also included in a SharePoint 2010. This allows for easier migration and backwards compatibility, but they are not activated by default. Even if their features are activated they still cannot be modified and will not appear in SharePoint Designer.

To modify an out-of-the-box workflow open your site using SharePoint Designer 2010 and open the Workflows site object from the left hand menu. You will see the three workflows listed and any custom workflows that may exist, as is shown in Figure 5.4.

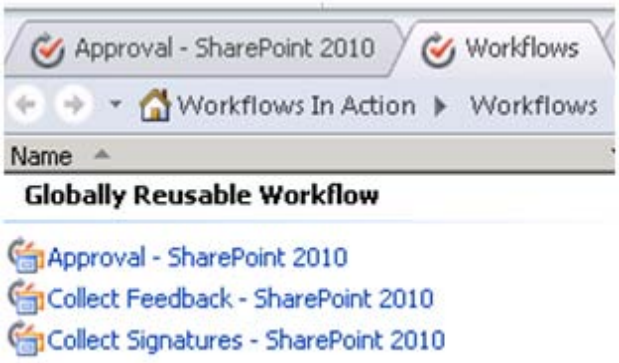


Figure 5.81 SharePoint includes several workflows by default, all of which are now editable in SharePoint Designer.

Clicking the name of any of the workflows will open the Workflow editing interface. From here you can click the Edit Workflow link to see the steps and actions associated with the workflow. If you edit the "Approval - SharePoint 2010" workflow and view the workflow steps it will at first appear to be a very simple workflow with only one step, see Figure 5.5.

Clicking the Approval task will reveal the true logic which includes Completion Conditions, Single Item Behaviors, and Overall Behaviors, see Figure 5.6 and Figure 5.7, all of which can be modified using the workflow editing interface.

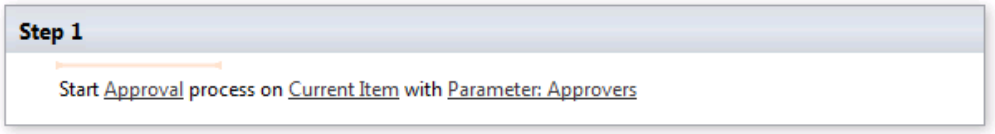


Figure 5.82 At first glance the Approval workflow looks very simple. Its Approval Action contains its true functionality including multiple Completion Conditions and Task Behaviors.

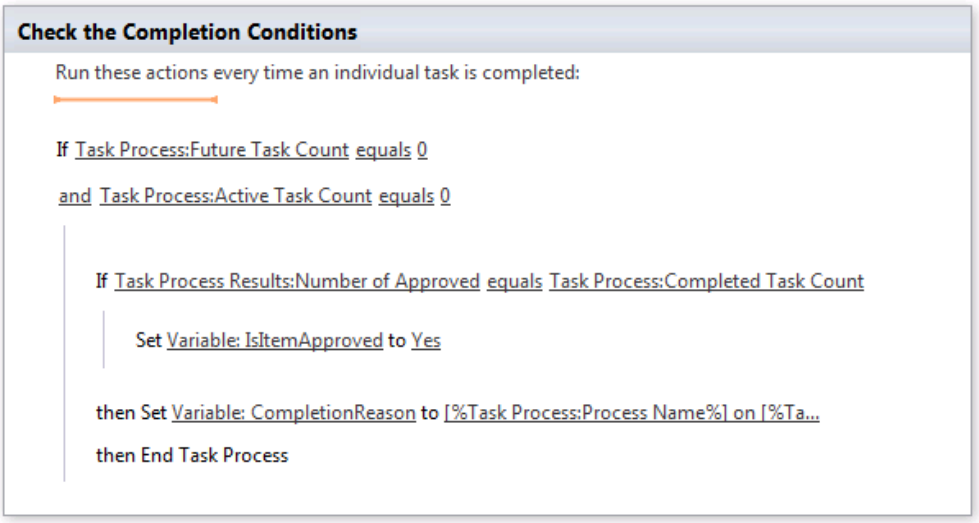


Figure 5.83 The Completion Conditions determine when the Approval Task is complete.

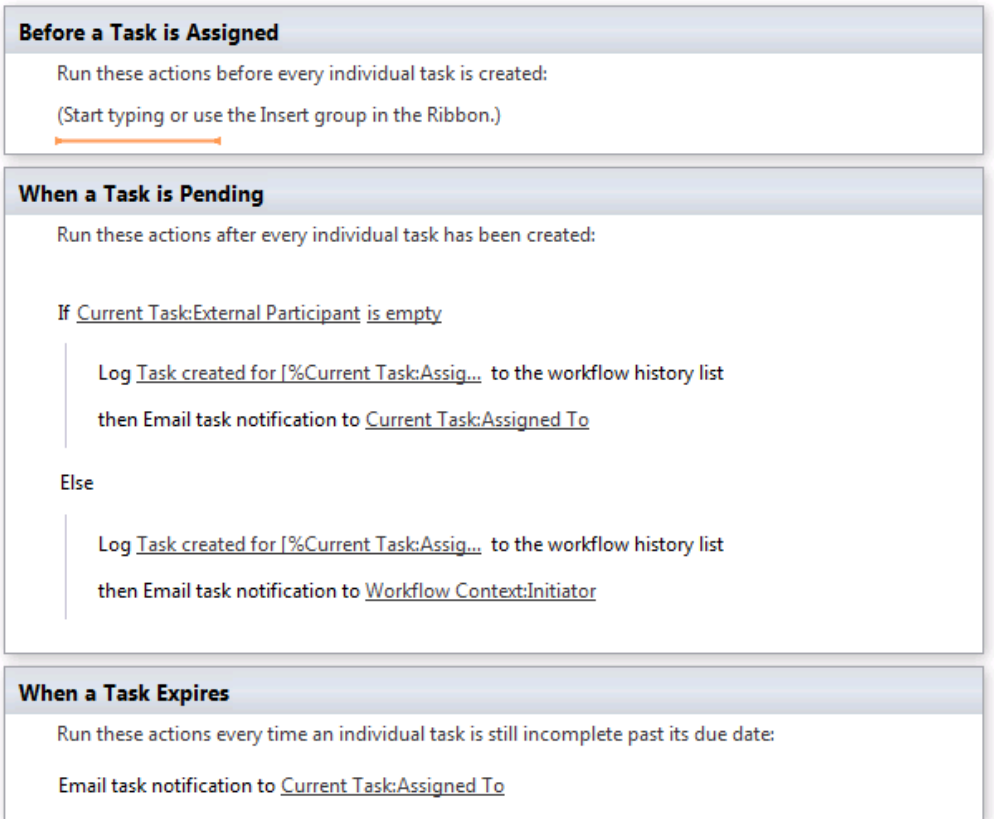


Figure 5.84 The Behaviors determine what happens at different stages of the process.

Because of the complexity of the out-of-the-box workflows it is recommended that you do not modify the originals. Instead, copy the original and make changes to the copy. To do this, right click on the workflow you want to copy and choose "Copy and Modify". You can also use the Copy & Modify Ribbon button to copy an out-of-the-box workflow.

After using either Copy & Modify option you will be presented with a dialog that will allow you to give your workflow a unique name. Enter a name for your new copy. Optionally, you can bind the new workflow to a content type, and limit the scope of where that workflow is available to a particular content type. After making you content type selection, click OK.

When you click OK you will be taken directly to the Workflow editor screen. You may notice that the action names are modified slightly due to the copy process. The text "(en-US) Copy" has been added to the Approval action name in Figure 5.11. You can leave these as is or adjust the names too. At this point you can save and publish the workflow, with or without making any actual changes and you'll be left with an exact copy of the original default workflow, except it has a different name.

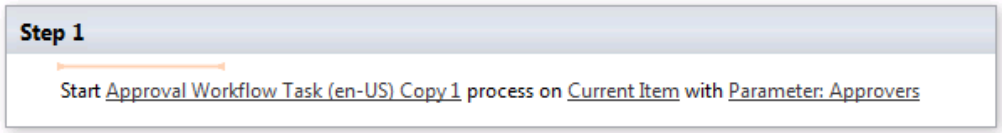


Figure 5.85 The action names may be automatically adjusted when you copy a reusable workflow. They names can be adjusted as required.

Adjust the workflow settings, conditions, and actions as required then save and publish the workflow like usual. The workflow will now be listed under Reusable Workflows along with the default global workflows (Figure 5.12).

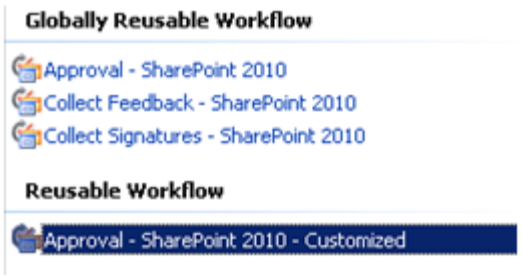


Figure 5.86 Copying a Globally Reusable Workflow results in a new Reusable Workflow.

Remember that Reusable Workflows are available only to the site they were created in and not the entire site collection while Globally Reusable workflows can be used throughout the site collection. To promote the copied and customized workflow to be Globally Reusable, open the customized workflow and return to the workflow editor. Then click the Publish Globally Ribbon button seen in Figure 5.13.

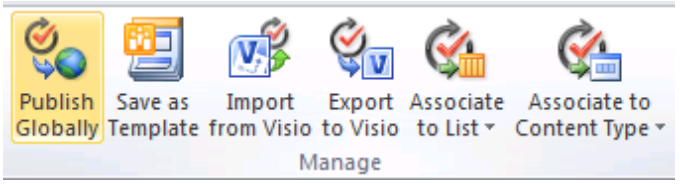


Figure 5.87 The Publish Globally Ribbon Button will promote a Reusable workflow to Globally Reusable and make it available across the entire site collection.

When publishing a reusable workflow globally you will always get a warning about making it available to all lists in the site collection and visible to all users. Click OK on this warning when you see it. Thereafter, the workflow will be copied again and published, but this second copy will be published as a Globally Reusable workflow. Note in Figure 5.14 that the Reusable workflow will not be deleted by this process and you will likely want to manually delete it to avoid any confusion.

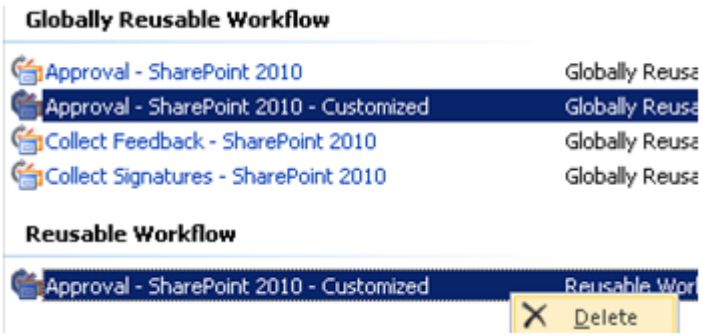


Figure 5.88 If the Reusable Workflow is not deleted, both workflows will be displayed on the Add Workflow screen. It is recommended that you delete the Reusable workflow to avoid confusion.

Using this process you can modify the out-of-the-box workflows to meet your organizations needs. This may mean that you make a single customized version for your entire organization or several versions for different purposes. In either case, starting with the out-of-the-box workflows may significantly reduce the amount of development time since much of the work has been completed already.

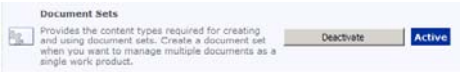
5.3 Workflow Actions for Document Sets

Document Sets are a new concept in SharePoint 2010. They represent a collection of documents, similar to folders, except that they allow for additional functionality. For example, it is possible to secure all of the documents in a document set at one time. If the security was applied to just a folder, documents within that folder could be breaking inheritance and would not be effected by the change. It is also possible to run workflows against document sets, which you cannot do with simple folders. Before we dig into document sets and workflows, we first need to learn how to create a document set in the first place. That will help set the stage for what the new actions bring to the table, as well as an example that puts them to use.

5.3.1 Creating Document Sets

Before you can interact with document sets in a workflow you must activate the Document Set feature and apply the content type to a document library. Follow these steps to enable document sets in your site. Note that you must be a Site Collection Administrator to activate the document set feature.

Creating a Document Set in SharePoint 2010

	Action	Result
1	Activate the Document Sets feature in your SharePoint site: A) In your root site collection site, browse to Site Settings under the Site Actions menu. B) Click the "Site Collection Features" link under the	The Document Sets feature will now become Active. 

Site Collection Administration category. C) Click the "Activate" button next to the Document Sets feature.	
2 Add the Document Set content type to a Document Library: A) In a document library such as Shared Documents, browse to Library Settings using the Ribbon. B) Click the Advanced Settings link under General Settings. C) Set "Allow management of content types" to Yes and click OK. D) Back at the Library Settings screen, under Content Types, click the "Add from existing site content types" link. E) Highlight Document Set in the available content types list and click the Add button, then click OK.	Document Sets can now be created on the library by clicking the New Document menu item and choosing Document Set (Figure 5.15).

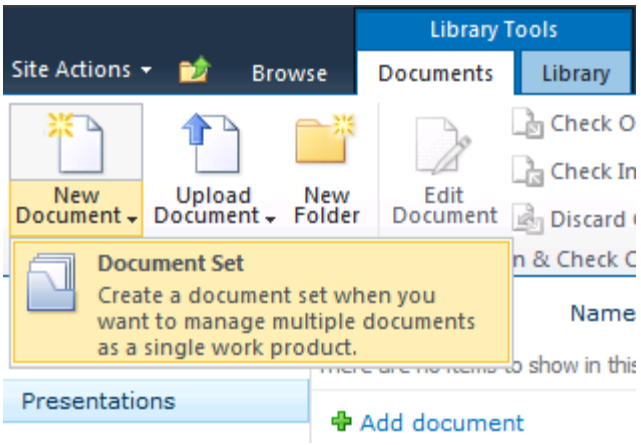


Figure 5.89 Document Sets are created using the New Document menu the same way that new documents are created, as long as they have been added as a content type to the list.

	Action	Result
3	Create a new Document Set:	The Document Set will be shown, with no documents. To upload

A) Click the New Document dropdown and select the Document Set content type (Figure 5.15). B) In the New Document Set dialog. give the document set a name such as "Sales Presentation" and description and then click OK.	documents into the document set, simply use the Upload Document Ribbon button.

5.3.2 Document Set Workflow Actions

With a document set in place, it's now time to transition toward how our SPD workflows can interact with it. In order to interact specifically with Document Sets, four new workflow actions have been created.

- **Capture a Version of the Document Set** - captures a snapshot of the document set and saves it in the version history.
- **Send Document Set to Repository** - moves or copies the Document Set to a SharePoint 2010 Records Center, which is used to permanently store documents for archiving purposes.
- **Set Content Approval Status for the Document Set - approves/rejects the set from within the workflow.**
- **Start Document Set Approval Process - allows the user to approve or reject the set.**
-

CAPTURE A VERSION OF THE DOCUMENT SET

The Capture a Version of the Document Set action captures the current state of either the Major or Minor versions of all documents in a document set and saves them to the document set's version history. This allows you to take a snapshot of the document set as a whole, instead of relying on the major and minor versions of the individual documents. The action accepts two parameters, type and comment (Figure 5.17). The type parameter determines the versions that will be captured. The choices include capturing the latest minor versions or the last major version. The comment parameter allows you to add a comment that will be stored along with the document set version that was captured.

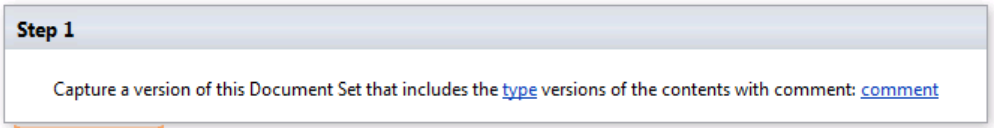


Figure 5.90 The Capture a Version of the Document Set action takes a 'snapshot' of all the documents in the document set and saves the set to the version history along with a comment.

After using this action on a document set, you can view the document sets version history using its context menu in the SharePoint interface. You'll see the latest major or minor versions, along with the comment, that was captured with the action. From here it is possible to restore the previous version of a document within the set or the entire document set.

The Capture a Version of the Document Set action could be useful when creating a presentation that includes several documents containing related information. As the presentation is developed and its

documents are modified, saving a snapshot of all related documents will allow you to more easily track the progress of the presentation. You can of course still manage the documents individually, including their version history if required. After capturing a version of the document set you can view it by opening the document set from within a document library. You'll notice that the Ribbon will include a new tab called Document Set. Opening this tab will reveal the Capture Version button, which is similar to the workflow action but is used manually. Also there is the Version History button, which will show you the captured document set versions (Figure 5.19).

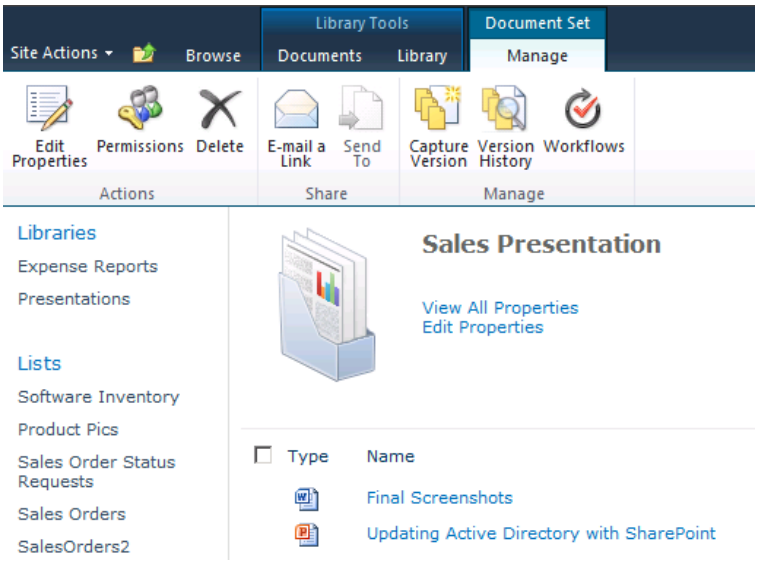


Figure 5.91 A Document Sets version history shows all of its related documents at a point in time, which individual document versions.

Clicking the Version History button for a document set will show you a version screen similar to the one used for documents, except that documents within the document set are also shown, along with their respective versions (Figure 5.20). Note that modifying the properties of the document set such as the set's title, will still generate a new version of the set. The documents will not be captured or shown in the history unless you use the Capture a Version workflow action or Ribbon button again.

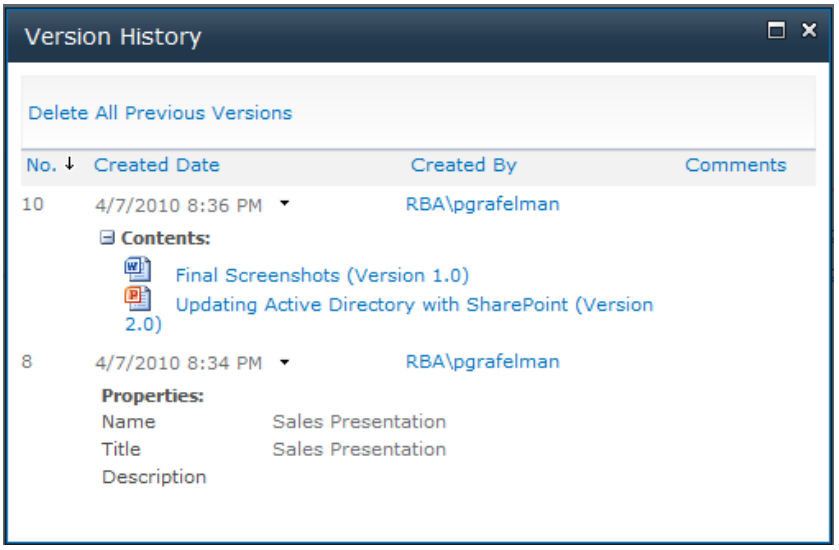


Figure 5.92 A Document Sets version history shows all of its related documents at a point in time, which individual document versions.

SEND DOCUMENT SET TO REPOSITORY

The Send Document Set to Repository action is used to send a document set to a Records Center, which is used for permanent storage of important documents. The action requires three parameters, this action, this destination content organizer, and this explanation (Figure 5.21). The 'this action' parameter determines if the document set will be copied, moved, or moved with a link (a link to the final destination is left in place of the original document). The destination content organizer is the address of the Records Center Router that you want to process the document set. A Records Center Router is configured within the Records Center to allow for the management and storage of multiple types of documents. See the example at the end of this section to learn how to configure a Record Center and Record Center Router. Finally, 'this explanation' is text that is associated with the record when it is placed in the Records Center.

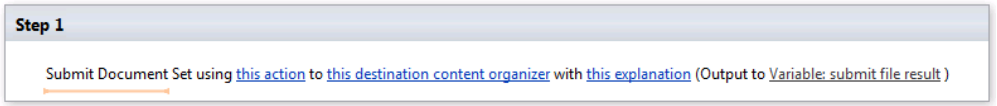


Figure 5.93 The Send Document Set to Repository action sends a document set to a record centers using data supplied in several parameters.

A Brief intro to Records Center in SharePoint

A SharePoint Records Center is used for permanent storage of important documents. Records Centers are usually used when a company has documents that are sensitive for compliance or legal reasons.

That company can use Records Center to house those documents in a non-editable fashion. Those documents can also have a retention policy such as seven years, and after which they are deleted.

The record center can be configured to manage multiple types of documents be creating rules. These rules allow documents to be sent to one common location by a workflow or by a manual process. Once in Records Center they can then be routed to the correct library based on the type of document or other metadata.

The Document Set can be copied, moved, or moved with a remaining link. The first two options are self explanatory, but note that a document set is converted to a Zip file when it is placed in a Record Center (Figure 5.22). Converting the document set to a Zip saves storage space and allows all of the documents to be downloaded at once. The Move and Leave a link option will store the document set in the Record Center but will leave behind a link in the list allowing users to still find the document set easily. The results of the action are stored to a variable, allowing you to take further action if the action fails or succeeds.

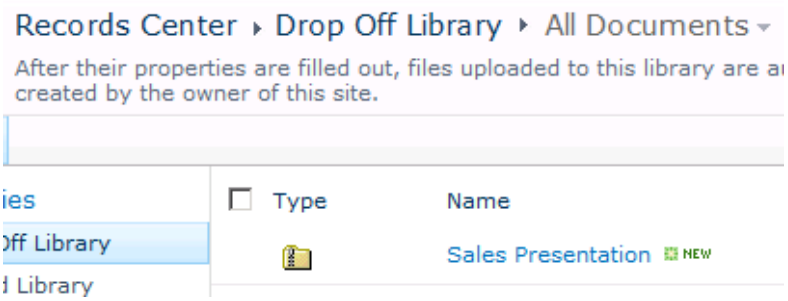


Figure 5.94 Document Sets that are saved to a Records Center are automatically converted to Zip files, allowing them to be downloaded as a set and saving storage space.

Set Content Approval Status for the Document Set

Document Sets, just like Documents, can require approval if their library is configured to require it. The Set Content Approval Status for a Document Set action is used to set the approval status of a document set. This process is usually done manually using the SharePoint interface, but this action allows it to be automated when required. The action only has two parameters, the status to apply and comments to include (Figure 5.23). The action can only be run against the current item and cannot be used to set the status of a document set in a different library.

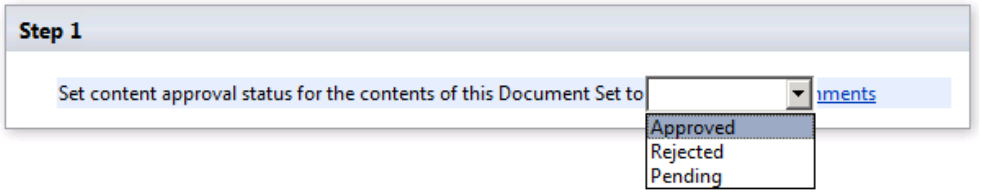


Figure 5.95 Document Set Approval status can be adjusted using the Set Content Approval Status for the Document Set workflow action.

This action would be useful to automatically approve or reject a document set after a given amount of time. It would also be wise to include logic that would stop the workflow if a user manually adjusted the approval status before the automatic approval is started.

START DOCUMENT SET APPROVAL PROCESS

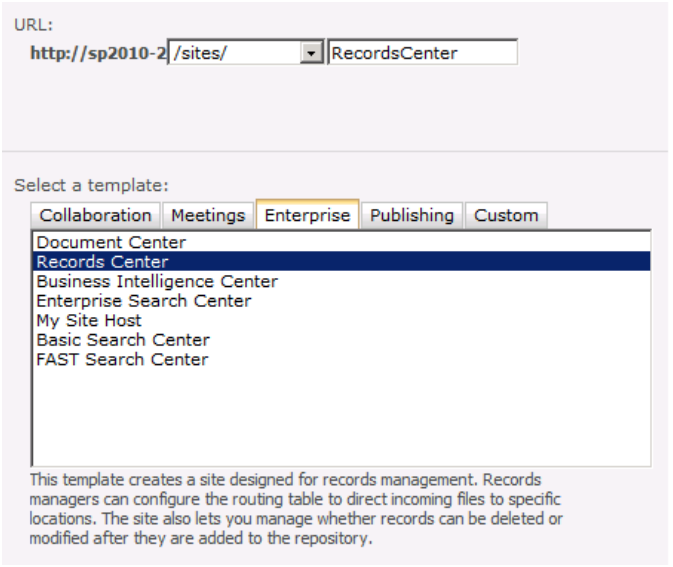
The Start Document Set Approval Process action is related to the Set Document Set Approval Status action, but it is used differently. Instead of performing the actual approval/rejection, this action is used to start an instance of the Approval workflow on the set, prompting a user or users to review and approve or reject the document set. The Approval link in is not related to a parameter; instead it opens a new instance of the default Approval workflow as a child to the original workflow.

5.3.5 Document Set and Records Center Workflow Example

The following example will illustrate the use of a workflow to send a Document Set to a Records Center. The scenario is that a document set is used to create and collect data required for a sales proposal. Once the proposal is complete it must be approved by a manager. Finally, after the proposal is accepted by the client, the document set must be sent into a Record to prevent changes for compliance and legal reasons. The first set of steps involves configuring a records center that we can route document sets to from our workflow. Follow these steps to configure a records center in SharePoint:

Provision a new Records Center in SharePoint 2010

	Action	Result
1	Create a new site collection using the Records Center site template: A) From within Central Administration -> Application Management, create a new site collection using the Records Center site template:	A new site collection will be created using the Records Center template.



2 `Create a new library in the records center to store our Sales Proposals document sets:` A) Browse to the newly created Records Center and select "Manage Records Center" under Site Actions. B) Click "Create records libraries" and then click "Record Library" to create a new library to store Document Sets. C) Name the library "Document Set Records". D) Add the Document Set content type to this new library using the Content Types settings on the Library Settings page, similarly to step 2 in 5.3.1.	A new document library will be created in records center that will receive routed document sets.
3 `Setup the set's content organizer rules:` A) Click Manage Records Center under Site Actions again and click "Create Content Organizer Rules". B) Click Add New Item and Fill in the Rule properties to match Figure 5.24 and Figure 5.25. Note: the Content Type must be set to "Document Set" and that the target location must point to the Document Set Records library that you created above.	After saving the rule, documents sent to the records center's "Drop Off Library" will be automatically moved to the Document Set Records library, created in step 2, if they have the Document Set content type.

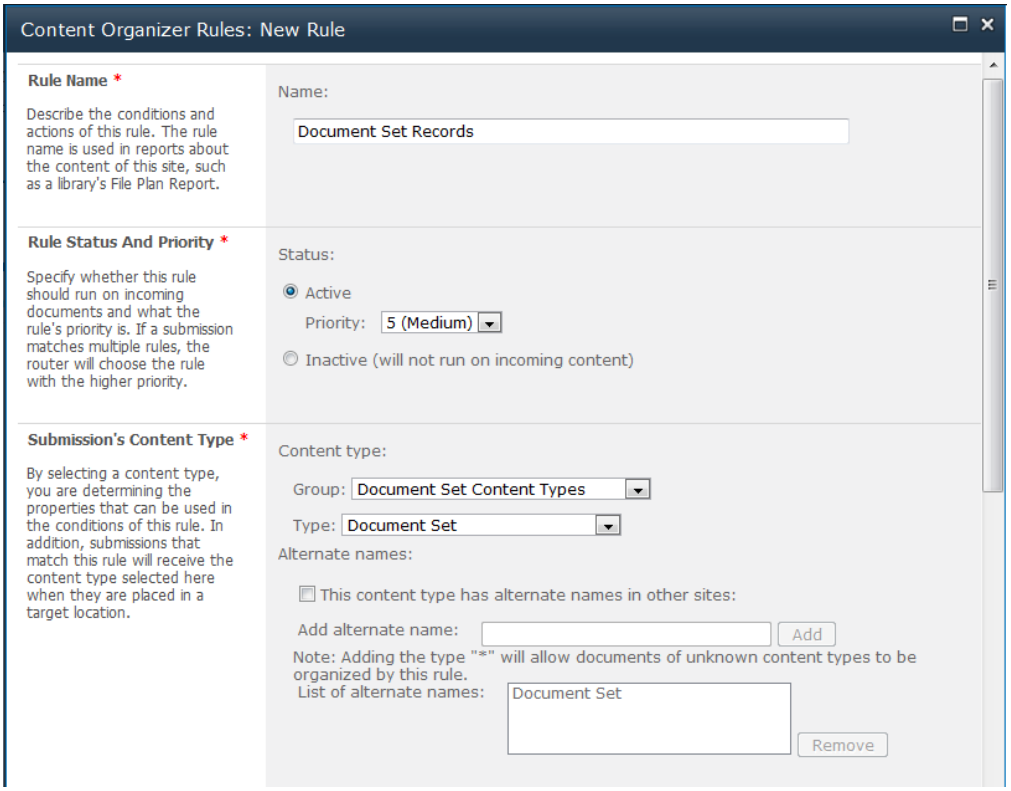


Figure 5.96 Records Center Rules are used to route submitted items based on their metadata. In this case we are going to route all Document Sets into a specific Record Library by examining the Content Type.

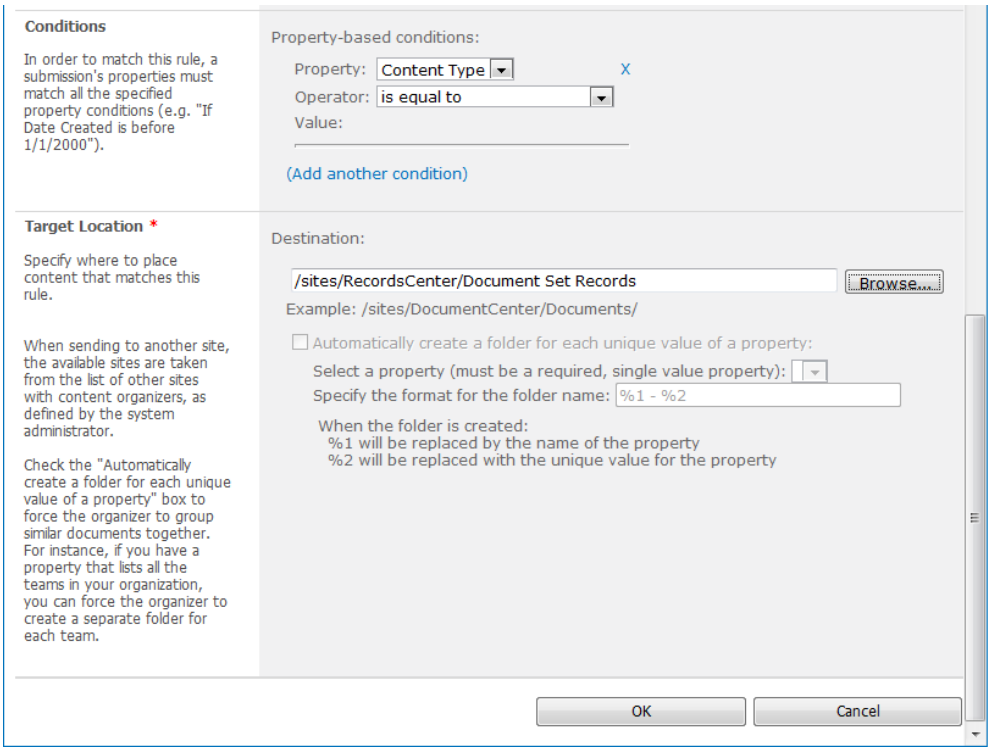


Figure 5.97 Additional conditions can be added using the Conditions settings if needed, but it is not required. It is required to indicate where the submitted items need to be stored. Note that this target location must be configured to allow the Document Set content type or you will not be allowed to save the new rule.

Now that you have configured a Records Center and have configured a library to use Document Sets you can use the new Document Set specific workflow actions. Records Center in and of itself is a very vast feature within SharePoint 2010, but for the scope of this book, we simply want to focus on routing documents via workflows. With Records Center all ready to go, we can finally build our workflow. Follow these steps to build a SPD workflow that routes Document Sets to records center:

Create a SPD workflow that routes document sets to Records Center

	Action	Result
1	Create a new document library to hold the proposals: A) Create a Document library called Proposals B) Add the Document Set content type to the library using the Library Settings menu. C) Under the libraries Versioning Settings option, set Require content approval for submitted items to Yes.	A new document library will be created to store customer proposals.

2	Create a new List workflow called "Approve Proposal": A) Using SharePoint Designer, create a new List workflow on the Proposals library called "Approve Proposal". B) Add a "Set Content Approval Status for the Document Set" action to the workflow and configure it to set the status to Approved and add a comment, as shown in Figure 5.26. C)Save and Publish the workflow.	A new workflow called "Approve Proposal" will be created that will handle the approval/rejection of the document set.

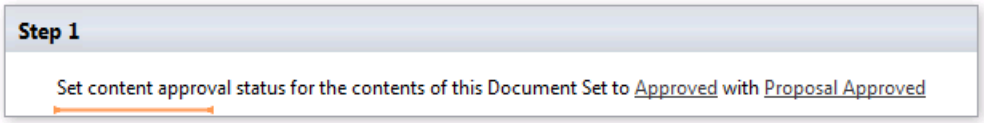


Figure 5.98 The Set Content Approval Status action can be used to set the status to Approved, Rejected or Pending to meet your needs.

	Action	**Result**
3	Create another workflow called "Send Proposal to Records Center": A) Create another List workflow on the Proposals library called "Send Proposal to Records Center". B) Add a "Send Document set to Repository" action to the workflow and configure it to move the document set to the Records Center "Drop Off Library" (Figure 5.27). C) Save and Publish the second workflow.	A second workflow will be created titled "Send Proposal to Records Center". This workflow will route the document set to Records Center.

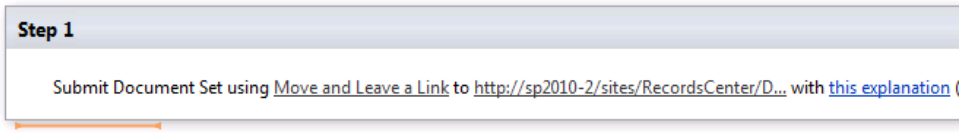


Figure 5.99 The Send Document Set to Repository can copy, move, or move and leave a link. In this case it is set to Move and Leave a link to allow users to find the Proposals easily after they are sent to the Records Center.

Action	**Result**

4	Create a new document set called "Acme Proposal": A) Within the Proposals document libraries, create a new Document Set named "Acme Proposal" by using the New Document Set option under the New Document Ribbon button. B) Add some documents by opening the document set and using the Upload Document Ribbon Button.	A new document set will be created that our two workflows will execute on.
5	Run the Approve Proposal workflow: A) Switch to the Manage tab and click the Workflows Ribbon Button. B) Run the Approve Proposal workflow.	The status of the document set will change to Approved
6	Run the Send Proposal to Records Center workflow.	The Document Set will be moved to the Records Center and the will change to a Document Link in the Proposals library

5.4 Workflow Actions and Conditions for Security

Sites, Libraries, and even documents can all be managed with different permissions. Additionally, multiple levels of permissions are available right out-of-the-box, such as Full Control, Designer, Contributor, and Reader. Also custom permission levels can be created to suit your needs. New to SharePoint Designer 2010 is the ability to manipulate the permissions on list items within workflows. This previously could only be done manually within SharePoint or using custom .NET code. It is also now possible to run workflow actions as a different user, taking advantage or their permissions. All this is done using Impersonation Steps and a slew of new security related actions and conditions within SharePoint Designer.

5.4.1 Impersonation Steps

When a user starts a workflow, most of the actions taken by the workflow will be attributed to that user. In some cases, the user running the workflow may not have the permission required to execute the required action. For example, you may want to prevent users from deleting list items directly, but you still want to allow a workflow to delete the item. If you remove the user's Delete permission level, they will not be able to run a workflow that deletes an item. If this is not the desired behavior, you can use Impersonation Steps to allow the user to run the delete action as another user that has the necessary permissions to delete items. There are a few limitations around impersonation steps that you need to be aware of:

- The impersonation step can only be run as the user that is authoring the workflow. You cannot set the step to run as a different user.
- Messages logged to the workflow history as part of an impersonation step will still be logged as System Account and not the workflow author.
- You cannot create an impersonation step within another step; they can only be created as top

level steps.

The following figures show the creation of two new list items in the Sample List. The first list item is created as part of a standard step and the user that started the workflow will be listed in the Created By and Modified By fields. The second item is created as part of an Impersonation Step and the workflow author, Paul Grafelman, will be listed in the Created By and Modified By fields. Figure 5.28 shows the two steps and their create list item actions.

Although Todd Rowe ran the workflow that created the list items in Figure 5.29, the second item created (the first item shown below) was created as part of the impersonation step and is shown as created by Paul Grafelman, the author of the workflow.

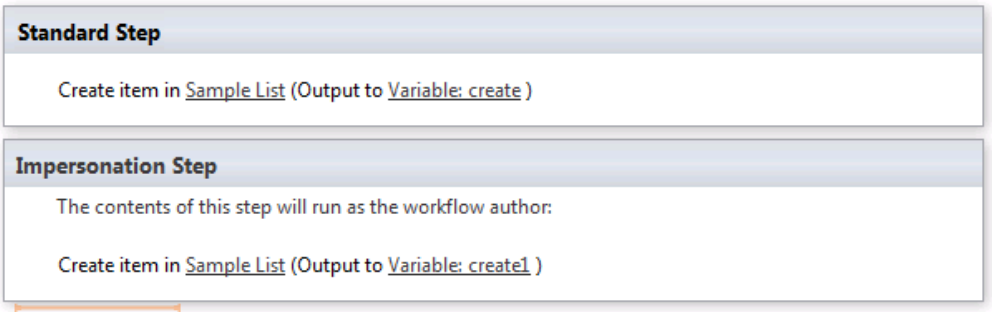


Figure 5.100 Although the actions within are nearly identical, the Author will be different for each item.

Figure 5.101 Although the two items were created by a single workflow, only the item created by the Standard step is attributed to the end user.

5.4.2 Security Related Conditions and Actions

In addition to the impersonation step there are several new security related conditions and actions available in SharePoint Designer workflows. These allow a workflow to not only inspect a user's permissions for an item or document, but also modify those permissions. Note that these additional conditions and actions are only available within an Impersonation Step and cannot be added to a conventional Step.

CHECK LIST ITEM PERMISSION LEVELS CONDITION

The Check List Item Permission Levels condition checks to see if the specified user has been given explicit permissions on the specified list. The first parameter indicates which user's permissions should

be checked. The second parameter indicates what permission level is being checked for. Finally, the third parameter indicated which list item is being examined. The list item can be set to 'Current Item' or to an item in any other accessible list. The following table shows the results of some different condition configurations.

> **Note: Inherited versus Explicit permissions**
>
> The two security related conditions only look at explicit permissions. Inherited permissions will not be taken into account.

Table 5.1 this table describes how user permission levels are compared in an impersonation step.

User's Item Permission Level	Condition Checks For	Condition Result
Contribute, Approve	Approve	True
Contribute, Approve	Full Control	False
Full Control	Contribute, Approve	False

The most interesting thing in Table 5.1 is how if the user has full control on the item, but the condition is checking for say Contribute, the condition will return false. This is interesting because you're typically given contribute rights when given full control to an item. If this is problematic for your workflow use the next condition, Check List Item Permissions.

CHECK LIST ITEM PERMISSIONS CONDITION

The Check List Item Permissions condition checks to see if the specified user has the specified permissions on the specified list item. It is configured the same way as the previous condition, but the results are potentially different. This is because it is not looking for the exact permission specified. Instead, it is checking to see if the user has at least the permission specified. The table below shows some examples.

Table 5.2 this table describes how user permissions are compared in an impersonation step.

Users Item Permissions	Condition Checks For	Condition Result
Full Control	Contribute	True
Contribute	Read	True
Read	Contribute	False

ADD LIST ITEM PERMISSIONS ACTION

The Add List Item Permissions action is used to give a user explicit permissions for a list item. This will also have the effect of breaking inheritance on the item's permissions. The specified permissions will then be added to existing permissions. It can also be used to give multiple users permissions for the list item at one time (Figure 5.30).

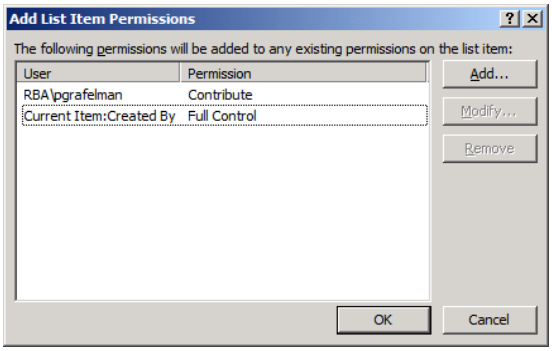


Figure 5.102 Multiple users can be given a variety of permissions with one Add List Item Permissions action.

INHERIT LIST ITEM PERMISSIONS ACTION

The Inherit List Item Permissions action removes all explicit permissions for the item and sets it to inherit permissions from its parent. An example of this is a workflow that sets explicit permissions on a list item while the workflow is running in order to prevent anyone from editing the item. Once the workflow is complete users need to be able to edit the list item again. Using the Inherit List Item Permissions action would allow anyone with contribute or edit permissions on the list to again edit the list item.

REMOVE LIST ITEM PERMISSIONS ACTION

The Remove List Item Permissions action removes the specified explicit permissions from the list item. Other explicit permissions will not be removed. Note that if the item has inherited permissions, this action will break the inheritance.

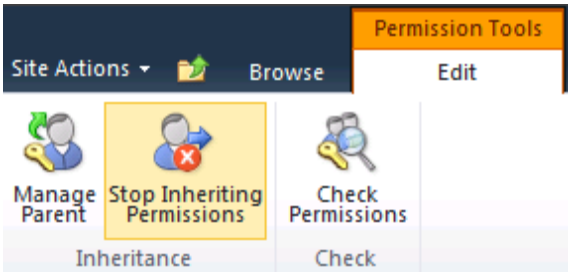
REPLACE LIST ITEM PERMISSIONS ACTION

The Replace List Item Permissions action removes all existing explicit permissions, if any, and applies the permission specified in the action instead. This action will also break inheritance if it is in place when the action starts. Also, a Workflow Users SharePoint group will be added to the list item automatically. This group will allow users that do not have access to the parent list to be added to the list item permissions through workflows.

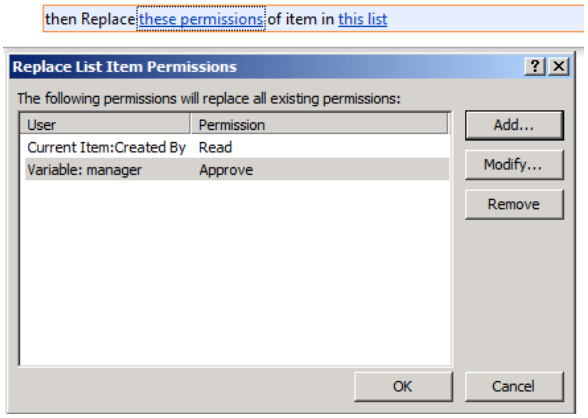
5.4.3 Working with Permissions and Security in a Workflow

This workflow example will demonstrate the use of the above conditions and actions to allow a user without permissions to a list to approve an item. The scenario is an expense report approval system in which employees need to be able to submit expense reports to a document library, but they should not have access to view other employee's reports. In addition, the employee's manager must be given the ability to approve expense reports submitted by their employees only. Managers should not be able to read the reports submitted by other employees. To accomplish this, the workflow will include actions that will break inherited permissions and give the submitting user Read access only. In addition, the user's Manager will be given Approve permissions on the item. This will all happen automatically when the item is uploaded. To do this, follow these steps:

Working with Security related Actions and Conditions in a SPD workflow

	Action	Result
1	Create a document library called Expense Reports and configure security on that library: A) Create a new document library called "Expense Reports" and navigate to the library's Library Settings page and then choose *Permissions for this document library*. B) Click the "Stop Inheriting Permissions" Ribbon Button:  C) Select all users and groups in the permissions list and click the "Remove User Permissions" Ribbon Button to remove all existing permissions from the library. D) Click the Grant Permissions Ribbon Button and type "NT AUTHORITY\AUTHENTICATED USERS" in the Users/Groups field and then click the check box next to Contribute, followed by clicking OK.	A new document library will be created for users to upload their expense reports. The library is set to not inherit permissions for its parent site and all authenticated users have the ability to upload documents.

2	`Create a new SharePoint Designer list workflow to manage the approval of expense reports:`
	
	`A)Create a new List Workflow on the Expense Reports library titled "Submit Expense Report for Approval".`
	
	`Note:` Be sure you are logged in as a user with Full Control on the new list, such as a site collection administrator
	
	`B)` Add an impersonation Step to the workflow:
	
	 
	
	C) Add a 'Lookup Manager for a User' action into the impersonation step and set it to lookup the manager of the user that created the current item:
		A new SPD workflow will be created that will break inheritance on the expense report and add the submitter as a reader on the report, and will also add the submitter's manager as an approver.

	Find Manager of Current Item:Created By (output to Variable: manager) D) Add a Replace List Item Permissions action and set it to give the user that created the item Read permission and that same user's manager Approve permissions:  Note: The user that created the item has Contribute permissions on the list item, but to prevent them from changing a submitted report, they will be given Read permissions. The users Manager is then given Approve permissions	
3	Publish the workflow: A) Set the workflow to run automatically when an item is created. B) Click Publish.	The workflow is set to automatically start and is published onto the Expense Reports library.

Notice in Figure 5.31 that after the workflow runs you are no longer able to edit the document, but you are still able to view it. This is apparent because the Delete and Edit item options are disabled. Also notice that the manager of the submitting user now has approval rights on the document and neither user has access to other documents in the library. Sign in as the user's manager to verify that Approve is available when viewing the item.

These new conditions and actions will allow you to give access to otherwise non privileged users dynamically and automatically. This workflow could be extended to re-establish edit rights for the user if the manager rejects the item (allowing the user to made changes again).

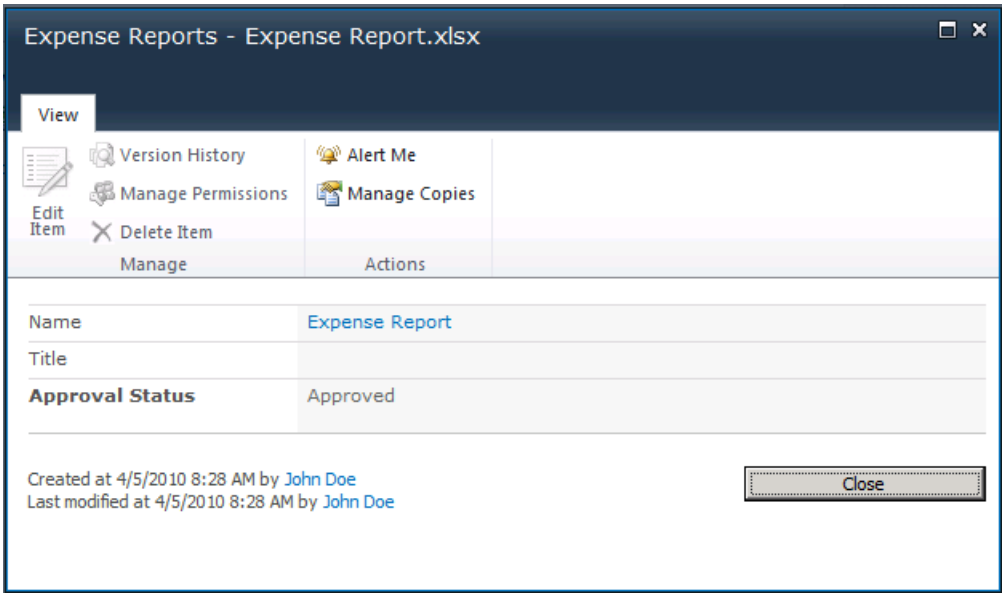


Figure 5.103 The Edit Item and Delete Item links are now disabled because John Does permissions have been removed by the workflow.

5.5 External Data in a SharePoint Designer Workflow

So far, all of the workflows you have looked at in this book have referenced and modified SharePoint list data. It is also possible to use SharePoint workflows to interact with external data, such as SQL databases. Using a new concept called External Content Types it is possible to display and create external data using the SharePoint environment. By extension, it is also then possible to write workflows that leverage this external data. It is not possible to create SharePoint Designer workflows that interact directly with external data. Also, workflows cannot be assigned to lists that use External Content Types. Despite these limitations there is a work around to write workflows that interact with External Content types and even create data in lists that contain external data.

In order to display external data and create workflows that use this data, external content types must be created. The most important thing to know when creating an external content type to use with workflows is that they must be configured to use the Secure Store Service when authenticating. This is the only supported authentication method when workflows are interacting with the data. It is also important to know that the service account is use by the SharePoint application pool will be used to access the external data and not the current user's credentials. This will require that the service account have read or read/write access to the external data in question. Each of these topics are moderately complex and beyond the scope of this book, so instead of describing them in detail a specific example is shown which explains one path to complete configuration. This example is broken into sections for each major topic. The example uses the AdventureWorks database, which is included as a test database with SQL server, to display and edit sample purchase order data.

5.5.1 Configuring a Secure Store Service

This section will describe how to configure the Secure Store Service. The Secure Store Service is used to store credentials that will be used when accessing other systems. This can be useful when individual users do not themselves have credentials to access the back end system, which is the case in this example.

Configure Secure Store Service application

	Action	Result
1	Create a new target application in the default Secure Store Service application: A) In Central Administration, click Manage Service Applications and the Scroll down and click Secure Store Service. B) Click the New button on the Ribbon to create a new target application. C) Enter the values shown in Figure 5.32 for the ID, Name, E-mail and be sure to select the Group type, which indicates that this Application will use shared credentials for all users and then Click Next. D) Add the account you are logged into Central Admin with as an administrator and All authenticated Users in the members box, shown in Figure 5.33, and then click OK.	A new target application will be half-way configured with the general settings configured and the administrator and members configured.

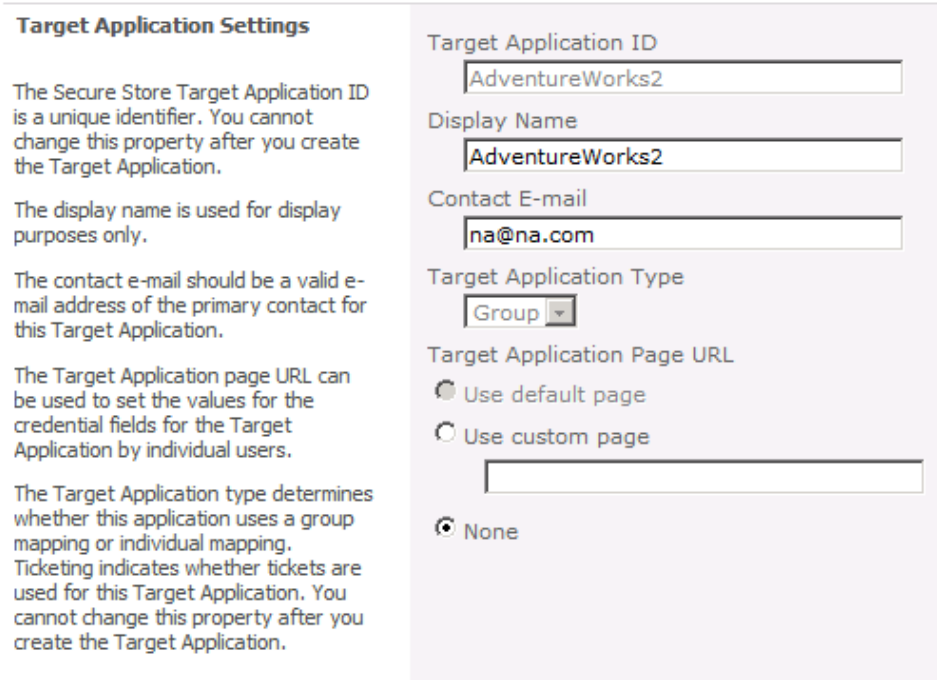


Figure 5.104 Enter a name and email address for the application, along with the Group application type.

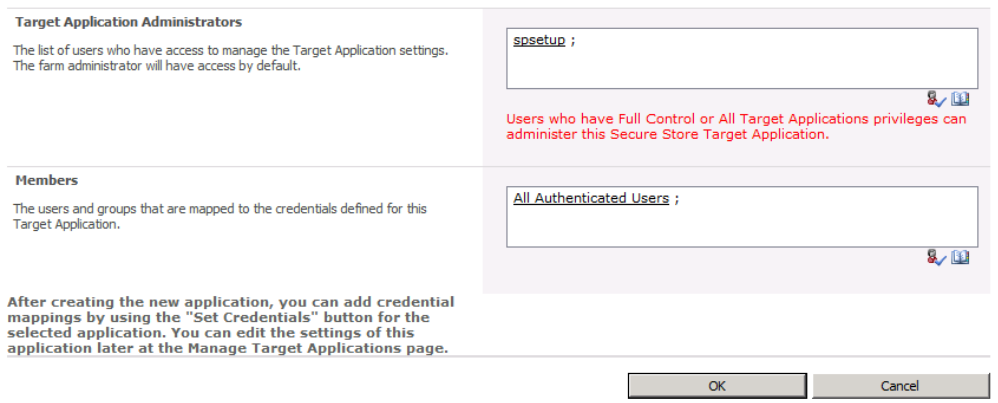
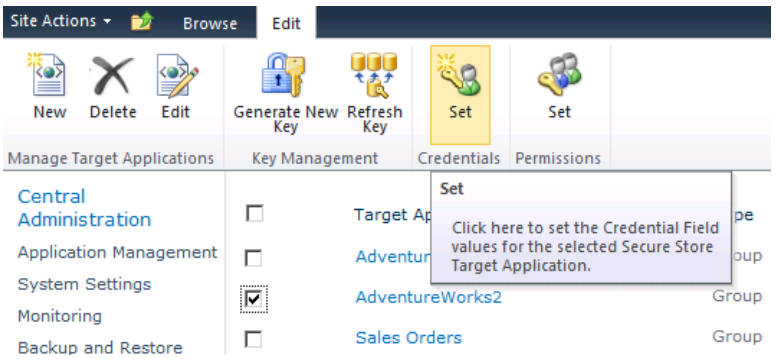


Figure 5.105 Adding All Authenticated Users to the Members field will allow anyone to use this Secure Store Application, and BCS applications associated with it.

	Action	Result
2	Continue configuring the new target application: A) Leave the default values as shown in Figure 5.34, which determine the credential fields that will be stored and click Next. Note: In most cases the default User Name and Password fields are the only fields that need to be stored in order to access a back end system. The Secure Store Service can also be configured to store additional credential values, such as a PIN number or a Key. If the Masked setting is enabled for a field, the typed text will not be displayed when entered by the user. This is commonly used on password entry fields to prevent anyone from reading the users password. B) To allow all system users to authenticate using this application, add yourself or an applicable account in the administrator field and add 'All Authenticated Users' to the Members field and then click OK. Note: this may not be appropriate for your production environment!	The target application named AventureWorks2 will be created and all authenticated users will be able to leverage the credentials to connect to remote data.

Figure 5.106 The standard username and password are stored by default, but other fields can be stored if needed. For this example, use the default username and password only.

	Action	Result
3	Set the credentials of the new target application: A) Select the Application you just created by clicking the check box and click the Set Credentials button in the Ribbon:  B) Enter the credentials for an account that has permissions to read the external database. Note: In Figure 5.35 the spsvc account has read and write permissions. You may need to check with your SQL DBA to confirm that the correct permissions are in place C) Click OK to save the Credentials for the Secure Store Application.	The actual credentials are now stored in the newly created target application.

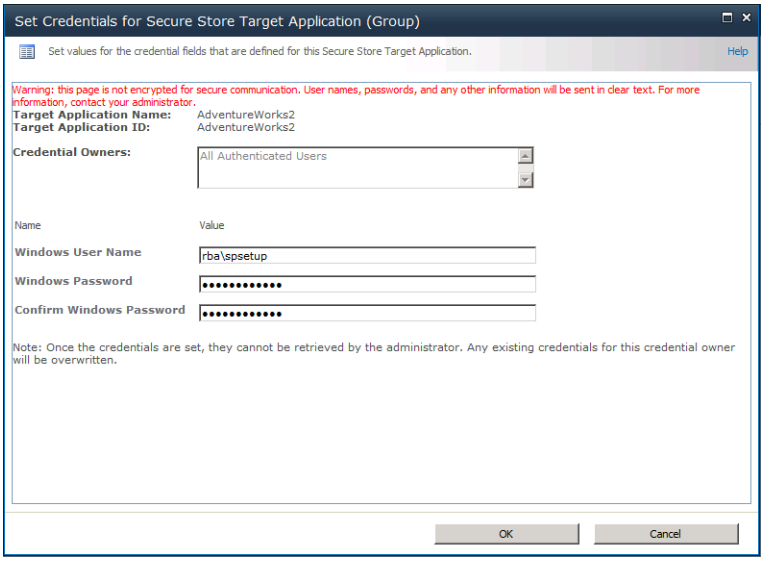


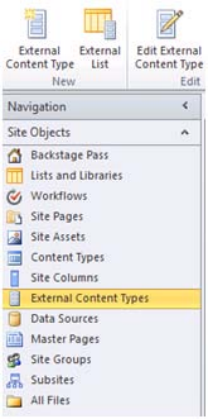
Figure 5.107 After setting permissions, which determine who can use the Secure Store Application, the Credentials that are used to connect to SQL must be set. This is done using the Set Credentials button.

5.5.2 Creating an External Content Type

The Secure Store Service has now been configured and it is time to create an External Content Type. Remember that the Secure Store Servers simply defines what credentials we will use to connect to the external data source. The External Content Type will do the actual work of retrieving, updating, and deleting data within SharePoint. Follow these steps to create a new External Content type:

Create a new external content type

	Action	Result
1	Open SharePoint Designer and create a new External Content Type within your SharePoint site: A) Click the External Content Types link under Site Objects on the left navigation:	An external content type will be created that points to a SQL database.

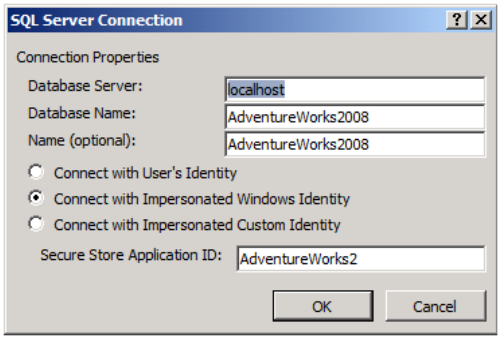


B) Click the External Content Type link in the Ribbon, which will create a new External Content Type.

C) Give the new content type a Name and Display Name and click the link next to External System.

D) Click the Add Connection button and choose SQL Server from the dropdown. Click OK.

E) Fill out the form as follows and then click OK:

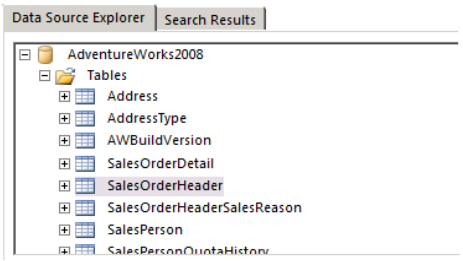
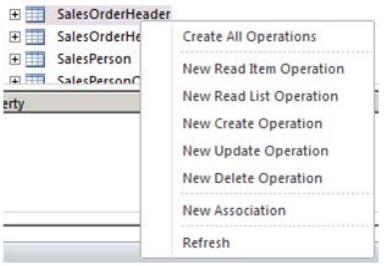


Note: be sure to specify the appropriate SQL Server name, which may or may not the "localhost". Be sure to select Impersonated Windows Identity and enter the name of the Secure Store Application that you created earlier.

2 `Configure the new External Content Type to pull out Sales Order information out of the database:`

A) Expand the Database that you just added and open the Tables folder.

The content type is now specifically pulling records out of the SalesOrderHeader

	Scroll down until you find the SalesOrderHeader table:  B) Right click the SalesOrderHeader table and choose Create All Operations which will create all of the necessary connection settings to Read, Create, Update, and Delete data from the SQL database:  	table.
3	`Complete the 'All Operations' wizard:` A) Click Next on the first two screens of the 'All Operations' wizard. B) On the Filter parameters screen click the Add Filter Parameter button. C) Click the 'Click to Add' link next to the Filter definition and set the filter to match Figure 5.36, and then click OK. D) In the Default Value dropdown, type '100' and click OK and then click Finish. Note: by specifying a default value of 100, only 100 records will be returned by the database. Returning all records can cause time-out errors in some situations and setting a limit in this manner can prevent these errors.	After the wizard completes, all the necessary Business Connectivity Services will be created on your behalf that the content type is using to retrieve the external data.

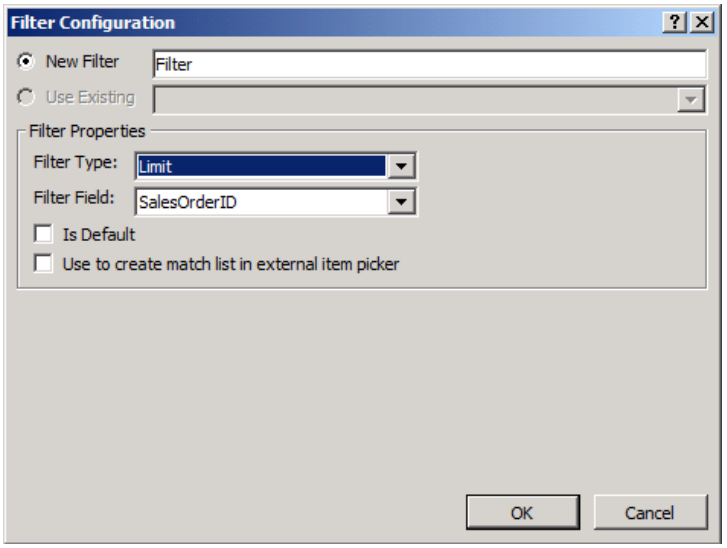
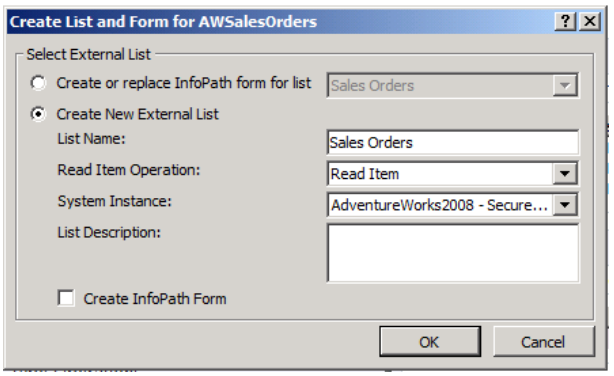
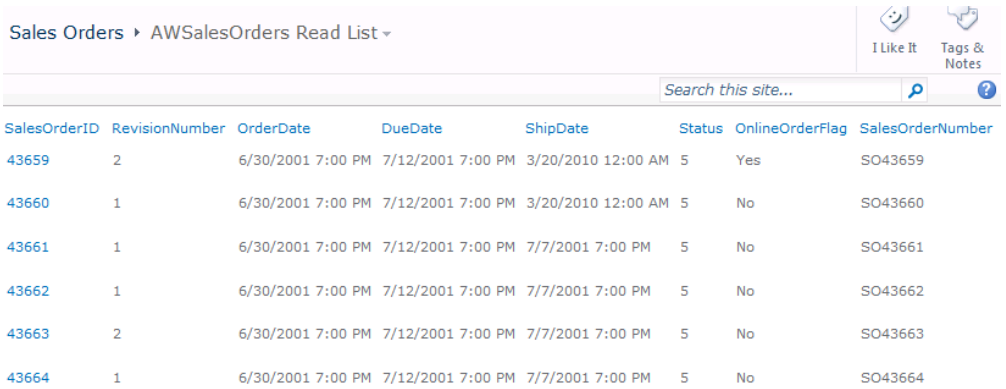


Figure 5.108 Adding a Limit filter will ensure that your application does not return too many records that may cause a timeout.

	Action	Result
4	Create a new external list of the newly created external content type: A) Click the Create Lists and Form button on the Ribbon and if you see a popup message about saving the content type, click OK. B) Give the list a name of "SalesOrders" and click OK: 	This will create a new custom list in your SharePoint site and configure it to use the external content type you just created

You have now configured an external content type and created an external list that uses that content type. Browse to the new list in SharePoint, you should see data from the AdventureWorks SalesOrderHeader table displayed similar to Figure 5.37.



SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber
43659	2	6/30/2001 7:00 PM	7/12/2001 7:00 PM	3/20/2010 12:00 AM	5	Yes	SO43659
43660	1	6/30/2001 7:00 PM	7/12/2001 7:00 PM	3/20/2010 12:00 AM	5	No	SO43660
43661	1	6/30/2001 7:00 PM	7/12/2001 7:00 PM	7/7/2001 7:00 PM	5	No	SO43661
43662	1	6/30/2001 7:00 PM	7/12/2001 7:00 PM	7/7/2001 7:00 PM	5	No	SO43662
43663	2	6/30/2001 7:00 PM	7/12/2001 7:00 PM	7/7/2001 7:00 PM	5	No	SO43663
43664	1	6/30/2001 7:00 PM	7/12/2001 7:00 PM	7/7/2001 7:00 PM	5	No	SO43664

Figure 5.109 An External List displaying sales order data from the adventureworks SQL Database in SharePoint.

5.5.3 Creating a Workflow using the External Content Type

Now that you have created a Secure Store Application, External Content Type, and a List that uses that content type, you can create a workflow that leverages the external data. In this example you will create an Order Status Request system. This will allow a user to request the status of an adventureworks order by entering an order ID. The workflow will look up the order using this ID and email to results to the supplied e-mail address.

Create a new workflow that leverages external data

	Action	Result
1	Create a new list to store sales order requests:: A) Create a new custom list called list "Sales Order Status Requests". B) Create two columns on this list, one called "E-mail address" that's a single line of text column, and the other called "Sales Order Lookup" and configure it as a Lookup type column. C) While configuring the lookup in the *Get Information From* dropdown, choose the "Sales Orders" list and check the box next to SalesOrderID. Note: this will instruct the lookup to get items from the Sales Orders external list, will show the user all of the available Sales Order ID's when they use the field. D) Leave all the rest of the defaults on the create lookup column screen and click OK.	A new list will be created that users can use to request the status of an order. The list should look similar to Figure 5.38.

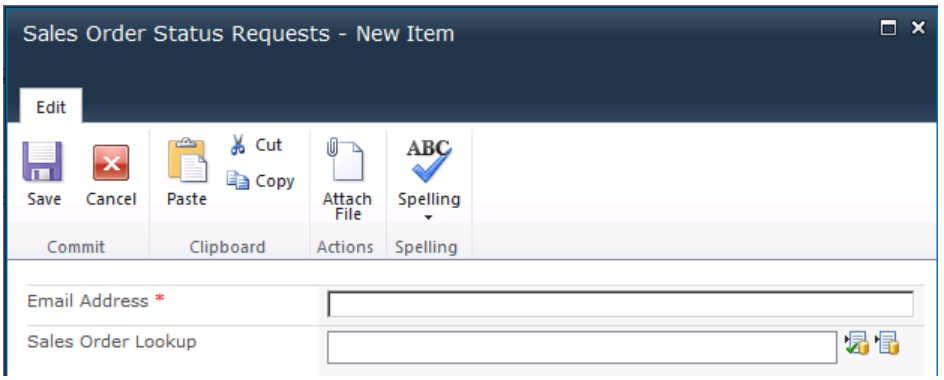


Figure 5.110 The user can validate a lookup value using the Validate link to the right of the field.

Notice the two icons next to the lookup column in Figure 5.38. The first icon allows the user to validate a value that they typed manually. The second button will show a popup where they can select from all available Sales Orders (Figure 5.39). This will help ensure that invalid Sales Order ID's are not passed to the workflow and allows the user to see additional data to help them select the correct item.

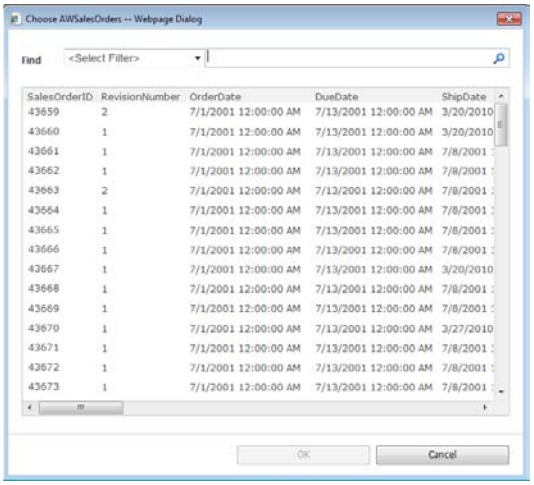
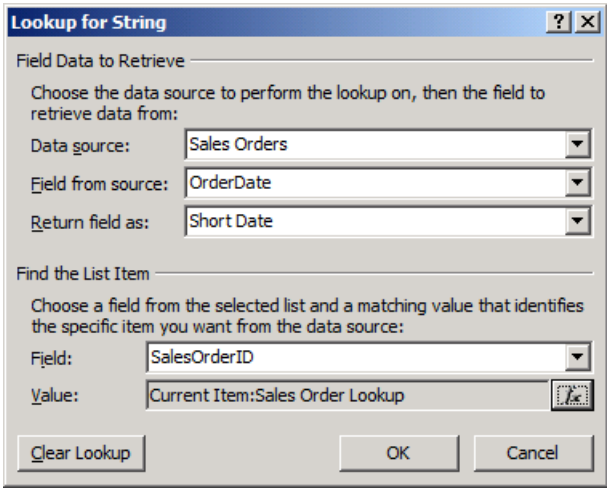


Figure 5.111 This dialog will allow the user to see additional data, helping them select the item they want.

	Action	Result
2	Create a new workflow named "Sales Order Status Request": A) In SharePoint Designer, create a new List Workflow using the "Sales Order Status Requests" List and name it "Sales Order Status Request".	A new workflow will be created that when run will

B) Edit the workflow and add a single Send an Email action. C) Configure the email action to send using the "Email Address" from the status request in the To: field. Also add a descriptive subject line D) In the Body of the email, add information about the order the requestor is requiring by using the Add or Change Lookup button and looking up information in the Sales Orders external list. Note: You can add references to any field that was included in the external content type created earlier. Be sure to configure the Data Source to reference the 'Sales Order' list that was created as part of the external content type configuration. To find the correct sales order item, use the "Current Item:Order ID" field and the SalesOrderID from the Sales Order list. You will need to configure this lookup for each field that you want to return. If you see a warning about returning unique items, click OK. You will always see this warning when working with external data. The figure below shows a configured lookup that's getting the order's order date:  E) Publish the workflow.	email the requestor the status of the sales order.

To test the workflow, create a new list item in the Sales Order Requests list and enter a Sales Order ID manually or use the popup to select one and save the Request item. Manually initiate the workflow. You should again receive an e-mail message with the relevant Order Date and Ship Date, or whatever looked up external order information you entered into the body of the email.

5.6 Summary

This chapter walked you through five advanced techniques with SharePoint Designer 2010 workflows. The first of the five involved creating workflow templates. You can save a workflow as a template, and then deploy that template across the entire farm, or to a separate farm entirely. The template is saved as a WSP file, and deployed using the stsadm command line utility.

Another advanced trick is to customize the out-of-the-box workflows. With SharePoint Designer 2010, you can now extend the default workflows if they don't meet you needs. Rather than start from scratch, you can make whatever tweak are necessary, thus saving you time.

SharePoint Designer 2010 also comes with a bunch of new workflow actions and conditions around working with Document Sets and Security. You can use workflows now to route document sets to Records Center. You can also use workflows to manage security, such as breaking permission inheritance or adding/removing user permissions.

Lastly, we took a look at how to leverage external data in your SharePoint Designer workflows via the Business Connectivity Services suite. With SharePoint 2010 you gain the ability to not only quickly configure access to external data, but you can read and modify that external data through SharePoint and SharePoint Workflows.

6

Custom Visio SharePoint Workflows

Designing and developing SharePoint workflows quite often require strong communication and collaboration skills. This is needed to clearly define and document what the stakeholders are asking for and what the developer needs to build. Sometimes someone that is not very technical is the best choice for gathering these workflow requirements. With Office Visio 2010, a non-technical person can create what's called a Visio SharePoint workflow. They can then hand this Visio workflow off onto someone more technical in nature. That more technical person can then import the workflow in SharePoint Designer and fill in all the details. This act of importing can be a great time and cost saver. Rather than have the developer start from scratch, he or she can start where the business analyst left off.

Office Visio 2010 is an application apart of the Microsoft Office product stack. Specifically, Visio is used to create diagrams. Some example diagrams that be created in Visio are:

- Diagrams showing an organizational hierarchy (reporting structure)
- Floor plans for office buildings
- Network diagrams showing servers, firewalls, etc.
- Gantt charts and timelines for project managers
- Flowcharts (aka, workflows)

A flowchart is the foundation of every workflow. If you remember back to chapter 2, we discussed using a "napkin drawing" to first get the "flow of work" on paper. These are the requirements and the purpose of the workflow. Office Visio is a great tool to use to document these requirements, because (as is true in many cases) a picture is worth a thousand words.

In this chapter you will learn the steps to build a workflow using Visio. Unfortunately, you can't really create a fully functional workflow in Visio, but you can certainly use it as a starting point. There are some new features with Visio 2010 that makes all this possible. Namely, there is a new diagram template called the SharePoint Workflow template. Inside this template there are connectors and actions that can be used to model your workflows.

After you have your diagram built out, you can then import that diagram into SharePoint Designer, and thereafter publish to SharePoint. Another great new feature is the ability to use this diagram as a status indicator of the workflow. After the workflow starts, the diagram will show where in the process the workflow is currently executing.

6.1 Introducing Visio Workflows

Visio SharePoint workflows are a way for a business analyst to diagram the workflow that is to be built in either SharePoint Designer or Visual Studio. A Visio workflow in and of itself can't do anything is SharePoint. It's just a diagram. The "power" of the Visio workflow is that it can be imported into SharePoint Designer (and subsequently Visual Studio). With this import feature, now a more technical person can add the technical aspect into the workflow and publish it into SharePoint.

Before we get into Visio SharePoint Workflows, let's take a brief look at what is new in the latest version of Visio, in general. Visio 2010 focuses on three major areas of investment; Ease of Use, Process Management and Visio Services.

Ease of Use - Visio 2010 incorporates the Office Fluent User Interface and design philosophy, which is also more commonly called "the ribbon". The improved organization and presentation of Visio's capabilities helps you complete tasks and create better looking diagrams with greater efficiency. The Shapes Window gets a new look and new capabilities to make organizing shapes and adding them to the drawing even easier. Within the drawing window Microsoft added productivity improvements like shape insertion and automatic alignment and spacing to speed up initial diagram creation and assist with editing and maintaining diagrams over time. You will find these improvements saving you time as soon as you start using them, it has been awhile since Visio has had a major enhancement so it may be confusing at first.

Process Management - Visio 2010 delivers a great new experience for working with process diagrams. Microsoft has redesigned the cross-functional flowcharts to be simple, scalable, and reliable. Also, added are new diagram types for the Business Process Modeling Notation standard and for designing SharePoint workflows, which can be configured and deployed with SharePoint Designer 2010. Sub-processes and containers break up a diagram into understandable pieces, and the Validation feature can analyze a diagram to ensure it is properly constructed. Visio integrates with SharePoint to provide a process diagram library for centralized storage of process documents.

Visio Services - Visio 2010 can take data-refreshable diagrams and publish them to SharePoint for broad distribution to anyone with a web browser. Visio Services performs data refresh and rendering on the server and delivers up-to-date diagrams in the browser. The diagram author no longer needs to repost the diagram every time the data changes, and diagram viewers no longer need the Visio client to see the diagram. This is much the same as Excel services in SharePoint 2007, and even easier to get setup and running.

In Visio Premium 2010, Microsoft has introduced many new high quality drawing templates. One of these is a new template called the Microsoft SharePoint Workflow template (Figure 6.1). The first thing

you will notice when you create your first Visio SharePoint workflow diagram is new improved menu navigation (Figure 6.2).

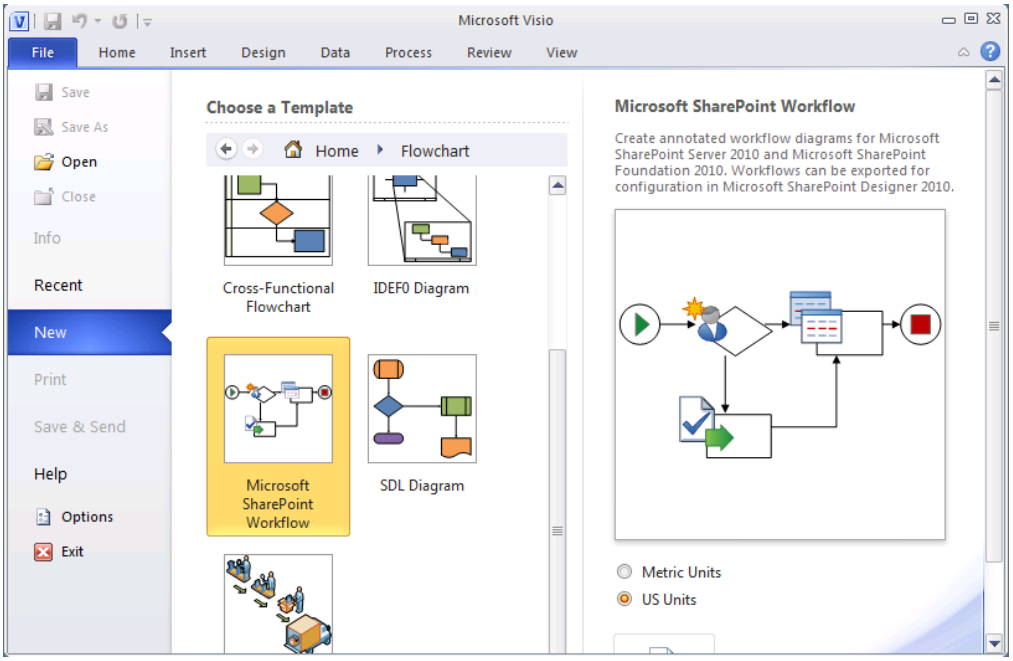


Figure 6.112 With Visio 2010, there is a new diagram template called Microsoft SharePoint Workflow. This template can be used to model workflows that can then be imported into SharePoint Designer.

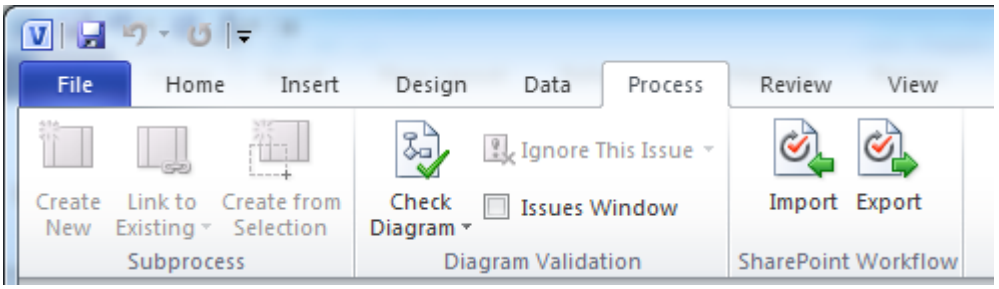


Figure 6.113 The ribbon in Visio 2010 features a Check Diagram button to validate the configuration, as well as import and export buttons to go between Visio and SharePoint Designer.

The next thing you will notice is the new and improved shapes navigation window on the left side of the screen (Figure 6.3). With the new look comes new features that make it easier than ever to find the right shapes. Within the new shapes navigation window you will notice some new content that wasn't there in the previous versions. The new content is specific to SharePoint Workflow template, and

includes SharePoint specific actions, conditions and terminators. These are the shapes needed to model the workflow functionality in a way that SharePoint Designer understands and can import.

By default the Quick Shapes item is selected, displaying a combination of actions, conditions and terminators that Visio thinks you will want to use. You can change your view to see more actions by clicking SharePoint Workflow Actions (Figure 6.3).To start authoring a SharePoint workflow, simply drop these shapes on to the canvas and connect them together in the appropriate manner.

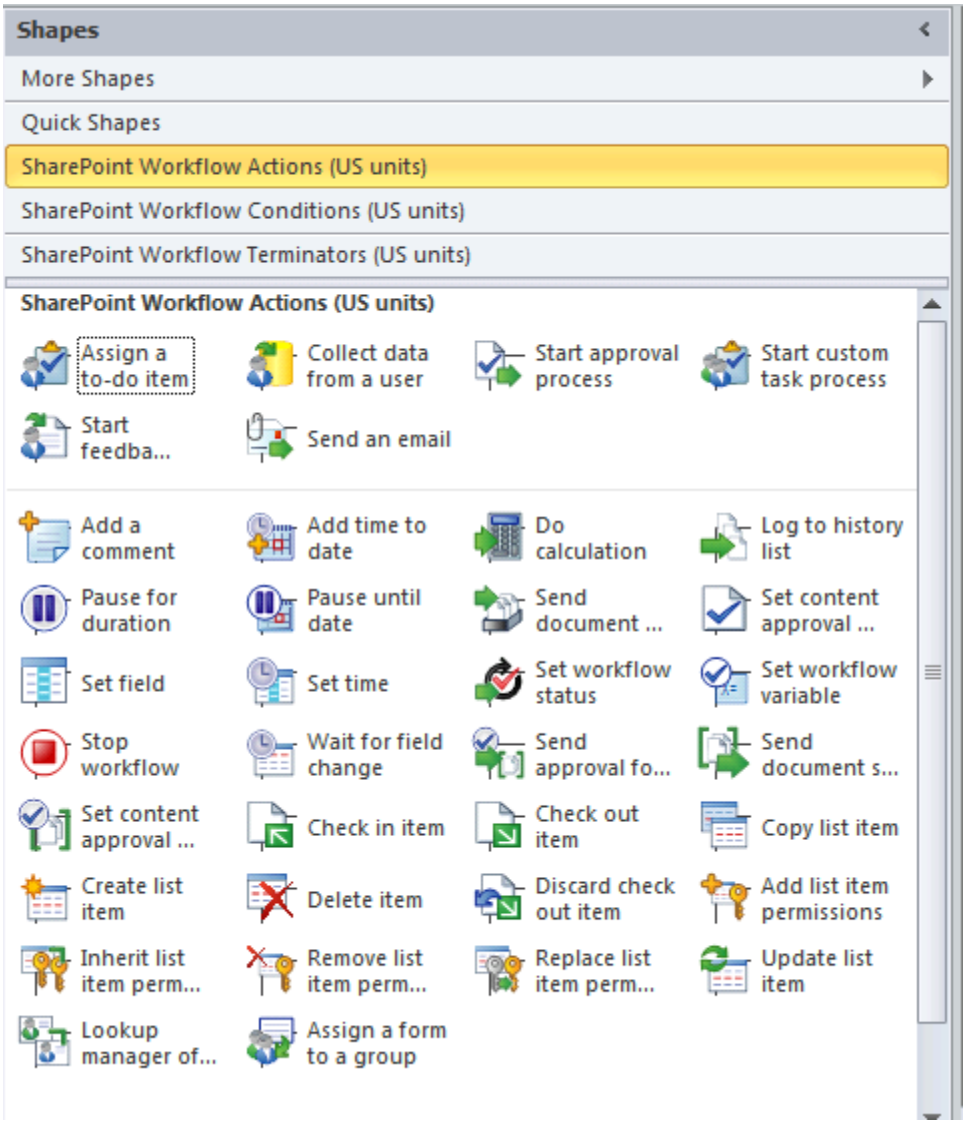


Figure 6.114 There's a host of actions and conditions that can be used in a Visio workflow to model your business processes. Each action and condition map to an action and condition in SharePoint Designer.

Notice how in Figure 6.4, you can drag and drop these action and conditions onto the canvas surface. The figure looks very similar to any ordinary diagram you'd find yourself building in Visio. The biggest difference between the standard "flowchart" template and the SharePoint Workflow template is that SharePoint Designer understands how to import the SharePoint Workflow template, but cannot import the flowchart template.

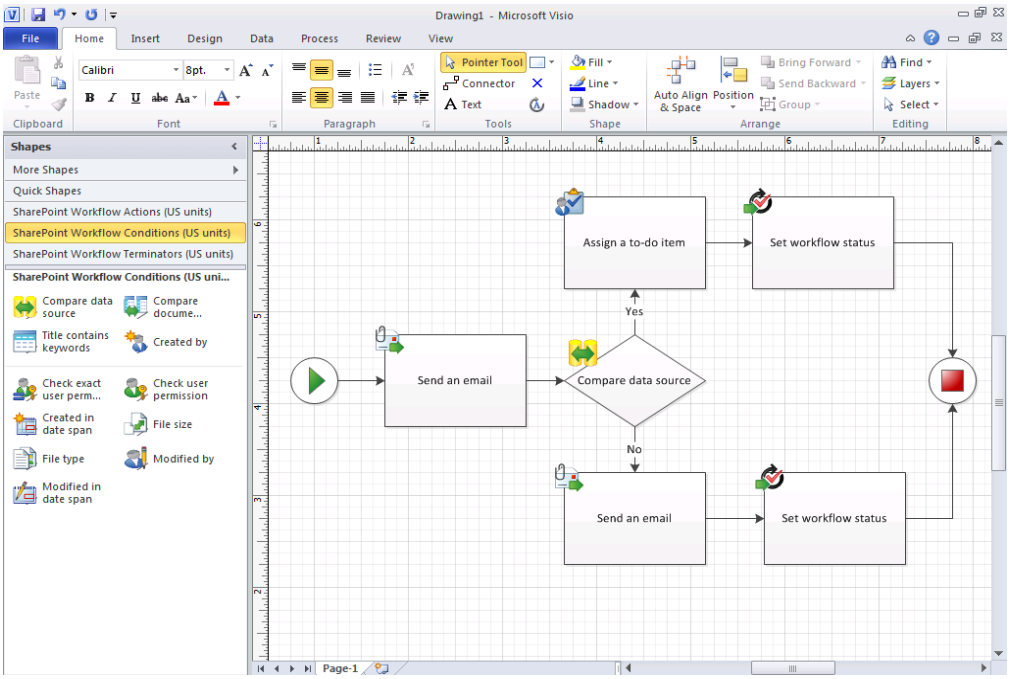


Figure 6.115 Notice how a Visio SharePoint workflow looks very similar to a flowchart diagram in previous version of Visio. The big difference now is SharePoint Designer now understands how to import Visio SharePoint workflow diagrams.

The same workflow in Visio 2010, like the one above looks much different when it is imported into SharePoint Designer (Figure 6.5). When in SharePoint Designer it is given a much more detailed and scripted look and feel. The diagram is now ready to be configured for SharePoint:

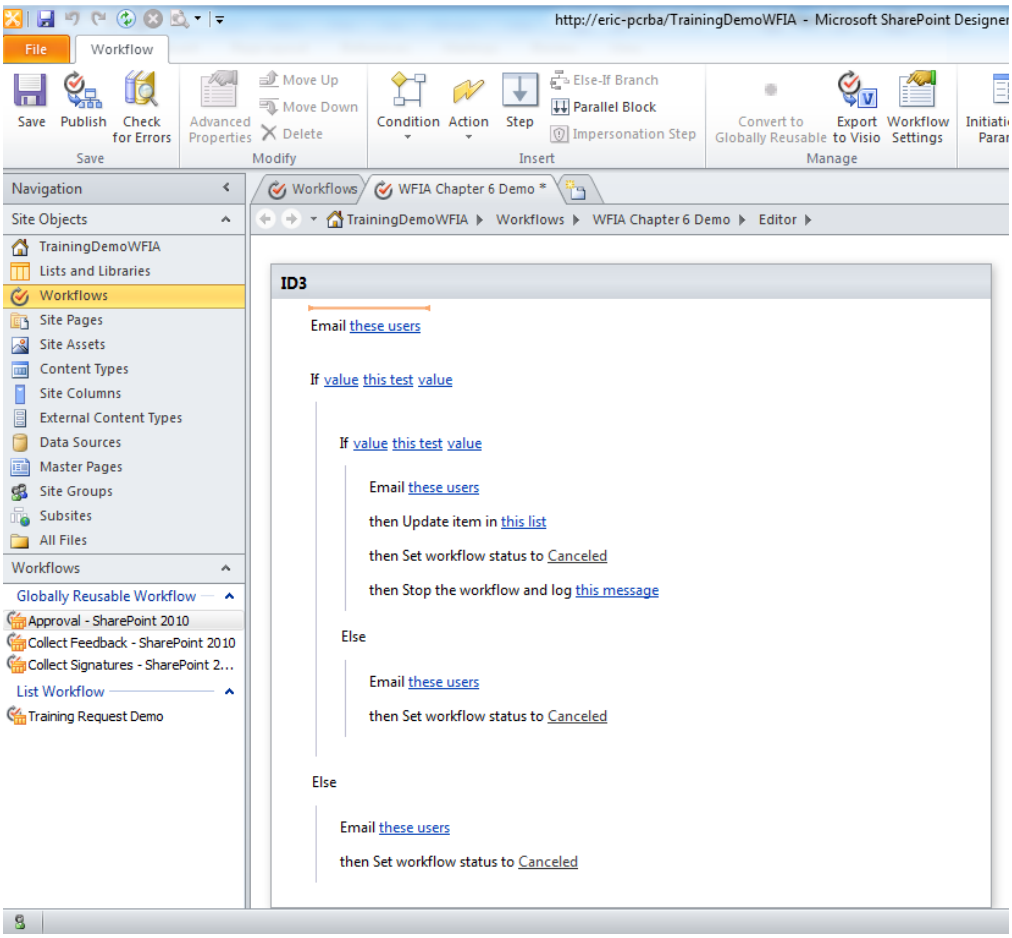


Figure 6.116 When the Visio diagram is imported into SharePoint Designer, it looks quite different. Instead of shapes showing the flow of work, you now have a more textual representation.

6.2 Building a Visio Workflow

SharePoint workflows can represent an infinite number of different business processes such as On-boarding, Approvals, Purchase Orders, etc. For this chapter the example of a Training Request system will be used. We'll start by modeling the training request workflow in Visio. Then in the next sections we'll import that workflow into SharePoint Designer and publish it into SharePoint.

Authoring a Visio workflow for SharePoint requires the correct configuration of shapes and validating that they are connected properly. The two terminator shapes are Start and Terminate, and these are required to start and end the workflow (Figure 6.6). If you do not use these terminators, you will not be able export the Visio file to the Visio Workflow Interchange (VWI) format. This format is required for importing the workflow into SharePoint Designer. To use these shapes simply drag and drop them onto the blank Visio canvas (Figure 6.6).

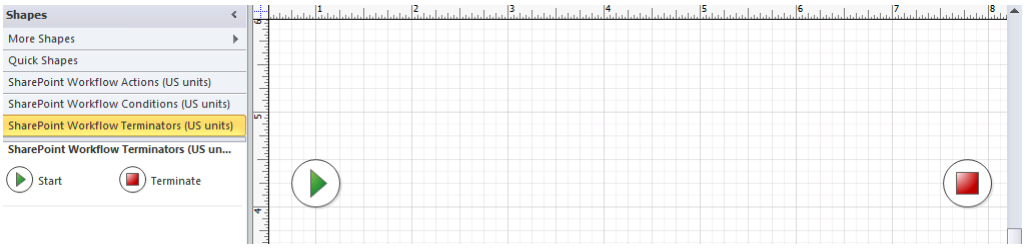


Figure 6.117 To begin modeling a SharePoint workflow in Visio, start by dragging and dropping the start and end terminators.

Now that you have begun to author a SharePoint workflow the next step is to add an action to the workflow. There are thirty six different actions to choose from and 11 different conditions. For this scenario we will add a few to simulate a training request process.

The workflow will be added to a listed called ACME Training Attendance. In this list is a column called Training Class. When the user adds a new item into the list, they specify what training class they are registering for. After they submit the request the workflow needs to ensure that the user has already attended the prerequisite class, "Introduction to SharePoint". If the user has taken the class, the user's manager needs to be assigned a task to complete their registration. If they haven't attended the prerequisite, the workflow needs to send an email notifying the user of the dependency.

As the business analyst, you can easily diagram this workflow in Visio. Follow these steps to create a new Visio SharePoint Workflow and add the necessary actions and conditions for the training request system:

1. Add a *Send an Email* action to the right of the *Start* terminator. This action will be used to send an email to the requestor, notifying them that the process is being processed.
2. Use the *Compare data source* to add a condition for this workflow to validate a prerequisite training class has already been taken. This shape will be configured later in SharePoint Designer to validate against another list in SharePoint.
3. Add another *Send an email* action below the *Compare Data Source* condition. This will be used in case the initiator does not meet the prerequisite and is the request is rejected.
4. Add a *Set Workflow Status* action to the right of the second *Send an Email* action. This will be used to update the workflow's status column to be cancelled.
5. Add the *Assign a to-do item* action above the *Compare Data Source* condition. This will represent the second path for when requests meet the prerequisites.
6. To the right of the *Assign a to-do item* action, add the *Set Workflow Status* action. We'll use this to change the workflow's status column to be approved.
7. Finally add a *Stop Workflow* action to the diagram, this may seem redundant, but for larger diagrams you may not want dozens of lines all connecting back to a single terminator.

When these steps are complete, your diagram should look something like Figure 6.7:

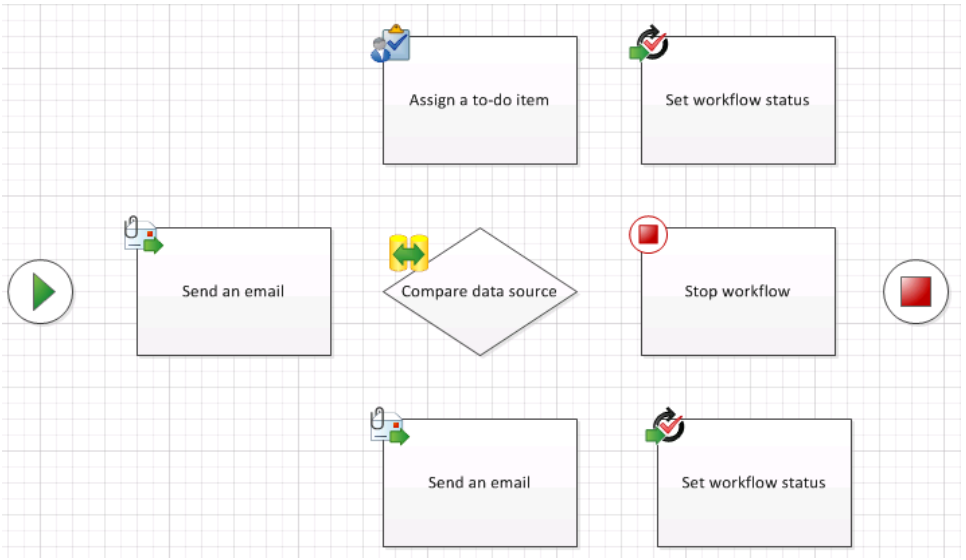


Figure 6.118 Continue by dropping the necessary actions and conditions onto the canvas. These actions and conditions have counterparts in SharePoint Designer that the developer will need to fill in.

Now that all the shapes are laid out, the next step involves connecting the shapes using the various connectors. In Visio 2010 Microsoft has included a new shape insertion feature that makes it much easier to connect shapes. Simply hover your mouse over the shape you want to connect; you will then see four blue arrows appear. Click the arrow (Figure 6.8) that points in the direction of the shape you want to connect to. You can also use this method to create new shapes, in addition to connecting existing. Notice in Figure 6.8 the popup to the right of the Right facing blue arrow. This shows new shapes that can be created, if you so desire.

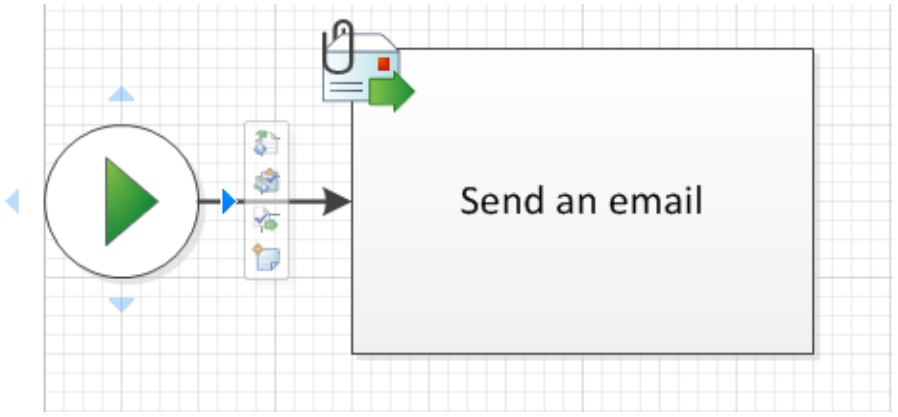


Figure 6.119 After your shapes are on the canvas, it's time to start connecting them. Simply hover over a shape and click the blue arrow. The nearest shape in the direction of the arrow will now be connected.

Note

Another shape insertion enhancement is the ability to hold down the mouse button on the arrow you want to connect and then drag the connector to another shape. Doing so eliminates a lot of extra clicks when it comes to connecting shapes.

Connect all the shapes together in a similar fashion as Figure 6.9. Once you have connected all the arrows you will need to do one more thing. The "compare any data source" action is a bit different from the other actions. You have to identify which path represents the "yes" as well as the opposite "no" path. To do this, right click on the arrow you want to change. In the case of our training request process, click the arrow leaving the conditional shape going to the "Assign to-do item" shape. Then select yes or no from the menu. Once you have connected all of the shapes and assigned the appropriate arrows your workflow should look like Figure 6.9.

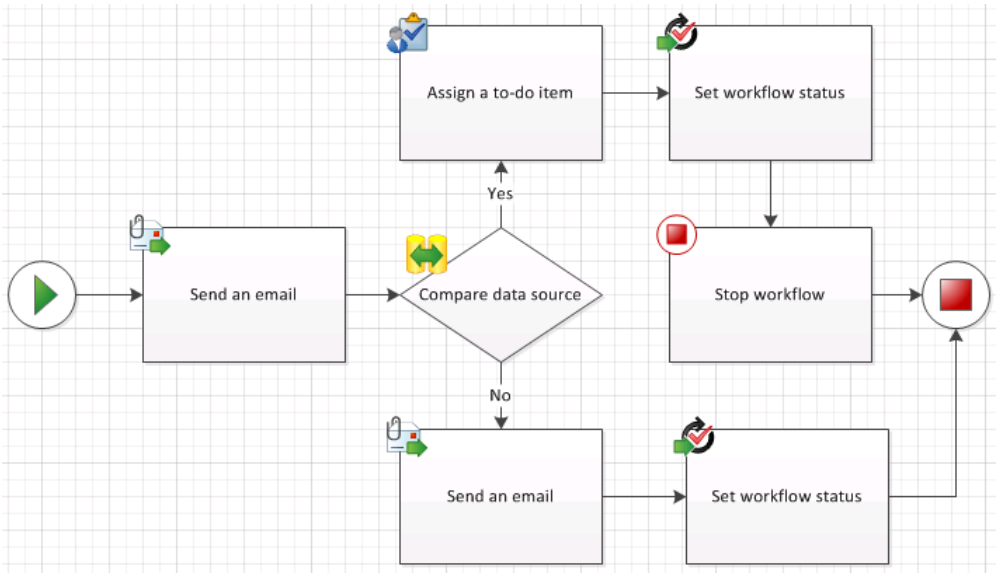


Figure 6.120 A Vision SharePoint workflow should be validated by clicking the Check the Diagram button. All "paths" should lead toward the Terminate action or terminator.

The next step is to validate the workflow by clicking the *Check the Diagram* icon at the top of the window located under the process tab. This check will scan the document to check it against validation rules and conditions related to SharePoint workflow standards. Only if it passes the check will the diagram be able to be successfully imported into SharePoint Designer.

Once you click check diagram you should get a message saying it passed successfully. Now you need to do the following to get the diagram into a format that SharePoint Designer can use. This is done by clicking on the Export button at the top of the screen also located under the process tab. When you click Export, simply accept the defaults and create a name for the workflow. Save the exported file to a place

that you'll remember, because in the next section we're going to import the workflow into SharePoint Designer, and thereafter add it to SharePoint itself.

6.3 Importing a Visio Workflow into SharePoint Designer

Visio is used to diagram out the workflow and SharePoint Designer is used to actually get the workflow into SharePoint. Also, SharePoint Designer is needed to add actually fill in the details of the workflow actions. Visio simply creates the "placeholders" of functionality. SharePoint Designer fills the gap.

To start, create or open a site in SharePoint Designer 2010. Click on the Workflows link on the left pane. When you switched over to the workflows section of SharePoint Designer the ribbon adjusted to show you workflow related commands (Figure 6.10). The one we are interested is the "Import from Visio" command, click that and navigate to the Visio diagram we created and exported earlier.

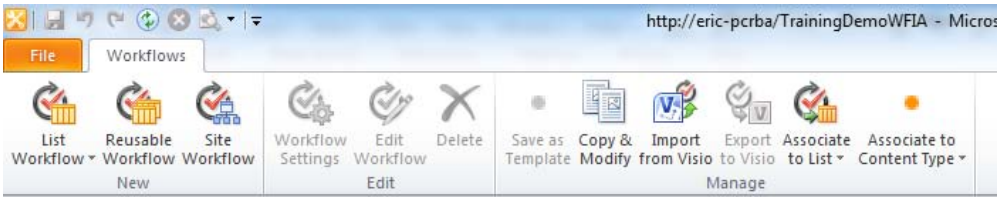


Figure 6.121 The new ribbon in SharePoint 2010 features an "Import from Visio" button. It also features an "Export to Visio" button so you can easily go back and forth.

Remember the Visio diagram we exported is in the new VWI format. When you import the file you will be prompted to choose what type of workflow you want to import it as. You can either choose a list workflow or a reusable workflow. For this scenario we will choose the list view and attach it to the list we want the request for training to be generated from (Figure 6.11).

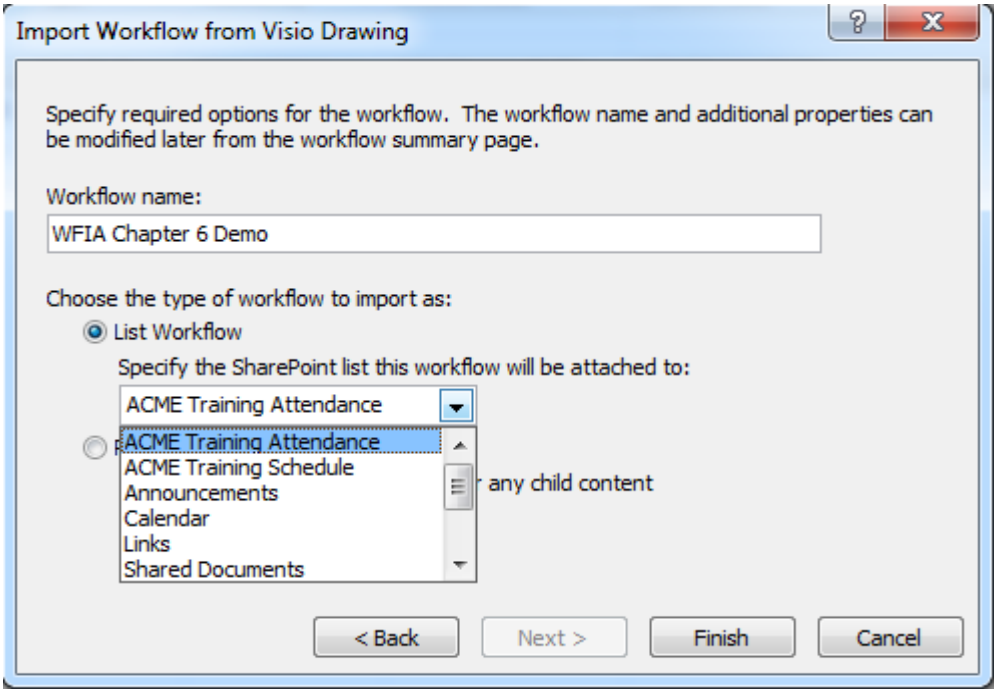


Figure 6.122 When you import a workflow you need to choose if it's a List workflow or a reusable workflow. If it's a List workflow, you need to choose the list to bind it to.

We are now looking at our Visio workflow that we built earlier, but now in the SharePoint Designer workflow editor it looks very different. Instead of shapes connected to one another, you see a list of actions. The list of actions still reflects the same placeholders for our training request workflow (Figure 6.12):

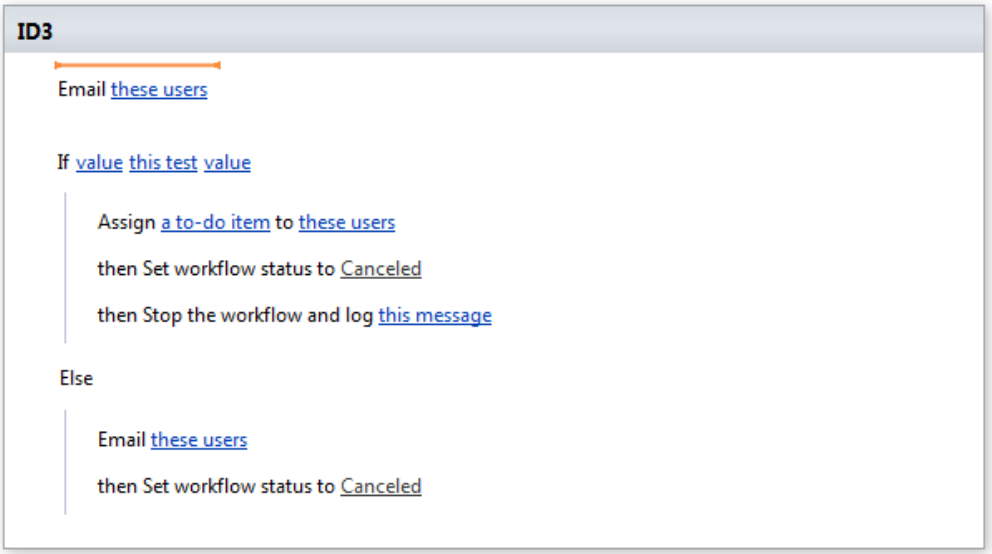


Figure 6.123 When the workflow is first imported into SharePoint Designer, you'll quickly notice that all the actions and conditions need your attention. This is noted by the blue underline, and is because importing from Visio only adds "placeholders" that later need to be filled in.

You can see all the steps from the diagram we authored earlier in Visio in this different format. Basically we have an empty workflow with place holders in it. Now we need to finish out each step. The first step is the send email action we created earlier. The blue underline indicates more information is needed. Click on it to define how the email should function.

When you click on the blue underlined for the email action, a window called "Define email message" pops up. We will configure the email to send a response back to the initiator of the request that his\her request has been submitted and is currently being processed. Configure your email similarly to Figure 6.13:

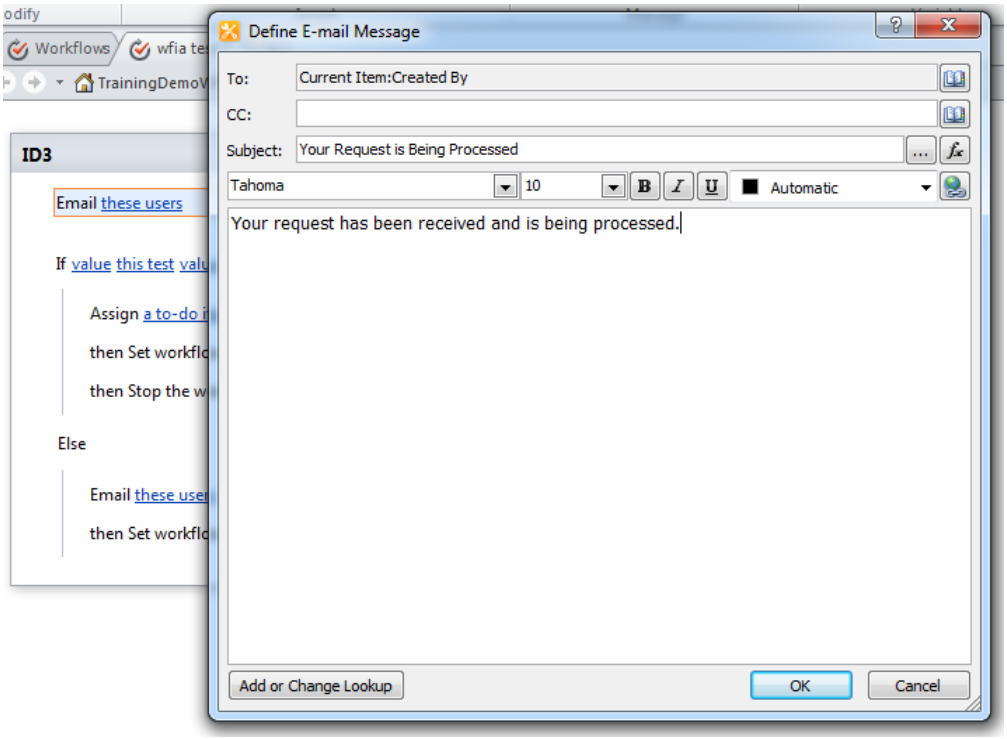


Figure 6.124 Click on the blue underscores to start providing the actions and conditions with their needed configurations.

For this email we chose to send the email to the current user. We also added a subject line and some text in the body of the email to indicate to the user that their request is being processed.

Next we need to configure the lookup to see if the current user who is initiating this workflow has already taken any training classes before. Specifically, we need to make sure that the request has taken the prerequisite class, *Introduction to SharePoint (*Figure 6.14*)*.

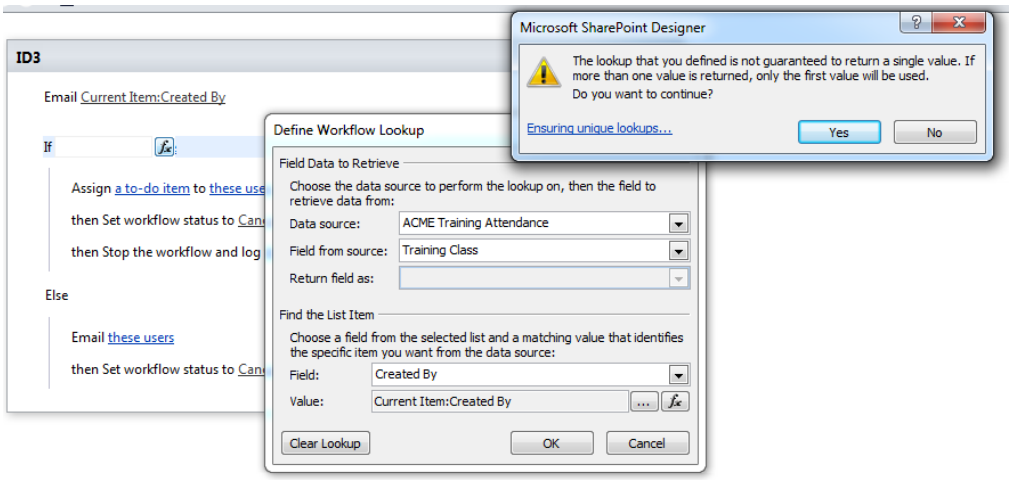


Figure 6.125 For the training request system, we want to confirm that the requestor has met the needed prerequisite classes.

After configuring the lookup set the rest of the condition to "if equals", "Introduction to SharePoint". This will make the workflow check to make sure the user has already take the prerequisite class of Introduction to SharePoint before he\she can take the next classes. I am not going into a great deal of detail here because this chapter is not about all the ways to configure SharePoint Designer, which is covered in chapter 3 of this book. The main point for this chapter is to show you the process of moving a workflow through SharePoint Designer into SharePoint.

One SharePoint Designer specific item we will go into here is the ability to know lookup someone's manager and assign a to-do item or an e-mail to them. SharePoint Designer introduces the ability to use the User Profiles as a data source in workflows or just about anywhere else in SharePoint Designer. This is one of the most requested features of workflows that end users would ask for and it was not easy to do and it certainly could not be done in SharePoint Designer, well now it can.

In the *Assign a new task* action, click "these users". What we want to do is lookup the requestor's manager (Figure 6.15). In the popup, click "Workflow Lookup for a user". Select "User Profiles" as the data source, and "Manager" as the field. The workflow will now be setup to send a task to the requestor's manager. In our training request workflow, this task could be to finalize the registration through some external business process.

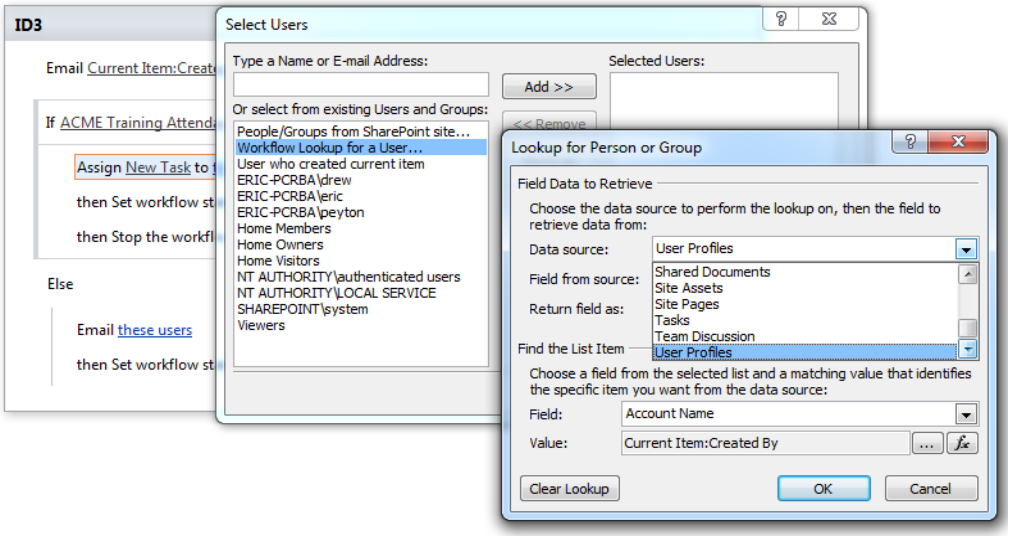


Figure 6.126 If the prerequisites have been met, email the requestor's manager to inform them that they need to finalize the registration.

After we fill out the rest of the actions and conditions the workflow should now look something like the Figure 6.16. The next step is to publish the workflow into SharePoint. We will also enable workflow virtualization on the workflow so that our diagram will be used to show the workflow's status pictorially.

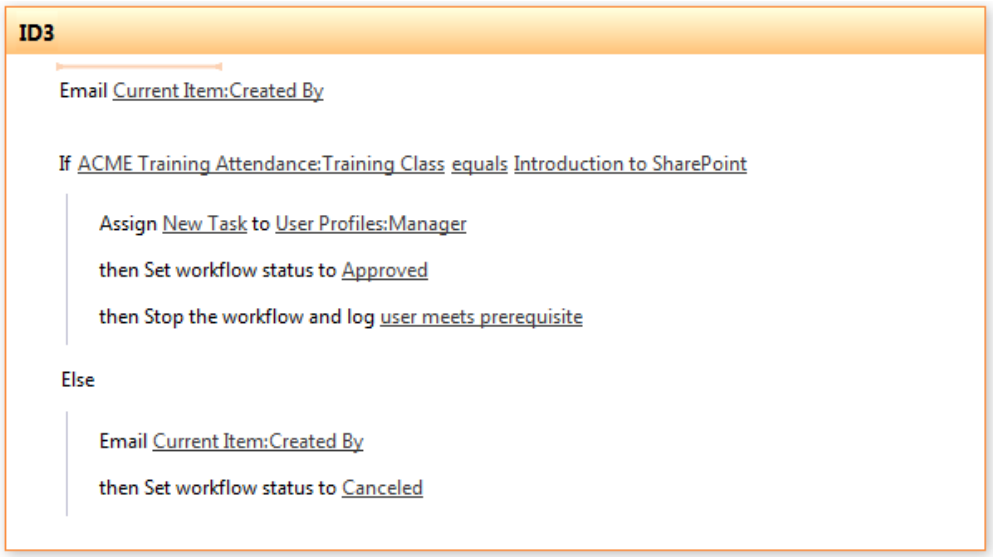


Figure 6.127 When the workflow is complete, you'll notice that all the blue underscores have been replaced with black underscores.

6.4 Publishing and Visio Graphic Services

Before we publish the workflow, let's look at the settings on the workflow. In particular, there's the "Show workflow visualization on status page" setting (Figure 6.17). When this box is checked, and after the workflow is published, all running or completed workflows will show the diagram on the workflow status page. Even more significant is that on the diagram will be a green check box next to actions that have completed as well as progress symbols on the actions that the workflow is currently executing on. This will give your users a visual representation on where in the process the workflow is currently executing.

To enable this functionality, from within SharePoint Designer, go to the workflow's settings tab. Check the box next to "Show workflow visualization on status page" (Figure 6.17).

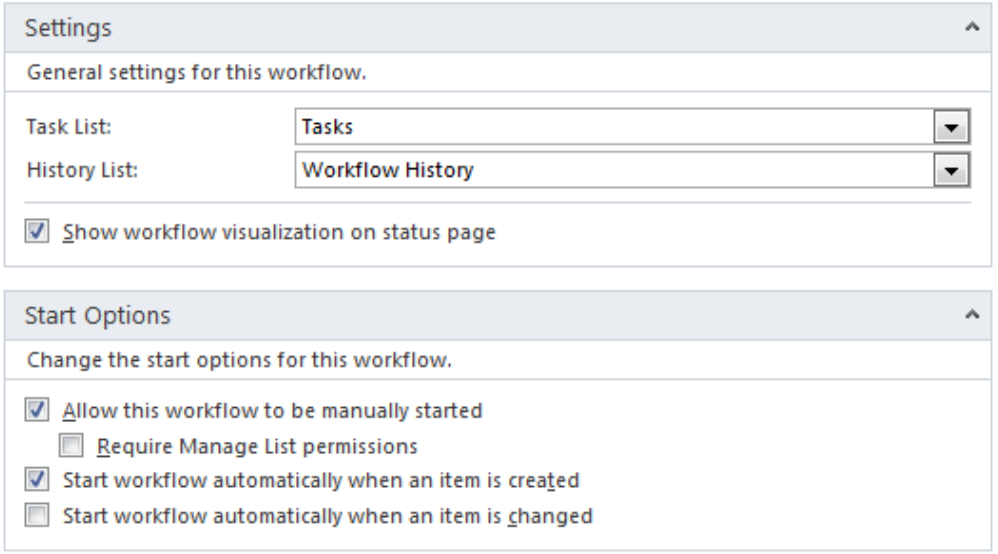


Figure 6.128 To enable the Visio diagram to display on the workflow's status page, check the checkbox next to "Show workflow visualization on status page".

After this box is checked, go ahead click the Publish button to publish the workflow. Go to your SharePoint list you added the workflow to, and add a new item. If you specified to "Start the workflow automatically when an item is created" (Figure 6.17), you will find your workflow already running. Navigate to the workflow status screen by clicking the "In Progress" column. There's a pretty major difference with this status screen, than what you've seen in previous chapters. Now at the bottom of the screen is the workflow visualization as seen in Figure 6.18.

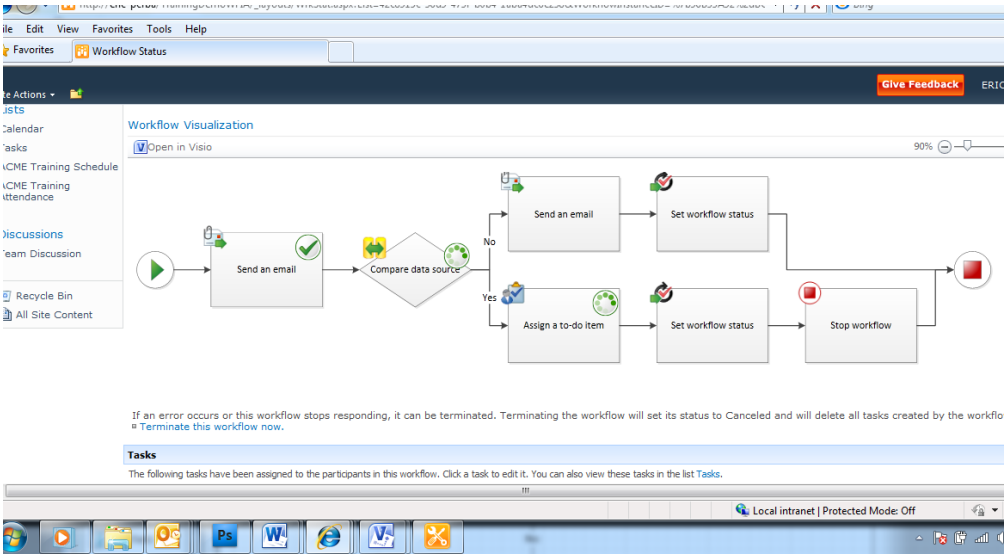


Figure 6.129 The workflow visualization features the diagram we created in Visio. This enables the user to more quickly see the status of the workflow, and the path it took.

Reviewing the Workflow Visualization above you can see that this workflow is on the assign a to-do item list action. Further details tell us that the first step has been completed successfully by the presence of a green checkbox. With SharePoint 2010 and Visio Services anyone can easily see what state a workflow is in without have to decipher cryptic messages and statuses.

6.5 Summary

In SharePoint 2007 there was a chasm between the SharePoint Business Analyst and the SharePoint Developer. In SharePoint 2010 this chasm has been made much smaller for SharePoint workflows. This is largely because of the new features in Office Visio 2010, namely, the new SharePoint Workflow diagram template.

The new template gives non-technical analysts the ability to diagram the high level business process in Visio. After they've produced this diagram, the SharePoint Developer can then import that diagram into SharePoint Designer, at which case they'll fill in the placeholders with actions, conditions, and the like. Following this involves the publishing of the workflow into SharePoint. The power with this is the ability for the developer to not have to start from scratch. They can take the product of the analyst's work, and run from there.

Another great feature is the ability to show the Visio diagram on the workflow's status page. Now the end user can see pictorially where in the process the workflow is currently executing. Before, your best option with to set the workflow status column, or perhaps write the log. This left the user sifting through logs to see the course the workflow took. The visualization of the workflow's status will grant end users the ability to find the workflows current state, and execution path, much more quickly than ever before.

7

Custom Form Fundamentals

Workflows and forms are a powerful combination. This is because most business processes usually revolve around a form of some sort. To have a holistic understanding of how to build custom workflows in SharePoint, you really need to know how to build custom forms. This includes knowing what tools are available to build your custom forms with.

There are three tools available in SharePoint to build custom forms: out of box forms (auto-generated), InfoPath forms, and ASP.NET forms. It is important to know when to use which type. Each type comes with its own unique pros and cons. For instance, there is a lot you can do with simply the out of box forms without ever needing to use a more advanced tool like InfoPath or ASP.NET. Working with the out of the box forms is very simple and fast, saving you time and money. Because of this, it remains a good place to start whenever considering a forms solution. There are shortcomings that necessitate InfoPath, which will be discussed in further detail in this chapter. Since ASP.NET forms are more of a programming topic, they are covered in chapter 9.

Along with these tools, there are many fundamental ways of using the forms to build workflow solutions. These include: customizing the out of box forms; using form libraries; and working with initiation and association forms in SharePoint Designer workflows. Throughout this chapter we'll take you through activities such as publishing custom forms into form libraries, updating a library's default template, and making shared templates across multiple libraries..

7.1 Tools used in the Building of Custom Forms

There are three main tools you can use to build custom forms with in SharePoint. The out of box forms, InfoPath forms, and forms built in Visual Studio with ASP.NET. The out of box forms help users edit list items and SharePoint data and have a limited set of customizability, whereas InfoPath forms bring a deep level of customization to the table. ASP.NET forms built in Visual Studio are used when the most technically complex forms needed. As you design your workflow solutions you will want to carefully think through which form technique to go with, because there are clear pros and cons that come with each. Thinking through your form requirements ahead of time may save you from having to rewrite functionality later.

7.1.1 Out of the Box New and Edit forms

It's best to start with these out of box forms rather than go straight to a tool like InfoPath or Visual Studio. This is because it is most likely the easiest and fastest avenue assuming your requirements are not too complex. There's no sense in reinventing the wheel if you don't have to. This is because the out of box forms will be generated for you depending on what data elements you have in the list, whereas with InfoPath and ASP.NET, you'll have to build the form from the bottom up.

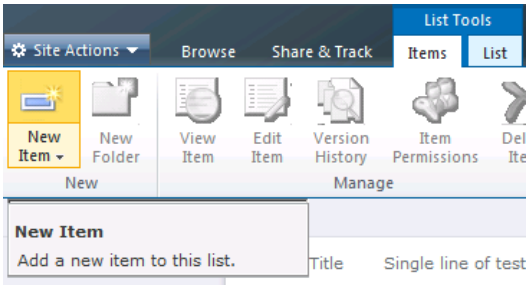


Figure 7.130 When you add a new item in SharePoint, a form will appear where you can edit the metadata around that item. It is common to have a workflow run on an out of box item like this.

Consider with me for a moment the process of creating a new list or library in SharePoint, and adding a workflow onto that list. When you create a new custom list or library in SharePoint, most of the time you'll go back into that list or library and add a bunch of custom columns. These columns make up the metadata that resides on a new list item or document when one is created or added. On a list you have the opportunity to enter this metadata through an out of box form after you click the "New" button (Figure 7.1). Additionally, after it has been created, there's an "Edit" button available (Figure 7.2). Both these buttons take you to automatically generated forms where your users can enter information (Figure 7.3 Sample out of box, auto generate new item form). It is then very natural to start a workflow on that item that will respond to the creation of new items, or the editing of existing items.

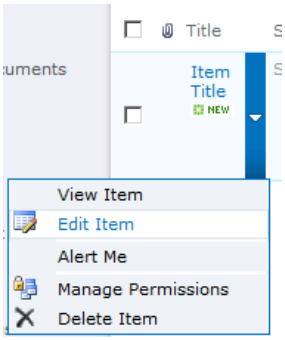


Figure 7.131 Similarly to creating a new item, editing an item will also give you a form where you can edit that item's metadata

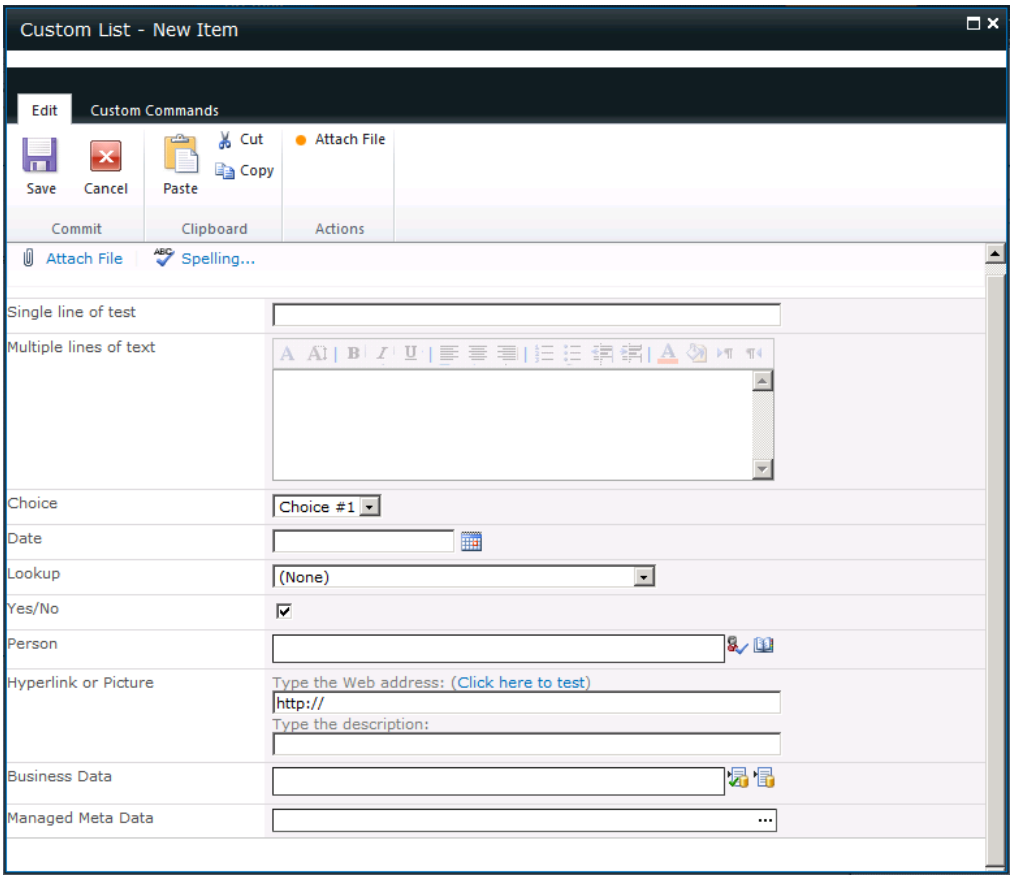


Figure 7.132 Sample out of box, auto generate new item form

So, the question is, are the requirements for the form in your custom workflow complicated enough where the auto generated forms will not work? At the very least, it only takes a few minutes to add a bunch of columns into a list or library to see if you can get by with the auto generated functionality.

Let's take a quick look at what types of columns you can add to a list out of the box, and what that translates into on an auto generated form. Table 7.1 and Figure 7.3 illustrate this.

Table 7.1 listing of all the out of box column types and how they're rendered on the form(s)

Column Type	Render Style on Form
Single line of text	Text box
Multiple lines of test	Text box - multiple lines
Choice	Drop Down (or left/right chooser, "Allow Multiple")

Date	Text box with date picker
Lookup	Drop Down (or left/right chooser, "Allow Multiple")
Yes/No	Check box
Person	Text box with people picker
Hyperlink or Picture	Two text boxes, one for URL, one for description
Business Data	Text box with data picker
Managed meta data	Text box with meta data picker

It's easy to see that there's quite a bit of functionality and requirements that can be met simply by using these out of box forms. As was said, a workflow can sit on top of the list item or document, and the metadata on the item entered through the form will be readily available to act upon.

In a custom SharePoint Designer workflow, you can check the data in a custom column through the *"If current item field equals value"* condition. With this condition (Figure 7.4), you can select which field you want in the current item, and compare that with some other value. Similarly, in a Visual Studio workflow, the list item data entered in the form is readily available to the programmer and workflow to utilize.

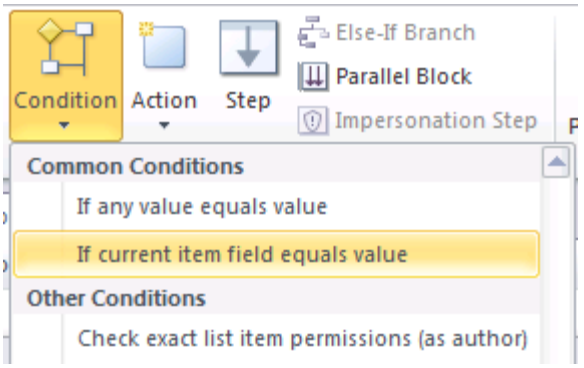


Figure 7.133 SharePoint Designer workflow condition that compares the data in the workflow's list item

So why then wouldn't you use the out of box forms all the time? Table 7.1 accurately shows the data types that can be used, but what if you want check boxes instead of radio buttons? Or possibly, you need more detailed instructions, logos, or graphics on the form to make it more graphically appealing? Better yet, the data in the drop downs may need to come from an external line of business application. All these examples represent where the out of box forms fall short. So again, are you keeping it simple, or going more complex? With this, here is how the pros and cons around the out of box forms stack up:

PROS

- Easiest approach - all out of box functionality.
- Fastest approach, it is very quick to configure and add a few custom columns onto a SharePoint list

CONS

- Least flexible in supporting customizations
- Limited ability to meet complex requirements

7.1.2 InfoPath 2010 Forms

Even with the strengths that come with the out of box forms, there's no doubt that they are only able to meet basic requirements. An easy example of something more advanced is having a requirement to populate a drop down with items that came from some external data source outside of SharePoint. Or possibly there are multiple drop downs that need to be dynamic, and change their items depending on some prior selection. In any case, all these examples demonstrate where the out of box forms fall short, and where a new tool is needed. The next tool in the lineup is Info Path.

InfoPath brings a lot to the table in its ability to offer solutions for these more advanced requirements. Requirements such as pulling business data out of web services, dynamic filtering of controls, and advanced validation of form fields upon submission can all be easily accomplished with InfoPath. Another great advantage is you have way more flexibility in building a unique, branded user interface that is easier for end users to understand and work with. There's no doubt that InfoPath could be a book in itself, but several of the most common techniques will be described in this chapter's InfoPath examples.

An InfoPath form may reside as a document in what's called a Form Library within a SharePoint site. While this isn't always the case, it is the most common InfoPath scenario. Form libraries are essentially document libraries, but they have connectors to InfoPath and are designed specifically to house forms. It is in this way that InfoPath supplements the out of box forms, because rather than work with the *new and edit* forms, you now will edit the InfoPath form directly. When you save or submit a form in a form library, by default the data in that form is stored in the library as XML.

Another way in which InfoPath interacts with SharePoint lists is through the customization of a list or library's *new and edit* forms. When you customize the out of box forms, the data in those forms is always mapped to columns in the list or library. Whereas in the previous example, the form itself was uploaded into the form library as an attachment, and that attachment contained the data as XML.

So how do you know when to customize the out of box forms, versus using a form library? For the most part, consider how much data you will be interacting with. If you customize the out of box forms, you need a column for every piece of data you want to save. If you use a form library, that data is stored as XML in the form itself, so you don't really need to have any extra columns on the document if you don't want to. Fifteen is a good rule of thumb. If your form needs to save more than 15 pieces of data, a form library is the better route to go, rather than create 15 columns on a list.

PROS

- Drag and drop experience and wizard based experience. You don't have to be a programmer.
- Supports advanced form customizations like connecting to external data, rules, and conditions.

CONS

- Requires a SharePoint Server license to host the form in the browser, otherwise the InfoPath client application is need to fill out a form.

Browser enabled InfoPath Forms and its Dependencies

The InfoPath client is a client application much like how Microsoft Word is a client application. You can use the client to design or edit a form template, but you can also use the client to fill out and submit

a form. There are two main versions of SharePoint, SharePoint Foundation and SharePoint Server. SharePoint Server is an add-on on top of SharePoint Foundation. This add-on includes a feature called Form Server that enables InfoPath forms to be filled out and viewed right from within the browser window. Without SharePoint Server, your users will be force to load the form in their InfoPath client application. Obviously this can be problematic because not having SharePoint Foundation forces your users to have a copy of Microsoft Office installed on their workstations. You will have to compare the cost to buy SharePoint Server with the cost to by copies of Microsoft Office, along with the headache of downloading the form first, to decide which avenue is best for your company.

7.1.3 ASP.NET Forms built in Visual Studio

Do you need the utmost flexibility in form customization? Do you have existing forms and code that you built long ago that you'd like to reuse? Are you an ASP.NET developer who's not interested in learning InfoPath? If you said yes to any of these questions you'll be happy to hear that ASP.NET forms still play a role in SharePoint 2010. For example, you can replace the *new and edit* out of box forms with a custom ASP.NET form. You could write a custom ASP.NET form and imbed that form within a web part. Then, when a user submits that form you could use the SharePoint object model to add a new list item in a list and start a workflow on that item. There are other examples where ASP.NET forms are used in SharePoint workflows. Forms such as custom workflow initiation and association forms are commonly built in ASP.NET. For more on this topic in particular, reference chapter 9.

Despite the ability to support complex customizations, ASP.NET forms are growing less and less common because InfoPath is becoming such a versatile and powerfully simple tool to use. With ASP.NET, you literally need to start from scratch, but with InfoPath you have a host of controls and wizards that help you add the needed functionality into your form. Because of this, and because the ASP.NET form topic is rather generic to the ASP.NET platform, it will not be discussed further in this chapter. Chapter 9 has a few touch points with ASP.NET forms specifically as it pertains to SharePoint workflows, and will be covered in more detail there.

PROS

- Supports from the ground up customizations.
- You may be able to leverage legacy code.
- Possibly a smaller learning curve if you already have a lot of ASP.NET experience.

CONS

- Most likely this is the most time consuming approach since you have to build everything from scratch.

7.2 Customizing the Out of Box Forms with InfoPath

We come to a place now where InfoPath has had plenty of introductions, and now needs to prove itself and actually do some work. A new feature in InfoPath 2010 is the ability to customize the out of box, auto-generated forms. You heard it discussed that there is a lot of benefits to using the out of box forms because they are really fast and easy to setup. Even if you choose the out of box approach, but later

decide you need more functionality; you can still customize those forms with InfoPath. This section will walk you through the steps involved in customizing the out of box forms.

A big difference between a form in a form library and using InfoPath to customize the out of box *new and edit* forms is how the data is stored. A form in a form library stores its data as XML that is uploaded as a document in that library. When you customize the out of box forms, that data still needs to be mapped to columns in the list item. If you have a lot of data you may want to go the form library route, rather than add 20 columns to the list item. You may just want to change the behavior of one or two columns, or possibly brand the form a bit with a company logo, or more detailed instructions. In this case, customizing the out of box forms is a great approach.

Note: SharePoint Server required when customizing the out of box forms

There are two main versions of SharePoint 2010, SharePoint Server and SharePoint Foundation. This section depends of features available only to the Server edition of SharePoint 2010.

To get started with this effort, first create a SharePoint list or library, and onto that list add all the columns you need to track. You can do this through the *List Settings* and then *Add a column* option. After you have all your columns setup in a way that suites your data requirements, it's time to edit the forms to suite your user interface requirements. When in the ribbon, in the *List* tab, click the *Customize List* drop down and choose *Customize Form* (Figure 7.5).

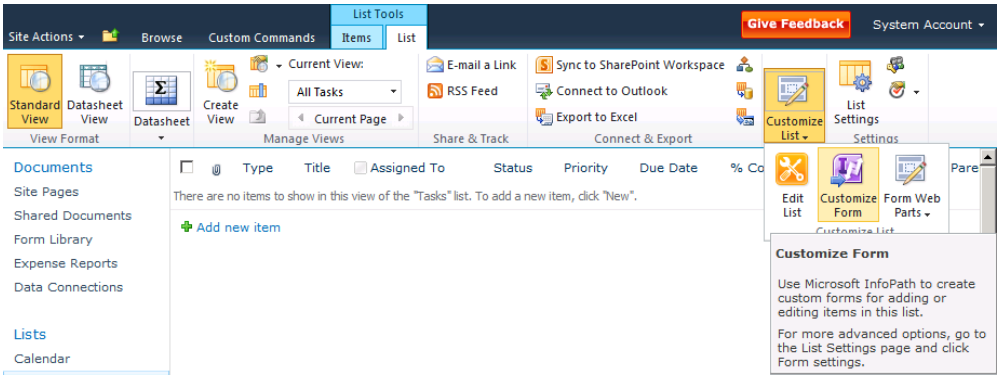


Figure 7.134 To customize the out of box forms in a list or library, click the Customize Form button in the ribbon to launch InfoPath.

Clicking this button will launch InfoPath and will be defaulted to a view with all the columns in the list you're trying to edit. The example we're going to build involves a tasks list, and each task has an associated Project and Sub Project. What we want to do is change the lookup columns for these associated projects to be dynamic, so when you select a Project, the Sub Project drop down will change its items based on what parent Project was selected. Figure 7.6 shows how the tasks form looks when it is first opened in InfoPath. Notice the Parent Project and Sub Project columns that we want to alter.

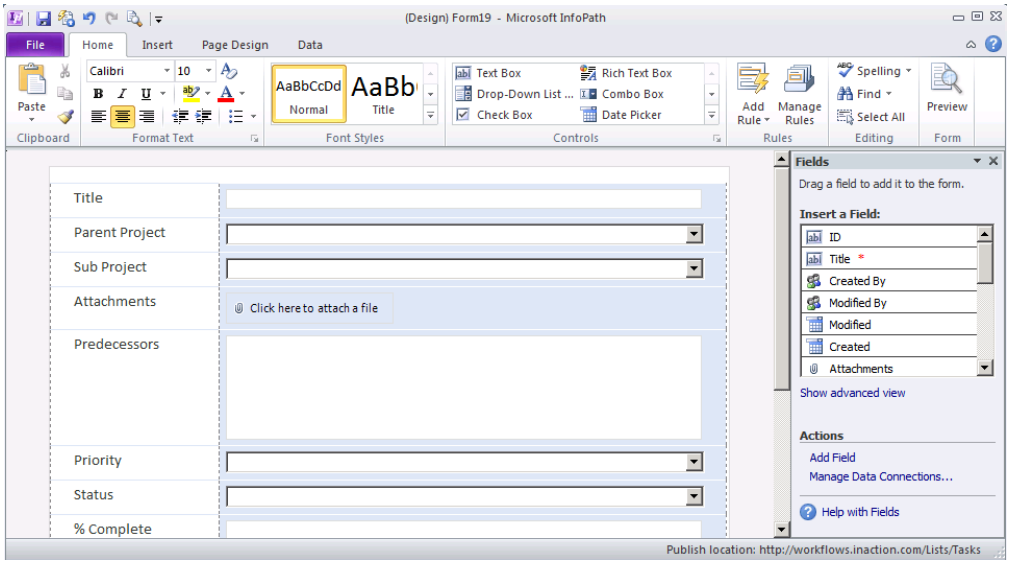


Figure 7.135 When you first select to customize the out of box New and Edit forms on a list or library, InfoPath is loaded with all the columns on that list or library, and you can start editing their behavior or look and feel.

The Parent Project column is a lookup column to a separate list called Project Names. Sub Project is another lookup column to a second separate list called Sub Project Names. The Sub Project Names list has two columns, Title and Parent Project. Parent Project again is a lookup to the Project Names list.

We want to make the Sub Project drop down in our tasks form dynamically select its items based on what is selected in the Parent Project drop down. The first thing we need to do is create a new data source for the Sub Project drop down. The default data source for this drop down only has two fields in it, Title and ID. Since we need to filter on the Parent Project column, we need to add a new data source. Secondly, we will then filter that data source on the selected Parent Project. And lastly, we'll setup a rule on the Parent Project drop down to ensure the data source is updated each time the drop down is changed.

Before running these steps, setup the Project Names and Sub Project Names lists as was just described. Then add a Project Name lookup column and a Sub Project Name lookup column onto a Tasks lists. Lastly, edit the out of the box form in InfoPath as shown in Figure 7.5.

Setting up Dynamic Drop Downs in an InfoPath Form

	Action	Result
1	While customizing the out of the box task edit form in InfoPath, setup a new data source for the sub project dropdown menu: A) In the Data tab within the ribbon in InfoPath, click the Data Connections button. Click the Add button to add a new data source.	You'll be half way through the new data source creation wizard.

B) Select the radio buttons to create a new data source that receives data. Click next.

C) Choose to receive data from a SharePoint library or list. Click next.

D) Type the URL to the SharePoint site that contains the Sub Project Names lists and then click next and continue wizard in step 2.

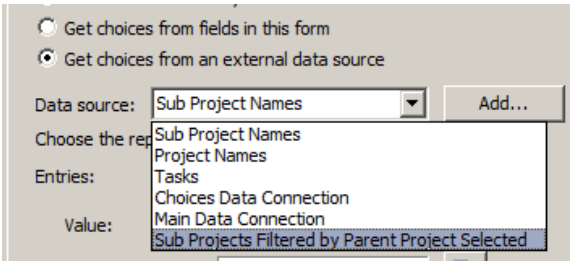
2 Specify your list and columns to be included in the data source:

A) Select the list name that contains the data for your connection. In this case, Sub Project Names and click next.

B) Select the columns to include in the data source, in this case Title, Parent Project, and ID columns. Click next twice.

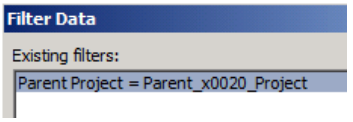
C) Give the data source a name, like Sub Projects Filtered by Parent Project Selection. Click Finish to complete the data source wizard, and thereafter Close.

D) Change the Sub Project drop down to use this data connection instead of the default connection by right clicking on the drop down and choose *Drop Down List Box Properties* and then change the Data Source drop down menu to be the data connection that was just created and finally click Ok.



A new data connection will be available and the Sub Project drop down will be set to use this new connection.

2 Filter the new data source on the selected parent project. In the entries box directly below the Data Source selection on the Sub Project drop down, you'll see the XPath query that points to the data. Click the *Select XPath* button directly to the right of the text box (click the ellipses image). Follow these steps to setup the filter:



	A) In the Select a Field or Group dialog box, click the Filter Data button and then click Add. B) The first dialog box contains the fields in our custom data source. Select the Parent Project field. Leave the comparing field (middle drop down) set to "is equal to". C) In the third drop down, choose "Select a field or group". Another dialog box will appear, change the Fields drop down to Main to select the data that is represented on the form itself. D) There are two sub folders. The "queryFields" folder will go out and query fresh data, and "dataFields" folder contains the data on the form at the present time. Expand the "dataFields" folder. E) Expand the sub folder containing the list item data. Select the Parent Project field. Click Ok 5 times.	
3	Now our Sub Project drop down is correctly filtering its data based on what the Parent Project drop down's selected value is set to. The last thing we need to do before we publish the form back into our SharePoint tasks list is to setup a rule on the Parent Project drop down. Each time the Parent Project drop down's value is changed, we need to tell the Sub Project drop down to update its items. A) Right click the Parent Project drop down, and under the Rules fly out, select Manage Rules. B) Click the New drop down in the Manage Rules tool bar, and select Action. C) Give the rule a name, like Update Sub Project Drop Down. D) Next to "Run these actions" click the Add drop down and select "Query for data". E) In the Data Connection drop down, select the custom data connection we created earlier, in this case, select Sub Projects Filtered by Parent Project Selection and then click Ok.	With the rule in place on the parent project dropdown, the data source for the sub project dropdown will update each time the parent dropdown changes.

That's it! Now all that is left is to publish the form back into our tasks list. Under the File menu in the Info tab, click *Quick Publish*. Now when we go to create a new task in our tasks list, our Sub Projects drop down will be dynamically populated based on what Parent Project is selected (Figure 7.7)!

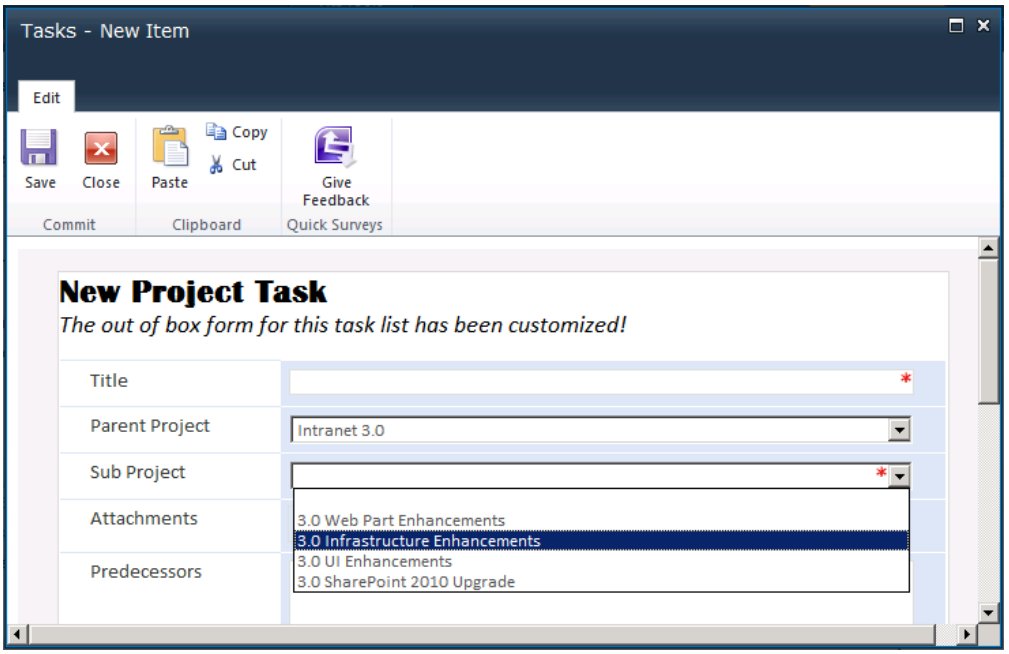


Figure 7.136 You can customize the out of box new and edit forms to make lookup columns in your lists or libraries dynamically generate their items based on custom rules.

Only Positive Integers Allowed Error

If you get an error that says "Only Positive Integers Allowed", you're trying to save a string into a column that is expecting a number. Most likely the sub project drop down's value is set to a string but since the column is a lookup it needs to be a number instead. To fix this error, right click on the Sub Project drop down and choose Drop down list properties. Then under neither Entries, change the Value from d:Title to d:ID. After you republish the form, the form should start saving properly.

7.3 Publishing a Template to a Form Library

We just wrapped up a section on how to customize the out of box forms, and now it's time to take a closer look at how forms work with form libraries. When a user "fills out a form" in a form library, new form (e.g. XML document) is created off a form template associated with that library. This template by default is blank and not very useful. What we want to talk about in this section is how to build a custom form template, and update the default template with our custom template.

There are many different ways to get an InfoPath form template into a form library. You can publish the form template straight from the InfoPath client up into SharePoint. Or, from within SharePoint, you can browse down into your hard drive and select the form template. No matter which direction you go, you need to decide the scope at which the form template needs to be available. Do you want the form template to only be on one library, or do you want your end users to be able to create a new form off a form template on any library in the entire site collection? If you need a "global" deployment of the form

template you will need to publish the form template to a content type (see section 7.4), otherwise simply updating the out of box template in a form library is sufficient.

When a form like a company's expense report only needs to be added to and edited within a single form library, updating a form library's out of box form template is the most straight forward approach to take. If you need a form template to be shared across several form libraries, see Section 7.4 on how to do this. Before you can edit a form template on a form library, you need to create a form library in the first place. To create a new form library, follow these steps:

1. Navigate to your SharePoint Site
2. Under the Site Actions menu, click "More Options…"
3. A popup will appear (Figure 7.8), in the left navigation, click "Library"
4. In the center navigation choose "Form Library"
5. Provide a unique name for the library and click create

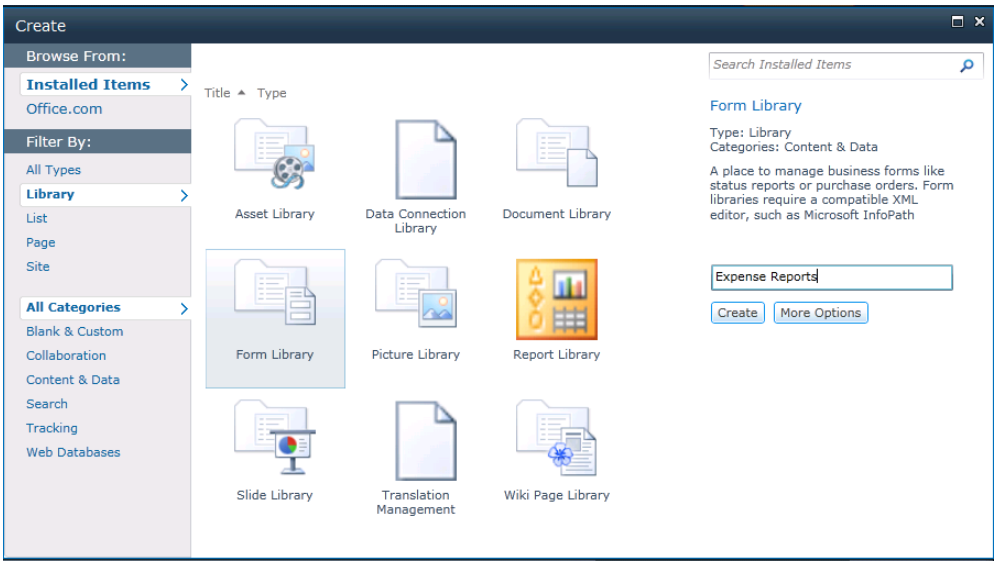


Figure 7.137 You can create a new Form Library through the "More Options" link under Site Actions on any SharePoint site.

Now that we have a form library to work with, let's take a look at how to update that library's form template. If you click on the *New Document* button in the ribbon, you get an empty form to fill out. It is at this point kind of boring and un-useful. You can tell what template the library is using by navigating to that library's *advanced settings*. In the ribbon, click *Library Settings*, and therein click *Advanced Settings*. Notice that the default template is stored within a "Forms" folder within the form library. This is the file we need to update (Figure 7.9).

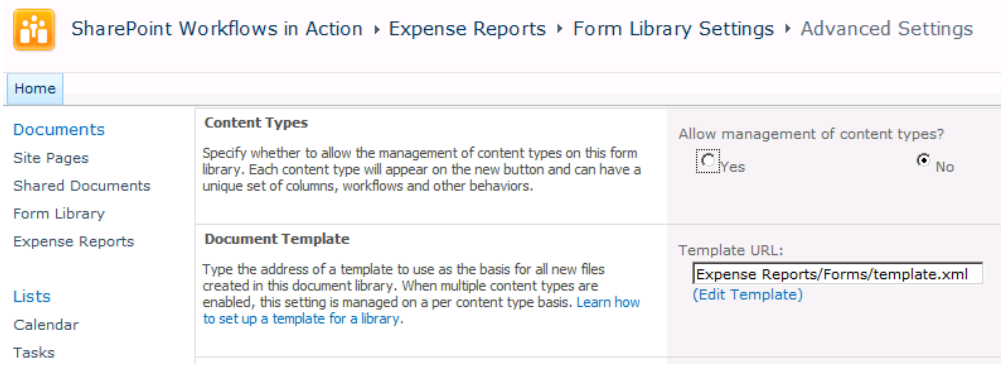


Figure 7.138 A Form Library by default works with a form template under the "Forms" folder within itself. Update this form to replace the blank form you get when you first click the "New Document" button on the ribbon.

To update this form, simply click the *Edit Template* link underneath the Template URL. This loads the blank form in the InfoPath 2010 Designer client. The designer surface is empty and has a grey background. We'll need to build our form from the ground up. Follow the steps below to setup a very basic expense report template for your company:

1. Let start with the grey background, click the Page Design tab from within the InfoPath form that just loaded and under *Page Layout Templates* choose *Title* and *Body*.
2. Give your form a title, such as "SP WFiA Expense Report" and a description.
3. Click below the description, add a *Repeating Table* by clicking into the *Home* tab, and selecting *Repeating Table* from the *Controls* menu. I chose a table with three columns.
4. Fill out the repeating table's table column headers with column descriptions such as "Date of Expense", "Dollar Amount", and "Description".
5. Save the template on your personal computer's hard drive by clicking the save icon.

When you're done, your form will look something like Figure 7.10.

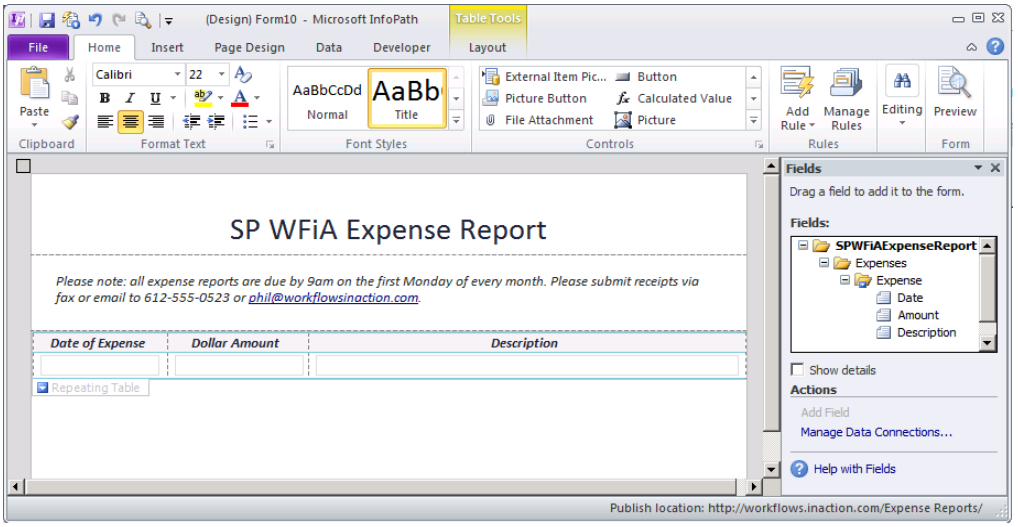


Figure 7.139 A sample form template that will replace the out of box blank template used by a new Form Library

InfoPath Best Practice: Naming your Underlying Data Fields Accurately

Figure 7.10 shows a sample form built in InfoPath. When building InfoPath forms, an important thing to watch out for is to make sure your fields are named in a way that makes sense. By default, when you add a field into the form, the underlying data is simply named "field1", "field2", etc. When the form is saved in SharePoint that data is stored as XML. If all your data has generic names like "field1", your XML is not going to be very compelling and will be difficult if not impossible to understand. Also, we will discuss in chapter 9 how to programmatically access InfoPath form data. It is very helpful when doing this to have your data named in an accurate manner. Notice how Figure 7.12 shows all the data renamed in a way that will build a nice XML structure, whereas Figure 7.11 (the default) will produce un-intelligible XML data.

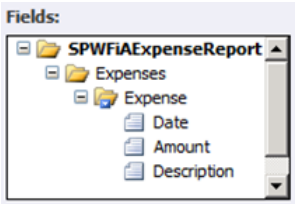


Figure 7.141 It is important to have well named data in your form like this figure shows.

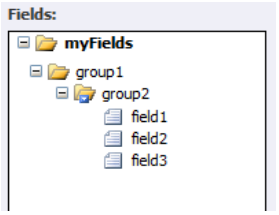


Figure 7.140 It is easy to see that the default way data is named is poor in comparison.

Now that we have our form template built out to our liking, the last thing we need to do is publish the form back into the form library. To do this within InfoPath click the *File* menu, and choose *Quick Publish* (Figure 7.13). After the form has completed publishing, go back to the form library and test out the form. Click the *New Document* button in the ribbon and the custom expense report template should appear.

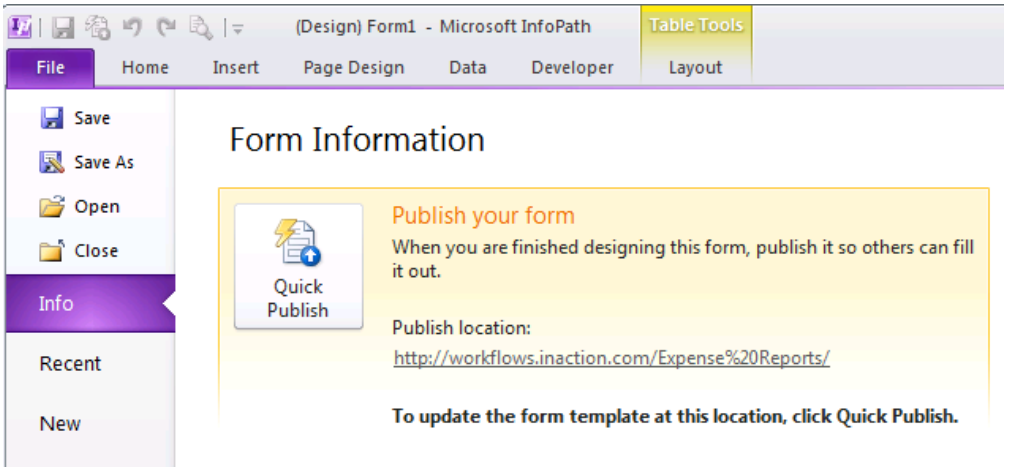


Figure 7.142 After you're done editing the form template, just click Quick Publish to save that template back into SharePoint

A quick publish worked for us in this case because we opened the blank template from the form library itself. So, InfoPath knew exactly what template in what library it should overwrite. You can also publish to a different form library if you like. When you first open the InfoPath Designer client, there are several templates you can choose from, one of which is called SharePoint form library. If you create a new form off this template, you'll essentially be doing what we just did, but in reverse order. Rather than first creating a form library, and then within *Advanced Settings* clicking *Edit Template*, you would be first creating the template from scratch, and then selecting which form library to publish it to (or to create a new form library).

In this case a quick publish won't be available. Instead, in the *File* menu, under the *Publish* tab, there's another option to publish to SharePoint Server (Figure 7.14). This action will load a wizard to walk you through the publishing process. Click the *SharePoint Server* button under the *Publish* tab within the *File* menu to launch the publishing wizard. Follow these steps to get through the wizard that will enable your form to be published to a different form library:

Publishing your Customized Form

	Action	Result
1	Enter the URL to a SharePoint site, eg: http://workflows.inaction.com. Click Next. Note: If you have SharePoint Server, there will be a check box to enable	The publishing wizard will know which SharePoint site to

	the form to be viewable in the browser. Leave this checked	publish the form to.
2	Leave the form library radio button selected. (more on content types in section 7.4). Click Next	Rather than publish to a content type, the form will be published to a form library.
3	Either select to *Create a new form library*, or to update an existing form library if you already have one created. In my case, I will choose to update an existing form library's template. Click Next twice, and thereafter click Publish	The wizard will update an existing form library's template with the newly created template.

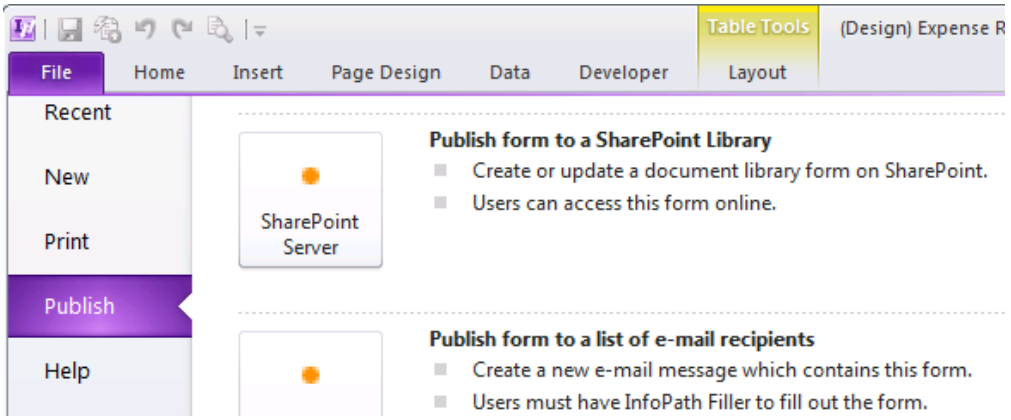


Figure 7.143 You can also publish to SharePoint Server, which loads a wizard that will walk you through where you want to publish to.

7.4 Publishing a Template to a Content Type

Up until this point all we've done is customized a form template for a single form library. What if you need that template to be available on multiple libraries in the same site collection? The best way to accomplish this is to publish the template onto a content type. Wherever that content type is added to a form library, that template will also be added to the library.

Flash back to Chapter 1: Content Types

In chapter 1 we discussed how content types can be used to package up workflows, templates, and metadata to make those "content types" reusable across an entire site collection. Creating content types saves you a lot of time because if you have 10 form libraries that all need the same template, metadata (columns), and workflows you can just add the content type to the library. Otherwise if you don't use content types you'll need to add the template, columns, and workflows onto all 10 libraries individually which can be time consuming and hard to maintain if subsequent updates are made later.

An example of when this would be applicable is, if you had departmental sites in the site collection. Say there was a site for Sales, another for Marketing, and a third site for Human Resources. Each site might need their own expense report library because the approvers are different for each department. Each could still share a common template. What you could do in this case is create an expense report content type with a custom template installed on it. Thereafter, you could add that content type onto each department's form library. With this example, if you need to update the template six months later, that change will be propagated down into the libraries using the content type.

To publish to a content type, you'll want to use the Publish to SharePoint Server feature again. Instead of selecting a form library to publish to, we're going to create a new content type.

Publishing a Form to a Content Type

	Action	Result
1	Publish the InfoPath form to a content type. Either open an existing InfoPath form template, or create a new form from scratch. Click the SharePoint Server button under the Publish tab within the File menu to launch the publishing wizard: A) Enter the URL to a SharePoint site, eg: http://workflows.inaction.com. Click Next. Note: If you have SharePoint Server, there will be a check box to enable the Form to be viewable in the browser. Leave this checked. B) Select the Site Content Type radio button. Click Next. Either select to create a new Content Type, or to update an existing content type if you already have one for another purpose. In my case, I will choose to create a new content type. C) Select to base the content type off the Form content type base, and then click Next. D) Give your content type a name, in this example I entered "SPWFiA Expenses" as the name. Click Next. E) Enter a URL to a form library where your form template will be stored and referenced from, eg. http://workflows.inaction.com/form%20library. Click Next and thereafter, click Publish.	This will publish that new content type into the site collection that was specified in the wizard. Additionally, the form template will be uploaded in the specified form library, and the content type will reference that form as its template.
2	Now the last thing to do is add this content type to all	Now within the New drop down on the form

the form libraries that need to share a common template. Find the library you want to add it to, and go to the Library's Advanced Settings screen to add it.

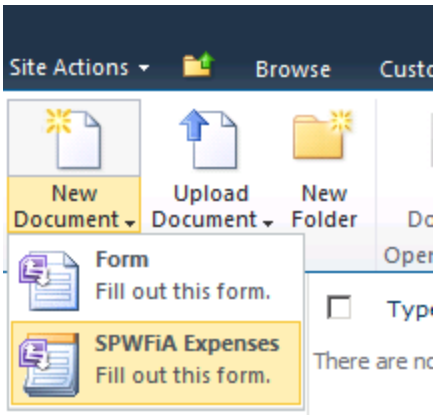
A) Select Yes, to allow the management of content types and then click Ok.

B) In the Content Types section of the Library Settings page, click Add from existing site content types.

C) In the content type category drop down, select Microsoft InfoPath.

D) Select the content type you created earlier, in this case, SPWFiA Expenses. Click Add. Click Ok.

library you'll have your content type listed.



Note: Repeat these steps for each library that needs to reuse this content type and form template.

7.5 Mapping Form Data to Columns

We just got done discussing how to customize the out of box forms, as well as how to use form libraries to house custom form data. If you remember, when you customize the out of box forms, the data is saved into columns on the list item, whereas when you work with a form library the data is stored as XML in the form itself. The trouble with keeping all the data in the form as XML is you won't be able to sort, filter, and group on the data. Perhaps you want to categorize all your expense reports by type of expenses (gas mileage, lunches, travel costs, etc). This is done by adding a field into the InfoPath form, and then mapping that field to a SharePoint list column, whereby you can sort, filter, and group on that column. In some sense, form libraries with mapped columns are the best of both worlds. You can store large amounts of data in the form, and yet extract out into the list item's columns what is especially interesting to your users.

To demonstrate this concept, we are going to go back to our SPWFiA Expense report and add a new drop down where the user can specify the type of expense. Also, we are going to add another field that will contain the sum of all the expenses in the expense report (logic to calculate sum discussed in chapter 9, section 1). We will then re-publish the form back into SharePoint; this time we will map our new columns into the form library that end users can use to sort, filter, and group. Follow these steps to add our two new pieces of data into our expense report form:

Adding Two Columns to the Form that will be Mapped to SharePoint

	Action	Result
1	Add a new Drop Down called "Expense Type" A) In the ribbon, under the *Controls* menu, click *Drop Down List*. Right click the drop down and choose Drop *Down List Properties.* B) Within properties, give the *Field name* a better name, like "ExpenseType" and check the *Cannot be blank* check box below the field name to require the user to select a value C) In the *List box choices* section, add three choices, "Gas Mileage", "Lunches", and "Travel Costs", and then click Ok.	A new piece of data will be added to the form that will track the type of expense. Later this piece of data will be mapped to a new column on the list item.
2	Add a text box called "Total Amount". A) In the ribbon, under the *Controls* menu, click *Text Box*. Right click the text box and choose *Text Box Properties.* B) Change the *Field name* to be "TotalAmount" and under the *Display* tab, make the field read only since this will be a calculated amount. Click Ok. Note: Since it is a read only field, removing the border makes it look cleaner. To take the border off, Right click the text box and click *Borders and Shading*. To take the border off, click the *None* preset. Click Ok	A new piece of data will be added to the form that will track the amount the expense is for. This piece of data will also be mapped to a new column on the list item.

When done, your form should look something like this:

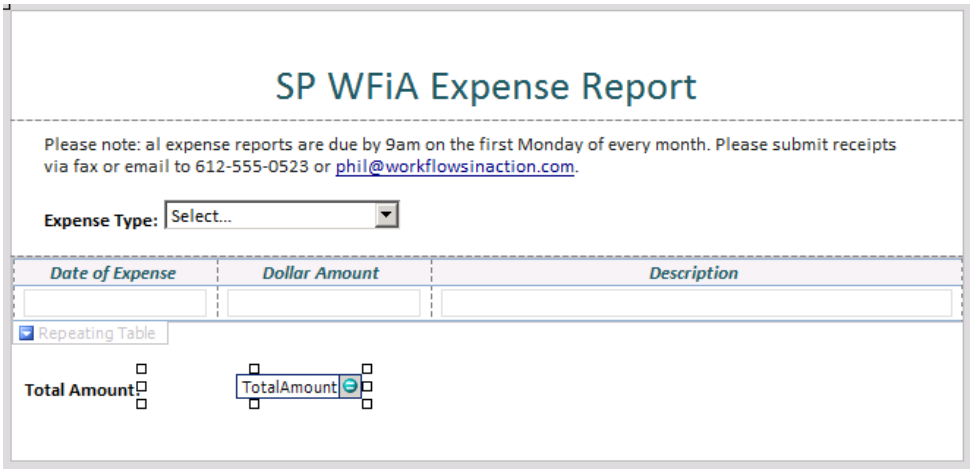


Figure 7.144 The Expense Type and Total Amount fields are added to the form and will be mapped to two new columns on the form's list item.

Instead of doing a quick publish, this time let's publish the long way and add our *ExpenseType* and *TotalAmount* fields as mapped columns onto the list item. This will give us the ability to sort our expenses by the total amount, and filter by expense type. Publish the form through the *Publish > SharePoint Server* in the *File* menu, and specify the URL to the SharePoint site again. It is up to you to decide if you want to publish directly to a library or to a content type instead. In either case you can map the data to columns. When you get to the step in the wizard asking "The fields listed below will be available as columns…", click the *Add* button and add the *ExpenseType* and *TotalAmount* fields (Figure 7.16). Click next and finish publishing.

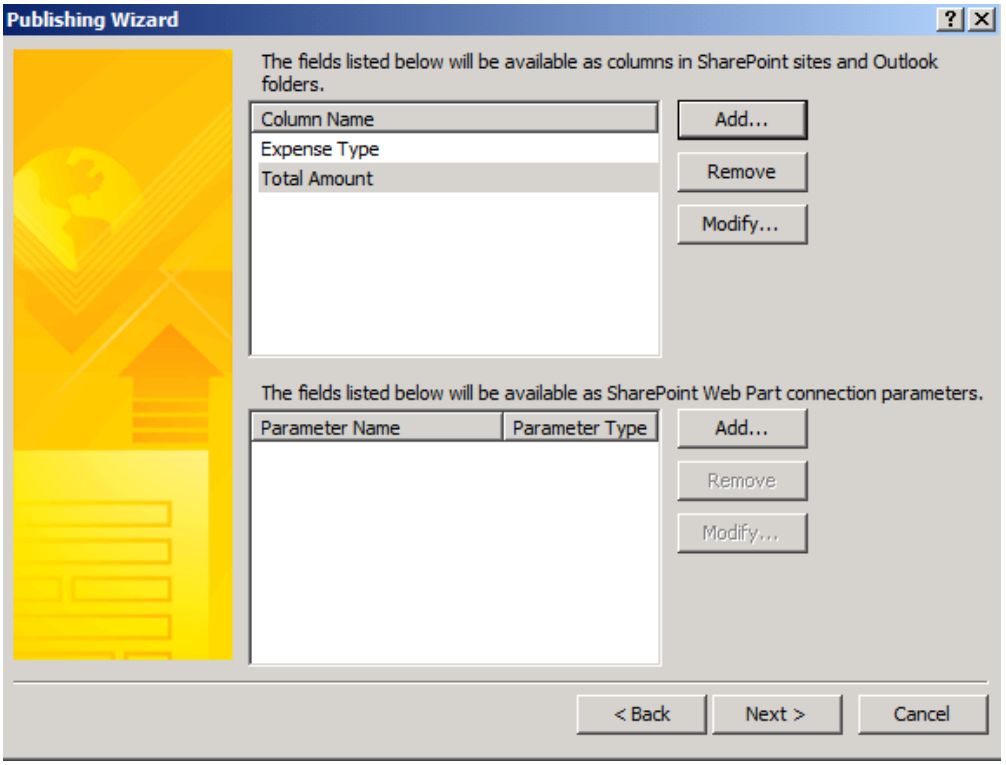


Figure 7.145 When you publish a form, you can specify fields within the form that can map to columns in the SharePoint list item.

Back in the form library that was recently updated, you'll notice two new columns available. If you create a new expense report, and enter several expenses, after you save and close the form you'll notice that the expense type and total amount fields were written to their corresponding columns (Figure 7.17). With this data being mapped into these columns, you can do neat things such as sorting, filtering, and grouping on that data. Additionally, you could use a SharePoint Designer workflow to respond to the submission, and those fields in particular if appropriate.

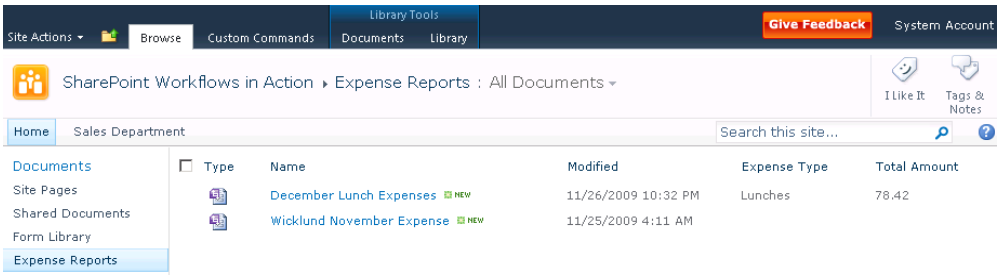


Figure 7.146 When you map fields in the InfoPath client, the corresponding columns are automatically created in the form library when you publish.

Check out chapter 9, section 1 to take this example a step further. At this point the total amount column is not automatically calculated. In chapter 9 we'll add some .NET code into the form that calculates this value by summing all the expenses. With this code in place, the submitter won't have to calculate this manually.

7.6 Forms in SharePoint Designer Workflows

You can use InfoPath to customize initiation and association forms within a SharePoint Designer workflow. By default when a SharePoint Designer workflow is first started, the initiation form is shown to the user, but there are only two buttons showing and the form is empty. With the buttons the user can confirm or cancel the start of the workflow. The association form is shown to the user when they add the workflow to a list, whereas the initiation form shows when the workflow starts.

No Forms Server Dependency

All the previous examples need Forms server to render the form in the browser when the form was edited. SharePoint Designer workflow forms are an exception to this rule. The forms we build in this section (as well as forms in chapter 9) will not require a Server license. Rather, SharePoint Foundation will do.

By default the initiation form includes Start and Cancel buttons and nothing else (Figure 7.18). In some cases you may need to collect information from the user before the workflow begins. You may also need to add additional information to the form to help users understand the workflow and how things may work.

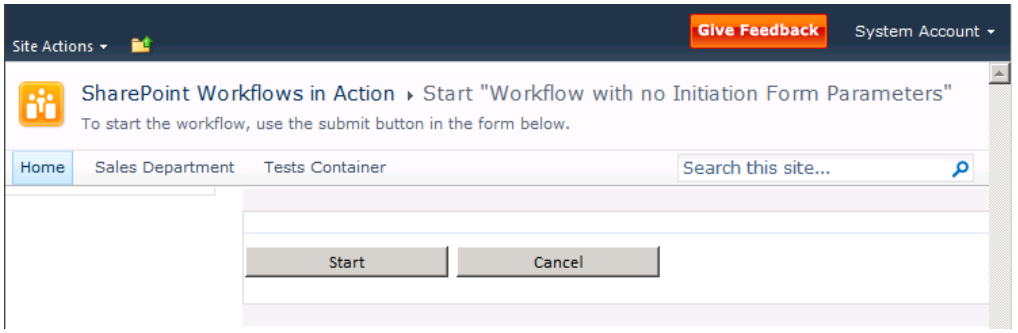


Figure 7.147 When you start a SharePoint Designer workflow it always shows a workflow initiation form. By default this form is empty.

When you need to collect information from the user before the workflow starts, Initiation Form Parameters can be used. These are simply fields that are displayed on the initiation form, which the user can fill out. After the workflow starts, these parameters are available to us in the workflow. These Initiation Form Parameters are very similar to local variables. Adding parameters to the initiation form is easy; just click the Initiation Form Parameters button on the Ribbon, which will open the Association and Initiation Form Parameters dialog (Figure 7.19).

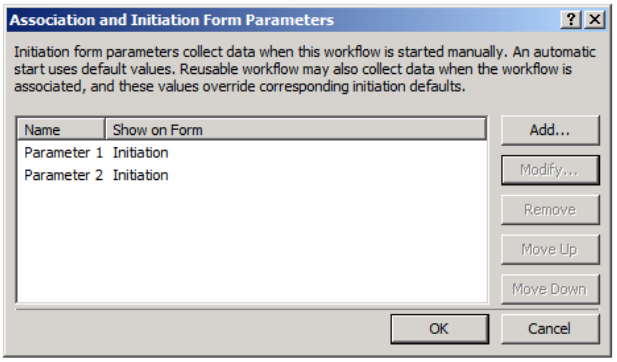


Figure 7.148 You can use the form parameters dialog to add fields onto an initiation form that users will need to fill out before they can start the workflow.

Clicking the Add button will allow you to add new fields to the form using the Add Fields dialog (Figure 7.20). The type of field you select will determine the available settings on the Column Settings dialogs after you click next. For List workflows, you will only be able to create initiation forms because SharePoint Designer will automatically associate the workflow to the list you selected. For Reusable workflows or Site Workflows, you'll be able to specify association parameters. Figure 7.20 shows a drop

downs where you can specify what form you want the parameter to be displayed on, initiation, association, or both.

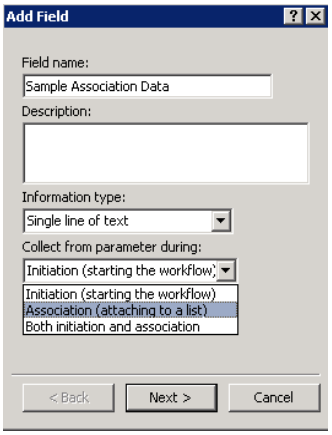


Figure 7.149 Use the Collect from parameter during drop down to choose if you want to parameter to be available on the association form or the initiation form.

When initiation parameters have been added to a workflow, the user will be able to interact with them when they manually start the workflow from a list item. Figure 7.21 shows what the initiation form can look like after the initiation parameters are published.

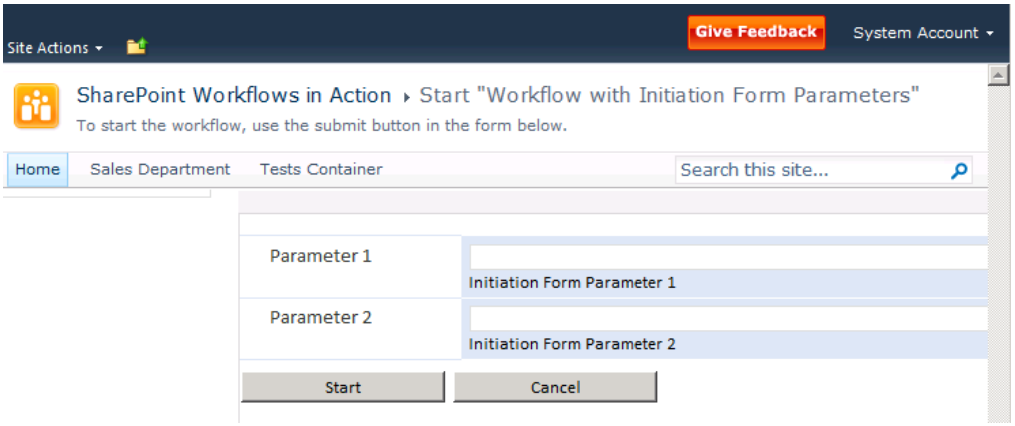


Figure 7.150 After you add some initiation form parameters and publish the workflow, the initiation form will show the parameters you specified.

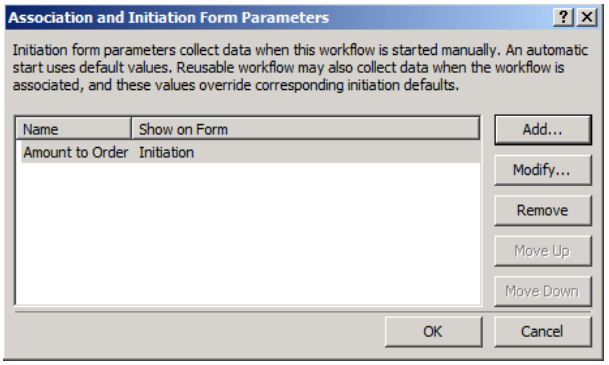
To help walk you through this concept of initiation forms from within SharePoint Designer, we're going to walk through an example workflow that handles inventory requests for parts. A workflow like this

could easily be used in an auto mechanic shop, where they manage the quantity of parts they have in stock, and employ a workflow to handle requests to order more parts by their mechanics.

The example will work off a custom list titled Parts Inventory. The Parts Inventory list will have 3 columns, Part Name, Quantity Available, and Base Cost. What we want is a workflow that our users can use to request additional parts. If they notice they are getting low on a given part, they can put in a request for more to be ordered. Then, depending on the costs and how many they want ordered, different approvers will be assigned. An initiation form will come in very handy in this workflow. When the user starts the inventory request workflow, he will see a custom initiation form prompting him to enter how many parts he wants to be ordered.

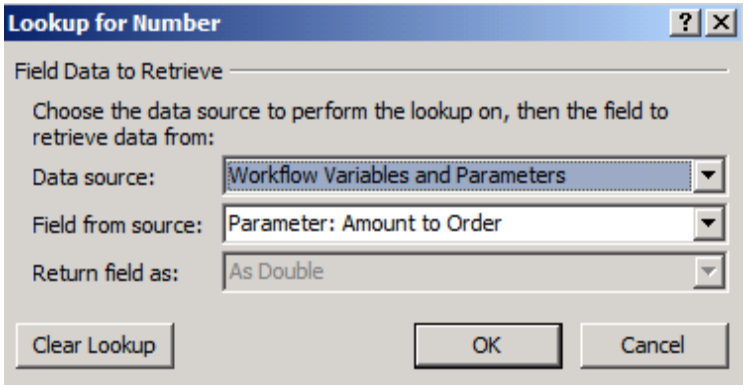
To setup this example, begin by creating a new SharePoint list titled Parts Inventory with the three columns. This list will store our parts and host the workflow. Then, create a new List Workflow in SharePoint Designer off this list. Open SharePoint Designer to the site where you have your Parts Inventory list, and on the workflows tab click List Workflow and select your custom list. At this point you'll be prompted to enter the title and description of your workflow. Enter "Parts Request Workflow" as the title and click ok. Once in the workflow, let's start by specifying our initiation form data.

Adding Initiation Data into the Form

	Action	Result
1	Add a initiation parameter into the form: A) In the Ribbon, click *Initiation Form Parameters and* click *Add.* B) Add a field titled *Amount to Order* and click Next. Users can use this field to enter how much of the part they need C) Specify a minimum value of 0, *click Finish, and then* click *Ok* to save the parameter.	

Now with this initiation parameter specified, the user will see a form when they first start the workflow. The form will ask them to enter the quantity of the part they want to order. What we can do now is add some business logic into our workflow to react to the submission. Let's use the Do Calculation action to calculate the cost of the request. We can then add an if/else block to calculate if the total cost is greater than $1,000, and if so Joe Bob the inventory manager needs to approve the request, otherwise Fred Fredrickson, the assistant manager, can approve it. Follow these steps to setup this action and condition:

Adding Initiation Data into the Form

	Action	Result
2	Create a workflow variable to hold the total cost. A) In the Ribbon from within the newly created Part Request workflow, click *Local Variables.* *B)* Add a new number variable called *TotalCost.*	A new variable will be created that stores the Total Cost.
3	Add the *Do Condition* action into the first step. Configure as follows: A) Click the first value and specify the *Base Costs* column under the *Current Item* data source B) Select *multiply* by as the operator C) Click the second value selection and this time change the *DataSource* drop down to be Workflow Variables and parameters D) Select the *Amount to Order* variable for *Field from source:*  E) For the last setting, set to output the calculation result to the *TotalCost* variable.	The final result of the calculation should look like Figure 7.22

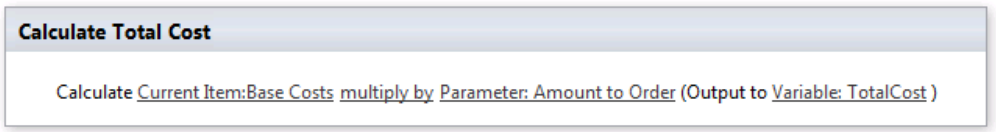


Figure 7.151 You can use the Do Calculation action to perform mathematical calculations. In this case we're multiplying the cost of each part by the amount the user wants to order.

Adding Initiation Data into the Form

	Action	Result
4	Add an *If any value equal value* condition into a second step. A) For the first value, specify the *TotalCost* variable, and set the operator to be greater than 1,000. B) If the condition it true, add an email action to email the Manager. If the condition is false, email the assistant manager.	

Now that we have the basics of the workflow lets publish and test the initiation form. Click the Save button in the Ribbon to save the workflow, and then click the Publish button to publish the workflow to the Parts Inventory list. On the list, select a list item and start the Parts Request Workflow. Instead of the workflow starting immediately, you'll be prompted with a form you need to fill out. The workflow won't begin until you click the start button. For our workflow, you'll notice a field to enter the quantity of parts you want to order (Figure 7.23).

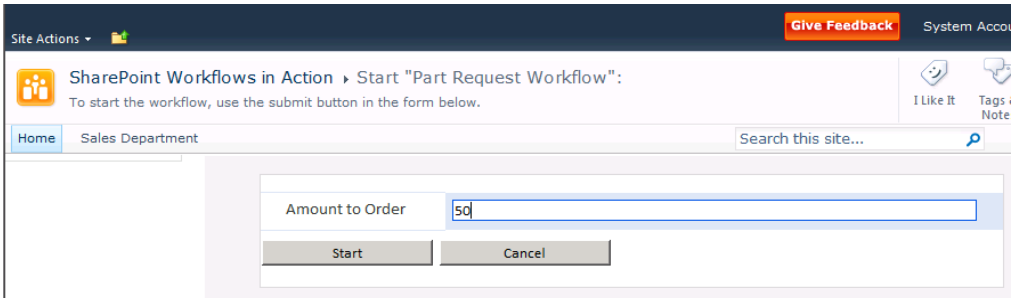
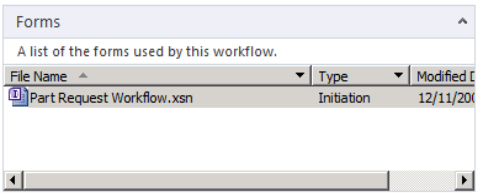


Figure 7.152 When you start a workflow that has initiation parameters set, an initiation form will automatically load asking the user to provide values for those parameters.

After we hit the start button, our workflow will perform the calculation to determine who should approve the order request, and an email will be sent to the corresponding approver. Now before we call this example done, we really need to go back and customize this form a bit more. It has the minimum data fields we want on it, but it isn't really super intuitive to the user. Wouldn't it be nice to show to the user some descriptive text that describes the $1,000 milestone a bit more? We can accomplish this by editing this form in InfoPath. Follow these steps to do this:

Customizing the Initiation Form

	Action	Result
1	Open the Initiation Form in InfoPath. A) Back in the workflow within SharePoint Designer; click *Workflow Settings* in the ribbon to navigate to the settings page. B) Within the settings page you'll notice a Forms box that shows all the InfoPath forms that this workflow is using. Click on the form and the InfoPath client will appear	
2	From within the client, make any changes you see fit, like adding more descriptive text. When your changes are complete, under *File* click *Quick Publish*	The form will reflect any unique look and feel requirements you may have.

After you publish your form, it will look something like Figure 7.24. This is a rather generic example, but don't be fooled because there are a lot of powerful things you can do with InfoPath in your initiation forms. Another common thing to do is populate a drop down with external data (covered in the previous chapter's example).

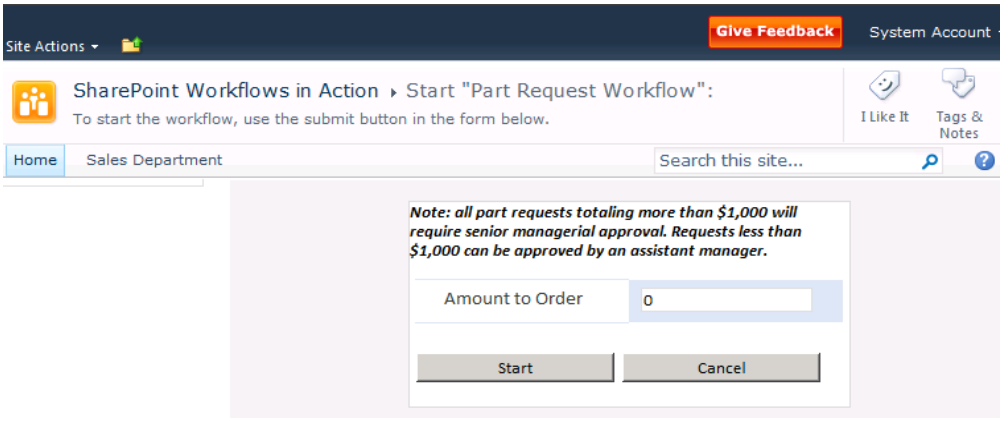


Figure 7.153 After you edit a workflow's form, and publish the changes through InfoPath, those changes will show when users go to edit the form. In this case the form now has some informational text at the top to help guide the user through the process.

7.9 Summary

Workflows and forms are two pees in a pod. You rarely have a workflow that doesn't have any forms, because most workflows rely heavily on human interaction. Most of the time that human interaction takes place via a form where the user is entering some information into the system.

There are several tooling options you must choose between after you decide to build custom forms for these workflows. There are three main tools you can use to build custom forms in SharePoint with: out of the box forms, InfoPath forms, and ASP.NET forms with Visual Studio (as seen in Chapter 9). The out of the box forms provide the easiest and fastest approach to giving your end users a user interface that can enter data and information. The out of box forms have their limitations. When a more complex form is required, InfoPath is the next best choice. Be aware that SharePoint Server edition is needed to host InfoPath forms in the browser, otherwise the InfoPath client will need to be installed on all your end user's workstations.

Beyond the tools related to form development, there are many foundational techniques to working with forms that really need to be understood to be a good workflow solution builder. This includes knowing how form libraries work and how to build a custom form with InfoPath. After you have a custom form built, you can update the default template in your form libraries to make those forms available to your end users to be filled out. The default template is blank, and not terribly usefully. By customizing that template you can really spice up your form libraries.

You can also display forms to those using your SharePoint Designer workflows. This includes two opportunities; when the workflow is added to a list (association form), and when the workflow first starts (initiation form). Using these forms is a good way to get some data from the user before the workflow starts executing.

8

Custom Visual Studio Workflows

Using Visual Studio to build your custom SharePoint workflows enables you to build the most complex workflows possible. The possibilities are really limitless. With Visual Studio, you're writing your workflows from scratch, albeit still using the Windows Workflow Foundation architecture.

Visual Studio isn't a new tool for SharePoint 2010. In fact, it was first released in 1997 under the name of Visual Studio 97, well before SharePoint even existed. This was Microsoft's first attempt to have one development environment for many languages (c++, j++, and InterDev). By the time we got to SharePoint 2003, Visual Studio 2003 had some meager SharePoint extensions, allowing SharePoint developers to more easily package up their development customizations and deploy them into SharePoint. Even with these extensions, you still needed a lot of "under the hood" knowledge, which made their value negligible.

As Visual Studio advanced, the SharePoint extensions did not. The 2005 and 2008 versions of Visual Studio did not bring much more to the table. It wasn't until the 2010 version of Visual Studio that SharePoint developers finally started to feel like first class citizens. Right out-of-the-box, SharePoint developers will get the famed "F5 Experience", where they simply hit F5 and all their customizations are auto-magically packaged up and deployed into SharePoint. Given that the learning curve for SharePoint developers was really high in the 2003 and 2007 versions of SharePoint, Visual Studio 2010 is definitely going to bring a sigh of relief, for new and experienced developers alike.

As far as workflow development goes, the same packaging and deployment benefits are still applicable. Simply create a new SharePoint workflow project, and all the necessary feature and package files will be created. Hit F5 and your workflow is sent to SharePoint, and ready to test!

Since Visual Studio 2010 makes it so easy, all you have to worry about is whether you want to build a Sequential workflow, or a State Machine. If you remember from chapter 1, sequential workflows always flow in one direction - forward. State Machines on the other hand simply go in and out of different "states" until the workflow logic tells it to end. No matter which route you choose, you'll use the same techniques for dropping activities onto the Designer surface, as well as debugging and versioning techniques.

Lastly, if you're ok with using a Sequential workflow, a great new feature with Visual Studio 2010 is the ability to import a SharePoint Designer workflow. This can be great for many reasons. One neat

reason is if you have an existing workflow in SharePoint Designer. After a year or so goes by, your business requirements change and become more complex. This increase in complexity necessitates a Visual Studio workflow. Rather than start from scratch, simply import what you've got into Visual Studio!

8.1 Introducing Visual Studio Workflows

A Visual Studio SharePoint workflow is made up of 3 basic things, a template, deployment artifacts, and a type. The template is often times called the "designer surface". This is where you'll drag and drop workflow activities and structure the flow of work that the workflow is navigating through. Activities are reusable pieces of code that interface with this template, and are droppable on its surface. The surface offers a visual representation of what the workflow is doing, simplifying readability and maintenance.

The deployment artifacts are one of the most exciting new aspects Visual Studio 2010. Visual Studio workflows are deployed into SharePoint via Features and Solution packages. A solution package packages up and deploys the workflow into SharePoint. The feature is what enables the functionality, when active. In SharePoint 2007, workflows were deployed in exactly the same way. You had to create these artifacts from scratch, or rely on an un-supported third party. With Visual Studio 2010, these artifacts are created automatically, and little knowledge of their workings is necessary.

Lastly, a workflow always has a type. It's either a sequential workflow, or a state machine. A workflow cannot change its type. It has to be recreated if that's necessary.

8.1.1 Working with the Workflow's "Template"

When you first create your Visual Studio project using one of the workflow templates, a auto-generated workflow will be in that project titled Workflow1. When you double click that file, the workflow "template" will load (Figure 8.1), showing you the workflow designer surface. This template is where you can start dragging and dropping activities and start structuring the flow of work.

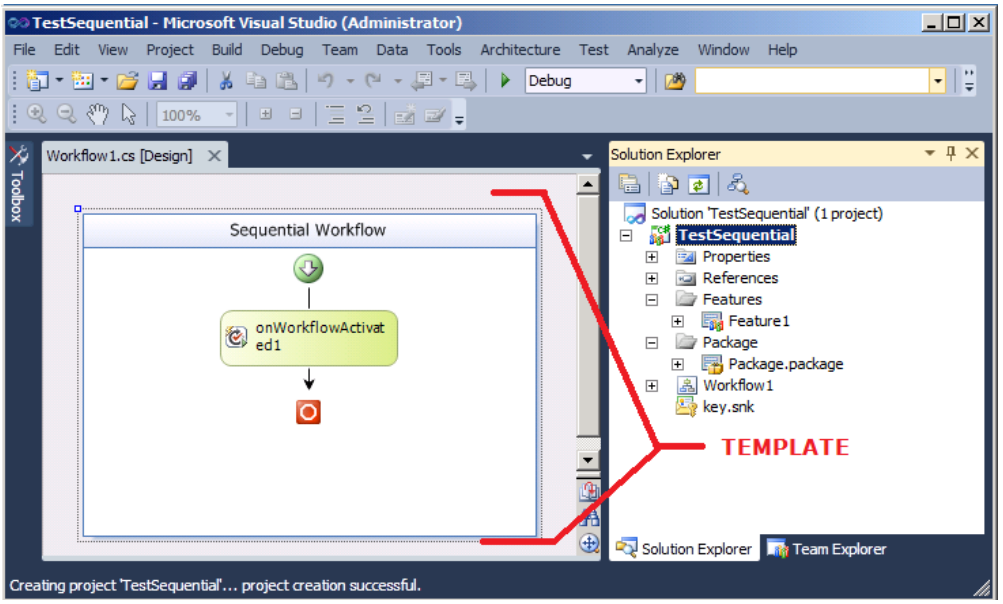


Figure 8.1 The workflow's template is where you can drag and drop activities and start positioning the flow of work.

Toward the left of the template is the toolbox (Figure 8.2). Within this toolbox are all the out of box activities you can drop onto the template. In general, the rule of thumb is to put all your code in activities, and only use the template to position the activities. Sometimes people just put all their code in the template, but that negates the visual benefit that the designer surface offers. For example, you can right click an activity and choose "generate handlers". This will create a method in the template's code behind, where you can start adding code. If this code needs to be reused in other workflows, it's better to encapsulate that code in a composite activity. A composite activity is a custom activity that calls other activities. Then, instead of adding that code to the template, you can simply drag and drop your composite activity onto the template. This discussion on custom activities is covered more deeply, in chapter 9, Custom Workflow Activities.

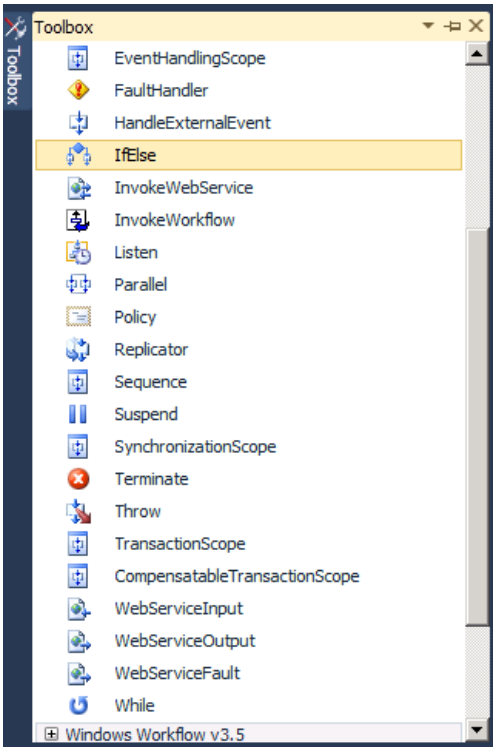
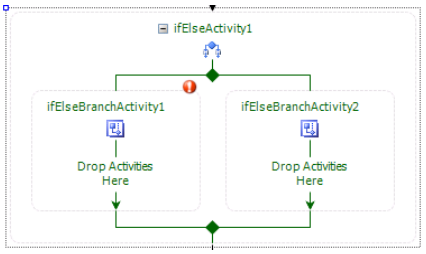
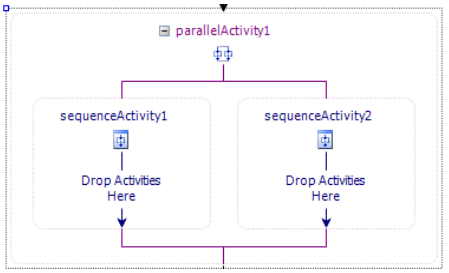
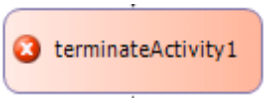


Figure 8.2 There are dozens of out of box activities available for use in your workflows. The activities are found in the tool box in two categories, Windows Workflow and SharePoint Workflow.

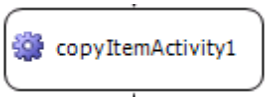
Before you go starting to build custom activities, notice how there are dozens of out of box activities available for you to use. Many are quite powerful, requiring little configuration to use. Below is a list of 10 fundamental activities, and what they can be used for:

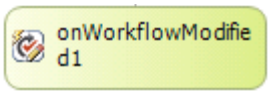
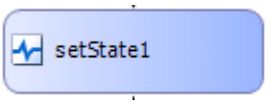
Top 10 most Fundamental Activities (Windows Workflow Activities)

	Name - description	Visual
1	**Code** - the Code activity allows you to drop code into the template. If you don't want to go through the process of creating a custom activity, or you don't think the code will ever be reused, this activity may be your best choice.	

2	**IfElse** - You guessed it, the IfElse activity allows you to make logical decisions in the workflow. You can add additional "else if" branches. Each branch requires a method that calculates if the condition is met or not.	
3	**Parallel** - the Parallel activity allow you to be able to run 2 or more "trunks" of activities in parallel. Otherwise, the activities would have to run in sequence.	
4	**Terminate** - the Terminate activity simply terminates the workflow.	
5	**While** - a While activity is used for looping purposes. You can have activities inside the workflow that repeat themselves over and over again while the While condition is still true.	

SharePoint Workflow Activities

6	**CopyItemActivity** - This CopyItemActivity activity allows you to create a copy of a list item or document in another document library or list. This activity is good for archiving or moving documents.	

7	**CreateTask** - The CreateTask activity creates tasks in tasks lists. See chapter on task processing for more information.	
8	**OnWorkflowModifed** (and **EnableWorkflowModification**) - OnWorkflowModified activity responds to when the workflow is modified. "Workflow Modifications" are a way for users to change the behavior of a workflow after it has already started (see chapter on workflow forms for more information).	
9	**SendEmail** - The SendEmail activity uses the exchange server specified in SharePoint Central Administration to send emails to users.	
10	**SetState** - The SetState activity is used to set the state of the workflow. When a workflow starts, a new column is created in the list or library. Instead of the default "In Progress" or "Completed", you can define custom states to show in this column.	

8.1.2 Workflow Deployment Artifacts

As was mentioned, a SharePoint workflow relies on a Solution Package to be deployed into SharePoint. Thereafter, it relies on a Feature to be activated in a SharePoint site, thus enabling the workflow to be added to sites, libraries, etc. When you create a new SharePoint workflow project in Visual Studio, the package and feature are automatically created and configured for you (Figure 8.2).

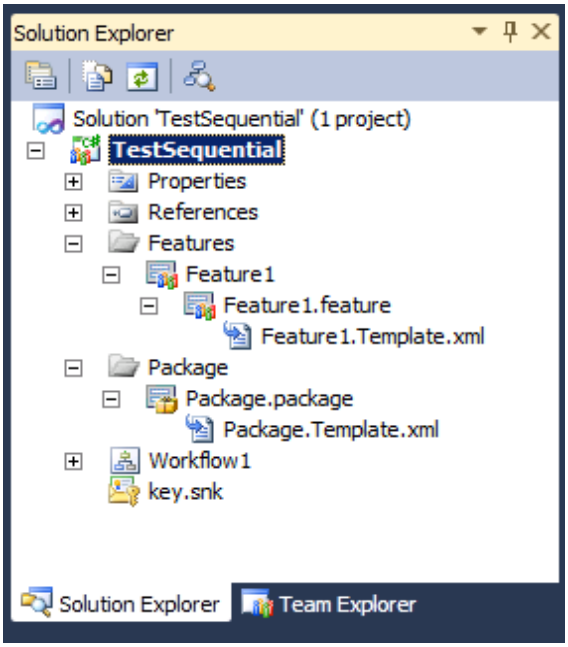


Figure 8.3 New with Visual Studio 2010, the Solution Package and Feature files are created automatically for you. A default feature, titled Feature1 is in a folder called Features, and a package titled Package.package is in the Package folder.

After you have built out your workflow, all you have to do is hit F5 (or right click the project name, and click Deploy) and your workflow's package will be deployed, and the feature will be installed. When you are first creating the project, the new project wizard will ask you what SharePoint site you want to deploy your workflow into (Figure 8.4). This is how the F5 knows where in SharePoint to deploy to.

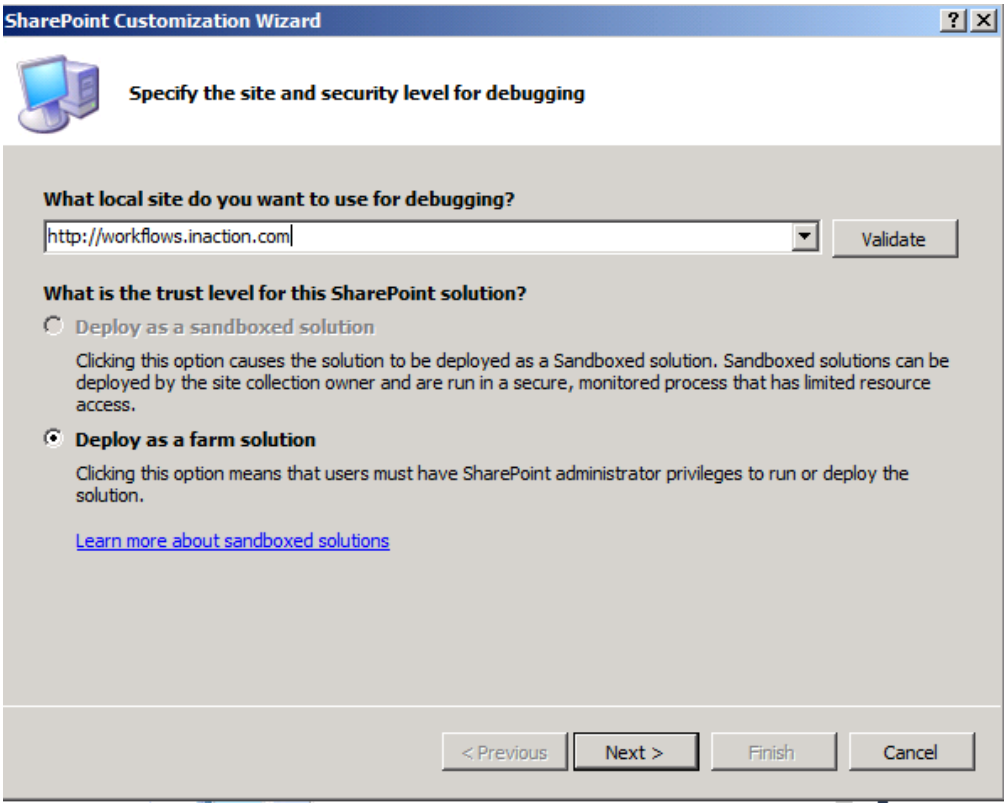


Figure 8.4 When you first create a workflow project, you'll be prompted to enter the URL of a local SharePoint site. Then when you're done building your workflow, you can hit F5 build, deploy and start testing the workflow.

What's important to remember here, is that the wizard is assuming a local installation of SharePoint. It doesn't support remote deployments. What you'll want to do is install SharePoint 2010 locally (like on your personal laptop) where you can do your development and unit testing. After you've finished testing your workflow, you can then manually install and deploy the solution package into another environment.

Even though the solution package and feature is created and configured for you, it's important to know a little bit about how to work with them manually. Some of the more advanced techniques require some "under the hood" knowledge, so let's briefly look at these files to see what they're made of and how they work.

Let's start by digging into the solution package. Now technically, you wouldn't have to use a package. A package is simply a deployment tool, allow you to guarantee consistency across your servers because instead of manually copying and deploying files, you're letting the package do all the work. Really all a workflow needs is a DLL and a Feature that activates it. But rather than manually copy files, let's stick with the package that is created for us. When you double click on Package.package, a GUI (graphical user interface) will appear looking similar to Figure 8.5. Notice how in Figure 8.5 there are two main panels, a left panel showing what's in the solution, and a right panel showing what's in the

package. The left shows SharePoint items in the solution that are available to be added to the package, but have not yet been added. If you click the right or left arrows, you can move object in and out of the package. Notice how there are 2 features, on is in the package, and one is not in the package:

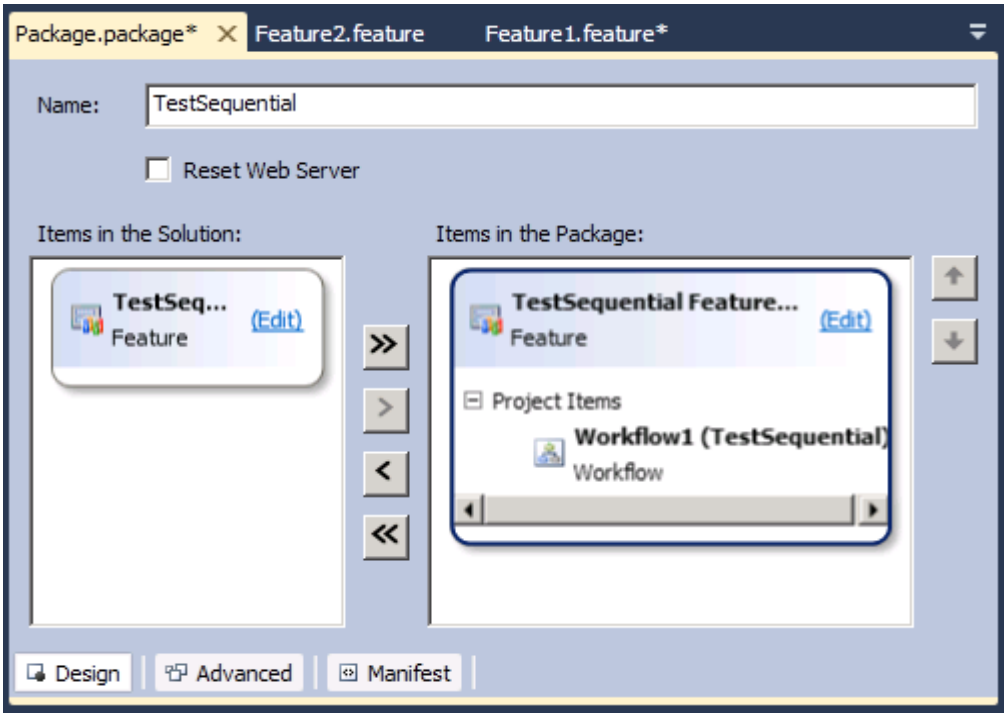


Figure 8.5 Visual Studio 2010 offers a nice GUI that can be used to manage your packages. Simply move SharePoint items in and out of the package, and deploy!

You can also use the Package Explorer (Figure 8.6) to drill into what is in the package. Figure 8.6 shows how the package has one feature in it. Inside that feature is a workflow. When that package is deployed, and the feature is activated, the workflow will thereafter be ready for use.

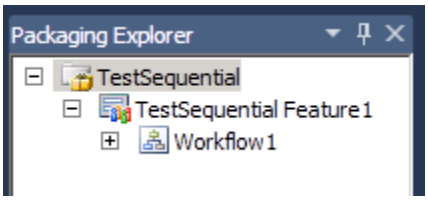


Figure 8.6 The Package Explorer gives you a hierarchical view into what's inside your package.

At the bottom of the package you'll see three tabs (Figure 8.7). The first tab, the Design tab, is the GUI that you first see when you double click the package in the solution explorer. A package is essentially just a CAB with and XML manifest file. This GUI is what is generating the XML manifest on your behalf. If you still want to manipulate the XML manually, switch over to the Manifest tab and you can start editing the XML directly. Lastly, the Advanced tab allow you to add and remove other assemblies from the package. Perhaps you have a third party assembly that you also want deployed with the package.

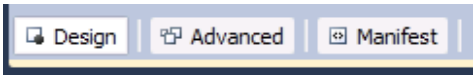


Figure 8.7 If you need to make more advanced customizations to your package, switch to the Manifest tab, where you can manually edit the package's XML.

Lastly, double click the Feature1 feature item in the solution explorer. A different GUI (Figure 8.8) will appear, it looks very similar to the package's GUI. On the left you'll see SharePoint customizations in the solution that are available to be added to the feature, but have not yet been added. On the right are items that have already been added. Figure 8.8 shows that the feature contains a workflow named Workflow1. There is a second workflow as well as a web part that have not been added. Only Workflow1 will be enabled when this feature is activated in SharePoint.

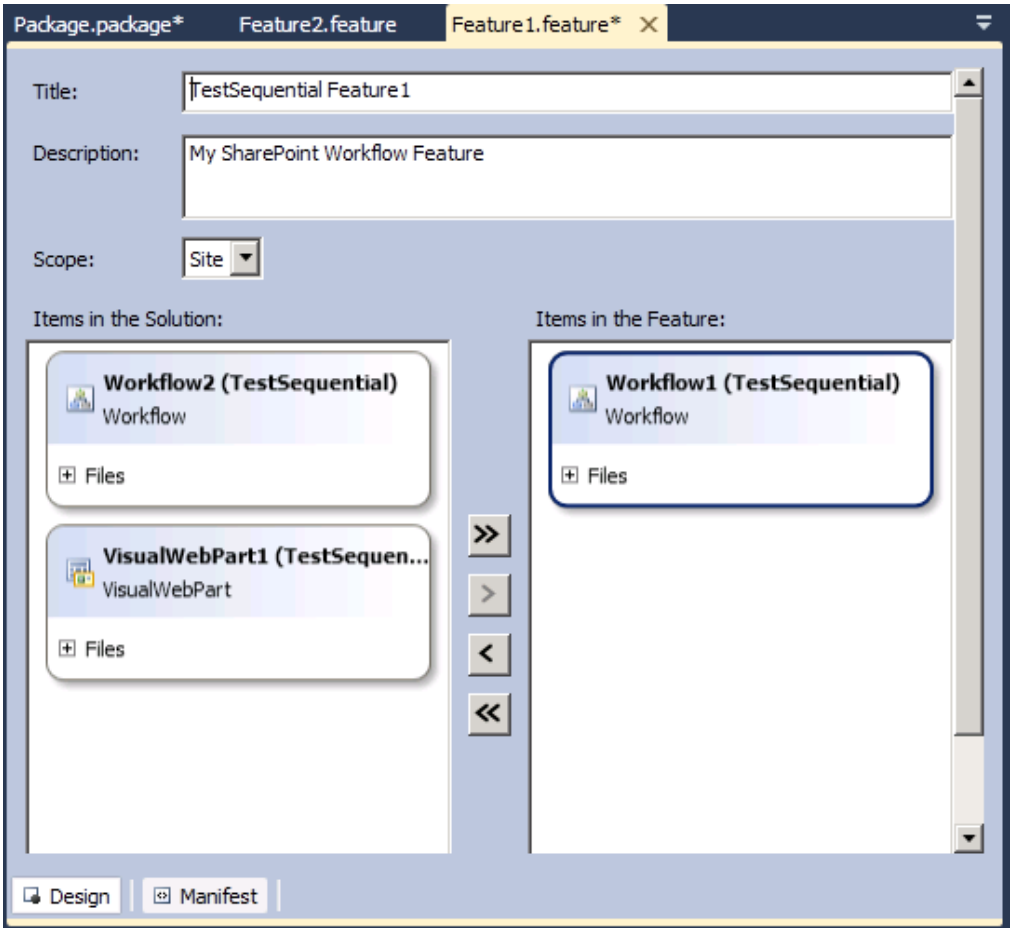


Figure 8.8 Similarly to packages, features are also configured with a nice GUI. You can move things in and out of the feature, thus controlling what functionality is enabled upon the feature's activation.

8.1.3 Sequential vs. State Machine Workflows

The way in which a workflow will execute from one step in the process to another, falls in one of two categories. A workflow is either:

- Sequential, in that the steps within the workflow execute sequentially, one after another.
- State Machine, where it executes in no particular order.

A sequential workflow always progresses forward, and never goes back to a previous step. Figure 8.9 shows how a sequential workflow looks on the Visual Studio workflow designer surface:

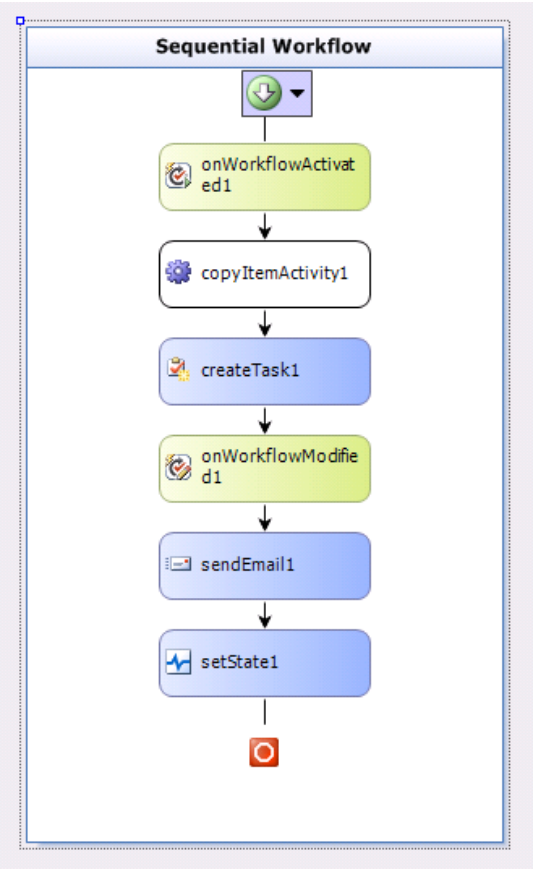


Figure 8.9 Sequential workflows only move in one direction... forward.

A State Machine on the other hand has no such constraint, but rather moves from one state to another, until some logic concludes the workflow has completed. A good example of a state machine is a bug tracking workflow that tracks bugs in a computer program. When the workflow first starts the bug may be placed into a "Pending" state, where it waits for a developer to be assigned to the bug and start working on the bug. Thereafter, the developer starts working on the bug and fixes it, putting the bug into a "Fixed" state. When a bug is fixed, a tester goes to confirm the resolution of the bug, in which time he finds that it was not fixed and places the bug back in a "Pending" state. This ability to go back in time, or to a previous state, is only available with State Machine workflows. Figure 8.10 shows how a state machine workflow looks on the Visual Studio workflow designer surface:

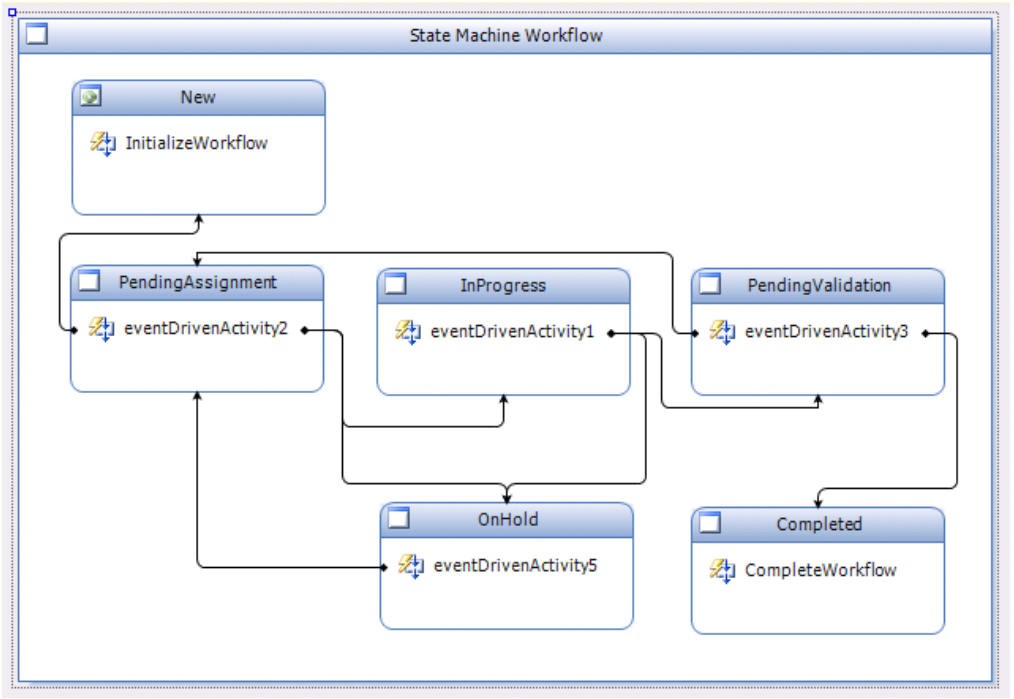


Figure 8.10 State machines on the other hand can go any direction you tell it to go.

When you create a new Visual Studio project, you'll need to decide which workflow type you're going to go with. The new project wizard (Figure 8.11) will show two workflow project types. The project template you choose isn't actually that critical. This is because a workflow is an item inside a project, not the project itself. If you choose the Sequential project template, you can always delete the auto-generated sequential workflow, and replace it with a state machine if you so choose. Remember it's not an upgrade, so it's still important to think through your requirements first.

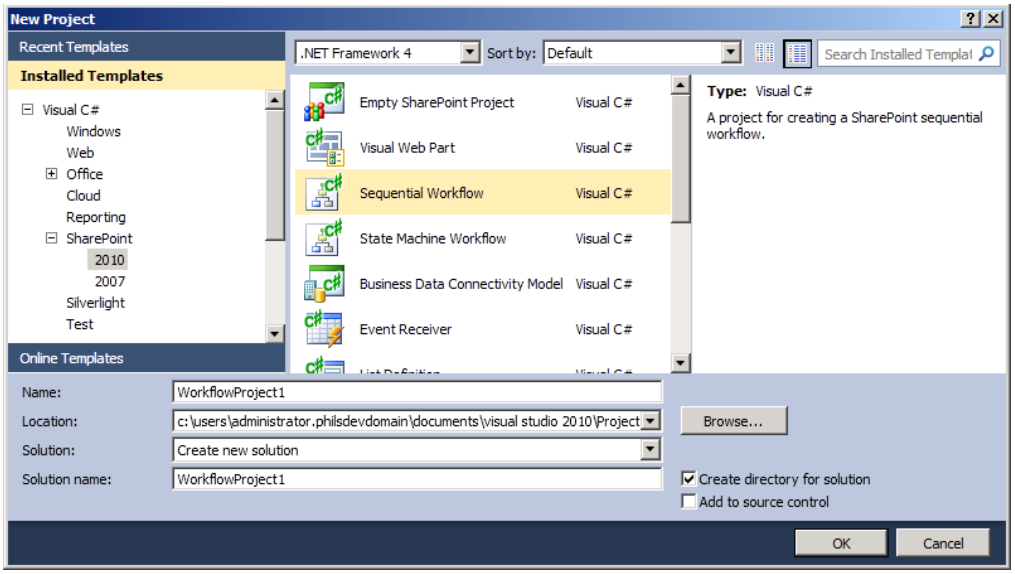


Figure 8.11 When you create a new project, you can use the Sequential or State Machine workflow project templates. By choosing one of these templates, the workflow, package, and feature files will be setup for you automatically.

Obviously some business processes will require a state machine and others won't. It is important to think through your business process's requirements before you start building a custom workflow, because it is difficult to change course after you have already started down a path. If you start building a sequential workflow, but later decide it really ought to have been a state machine, you're really talking about starting over. Bottom line - think through your business requirements before starting!

8.2 Building a Sequential Workflow

Now comes the time to build your first SharePoint workflow in Visual Studio. Since sequential workflows are less complex than their state machine counter parts, we're going to start here. To add some real world flavor into the mix, we're going to be building a workflow that tracks the progress of maintenance work orders. The setting will be a college that needs a system for students to submit maintenance requests for their dormitories. When a student submits a work order, the order is assigned to one of the maintenance workers who then performs the work, thus closing out the order and completing the workflow. Figure 8.12 shows the "Happy Path" this workflow will take:

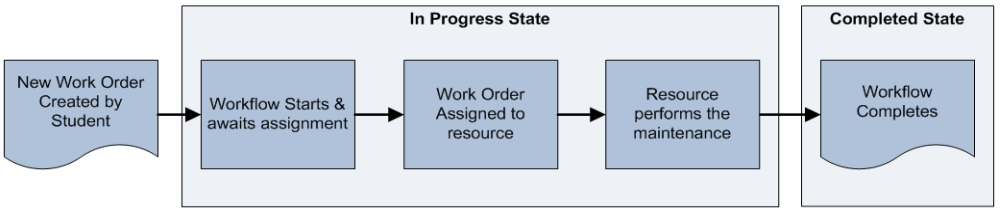


Figure 8.12 The example in this chapter features a Work Order processing workflow. A new work order is created, assigned, and then the work is completed.

I used the phrase "Happy Path" because we all know that business processes are never so simple. Oftentimes the process has "alternate paths" that the workflow takes depending on certain circumstances. Additionally, when you factor in "technical" aspects, the flow changes as well.

Consider with me how we will technically achieve this example in SharePoint. We'll need a list titled "Work Orders" and students can add items into that list. After an item is added, the workflow needs to wait for the status of the work order to change. We'll have 4 possible statuses:

- Pending Assignment
- Pending Completion
- On Hold
- Completed

When the manager assigns the item to a maintenance worker, at the same time they'll change the status to Pending Completion. Our workflow then needs to wake up and email the worker that they have a new order to complete. To accomplish this we're going to leverage the OnWorkflowItemChanged activity. This activity sits and waits for the item the workflow is running on to change. In our case, when a change happens we want to look at the status column to see if the status changed. If the status changed, we're going to take an action. If the status didn't change, we want to resume waiting. Figure 8.13 show's a closer representation into how the flow actually will look:

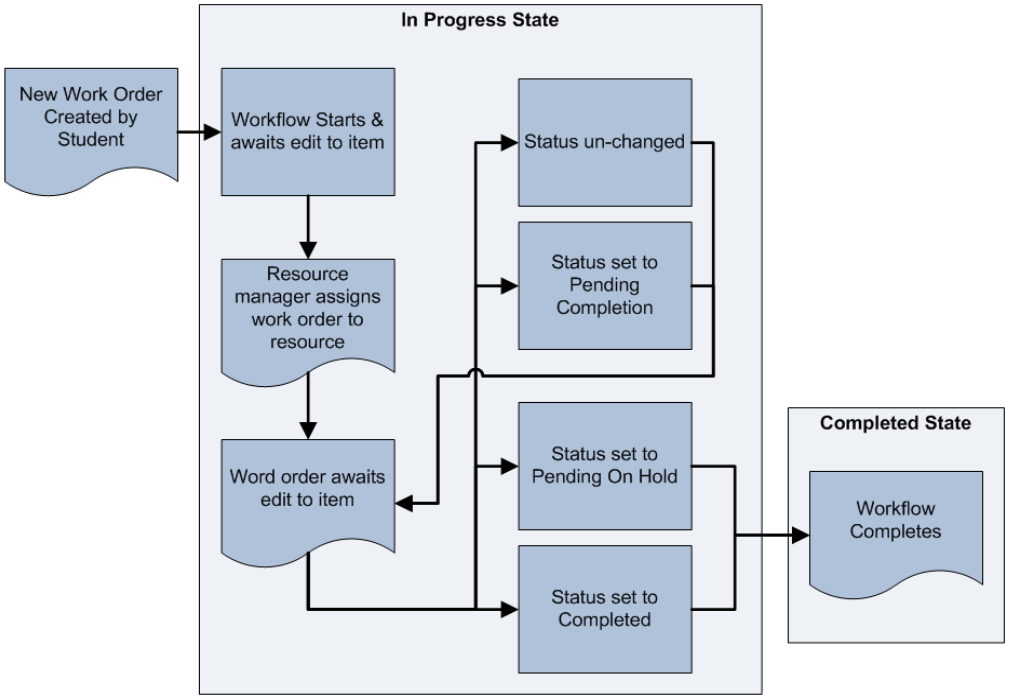


Figure 8.13 Technical challenges make the flow of work a bit more complicated. The workflow will wait for the item to be updated, and each time it is updated, it will check the item's Status column to know what to do next.

You may be wondering at this point, are we still building a sequential workflow? The answer is yes. We can use a While loop to keep looping until the status is set to either On Hold or Completed. If the status is not set to either of those, the workflow will loop back and use the OnWorkflowItemChanged activity again to wait. It'll keep waiting forever until the order is completed or put on hold.

Figure 8.14 shows what the new work order form may look for the student submitting the order. Alternatively, the maintenance worker could use the same form when they change the order's status to completed. Notice the Status column and the Assigned to column. Typically, you wouldn't want the student to see these fields. You could use a custom InfoPath form to hide these fields for students, but show them form managers and the maintenance workers. Reference the "Custom Form Fundamentals" chapter for more information on how to do this.

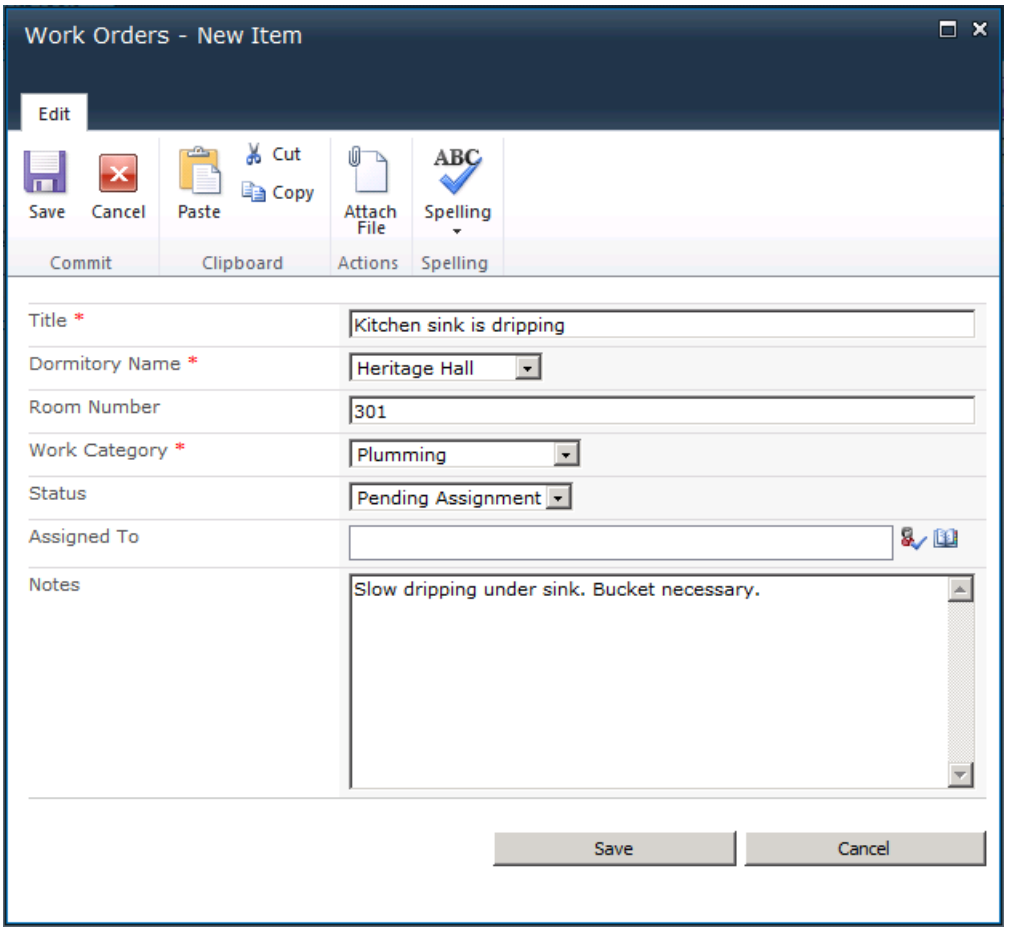


Figure 8.14 The example features a list that contains work orders. Each work order has several pieces of metadata including a title, dormitory name, and work category.

With the stage set, let's go ahead and start building the workflow. Open Visual Studio 2010 and create a new project by clicking File > New Project. When the project template popup displays, choose the Sequential Workflow template. Title the project "WorkOrderProcessing". When you create the project, enter the URL to your SharePoint site, and go ahead and bind the workflow to your Work Orders custom list that has columns similar to what's seen in Figure 8.14 (Figure 8.15 shows the wizard screen where you bind the workflow to the list).

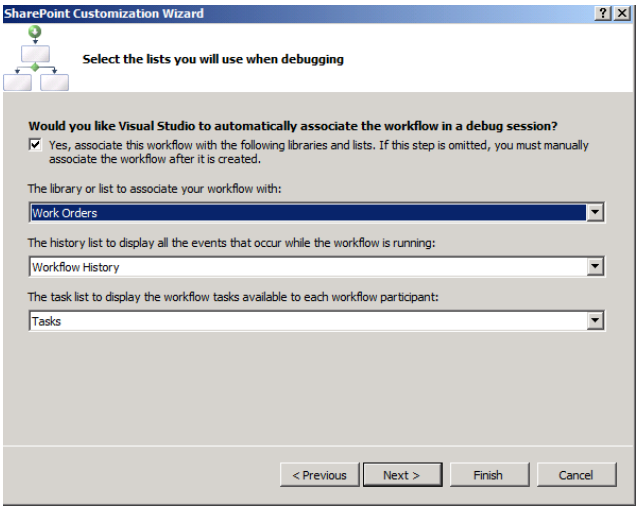
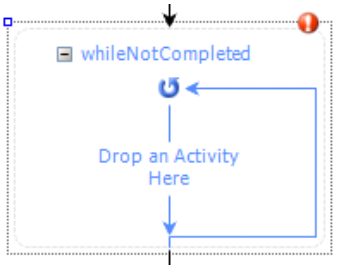


Figure 8.15 When you create a new project, go ahead and set it to bind the workflow, automatically, to your Work Orders custom list.

Double click Workflow1 and configure the high level activities such as the while loop, OnWorkflowItemChanged, and the if-else branch:

Configure the main activities for the Work Order Processing workflow

	Action	Result
1	Configure a While activity: A) Drag and drop a While activity from the toolbox onto the workflow template. B) Right click the activity, choose Properties, and change the name of the activity to whileNotCompleted. C) Notice the red exclamation point. To resolve part of this error, under Properties again, change the Condition property to be Code Condition. D) Click the plus symbol next to the Condition property, and type a method name of WhileOne. This is the method that the activity calls to see if it should keep looping. E) When you type WhileOne, you'll be taken to the workflow code file and the newly created WhileOne method. Enter this one line of code: `e.Result = true;` Note: setting the Result to true or false is what	A while activity will be setup to continue looping:  Note: the While activity will have the red exclamation point until at least one child activity is added.

keeps the While activity looping or not. True means keep looping, false means stop looping.

2 Add a Sequence activity and a OnWorkflowItemChanged activity:

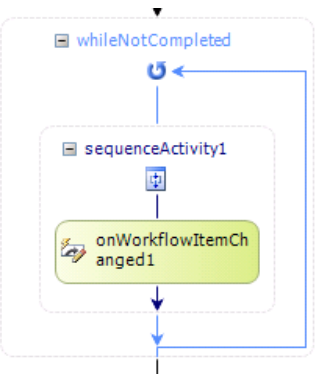
A) Drag and drop a sequence activity inside the While activity.

> Note: a While activity can only have one child activity. Since we need more than one, we'll use the Sequence activity to be a container for all the rest of the needed activities.

B) Inside the sequence activity, drag and drop a OnWorkflowItemChanged activity.

C) Another red exclamation point will show on the OnWorkflowItemChanged activity. Right click and choose properties. Click the crop down on the CorrelationToken property, and select "workflowToken".

> Note: correlation tokens are described in greater detail in chapter 9.



3 Add an If-Else activity that will check the status column:

A) Below the OnWorkflowItemChanged activity, drop an If-Else activity.

B) Right click the If-Else activity and choose Add Branch. Do this so there is a total of 4 branches.

C) Change the names of each branch by right clicking, choosing properties, and editing the Name property. Change the names to:

- ifStatusHasNotChanged
- ifPendingCompletion
- ifOnHold
- ifCompleted

D) On the fourth branch, change the Condition property to be Code Condition. Now all four branches should show a red exclamation point.

The resulting if-else activity will have four branches configured similarly to Figure 8.16.

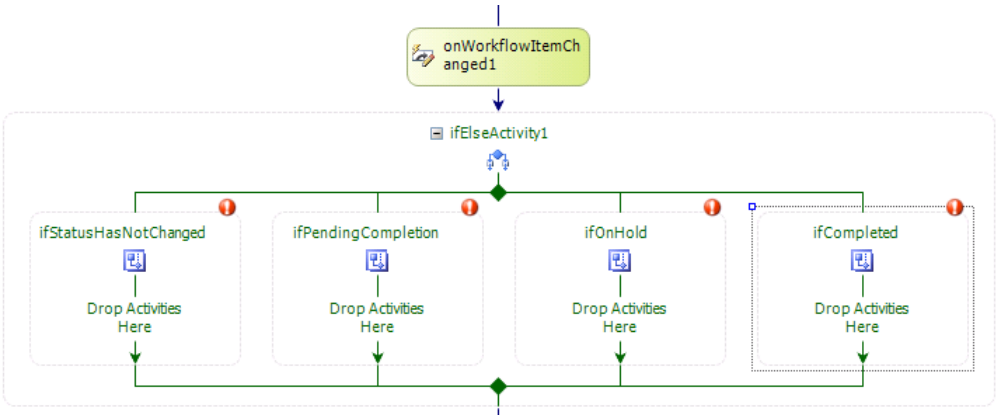


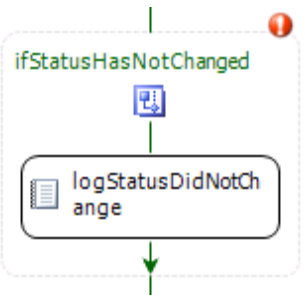
Figure 8.16 Each time the workflow item is updated, an If-Else activity will check the status column to see what the status is set to.

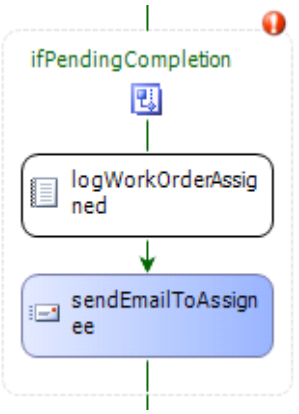
With the while loop and the if-else branches in place, the next step is to add activities into each if-else branch that will handle our business requirements. What we want to have happen is:

- **If the status doesn't change**: do nothing
- **If the status is set to Pending Completion**: email the maintenance worker informing them they have a new work order to complete.
- **If the status is set to On Hold**: email the student informing them their request is put on hold. Terminate the workflow.
- **If the status is set to Completed**: email the student informing them their request has been completed. Terminate the workflow.

With the branches now configured properly, let's add some activities into the branches to send the email, and perform some logging. Follow these steps to configure the branches:

Add actions into the branches

	Action	Result
1	For the first branch, we simply want to log that the workflow item did not change. A) Drag and Drop a LogToHistoryList action from the toolbox inside branch one. B) Rename the action to be titled "logStatusDidNotChange". C) Under the logger's properties, enter a message in the HistoryDescription property. This is the text that's logged to the workflow's history log. Enter something such as "The work order was updated, but the status did not change".	

2	Configure the isPendingCompletion branch to log and send an email to the assignee: A) Drop a LogToHistoryList activity into branch two and change the name to be logWorkOrderAssigned and give a HistoryDescription of "Status changed to Pending Completion". B) Drop a SendEmail activity below the logWorkflowOrderAssigned activity and change the name of the SendEmail activity to be sendEmailToAssignee. C) Click the correlation token drop down on the SendEmail activity and choose "workflowToken".	
3	Configure the To, Subject, and Body of the email message to the assignee: A) Right click the sendEmailToAssingee activity and choose Generate Handlers. B) Enter Listing 8.1 into the newly created event handler.	The sendEmailToAssignee's To, Subject, and Body properties will be set to send the email to the assignee, informing them that a new work order has been assigned to them.

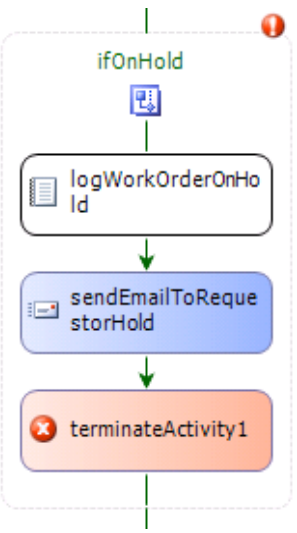
Listing 8.1 sendEmailToAssignee_MethodInvoking event handler

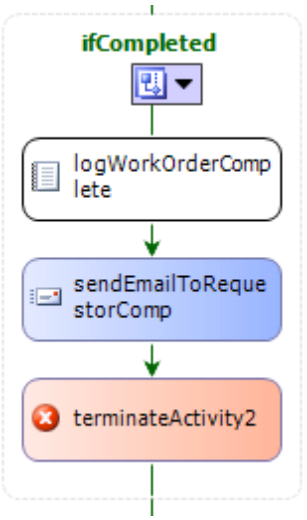
```
SPListItem wfItem = onWorkflowActivated1.WorkflowProperties.Item;
SPFieldUser assignedTo = (SPFieldUser)wfItem.Fields["Assigned To"];      1

SPFieldUserValue user = (SPFieldUserValue)assignedTo.GetFieldValue(
    wfItem["Assigned To"].ToString());                                   2
string assigneeEmail = user.User.Email;

sendEmailToAssignee.To = assigneeEmail;                                  3
sendEmailToAssignee.Subject = "New work order has been created.";
sendEmailToAssignee.Body = "Work order number " +
    onWorkflowActivated1.WorkflowProperties.Item.ID +
    " has just been created and assigned to you.";
```

1) Get AssignedTo column
2) Get AssignedTo column value
3) Set To, Subject, and Body properties

The first thing Listing 8.1 does is gets the AssignedTo column and casts it into a SPFieldUser object **#1**. Then, the value of the field is casted into a SPFieldUserValue object **#2**. From that object you can get at the SPUser object that is the person stored in the AssignedTo column. We need this object to get at that user's email address and assign it to the To property of the SendEmail activity **#3** (along with the subject and body).

	Action	Result

4	Configure the third branch to log, send an email, and terminate the workflow:
	
	A) Drop another LogToHistoryList activity and change the name to logWorkOrderOnHold and provide an appropriate HistoryDescription.
	
	B) Drop another SendEmail activity and change the correlation token to "workflowToken" as well as the name to "sendEmailToRequestorHold".
	
	C) Right click the SendEmail activity and choose Generate Handlers and set the To, Subject, and Body properties as follows:
	
	`sendEmailToRequestorHold.To = onWorkflowActivated1.WorkflowProperties.OriginatorEmail;`
	
	`sendEmailToRequestorHold.Subject = "Work order put on hold";`
	
	`sendEmailToRequestorHold.Body = "Work order number " + onWorkflowActivated1.WorkflowProperties.Item.ID + " has been put on hold.";`
	
	D) Drop the Terminate activity below the SendEmail activity.
	
	E) Right click the Terminate activity, and change the Error property to be a friendly message like "The work order was put on hold". This message will display on the workflow history page.
		

5	Configure the fourth branch to log, send an email, and terminate the workflow as well:
	
	A) Drop another LogToHistoryList activity and change the name to logWorkOrderComplete as well as provide an appropriate HistoryDescription.
	
	B) Drop another SendEmail activity and change the correlation token to "workflowToken" as well as the name to "sendEmailToRequestorComp".
	
	C) Right click on the SendEmail activity and choose Generate Handlers and set the To, Subject, and Body properties as follows:
	
	`sendEmailToRequestorComp.To = onWorkflowActivated1.WorkflowProperties.OriginatorEmail;`
	
	`sendEmailToRequestorComp.Subject = "Work order has been completed";`
	
	`sendEmailToRequestorComp.Body = "Work order number " + onWorkflowActivated1.WorkflowProperties.Item.ID + " has been completed.";`
	
	D) Drop the Terminate activity below the SendEmail activity.
	
	E) Right click the Terminate activity, and change the Error property to be a friendly message like "The work order has been completed".
		

With all our activities in place, we're almost ready to build and test our workflow. Before we can do that, we first need to resolve the red exclamation points next to each if-branch. The exclamation point is telling us that the branch doesn't have a method telling the branch if the condition is met or not. Follow these steps to add logic into the branches so they can determine if the condition is satisfied:

Add Code Condition Methods for each If-Else Branch

	Action	Result
1	All the conditions depend on looking into the work order's Status column. Configure the Before and After properties of the OnWorkflowItemChanged activity so we can gain access into the Status column's data: A) Go to the properties of the OnWorkflowItemChanged activity. B) Double click on the little database icon next to the AfterProperties property.	Two new fields will be added to the workflow's template, one for After properties and one for Before properties. These fields will contain the before and after result of the Status column.

	C) in the popup, select the *Bind to a New Member* tab and click Create Field, and then click Ok. D) Repeat steps A-C for the Before Properties property.	
2	Create a condition method for the first If branch: A) Change the Condition property of the first branch to be code condition. B) Click the plus sign next to the Condition property and type a name of a method, such as "HasStatusChanged". C) Inside the newly generated HasStatusChanged method, enter the code found in Listing 8.2.	The first branch will now be executing a method that will determine the outcome of the condition.

Listing 8.2 HasStatusChanged Method

```
string aStatus = onWorkflowItemChanged1_AfterProperties1["Status"].      1
    ToString();

if (onWorkflowItemChanged1_BeforeProperties1["Status"] == null)          2
{
    e.Result = false;
    return;
}

string bStatus = onWorkflowItemChanged1_BeforeProperties1["Status"].
    ToString();

if (bStatus == aStatus)                                                  3
    e.Result = true;
else
    e.Result = false;
```

1) Retrieve status after item changed
2) Return if the before status is null
3) True if before and after are equal

This code first retrieves the status out of the AfterProperties property **#1**. This will contain the value of the status column after the change is committed to the database. A check on the BeforeProperties for null is needed **#2** because if the work item is new, it won't have a before value. Lastly, if the before and after status are equal to each other, return true because the Status column had not been updated **#3**.

	Action	**Result**
3	Create a condition method for the second If branch: A) Change the Condition property of the second branch to be code condition. B) Click the plus sign next to the Condition property and type a name of a	Same as step 2 but for the second branch.

	method, such as "IsStatusPendingCompletion".
	
	C) Inside the newly generated IsStatusPendingCompletion method, enter the following code:
	
	`string astatus = onWorkflowItemChanged1_AfterProperties1 ["Status"].ToString();`
	
	`if (astatus == "Pending Completion") e.Result = true; else e.Result = false;`	

4	Create a condition method for the third If branch:
	
	A) Change the Condition property of the third branch to be code condition.
	
	B) Click the plus sign next to the Condition property and type a name of a method, such as "IsStatusOnHold".
	
	C) Inside the newly generated IsStatusOnHold method, enter the following code:
	
	`string astatus = onWorkflowItemChanged1_AfterProperties1 ["Status"].ToString();`
	
	`if (astatus == "On Hold") e.Result = true; else e.Result = false;`
		Same as step 3 but for the third branch.

5	Create a condition method for the fourth If branch:
	
	A) Change the Condition property of the fourth branch to be code condition.
	
	B) Click the plus sign next to the Condition property and type a name of a method, such as "IsStatusCompleted".
	
	C) Inside the newly generated IsStatusCompleted method, enter the following code:
	
	`string astatus = onWorkflowItemChanged1_AfterProperties1 ["Status"].ToString();`
	
	`if (astatus == "Completed") e.Result = true; else e.Result = false;`
		Same as step 3 but for the fourth branch.

With these last set of steps in place, we're now ready to deploy and test our workflow. To deploy, right click the solution and click Deploy Solution. This will build the project and generate the solution package. It will then add and deploy that package into SharePoint.

Navigate to your Work Orders list and create a new work order and assign the work order to a user. This action should kick off the workflow and the workflow should find itself in the "In Progress" state. At the point the workflow is sitting on the OnWorkflowItemChanged activity and is waiting for someone to edit the work order. Go ahead and edit the work order by changing Status to be Pending Completion. This will cause the OnWorkflowItemChanged activity to fire, and the corresponding if-branch will execute, sending an email to the person you specified in the Assigned To column.

The workflow is still "In Progress" and is again waiting on the item to be changed. Edit the work order again; this time set the status to be Completed. This action will send an email to the requestor (which in this case is you) and the workflow will terminate. If you click on the workflow status column showing "Completed", you'll be taken to the workflow history list that should look similar to Figure 8.17:

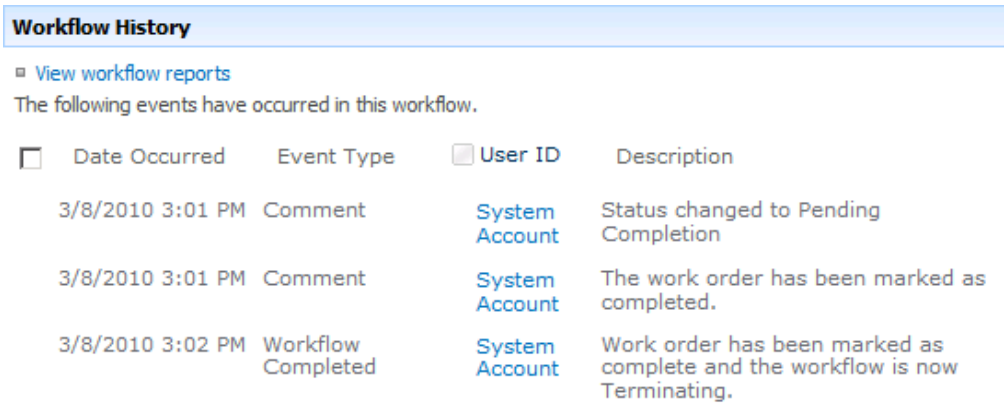


Figure 8.17 When all said and done, your testing would result in a workflow log showing the path the workflow took.

8.3 Building a State Machine Workflow

The sequential workflow we built in the previous section was actually quite neat. It helped us manage a maintenance request process for students on a college campus. That workflow starts to break down quite quickly if the business requirements get more complicated. For example, notice how in Figure 8.18 there's an extra step between the completion of the work order, and the workflow terminating. What if a business rule requires that the student confirm that the work was done to their satisfaction? If yes, the workflow will terminate. If no, the workflow needs to go back to the Pending Assignment status so the manager can send another worker out to finish the job correctly.

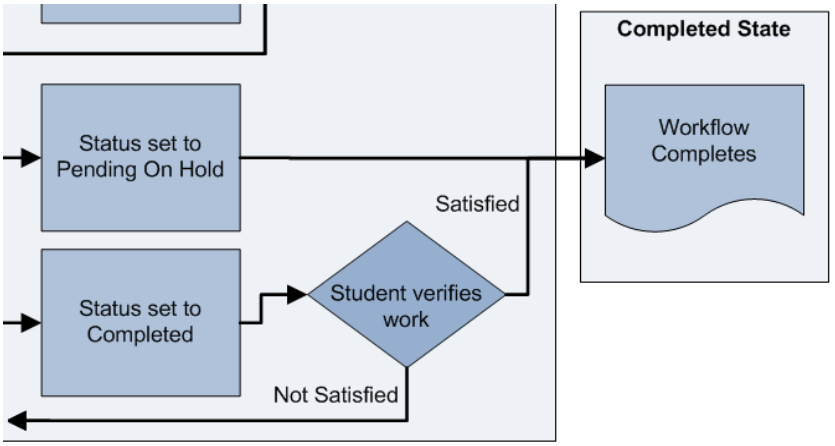


Figure 8.18 If a requirement was added that required students to verify the work order, the sequential workflow created in the previous section would get messy quick.

This requirement would be very difficult to meet with a sequential workflow. Rather, a state machine workflow would be the much preferred choice. I say "preferred" because you could hack your way to getting it to work in a sequential workflow, such as using more While activities or by adding a lot of complex logic that kept track of it all. Taking that path will create maintainability nightmares.

Notice Figure 8.19 showing how the diagram would look if we were to rewrite our sequential workflow into a State Machine. If you remember, the sequential workflow only had 2 states, "In Progress" and "Completed" with a bunch of logic crammed in the "In Progress" state. Figure 8.19 shows 6 states rather than 2, and notice how the workflow can more easily go "back in time" to a previous state. It's not sequential at all. Also, if you remember from the previous example, when the work order was put "On Hold" the workflow terminated. Now with a state machine the workflow is able to reside in that state indefinitely or until the manager decides to put that work order back on the docket.

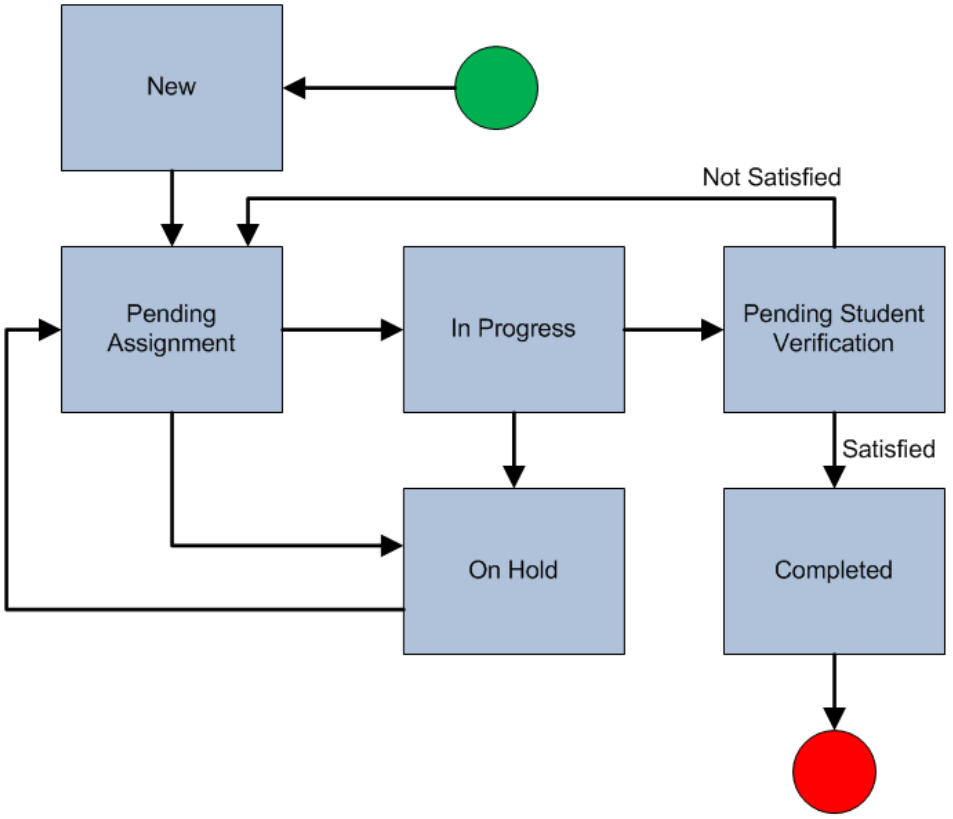


Figure 8.19 A better approach for the validation requirement would be to use a State machine workflow. It would be very easy to add a new state for student verification, even a year after the workflow was first created.

State Machines require a bit more clicking to configure, despite their neater diagrams. On a nice note you'll find that State Machines require a lot less code because the logic is modeled more, rather and coded.

Since the configuration of all the activities is nearly identical in both types of workflows, this section won't rehash the same steps to configure the activities. Rather, we'll take the high road and simply demonstrate how to setup and configure the states in a state machine. If you're curious how to configure a certain activity, go back to the previous section to see the full steps.

To get started, create a new project using the State Machine Workflow project template. When your workflow first loads, you'll see a template that looks very different than the sequential workflow's template. You'll get a single box (a state) with a StateInitialization activity inside it. Rename this default state to "New". When you do this, you'll notice a red exclamation point near the text "State Machine Workflow". Right click that text and choose properties. Change the InitialStateName property to New. This tells the state machine what state should be the state of the workflow when it first starts.

After you've configured the New state, add five more states onto the template by dragging and dropping the "State" activity from the toolbox. Use the following as state names:

- PendingAssignment
- InProgress
- PendingValidation
- Completed
- OnHold

Within each state, drop an EventDriven activity. For the Completed state, drop a StateInitialization activity. Each state needs either an EventDriven activity or a StateInitialization activity to work. All the states besides the Completed state need to wait for an action to occur before they execute. This action is the modification of the workflow item and is handled through the EventDriven activity. The Completed state doesn't need an event to occur before it executes, because it will simply terminate the workflow.

After you get all the EventDriven (and StateInitialization) activities inside the states, rename them to something a bit more descriptive such as:

- WaitForAssignment (PendingAssignment state)
- WaitForCompletion (PendingCompletion state)
- WaitForValidation (PendingValidation state)
- WaitForResume (OnHold state)
- CompleteWorkflow (Completed state)

With our states and EventDriven activities in place, it's now time to tell the workflow how the different states relate to one another. In essence, make the state machine look like Figure 8.19. The state machine template has a nice feature that allows you to connect the various states together with arrows. This is very similar to how a Visio diagram works, if you're familiar with Visio at all. Just like Figure 8.19, you "draw" the arrows on the template between the states. The workflow knows what state comes next based on the direction of the arrows.

To start connecting your states, hover you mouse over the New state and look for the four blue dots on the edges of the state box. Click and hold one of those blue dots and drag your cursor over to the PendingAssignment state. Hover over the PendingAssignment state's own blue dots and release the mouse. This will draw an arrow between the New state and the PendingAssignment state. Make sure the arrow is going in the right direction!

Repeat this step of drawing arrows until all the states are connected. Your template should look similar to Figure 8.19 and more specifically, Figure 8.20 when you're done.

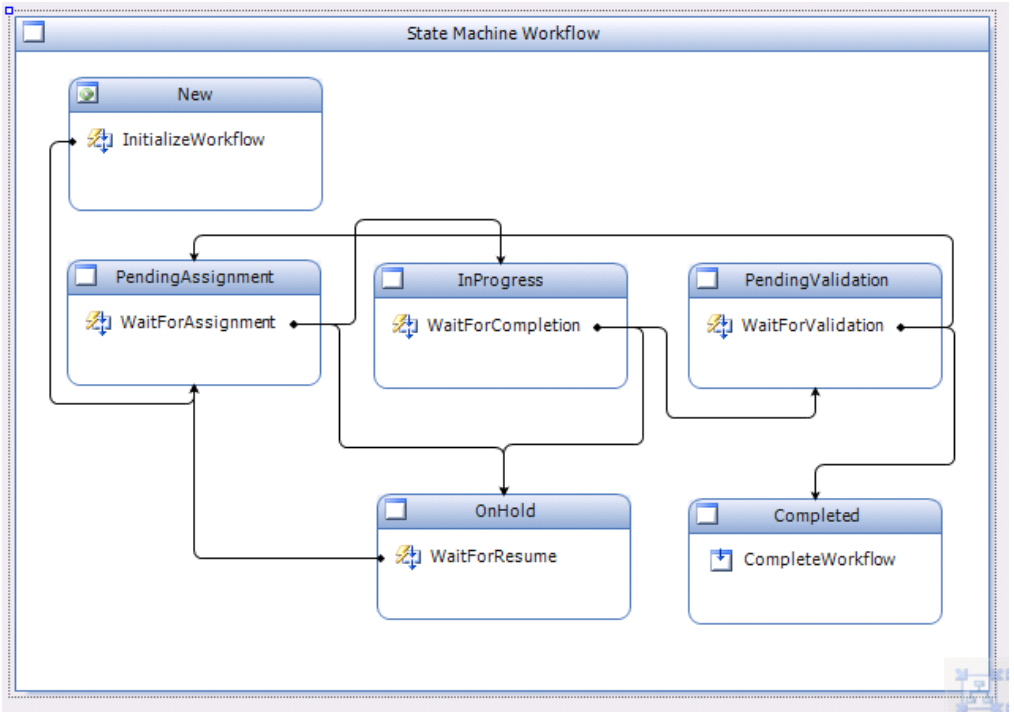


Figure 8.20 Similar to Figure 8.19, this figure shows the state machine as it looks in Visual Studio.

At this point you may be thinking, "this diagram is cute, but where's my logic going to go?". All the workflow activities live inside a state. For example, if you double click on the New state, you'll be taken to a template for that state. You can now drag and drop activities into that state. Notice how the New state has two activities in it (Figure 8.21). The first activity, OnWorkflowActivated, is the default activity for SharePoint workflows and needs to be the first activity that fires. The next activity is a SetState activity that essentially just tells the workflow which state to move to next. You can see that the New state simply initiates the workflow and moves the workflow to the PendingAssignment state.

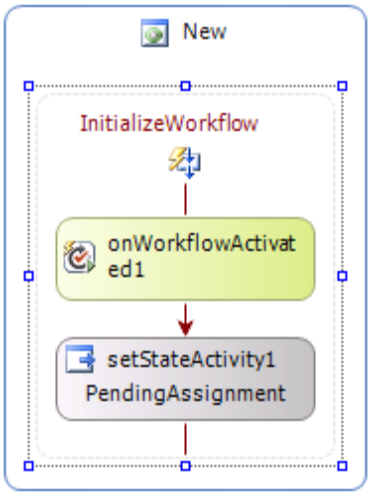


Figure 8.21 The New state does two things, initializes the workflow and moves the workflow into the PendingAssignment state.

So to build out of Work Order example, all we need to do is add our activities into the various states. Take the PendingAssignment state for example (Figure 8.22). Since the root activity is an EventDriven activity, the first activity must be an activity that waits for an event to fire. In our case we're using the OnWorkflowItemChanged activity again. When the workflow is in this state, it will sit and wait for the work order to be updated. When it is updated, there's a series of three if statements that determine what to do. The first checks if the AssignedTo column is empty. If it is, it simply repeats the PendingAssignment state. Otherwise it looks at the Status column. The second branch checks if the status is set to InProgress. If it is, the workflow moves to the InProgress state and emails the assignee. Lastly, if the status is set to On Hold, the workflow moves into the OnHold State.

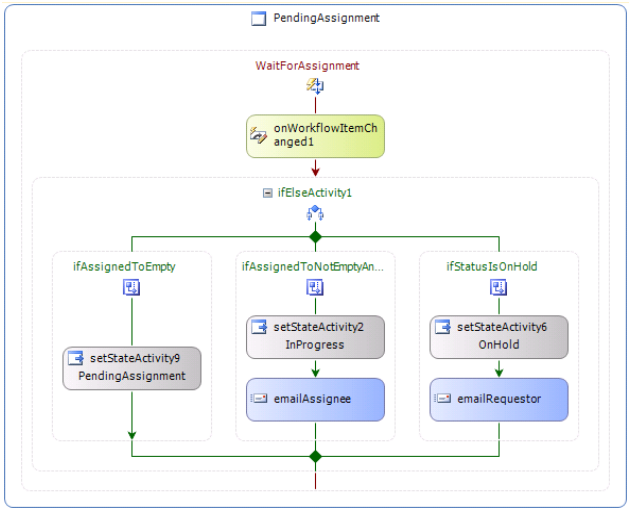


Figure 8.22 Once in the PendingAssignmet state, the workflow sits and waits for a person to be assigned to the work order and the status to be updated.

The InProgress state (Figure 8.23) behaves in a very similar way to the PendingAssignment state. The only big difference is the logic in the if branches checks for different things. The InProgress state represents when the workflow is waiting on the maintenance worker to finish the work order. When he or she finishes the work order, the individual updates the status column to be Completed. This causes the OnWorkflowItemChanged event to fire again, and it is handled by the second branch, which forwards to workflow onto the PendingValidation state. Otherwise the state repeats itself or goes into a hold.

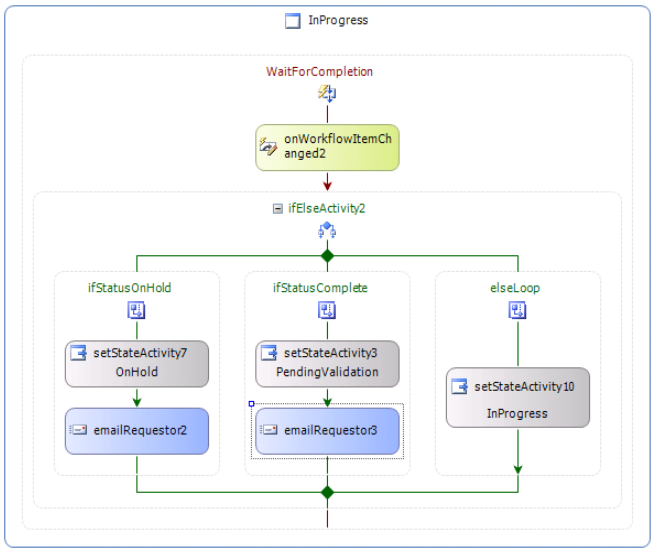


Figure 8.23 When the status column is changed to Pending Completion, the workflow state moves to In Progress where it waits for the worker to complete the work item. If the work order is set to Completed, the workflow moves into the PendingValidation state. Otherwise it either repeats itself or goes into the OnHold state.

When the work order is completed, the state goes into PendingValidation. This is where the requestor is emailed, asking them to validate that the work was completed to their satisfaction. There are several ways you could accomplish this. One option would be to imbed a link in the email. When the user clicks that link, they go to a page that reads a unique ID out of the query string and programmatically updates the correct work order. You could use a yes/no column in the work order to track if the student has validated the work. The PendingValidation state would simply sit and wait for that column to be updated. This is essentially what is depicted in Figure 8.24, the PendingValidation state. If the column is set to Yes, the workflow moves to the Completed state. If set to no, it moves all the way back to the PendingAssignment state.

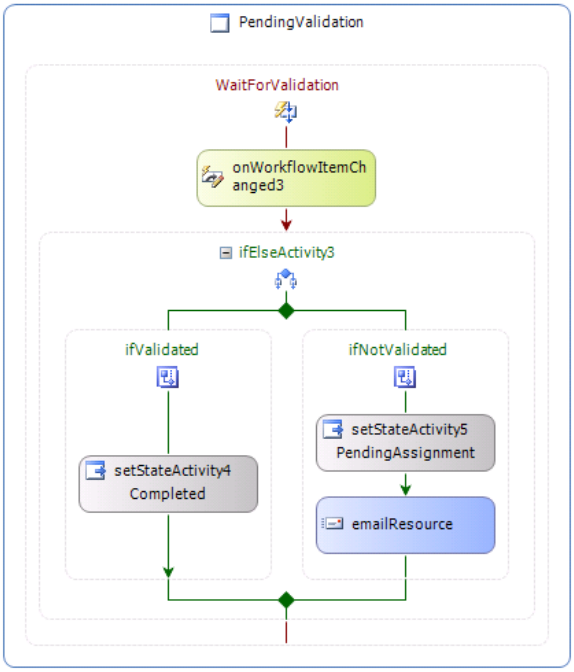


Figure 8.24 While in the PendingValidation state, the workflow waits for the student to submit their validation that the work has been completed. If so, the workflow moves to the Completed state.

Lastly is the Completed state (Figure 8.25). Not much interesting going on here, except that somewhere in the workflow you need to terminate the workflow. This essentially is the purpose of the Completed state. We could have put the Terminate activity in the second branch in the PendingValidation state, but then our workflow would terminate but still show "Pending Validation" in the workflow status column. Since "Completed" is a preferable status, its own state was desirable.

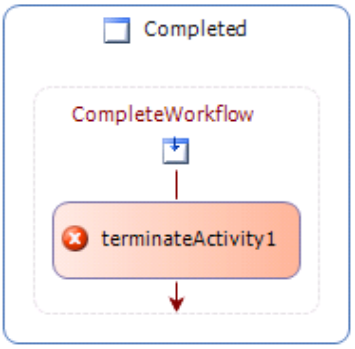


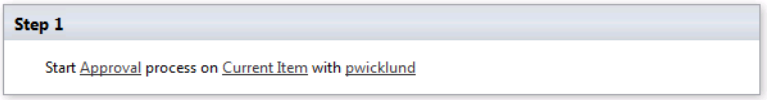
Figure 8.25 The Completed state simply terminates the workflow.

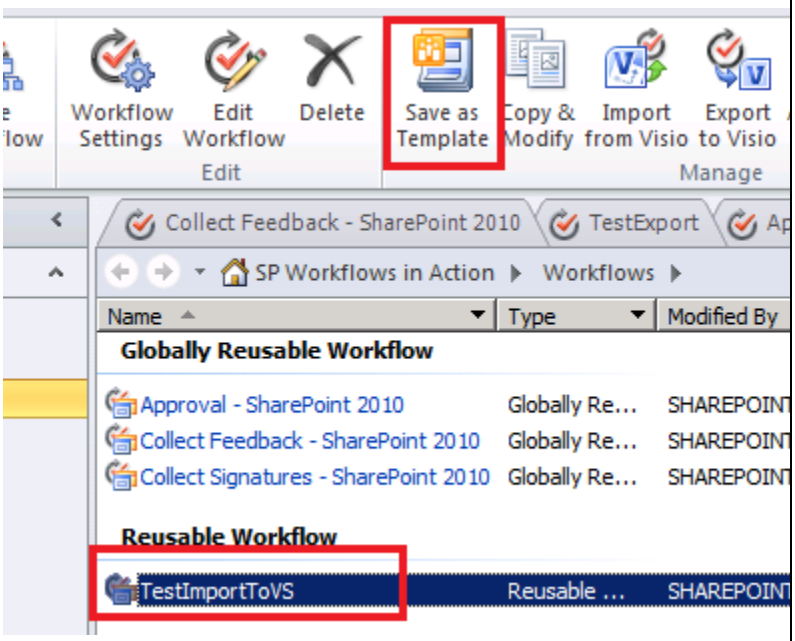
8.4 Importing a SPD Workflow into Visual Studio

One of the new exciting features with SharePoint 2010 and Visual Studio 2010 is the ability to import workflows from SharePoint Designer into Visual Studio. This will bring a lot of value to businesses because legacy workflows created in SharePoint Designer can easily be imported and expanded upon. Also, non-technical resources can prototype workflows in SharePoint Designer, and then hand off their work to someone more technical.

It's actually very easy to do this. The high level steps are to export the workflow template from SharePoint Designer. When you export the workflow, it saves it in the Site Assets library at the root of the site collection. The file has a WSP extension. You can download that file onto your computer and create a new Visual Studio project. When you select the type of project, choose Import Reusable Workflow, and browse to the WSP file you downloaded. It's that simple! Here are the more granular steps:

Export and Import a SharePoint Designer workflow into Visual Studio

	Action	Result
1	Build and export a SharePoint Designer workflow: A) Create a new Reusable workflow. Add some actions and conditions as you see fit. For example:  B) Save the workflow, then click publish. C) On the left navigation in SharePoint Designer, click Workflows and select the workflow you just created. Then in the ribbon click Save as Template:	By saving the workflow as a template, the workflow will be exported as a WSP file and sent to the Site Assets library at the root of the site collection.

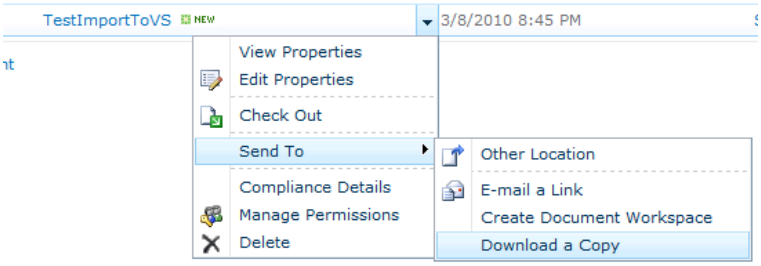


2 Browse to the Site Assets library and download the WSP file onto your desktop:

A) Browse to the root web in the site collection and click All Site Content.

B) Click the Site Assets document library and download the TestImportToVS STP file:

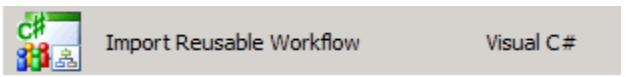
With the WSP file on your desktop, you can now import in into Visual Studio.



3 Create a new Visual Studio project using the new Import Reusable Workflow template:

A) Create a new project and use the Import Reusable Workflow project template:

The result will be a new project with all the activities imported form SharePoint Designer. Even



B) Enter the URL of a Site Collection. Click next, and then browse out to the WSP file you downloaded earlier. Click Finish.	the InfoPath forms were imported (Figure 8.30).

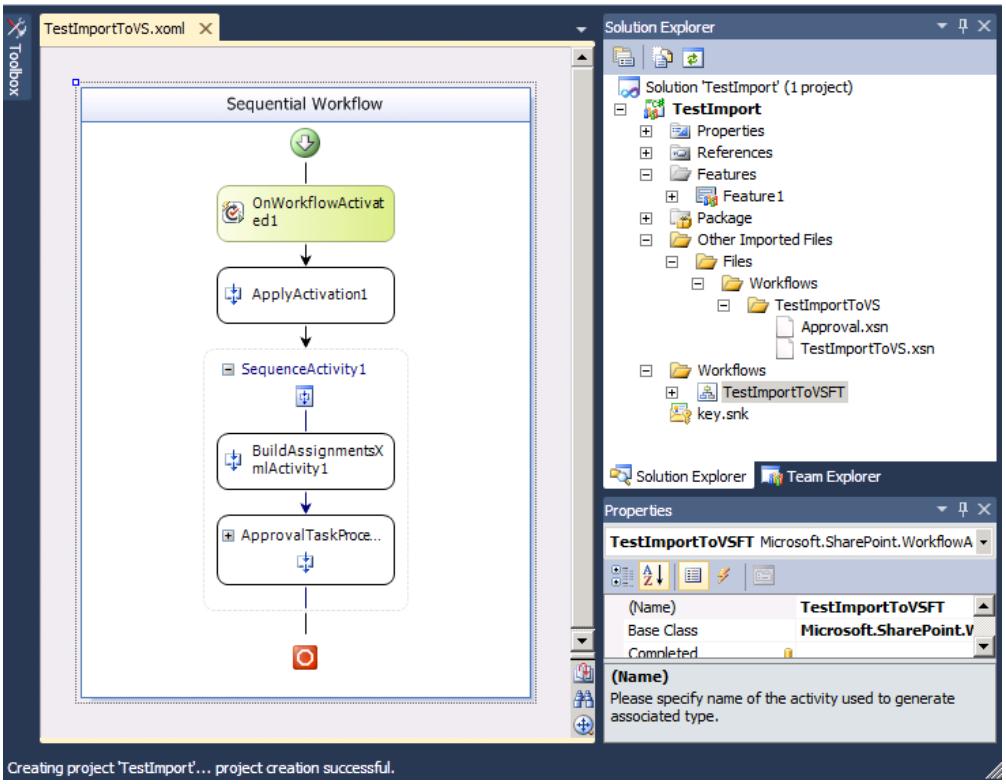


Figure 8.26 When importing from SharePoint Designer, you'll notice that all the actions are recreated as activities. Even all your custom InfoPath forms are carried over!

8.5 Summary

In many regards, this chapter is the most exciting chapter of the book. This is especially true for those developers who are new to SharePoint, because it can easily be seen that Visual Studio workflows for SharePoint are powerful, fast, and easy to create. Visual Studio 2010 has really made the life of a SharePoint developer so much easier from previous versions. All the packaging and deployment struggles are now a thing of the past. One doesn't need to know so much "under the hood" information that they used to need to know.

This leaves the developer solely focused on the workflows they're trying to develop. A developer can build Sequential workflows and State Machine workflows for SharePoint. Sequential workflows are similar to SharePoint Designer workflows in that they always flow forward. You can use looping

techniques to overcome this a bit, but the limitation remains that these workflows act in sequence. A new step always comes after the previous step.

On the other hand, State Machines are what's truly powerful. State Machines can go any direction you wish, even "back in time" you could say. How you model the state machine will dictate the flow of work. How you program the logic will dictate when the workflow should terminate. It's the uttermost in workflow flexibility. State Machines will accommodate the most complicated of workflow needs.

If you're ok with Sequential workflows, there is one great advantage that shouldn't be overlooked. That is the fact that you can import a SharePoint Designer workflow into Visual Studio. There is power, too, in this feature because your non-technical people can "pass the baton" to more technical people when the limits of SharePoint Designer are reached. This provides ROI because you may be able to more highly utilize your resources.

If you liked working in Visual Studio in this chapter, stay tuned! Of the 6 remaining chapters in this book, each has Visual Studio workflow techniques found within them. This chapter was only the start. Read on to learn about forms, tasks, custom activities, and events!

9

Forms in Visual Studio Workflows

Programmers are no exception when it comes to people who need to know how to build forms for workflows. InfoPath, for example, is a great tool for non-programmers, but programmers can enhance an InfoPath form to do more advanced tasks. A programmer can also programmatically retrieve data from an InfoPath form from within a workflow. These topics, and most of the topics discussed in chapter 7 all relate to forms *around* workflows, or forms that a workflow ran *on top of*. This chapter deals with forms that run *inside* a workflow. There are three main examples of a form that runs inside a workflow: association; initiation; and modification.

Initiation forms are used to gather information directly before the workflow starts. Association and Modification forms are two more avenues where users can enter information into the workflow. The difference is an association form is presented to the user when the workflow itself is added to the list, library, or site, and modification forms are presented to users during the workflow's execution.

Initiation forms come in handy when you need to gather data from the user before the workflow starts. The expense report example provides another real world case for this. When the workflow is initiated, it may not know who needs to approve the expense report. An initiation form could capture from the submitter who their manager is, and the workflow could then take that information and ensure that the proper individual receives the approval request.

For Association forms, the expense report example again provides some helpful justification. Rather than have the employee specify their manager every time they upload their expense report, you could use an association form to enter it only once when the custom workflow is first added to the library.

With a modification form you can actually change the behavior of the workflow after it has started; whereas the other two forms present themselves before the workflow starts. A good example of when this form is helpful is if the workflow assigns a task, and the assignee wants to reassign the task to someone else. You could use a workflow modification to accomplish this. The modification form comes into play via a link on the workflow status page when you enable a modification on that workflow. The modification starts after the user clicks the modification link on the status page, in which case they are redirected to the modification form where they may need to enter some information. After the form is submitted, the modification will execute. An important thing to remember is that modifications can only happen when your workflow says they can.

Whichever form you decide your workflow needs, you'll need to consider several implementation approaches. All three types of forms can be built in either InfoPath or ASP.NET. InfoPath's strength is its intuitive form designer surface, but its weakness in this scenario is that there are more hoops to jump through to deploy the form in the workflow. ASP.NET forms are straightforward to deploy, but less so to develop.

The steps to implement each form with each tool is the primary focus of this chapter. In addition, this chapter explains how to add .NET code to a InfoPath form and how to programmatically retrieving data. We'll go through a series of examples that add custom forms into workflows in both SharePoint Designer and Visual Studio. For Visual Studio workflows, both InfoPath forms and ASP.NET workflow forms are demonstrated.

9.1 Adding .NET Code to an InfoPath Form

In chapter 7, section 5, we setup two new pieces of data in an Expense Report form and mapped that data into columns on the form itself. One piece of data contained the expense type, and the other contained the total dollar amount of all the expenses in this report. Currently, the total dollar amount field isn't automatically calculated. To calculate this number, we need to add some code into our form that sums all the expenses together. What we want to happen is to fire some code any time someone changes the *Amount* field in any of the expenses in the repeating table. That code will then cycle through all the *Amount* fields in the table and calculate the total amount. This is rather generic example, but there are many instances when having .NET code in a form can come in very handy.

Before you can add this code to the form, you need to make sure you have *Visual Studio Tools for Applications* installed on your workstation. If you don't have it installed, you can re-run the Office 2010 setup, and choose to add new features. Under the InfoPath folder, select *Visual Studio Tools for Applications*, then "Run from My Computer" (Figure 9.1). Click continue, which will then install Visual Studio on your computer.

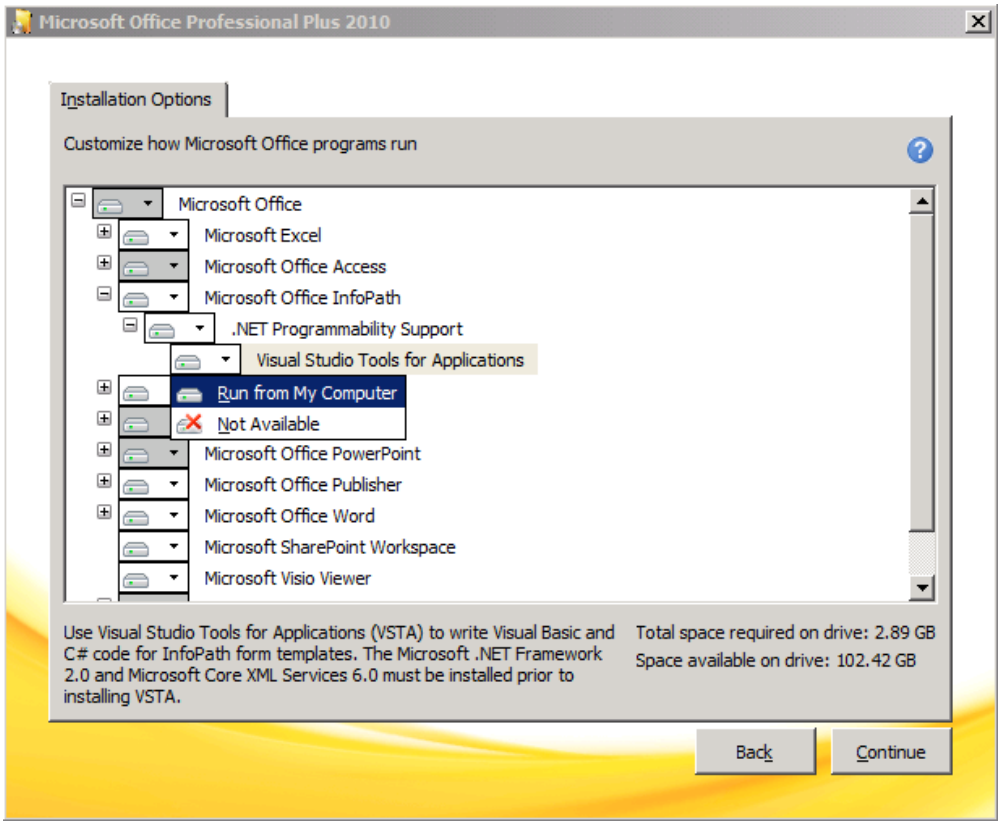


Figure 9.1 Visual Studio Tools for Applications is required to add .NET code behind to any InfoPath form. To install this application, run the Office setup, choose add Features, and select Run from My Computer under the Visual Studio Tools for Applications drop down.

With Visual Studio ready to go, it's time to add our code that will execute when the *Amount* field is changed. To get started, first click the *Amount* field. Then, on the ribbon under the *Developer* tab, click the *Changed Event* button. This will open up Visual Studio and a new project will be automatically generated that is linked to our form template (Figure 9.2). Inside the project you'll notice a single code file titled FormCode.cs. Within this code file there are two methods, *InternalStartup* and *Amount_Changed*. Since we selected to respond to the *Amount* field's Changed event, you'll see the *InternalStartup* method has an event already configured for us that will fire when changes occur. When a change occurs, it is handled by the *Amount_Changed* method.

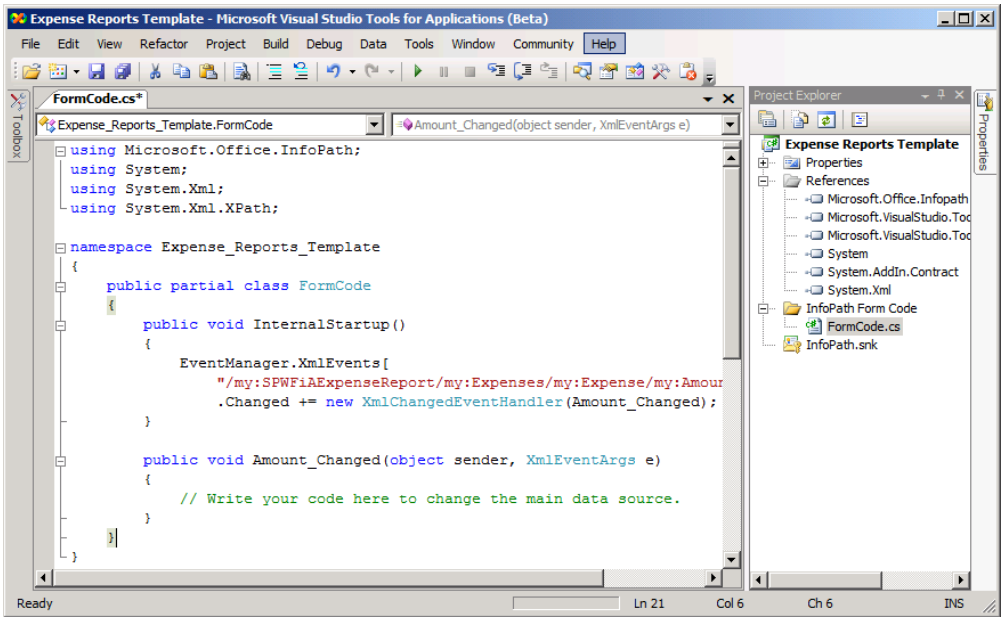


Figure 9.2 When code behind is first added to an InfoPath form, a new Visual Studio project is automatically created.

A couple of things are helpful to know when working in an InfoPath form's .NET code. First off, the InfoPath form and its code are linked together, so all you have to do is build the project, and then from within InfoPath, republish to get the code changes back into SharePoint. Another great thing is you have your expected debugging experience available to you. Simply hit F5 and the InfoPath client will appear with the form and if you have break points set you can walk through your code. Keep in mind that when debugging you are not going to have a web context, so if you are trying to run code that behaves differently based on who the running user is, it will be a bit trickier to unit test and debug.

Note: Setting your preferred programming language

By default all new Visual Studio projects created out of InfoPath are set to use Visual Basic as the programming language. This can easily be set to use C# instead. In the *Developer* tab in the ribbon, the very first button is *Language*. *Language* provides a form template code language drop down that you should update accordingly. If the drop down is disabled, you have already created a project and it can't be changed. If you are not worried about losing any code, click the *Remove Code* button to orphan that project, and after which you can change the language and start over by setting up a new project.

What we need to do now is add code into the *Amount_Changed* method that will crawl all the expenses in the form, and total up the amounts. Listing 9.1 has code that will do just that. Copy the listing and replace the "Write your code here..." comment in the *Amount_Changed* method.

Listing 9.1 Amount Changed Event Handler Code

```
XPathNavigator formData = this.CreateNavigator();

XPathNavigator expenseGroup = formData
   .SelectSingleNode("/my:SPWFiAExpenseReport/my:Expenses",              1
   NamespaceManager);

XPathNodeIterator expenses = expenseGroup.SelectDescendants(             2
   "Expense",
   expenseGroup.NamespaceURI,
   false);

double total = 0;
foreach (XPathNavigator expense in expenses)                             3
{
   string value = expense.SelectSingleNode(
         "my:Amount", NamespaceManager).Value;

   double amount = double.Parse(value);
   total += amount;
}

formData.SelectSingleNode(                                               4
   "/my:SPWFiAExpenseReport/my:TotalAmount", NamespaceManager)
   .SetValue(total.ToString());
```

1 Select just the Expenses XML data
2 Load Expenses into a collection
3 Iterate through each expense and sum expenses
4 Update TotalAmount with new total

The first thing that happens in this code is that an `XPathNavigator` is created to help us dig around in the XML data that contains all the values entered into the form. We then want to select just the *Expenses* data out from the form. To do this, we call the **#1** `SelectSingleNode` method on `formData`, which requires the XML path to the data element we want (along with the namespace). It is important to reiterate that when you create your forms in InfoPath, make sure to give the fields descriptive and meaningful names. The XML structure maps directly to the field names, as can be seen in the path that is specified in the `SelectSingleNode` method.

Now that we have our Expenses (repeating table's data), we need a collection object so we can iterate through each expense. To get this collection, we call **#2** `SelectDescendants` on the `expenseGroup` object and we pass in the element name in the XML that we want back which in this case is the `Expense` node. Next we **#3** iterate through each expense, and we pull out the amount of each expense. We then add that amount to a running total, and after we've gone through all the expenses that total amount is **#4** saved back into the `TotalAmount` field.

Note: TotalAmount field's data type

If you get an error when you publish , check the TotalAmount field's data type. It should be Double, not Text (the default).

Now that we have our code all neat and tidy, we should make sure it works as expected. Build the project to ensure there are no errors, and if there is none, hit F5 to launch the debugger. Unit test the

expense report form, and ensure that it is totaling up the expenses correctly and saving the total back into the *TotalAmount* field. Once you're certain it is behaving correctly, it is time to publish back into SharePoint.

> **Note: Sandboxes solutions must be enabled**
>
> When you publish, you may get an error saying sandboxed solutions are not enabled (Figure 9.3). To enable sandboxed solution, navigate to your server's Central Administration site. Click Application Management, and then click Services on Server. Ensure that the "Microsoft SharePoint Foundation Sandboxed Code Service" is active on all your web front ends.

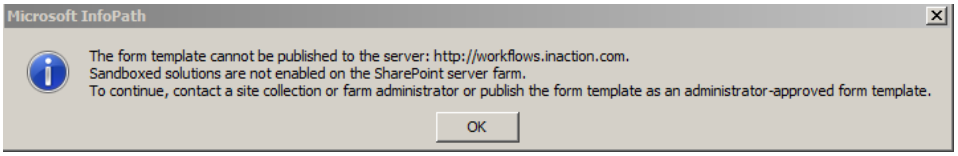


Figure 9.3 InfoPath forms with .NET code are packaged up as a sandboxed solution. By default, sandboxed solutions are not enabled on the farm, so you must enable them first.

> **Sandboxed Solutions versus Administrator Approved Templates**
>
> A step in the publishing wizard gives you the option to publish as an Administrator Approved template. If you select form library, or content type, and your form has .NET code in it, by default that code runs in what is called a "Sandboxed Solution". Sandboxed solutions are a new concept in SharePoint 2010. In SharePoint 2007, all browser enabled form templates needed to be uploaded into Form Server through SharePoint Central Administration. With 2010, a Site Collection administrator can upload their form templates right into their Site Collection's very own solution gallery. The difference between these two deployment options is mostly scope and security related. A form deployed into a sandbox obviously is only viewable within that site collection, whereas an administrator approved template (central admin) is usable across the entire farm. In addition, administrator approved templates can be set to have full trust, so the code in those forms can do things like cross domains (get/set data in separate SharePoint farms), and use things like SQL data, etc.

9.2 Programmatically Retrieving Form Data from within a Workflow

In the last section we saw how to programmatically read data out of a form from within the form's own code behind. What happens when an external system, like a workflow, needs to interact with that data? It is a common business requirement to have a workflow react to the submission of a form and then to have that workflow retrieve data stored in the form. After it has a handle on the data, it can do something with it, like ship it off to a separate line of business application or external process or maybe send some email notifications.

At a high level, to meet this business requirement you first need to create a .NET proxy class for the InfoPath form off the form's XML schema definition file. This class will be strongly typed, so it will be much easier to code against the form data because you will have intellisense into the form fields and their values. If you remember, this is unlike working in the code behind of the form where you had to

leverage XPath. Undoubtedly you will like this proxy approach, because the code will be much cleaner and easier to understand. You can use this proxy class to both read and write data in and out of the form. Note, this can't be done while the user is editing the form. The proxy class can be leverage before or after user interaction, and XPath must be used during, if necessary.

Therefore, to get started, let's build a new sequential workflow with Visual Studio 2010 called ExpenseReportWorkflow. This workflow will sit on top of the form, and will fire anytime a new expense report is entered into the system. We will add our proxy class into this new workflow project, and the workflow will have a generic code activity in it that will use the proxy to pull the form data out. Follow these steps to setup the sequential workflow project shell and the code activity:

Setting up a Sequential Workflow with a Code Activity

	Action	Result
1	Create a new Sequential Visual Studio workflow project with a name of ExpenseReportWorkflow. A) Enter the URL to your SharePoint site, and deploy as a farm solution and then click Next. B) Provide a name for the workflow as Expense Report Workflow, and use the List Workflow template and then click Next. C) Click Yes to automatically associate your new workflow with a SharePoint list. In the library or list drop down, select your expense reports form library to add the new workflow onto and then click Finish.	The project will be created with a workflow file called Workflow1.
2	In the workflow designer surface, add a Code Activity below the OnWorkflowActivated1 activity that was placed on the surface by default. A) After the Code activity has been placed on the surface, edit its properties and give it a name of RetrieveFormData. B) Right click the code activity and select Generate Handlers.	When done with these steps, your new project's workflow designer surface will look something like Figure 9.3:

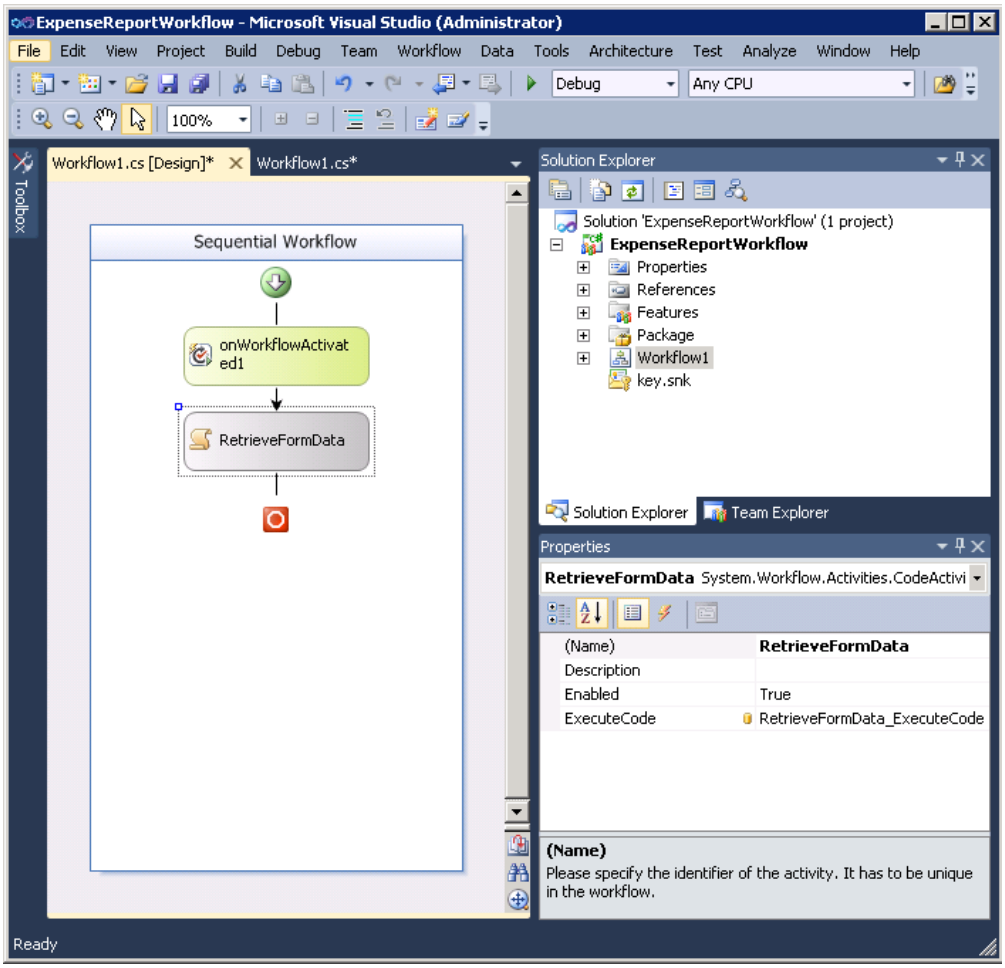


Figure 9.4 A new sequential workflow that will respond to new expense reports being submitted into the system.

Now that we have our workflow project started, we need to generate a proxy class from our InfoPath expense report template. To generate the proxy, we're going to run an XSD command from the Visual Studio Tools command prompt, and we'll point the command at our template's XML schema definition file. Every InfoPath form is actually a CAB file, that has several files all packaged up into one single file. Our schema definition file lives within this CAB. To save all these sub-files, click the *Export Source Files* button under the *Publish* tab in the *File* menu from within InfoPath. You'll have to browse to a place on your hard drive to save the files in a desired directory. Figure 9.4 shows the seven files that are exported.

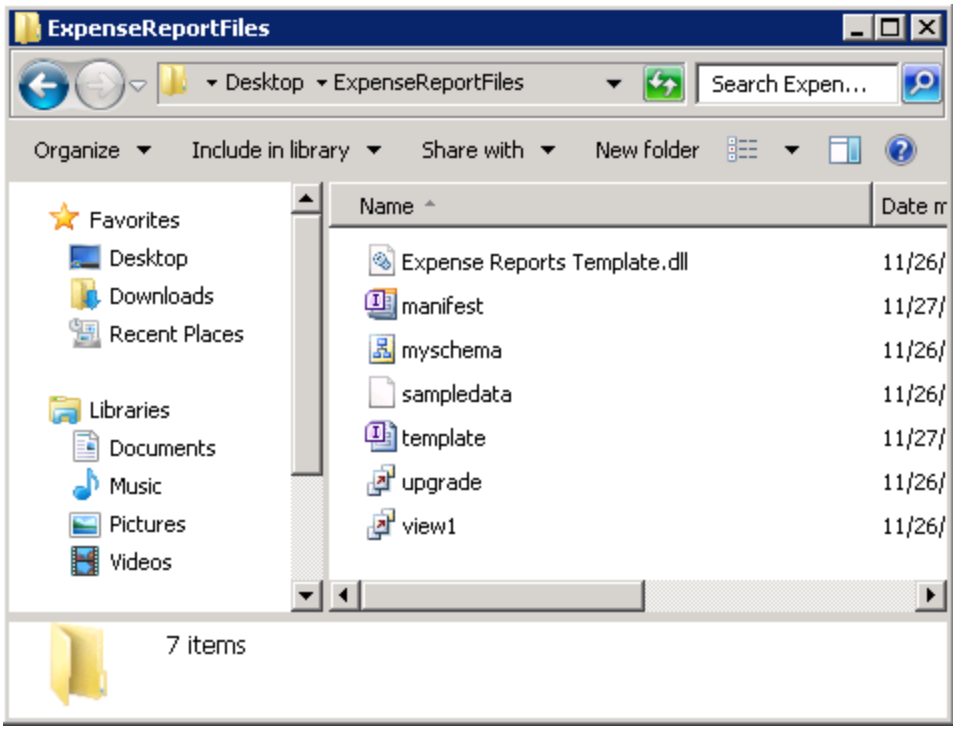


Figure 9.5 Every InfoPath form is actually a CAB that has several files contained within itself. You can export these files from the Publish tab in the File menu in the InfoPath client.

The file in interest is the myschema.xsd file, which contains our form's XML schema. This schema file will instruct the XSD command on what classes, properties, and fields to automatically provision for us when it generates and spits out our proxy class. There will be a class created for every group of fields in the InfoPath form. For example, our proxy file will contain three classes, one for the root, one called *Expenses* that has a collection of all the expenses, and another called *Expense*, that contains all the data in each expense. Follow these steps to generate this proxy class:

Generating a Proxy Class of an InfoPath Form

	Action	Result
1	With Visual Studio command prompt, run the following command against the exported files: `xsd myschema.xsd /c /l:cs` Note: If you get an error that is similar to "Error: The process cannot access the file 'c:\export\myschema.xsd' because it is being used by another process.", you have to close the InfoPath form first. When the form is open, it has	This command will spit out a new file named myschema.cs in the same directory as the exported files. This code file contains the proxy class we can use to code against the expense report data.

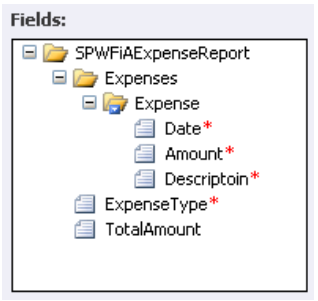
a lock on the schema file. Close InfoPath and rerun the command.

2 Add the myschema.cs file into the Visual Studio project.

A) By right clicking the project name, and choosing *Add*, *Existing Item*. Browse out and find myschema.cs, and click Add.

B) Rename the code file to something more compelling than myschema.cs, like SPWFiAExpenseReport.cs. I like to keep the file name the same as the root class name.

Note: Renaming the myschema.cs file name to SPWFiAExpenseReport.cs has no implications outside cleanliness and my obsessive compulsive tendencies. If you want rename the class name itself, please be aware that it's a two step process. The class name that is generated comes out of the root XML element name in the InfoPath form. You must change this field name first, and then rerun the XSD command to generate a new proxy class. The above figure contains the fields in my example expense report form. Notice how the root data element is named SPWFiAExpenseReport. Whatever this element is named is what the name of the proxy class will be. First change it in the form, and then rerun the XSD to update the class name to your liking.



The proxy class will be added to the Visual Studio project, and renamed SPWFiAExpenseReport.cs (from myschema.cs).

3 Add a namespace to SPWFiAExpenseReport.cs.

A) Open the code file. Above the class name, and under the using statement at the top, we need to add a namespace. Directly below the using System.Xml.Serialization statement at the top of the file add "namespace ExpenseReportWorkflow.Workflow1 {"

B) At the very last line of the file, add a close bracket, "}". Now our proxy class is contained in the same namespace that our

The SPWFiAExpenseReport class by default doesn't have a namespace, and with a newly added namespace, we'll be able to reference this class from other classes in the project.

workflow code is running in	

INSTANTIATING THE PROXY CLASS, AND CODING AGAINST THE DATA

Now that we have our proxy class added our Visual Studio project, let's instantiate that class and start working with the form's data. Listing 9.2 contains the code that we need to update the code activity's method with. Right click on the *RetrieveFormData* code activity and choose *View Code*. Replace the auto-generated *RetrieveFormData_ExecuteCode* method with the method in Listing 9.2.

Listing 9.2 Retrieving form data through a strongly typed proxy class

```
private void RetrieveFormData_ExecuteCode(object sender, EventArgs e)
{
    SPFile file = onWorkflowActivated1.WorkflowProperties.Item.File;     1

    XmlTextReader reader = new XmlTextReader(file.OpenBinaryStream());   2

    XmlSerializer serializer = new XmlSerializer(
        typeof(SPWFiAExpenseReport));

    SPWFiAExpenseReport fields = (SPWFiAExpenseReport)serializer         3
        .Deserialize(reader);

    double total = 0;
    foreach (Expense expense in fields.Expenses)                         4
    {
        total += expense.Amount;
    }

    file.Item["Total Again"] = total;
    file.Item.Update();
}
```

1 Load InfoPath form through SPFile object
2 Load file's data into XML reader
3 Deserialize XML into proxy instance
4 Sum each expense (much cleaner!)

Listing 9.2 starts by instantiating an `SPFile` object **#1**. The `OnWorkflowActivated` property has an instance of the form's list item (`SPListItem`), and a list item in SharePoint can have an attachment. This attachment is required in Form Libraries. The File property for `SPListItem` contains the `SPFile` object (the attachment), which also happens to be our InfoPath XML form data file.

We then **#2** open a binary stream on that `SPFile` object and load the stream into an `XmlTextReader`. Afterwards, our proxy class gets instantiated **#3** and is assigned to a deserialized version of that stream cast into our proxy class type. With this, our proxy class object will now contain all the properties and collections of data that is present within our InfoPath form. That data is mapped to the XML schema that is mapped to the fields in the form itself.

Just like in our code behind example, we're going to **#4** iterate through all the expenses in the expense report, and total up all the expenses amount field values. Also similar, we're going to take that value and **#5** assign it to a column in the form's list item, but this time the column is called "Total Again". This is a second plausible way to "Map columns", but is usually not the preferred since the other approach doesn't need any custom code. It serves to help demonstrate the proxy concept nicely.

The last thing to do is build, deploy, and unit test our workflow. Build the project and ensure there are no errors. If not, deploy the project by right clicking the project name, and selecting *Deploy*. This will package up our workflow and deploy it onto the list we specified in the begging of this section.

Now either create a new expense report, or edit an existing. Before we start our custom workflow on an expense report, make sure to create a new column titled *Total Again* on the form library, because our code will write to this column. After this column as been added, go ahead and start the *Expense Report Workflow* custom workflow we just deployed (Figure 9.5).

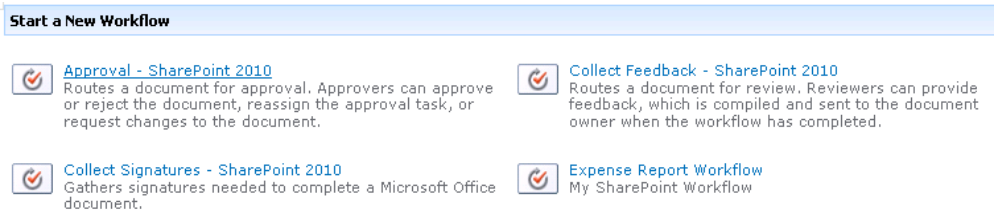


Figure 9.6 Out Expense Report Workflow was automatically packaged and published to our form library when we deployed form within Visual Studio.

The workflow should complete very quickly. Notice a new column called *Expense Report Workflow* has been added, as well as the *Total Again* column has also been populated correctly (Figure 9.6). The code activity successfully instantiated the proxy class to the form data, and crawled and totaled all the expense amounts in the form.

Figure 9.7 After you run the custom workflow, the workflow status should read Completed, as well as the Total Again column should show the total of all the expense reports.

9.3 InfoPath Forms in Visual Studio Workflows

As was mentioned, you can build forms for a Visual Studio workflow in either ASP.NET or InfoPath. This section will cover how to add InfoPath forms into your Visual Studio workflows. Specifically, this section will cover association, initiation, and modification forms. The example we're going to work with involves a workflow that manages the support needs of a Site Collection. If you have thousands of users working in a Site Collection, it may be helpful to have a workflow that can manage their help requests, such as if they need help adding or configuring a web part, or setting up a document library. The workflow will manage the help requests through each level of support. The first stage may be the IT help desk, and if they can't resolve the issue, the ticket can be elevated to the Site Collection Administrator. The third level then could be the technical architect.

9.3.1 Building a Custom Association Form

The first InfoPath form we want to add is an association form that captures who are the support personnel are for each support tier. When this support workflow is added onto a list, this form will prompt the user to specify SharePoint groups containing people that are responsible for supporting each level.

The goal for this project in this section is simply to wire up our association form, and deploy our workflow. In later sections we'll cover how to use initiation and modification forms. For the present, let's start simple and focus on this association form. Figure 9.7 shows what this form looks like.

> **Pre-work for the example**
>
> All the examples built through the end of this chapter are built on top of a Issues list. Create a new issues list named Support Requests and you'll be good to go.

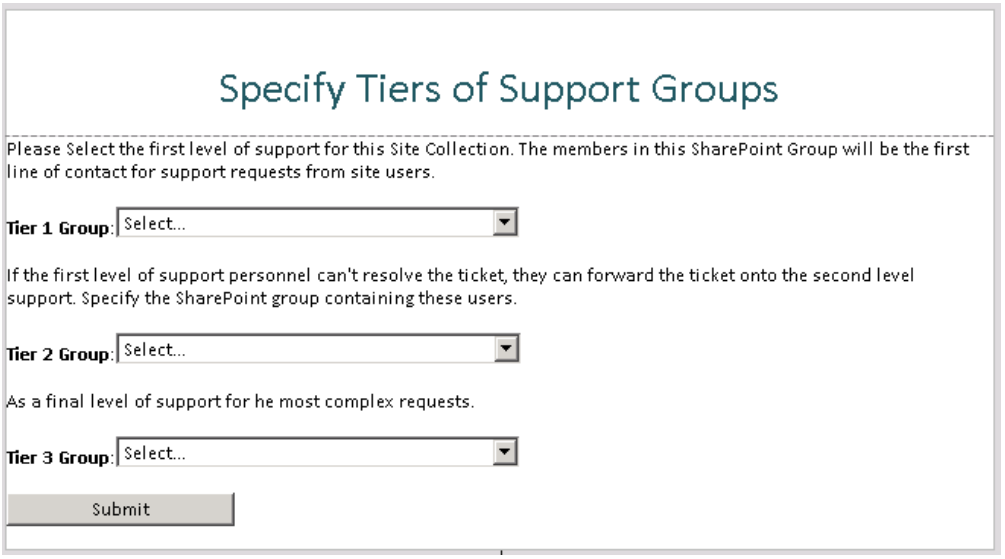


Figure 9.8 This sample association form built in InfoPath will capture from the user the SharePoint groups of users that will make up the 3 tiers of support in the system.

One thing that is important to note is how to configure the submit button on this form. The drop down boxes and verbiage is nothing complicated, but the submit button involves a few steps to get working correctly. Follow these steps to configure a submit button for any InfoPath form used in a workflow:

Configuring a Submit button to submit to the SharePoint Host

	Action	Result
1	Add the button on the designer surface: A) Right click the button, and choose Button Properties.	A new button will show on the form, allowing users to submit the form.

B) From the Action dropdown list on the *General* tab, select *Submit*.

C) Click *Submit Options*.

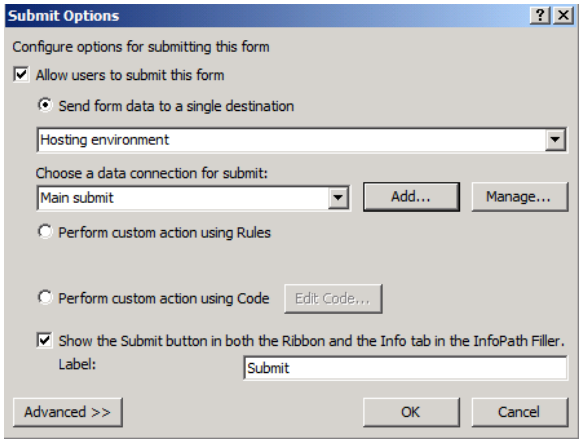
D) Check *Allow users to submit this form*.

E) Under the *Send form data to a single destination* drop down, choose Hosting Environment.

2 Add a new data connection for the button to use:

A) On the Submit Options popup still, below the *Send form data to a single destination* drop down, click the Add button to add a new data connection.

B) A new dialog will appear defaulted to Main submit. Leave the defaults and click Finish and then click Ok twice.



With the button now configured with a data connection, the button will now submit the form data appropriately.

Now that we have our InfoPath form all settled, let's create a new Visual Studio sequential workflow for our support process. In Visual Studio 2010, create a new project off the SharePoint Sequential Workflow template. Give the project the name of SiteCollectionSupportWorkflow and the workflow itself a name of Site Collection Support Workflow. You'll be prompted to create a List Workflow or a Site Workflow. Leave the type set to List Workflow because we want to bind our workflow to a list that contains our support requests. On the last screen in the wizard, uncheck the checkbox asking you to bind the workflow to a list. We want to do the binding ourselves so to test our association form. Essentially if you leave this checkbox checked, you're associating the workflow right now, and you won't see your association form later.

InfoPath Deployment Caveats

There are two other "tricks" that you should know about when creating your InfoPath form for your Visual Studio workflow. The first one is to make sure the form's security level is set to at least "Domain" when you publish. If you have it set to automatically detect the level, it may already be set to Domain. You can set this in Form Options under Security and Trust. Domain is needed to interact with SharePoint data. Full trust would be needed if you need to access your personal computer's hard drive, for example.

Secondly, the form that you add into the Visual Studio workflow needs to be the published version of the form. So before you add the form into your Visual Studio project, make sure to publish it first. You'll want to use the Publish to Network Location feature. When you publish to a network location, make sure to leave the Access Path blank. Our workflow will know the URL to access the form with from Forms Server, and specifying a network location or URL here will mess stuff up. Figure 9.8 shows how your form should look in the last step of the publishing wizard.

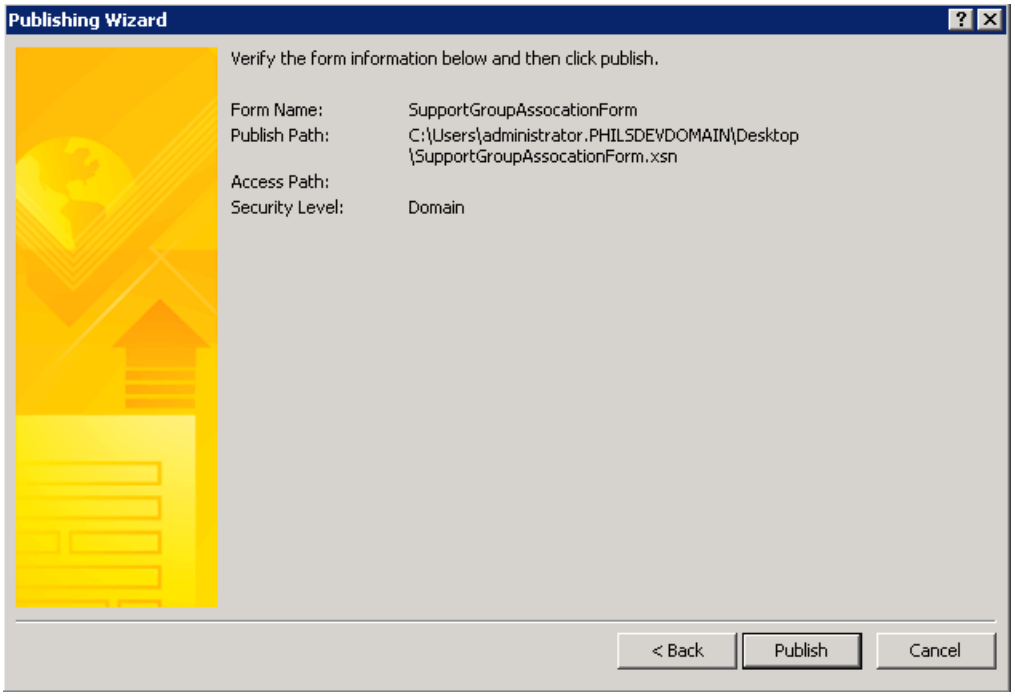
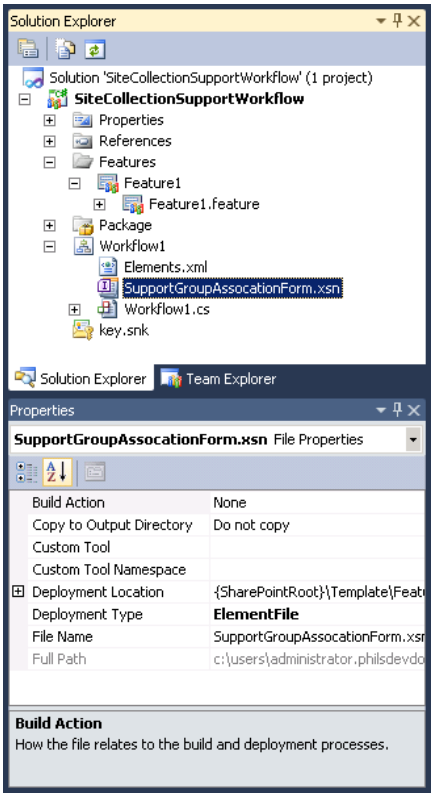


Figure 9.9 Make sure when you publish forms for Visual Studio workflow that you publish to a network location, your Access Path is blank, and your security level is at least set to Domain or higher.

When a new Visual Studio workflow project is created, there is a feature definition automatically created that deploys that workflow into SharePoint. What we need to do is get our InfoPath form into that feature definition, and instruct the workflow to use that form rather than the default blank form. The first step is to add the InfoPath form into the workflow. Once in the workflow, you can set its

deployment type property to be ElementFile. Once the form is set to ElementFile, the feature definition will automatically know to include it in the feature. Follow these steps to add the form into the feature:

Adding a custom Association Form into a Visual Studio Workflow

	Action	Result
1	Add the form inside the workflow itself, and update its DeploymentType property to include it in the feature: A) Right click on Workflow1, and choose Add, Existing Item. B) Browse out to your **Published (**Figure 9.9**)** InfoPath form and add it. C) Once it is added, right click on the InfoPath form and choose Properties. D) Change the DeploymentType property to ElementFile. Note: When we Build and Deploy our project, our InfoPath form will be placed in the Features directory under 12\Template\Features\[Feature Name]\[Workflow Name]\[Form Name].xsn.	

With our feature definition properly packaging up our form, we now need to instruct our feature to deploy it into Form Server. InfoPath association forms (as well as initiation and modification forms) must be deployed globally in Form Server to work. At this point you may be wondering why not simply upload the form into Form Server manually, via Central Administration? Well the trouble with that is those manually uploaded forms are not "Workflow Enabled", which our forms obviously need to be. Therefore, to get our forms workflow enabled, we need the feature to deploy the form into Forms Sever via an event Handler. Follow these steps to setup this event handler:

Adding a custom Association Form into a Visual Studio Workflow

	Action	Result
2	Instruct the feature to deploy the association form upon activation of the feature:	When these steps are completed, your

A) Open Feature1.feature, click Manifest, and then click Edit Options.

B) Add the ReceiverAssembly and RecieverClass attributes within the Feature element:

```
ReceiverAssembly="Microsoft.Office.Workflow.Feature,
    Version=14.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c"
ReceiverClass="Microsoft.Office.Workflow.Feature.
    WorkflowFeatureReceiver"
```

C) Add the RegisterForms Property element in the Properties element:

```
<Property Key="RegisterForms"
Value="Workflow1\*.xsn" />
```

feature XML should look something like Figure 9.9

```
<?xml version="1.0" encoding="utf-8" ?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
  ReceiverAssembly="Microsoft.Office.Workflow.Feature, Version=12
  ReceiverClass="Microsoft.Office.Workflow.Feature.WorkflowFeatur
  <Properties>
    <Property Key="GloballyAvailable" Value="true" />
    <Property Key="RegisterForms" Value="Workflow1\*.xsn" />
  </Properties>
</Feature>
```

Figure 9.10 When you're done customizing the feature file, your feature XML should be similar to this.

Before we move on, let's test that our form was successfully deployed into Forms server. It's important to test now, because we're going to need the form's unique id (URN) which we can easily get when it is in Forms Server. Right click on the project and choose Deploy. Wait a bit, and when it has successfully deployed, navigate to Central Admin to see if the template was deployed. Under General Application Settings, click Manage Form Templates under the InfoPath Forms Services heading. Note that the form was successfully added, and the Workflow Enabled column is set to Yes.

The last step in this process is to instruct the workflow to use our custom association form when the workflow is first added to a list. This is done by modifying the workflows Elements.xml file. There are two main things we want to configure in this XML file, the AssociationUrl attribute of the workflow, and a piece of meta data called Association_FormURN. The AssociationUrl attribute is set to an ASPX page in the Layouts directory in SharePoint. This ASPX page has a InfoPath Form viewer web part on it that renders our InfoPath form. The web part looks at the Association_FormURN meta data to figure out what form to render in Forms Server. We'll need to stick our form's URN in this meta property. Follow these steps to configure this Elements.xml file:

Adding a custom Association Form into a Visual Studio Workflow

	Action	Result
3	Configure Workflow1's Elements.xml file: A) Inside the Workflow element, right after CodeBesideAssembly, add AssociationUrl="_layouts/CstWrkflIP.aspx" B) Inside the MetaData element, add a Association_FormURN element with an open and close tag. C) Within the Association_FormURN element, add your custom form's URN. You can get this URN from within Forms Services, Manage Form Templates, and clicking View Properties on your form. You should see a property called Form ID.	Once complete, your Elements.XML file should look something like Listing 9.3.

Listing 9.3 Workflow1's Elements.xml file after adding assocation form

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Workflow
     Name="Site Collection Support Workflow"
     Description="My SharePoint Workflow"
     Id="719253c3-78d7-41b3-a0a8-bfa228b8588a"
     CodeBesideClass="SiteCollectionSupportWorkflow.Workflow1.Workflow1"
     CodeBesideAssembly="$assemblyname$"
     AssociationUrl="_layouts/CstWrkflIP.aspx">                    1
    <Categories/>
    <MetaData>
      <Association_FormURN>urn:schemas-microsoft-                  2
         com:office:infopath:SupportGroupAssocationForm:-myXSD-2009-12-
         12T17-59-13
      </Association_FormURN>
      <AssociationCategories>List</AssociationCategories>
      <StatusPageUrl>_layouts/WrkStat.aspx</StatusPageUrl>
    </MetaData>
  </Workflow>
</Elements>
```

1) Hosts InfoPath viewer web part
2) Loads form from Form Server

We finally come to it… let's test our workflow and custom association form. Build and Deploy the project again to push our latest changes. Then, find a SharePoint list and add our Site Collection Support workflow. You'll notice when you add the workflow to the list our association form loads to provide the user options for what groups represent each tier of support (Figure 9.10).

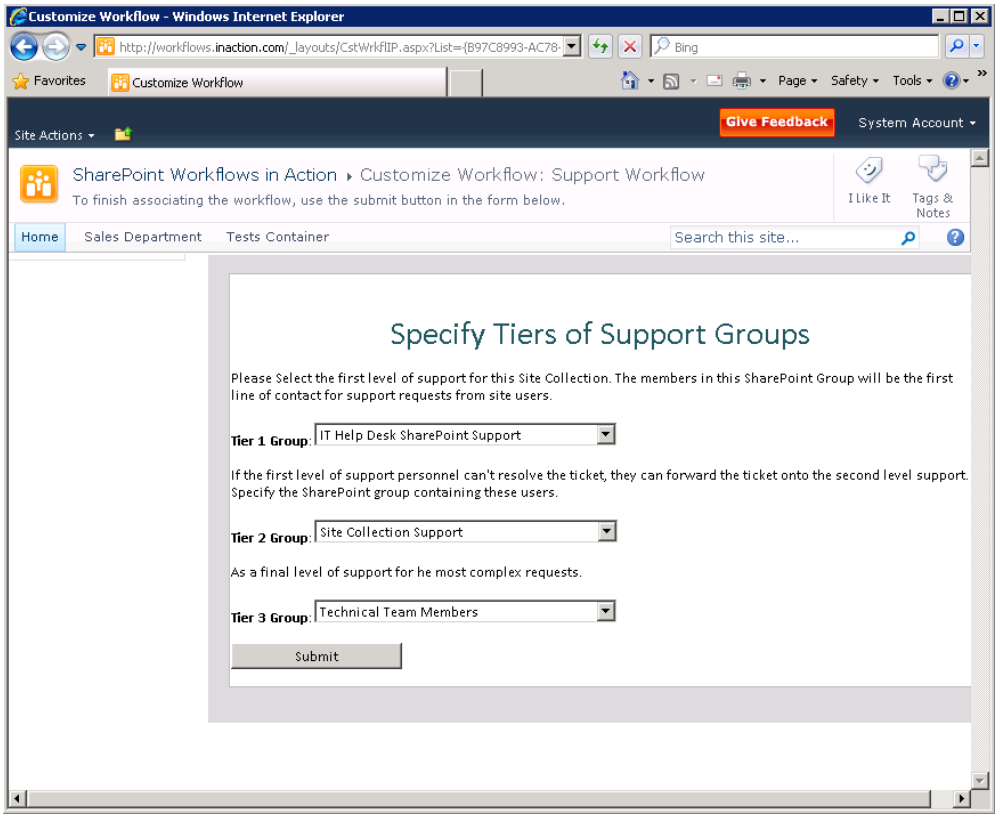


Figure 9.11 Here's our association for in the browser. The form loads whenever you add our Support workflow onto a list.

9.3.2 Building a Custom Initiation Form

Initiation forms are added into a Visual Studio workflow in almost the exact same way that association forms are added. The previous section walked through this process in great detail. Since the pluming for both association forms and initiation forms are so similar, we're not going to rehash that same process in this section. Rather, we will show how to add an initiation form into the project that was setup in the previous section. Before you start, create an InfoPath initiation form using Figure 9.12 as a guide.

Building a custom Initiation Form

	Action	Result
1	Add the form into the project, and deploy it to Forms Server: A) From within the Visual Studio project, right click Workflow1 and choose Add, Existing Item. B) Browse to and select the published version of the initiation form. C) After the file is added, right click the file and choose properties and change the Deployment Type property to ElementFile. D) Build and Deploy the project. This will deploy the form into Forms Server Note: All forms in Form Server must be marked at Workflow Enabled ("Yes") to be viable in the workflow.	Workflow Enabled Yes Yes Yes Yes
2	Copy Form's unique ID: A) Navigate into Manage Form Templates within Central Administration under InfoPath Forms Services B) Find your initiation form that was just deployed. C) Click on the form, and choose View Properties and copy the Form ID (URN). You will need this URN in just a bit	You'll need the form's unique ID in your clip board for step 3.
3	Back in the workflow, edit the Elements.xml file: A) In the Workflow element, add an InstantiationUrl attribute and set it to _layouts/IniWrkflIP.aspx. B) Inside the MetaData element, add an Instantiation_FormURN element. C) Within the Instantiation_FormURN element, paste in the URN from step 2.	After you paste in the URN from your initiation form that was just deployed into Forms Server, you can go ahead and Build and Deploy your project. If you go back into a list where the workflow is installed, and you start a workflow on an item in that list, you should see your custom initiation form (Figure 9.11).

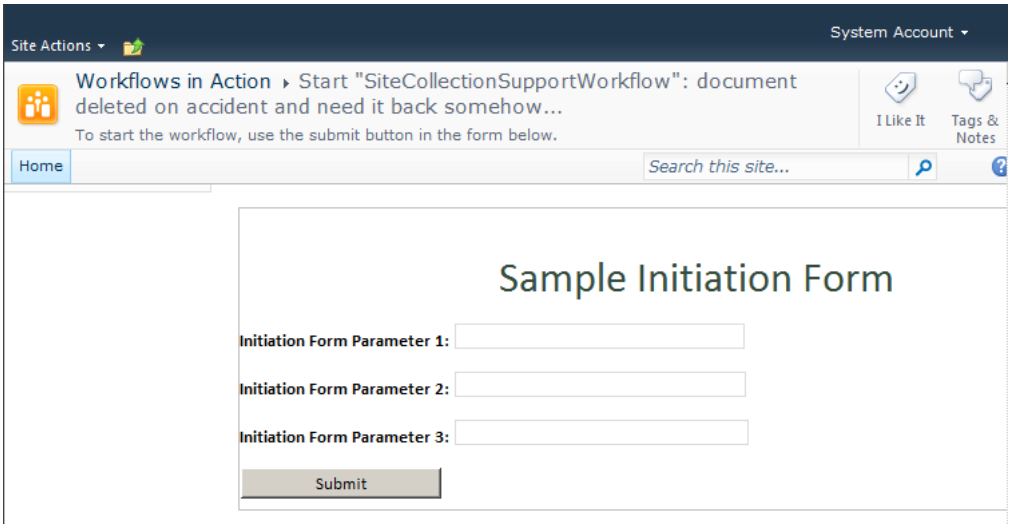


Figure 9.12 Initiation forms are built in a very similar manner as association forms. Rather than appear when the workflow is added to the list, initiation forms present themselves when the workflow first starts.

Auto-starting Workflows won't show the Initiation Form

Note: if your workflow is set to automatically start when a new item is added, the initiation form will not load when the new item is added. The initiation form will only show when the workflow is manually started.

9.3.3 Working with the Association or Initiation Form Data

Well it's pretty neat that we got our association and initiation forms integrated into a custom workflow. So, what does the workflow actually do with the data in those forms? This is something that is very important to know how to do. Essentially, the OnWorkflowActivated activity has a property called WorkflowProperties. Within WorkflowProperties, there are two more properties that store our association and initiation data, AssociationData and InitiationData. Each of these properties are strings that contain our form's XML data. What we need to do is cast this string into an .NET class that is built off the form's schema definition file. Going back to the association form, build a proxy class off the form's schema, and from within the workflow let's take the AssociationData and cast it into an object our workflow can work with. Follow these steps to do this:

Programmatically retrieving InfoPath Form Data

	Action	Result
1	Add the form's proxy class into your Visual Studio project: A) Open the form in InfoPath Designer, and under File click Publish, and then Export Source Files. B) Open the Visual Studio 2010 command prompt and browse to the directory that you exported the source files to.	A new class will be generated off the form's XSD schema file and added to the Visual Studio project.

	C) Run the following command on the schema definition file to generate our proxy class: "`xsd myschema.xsd /c /l:cs`". D) Add the myschem.cs file that was generated into your Visual Studio project and rename the file to its corresponding class name.	

2	Add a namespace and two using statements into the proxy class:
	
	A) By default, the class is not in a namespace. Wrap the class in the same namespace that your workflow is running in. This will allow you to create an instance of this class from within your workflow.
	
	B) In your workflow's code behind, add the following using statements at the top:
	`using System.Xml.Serialization;`
	`using System.Xml;`
		The proxy class will now be in the same namespace as the workflow, and the necessary using statements will allow the project to build successfully.

3	Add code into the OnWorkflowActivated activity that will serialize the form's data into the proxy class: A) Back to your workflow's designer surface, right click on the OnWorkflowActivated activity and choose Generate Handlers. B) This will generate an event handler and method. Inside this method, add the code from Listing 9.4 (note: replace "SupportGroups" with whatever your association form's class name is).	An event will fire when the workflow first starts that will serialize the Form's data into the proxy class.

Listing 9.4 Deserializing our Association form's data into an object

```
XmlSerializer serializer = new XmlSerializer(typeof(SupportGroups));                    1

XmlTextReader reader = new XmlTextReader(new System.IO.StringReader(                    2
   onWorkflowActivated1.WorkflowProperties.AssociationData));

SupportGroups associationFormData = (SupportGroups)serializer.Deserialize(reader);      3

string tier1 = associationFormData.Tier1Group;                                          4
string tier2 = associationFormData.Tier2Group;                                          4
string tier3 = associationFormData.Tier3Group;
string currentTier = tier1;
```

1) Create a serializer object off schema class
2) Read in AssociationData string into reader
3) Cast XML into a strongly typed object
4) Start programming against association data

The first thing that Listing 9.4 does is creates an XmlSerializer off our proxy class **#1**. In this case, the SupportGroups class is used that was built off the association form's schema definition file. Next we read the association data that was entered by the user into an XmlTextReader **#2**. This XmlTextReader will hold that data entered by the user in a format that is consistent with the association form's schema

definition. After we have this XML, we an cast it into a new instance of our proxy class **#3**. With this object we can now start coding against the data that was entered in the form.

This wraps up the discussion on initiation and association forms. Remember that both those form types always present themselves to the user before the workflow starts. Now it's time to discuss a form type that presents itself after the workflow starts, modification forms.

9.3.4 Configuring Activities for Workflow Modifications

Modifications in workflows give your end users the ability to alter their workflow's behavior after it has started. This is commonly done when someone wants to reassign a task to someone else. To play off this a bit, this section will extend the Support workflow that was built earlier in this chapter to provide a way for users to escalate the support ticket up the ladder. Essentially, they will be able to "modify" the workflow and specify a different SharePoint group of users who afterward will be responsible to complete the support request.

Before we can get to building and deploying our modification form, we need to tell our workflow that modifications are allowed. The way a modification works is there is a modification link presented on a workflows status page. When the user clicks this link they are taken to a modification form, and after they submit that form our workflow can react to the submission and the data the user entered. Well, the trick is to get this link to appear on the workflow status pages because by default it is not there. You must specifically allow modifications for it to show. Figure 9.12 shows this link's placement on the workflow status page. Note that the text the link uses is configurable.

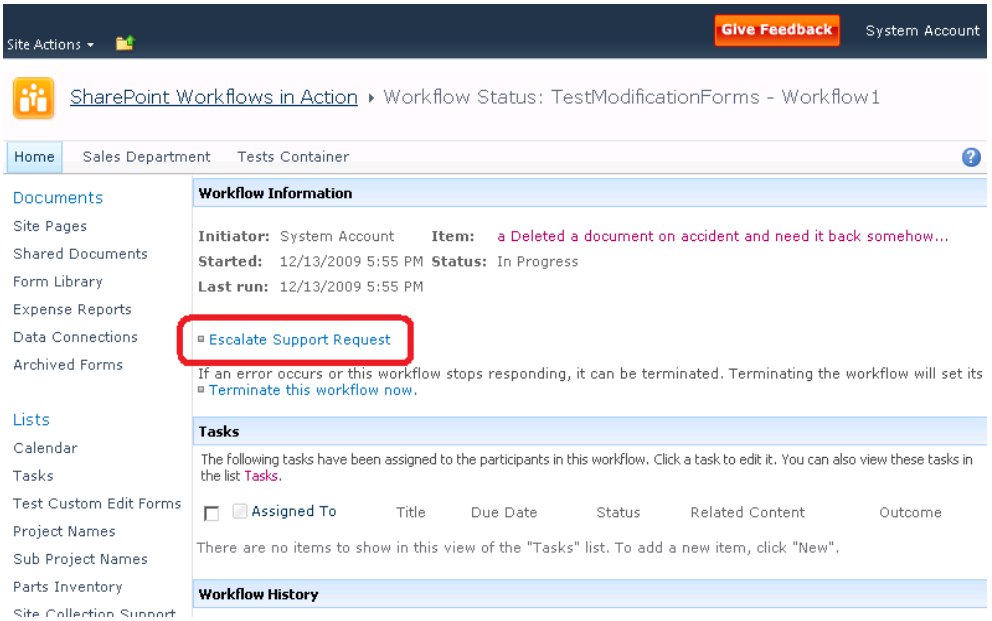


Figure 9.13 Workflow modifications are initiated from the workflow status page with a link that can present a configurable text to the user. The link only shows when modifications are specifically allowed on this workflow.

There are two main workflow modification activities used to enable this functionality, EnableWorkflowModification and OnWorkflowModified. You can use the EnableWorkflowModification activity to enable modifications, and in effect, display the link on the workflow status page. When the user clicks that link, they are taken to the modification form, and when that forms is submitted, the workflow framework fires a WorkflowModified event that a OnWorkflowModified activity reacts to.

As was mentioned, you can specify when a workflow can and cannot be modified. Typically, the EnableWorkflowModification activity is deployed inside an EventHandlingScope activity. You can use the EventHandlingScope activity to manage when a modification can take place. While the workflow is executing inside the EventHandlingScope activity, and the EnableWorkflowModificaiton activity is also inside it, modifications will be enabled. As soon as the workflow passes out of the EventHandlingScope activity, modifications will no longer be available.

Notice in Figure 9.13 that the EventHandlingScope activity has multiple "views". The OnWorkflowModifed activity is placed inside the Event Handlers view and thus can react to any workflow modifications that take place in the scope of the EventHandlingScope activity.

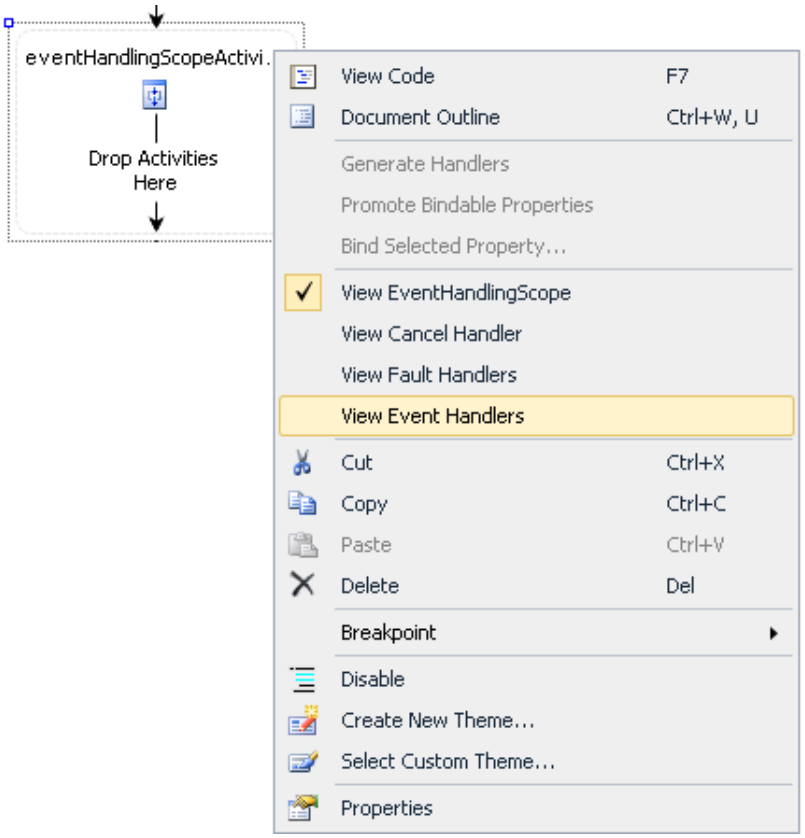
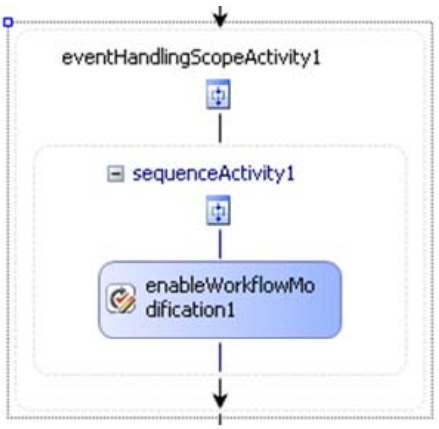
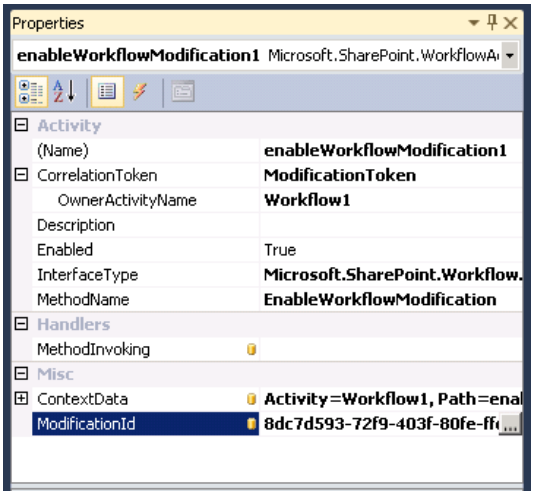


Figure 9.14 You can use the EventHandlingScope activity to manage when a workflow modification can and cannot take place.

You can toggle between views by right clicking the activity, and then selecting the view you want to edit. Then, within that view you can start dropping activities. Notice that besides generic event handlers, you can also react to exceptions (faults) as well as cancelations. Now it's time to configure these activities. Follow these steps to add this functionality into our support workflow:

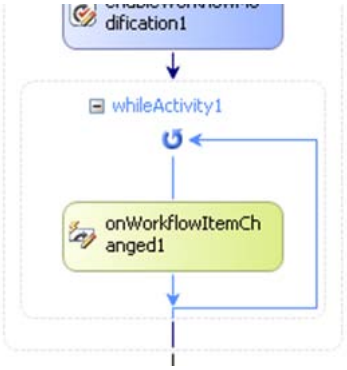
Configuring EventHandlingScope and EnableWorkflowModification Activities

	Action	Result
1	Add the two activities onto the Workflow designer surface: A) Directly below the OnWorkflowActivated activity, add a EventHandlingScope activity B) Inside the EventHandlingScope activity, add a Sequence activity C) Inside the Sequence activity, add an EnableWorkflowModification activity	
2	Notice the red exclamation point, signaling that we need to assign a correlation token. Assign a token to the ebableWorkflowModification1 activity: A) Set the correlation token to something other than what the OnWorkflowActivated activity is using, like "ModificationToken". Leave the OwnerActivityName property to be set to your workflow name. In this case it is "Workflow1".	The red exclamation point on the activity will go away.
3	The EnableWorkflowModification activity needs a way to pass data between the form and the workflow. In the activity's properties, click the ellipses on the ContextData property and Bind to a new member. Leave the defaults can click Ok	The ContextData property will now be bound to a field.

4	Every workflow modification needs a unique ID. Assign a unique GUID to the ModificationID property of the EnableWorkflowModification activity: A) Use the GUID generator under Tools, Create GUID (registry format) to copy a new GUID. B) Paste the GUID into the ModificationId property. Note: Make sure to remove the braces in the GUID generated from the GUID generator	

Now what we want to do is add some business logic into this workflow to make it feel more alive. Let's add a while loop activity and an OnWorkflowChanged activity to wait for the support request to be completed. If the request is completed, the workflow is finish executing. Follow these steps to configure the While and OnWorkflowChanged activities that the workflow will use to wait for the request to be completed:

Add activities to make the workflow wait for the request to be completed

	Action	**Result**
1	Add a While and OnWorkflowItemChanged activities: A) Below the EnableWorkflowModification activity, add a While activity. B) Inside the While activity, add a OnWorkflowItemChanged activity.	
2	Set the correlation token: A) Set the correlation token of the OnWorkflowItemChanged activity to the same token as the OnWorkflowActivated activity. The default would be "workflowToken". Note: You'll another red exclamation point on	The red exclamation point on the OnWorkflowItemChanged activity will go away.

	the While activity we still need to deal with. See step 3.	
3	Add code into the While activity: A) Right click the While activity and choose properties. Change the Condition property to be a Code Condition, B) Click the plus sign next to the Condition property, and type the name of a method to execute for this while loop. Type something like "WhileRequestIsNotComplete". C) This will kick you into the code editor. Enter the code from Listing 9.5 into the WhileRequestIsNotComplete method.	The red exclamation point on the While activity will go away.

Listing 9.5 WhileRequestIsNotComplete method

```
string status = onWorkflowActivated1.WorkflowProperties.
   Item["Status"].ToString();                                    1

if (status == "Resolved")
   e.Result = false;                                             2
else
   e.Result = true;                                              3
```

1) Check if support request is compete
2) Exit while loop if yes
3) Continue looping if no

This code first retrieves the value from the Status column **#1** into a standard Issues list list item. If the request is resolved, the while loop stops looping **#2**, and the workflow finishes. If the request is not complete, the while loop keeps looping **#3** and waiting for it to be completed. After this code is entered, we should be done with the body of our EventHandlingScope activity. Figure 9.14 shows how your workflow should look up to this point.

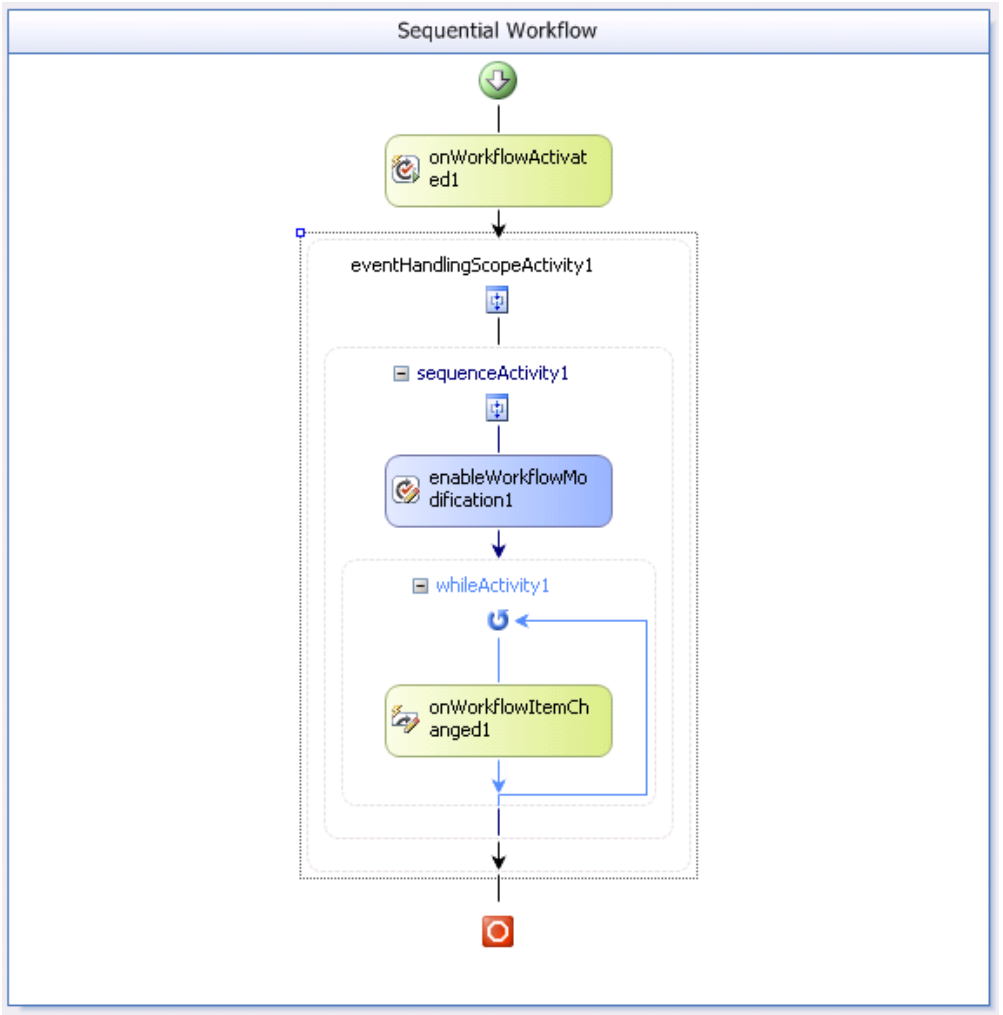
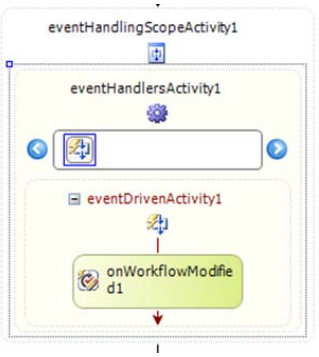


Figure 9.15 When the EnableWorkflowModification activity is inside an EventHandlingScope activity, workflow modifications will only be allowed while the workflow is executing in the EventHandlingScope activity.

After everything looks good, it's time to configure the modification event handler and setup the OnWorkflowModified activity. To do this, add an EventDriven activity into the event handlers' view of the EventHandlingScope activity. With our OnWorkflowModified activity inside this EventDriven activity, any time the workflow is modified in this scope, this activity will fire. Follow these steps to configure this:

Configure the Modification Event Handler

	Action	Result
1	Add the EventDriven and OnWorkflowModified activities: A) Right click the EventHandlingScope activity and choose View Event Handlers. B) Add an EventDriven activity into the EventHandlers activity that was automatically added inside the EventHandlingScope activity. C) Add the OnWorkflowModified activity inside the EventDriven activity.	
2	Set the correlation token of the OnWorkflowModified activity to be the same token as the EnableWorkflowModification activity. A) Set the correlation token to "ModificationToken"	The red exclamation point on the OnWorkflowModified activity will go away.
3	Configure three of the properties on the OnWorkflowModified activity: A) Edit the ContextData property of the OnWorkflowModified activity to be the same ContextData property that the EnableWorkflowModification activity is using. This will allow the workflow to pass and receive data between the forms. B) Edit the ModificationId property of the OnWorkflowModified activity, and make it have the same GUID that the EnableWorkflowModification activity is using. C) Bind the User property of the OnWorkflowModified activity to a new member. This property contains the username of the person who made the modification. D) Drop a LogToHistoryList activity below the OnWorkflowModified activity and then set the HistoryDescription property of the activity to be "Modification Success!". Note: we'll use the LogToHistoryList activity to log to so history so that we know the modification plumbing is working properly. To take this example a step further, we could add code to the OnWorkflowModified activity that updates the currentTier field, and perhaps	The OnWorkflowModified activity will now be complete. At this point your Event Handlers view of the EventHandlingScope activity should look something like Figure 9.15

emails the group members of the next tier, notifying them that they have a support request needing their attention.	

.

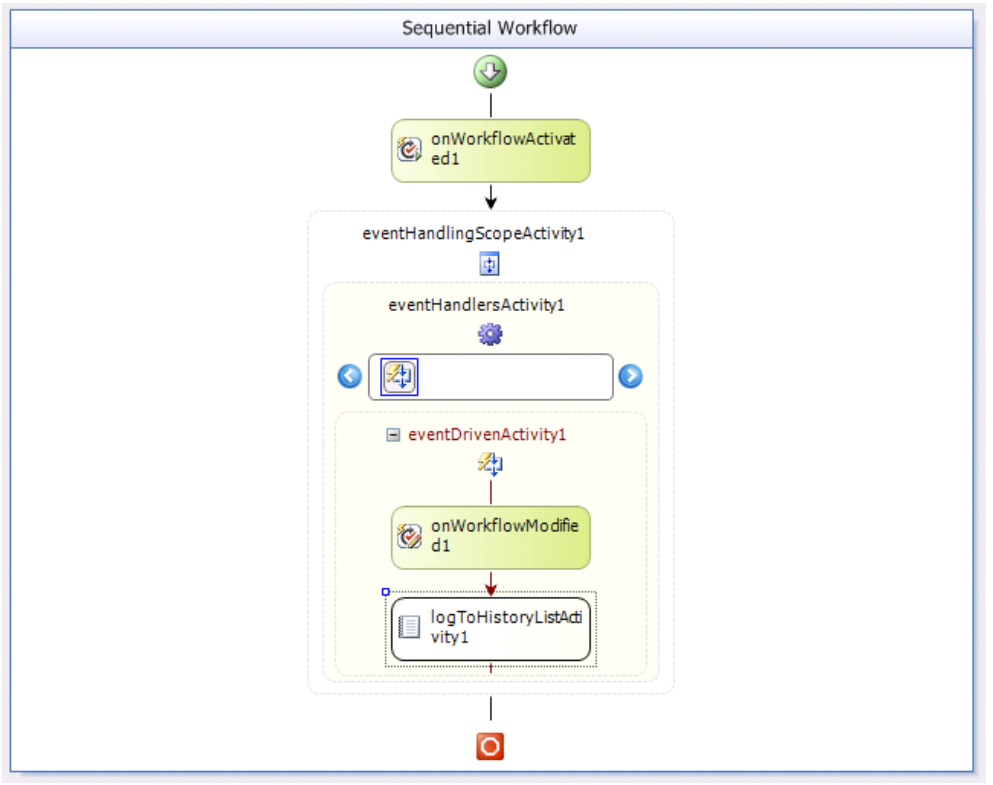


Figure 9.16 By placing the OnWorkflowModified activity inside the Event Handlers view of the EventHandlingScope activity, the OnWorkflowModified activity will respond to any workflow modifications made during the execution of the EventHandlingScope activity.

With these activities and code in place, we're now ready to start configuring our modification forms. The next section will walk you through how to integrate an InfoPath modification form.

9.3.5 Building a Custom Modification Form

Now that we have the foundation of our modification enabled workflow in place, it's time to add our modification form into the mix. An InfoPath modification form is the first in the batter's box. The steps to integrate a InfoPath modification form at a high level are to first create the form, second setup the context data that the workflow and the form use to communicate with each other, and lastly to edit the workflow's elements.xml file to tell the workflow which form to use. Let's start with the first step, which is setting up the form itself.

The InfoPath form we'll be using in the section will be an extension of the Support request system we setup earlier. What we want is a form that a user can use to modify the workflow with, and in doing so, escalate the support request to the next level of support. They would do this if they didn't feel they had the expertise to answer the requestor's question or problem. Figure 9.16 shows a sample form this demonstration will be using.

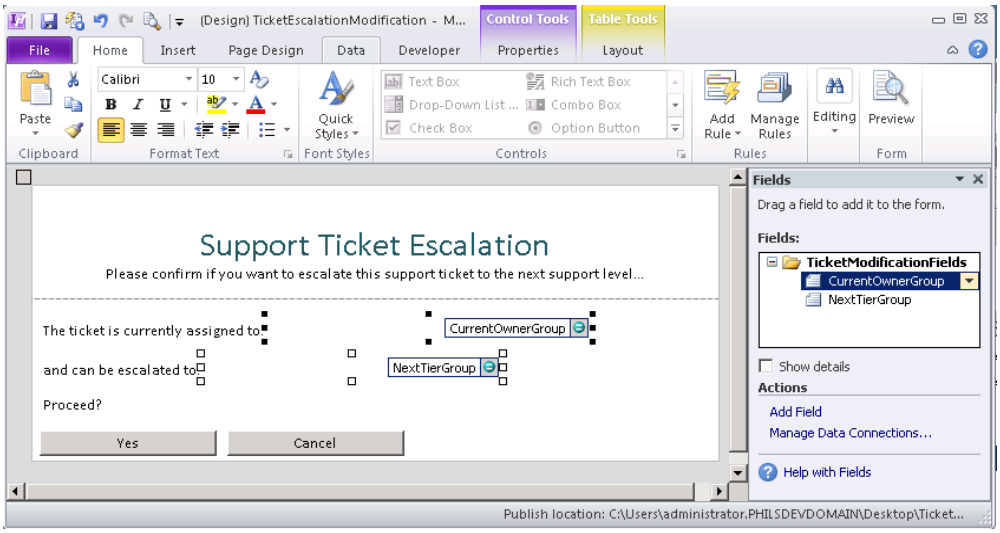


Figure 9.17 This InfoPath form will be used in our workflow modification to escalate a support ticket from one level of support to another.

This form has two pieces of data in it, the current level of support (CurrentOwnerGroup), and the level of support that comes next (NextTierGroup). If you remember, our workflow's association form allowed users to specify SharePoint groups that contain users who were responsible to answer support tickets for each tier of support. We want this modification form to let the user escalate the ticket from one group/level to another. Our workflow code will pass these two pieces of data into the form when it loads, and the user will simply confirm if they want to escalate the ticket or not. If they do, our workflow code will reassign the support ticket by flagging an owner column on the list item.

Before we can start configuring our workflow's xml file to tell it to use this form, we need to setup our context data. As mentioned, the workflow can pass data back and forth with the form via the ContextData property of the EnableWorkflowModification and OnWorkflowModified activities. At this point, this property is set to an empty string. It needs to be set to a string that contains the form's XML schema definition, as well as any preset data you want to pass to the form. This is how our workflow will pass the CurrentOwnerGroup and NextTierGroup values into our InfoPath form.

To do this, we first need to instantiate a proxy class off our form's XML schema definition. To get this class, we need to use the XSD command again. Follow these steps to build our form's proxy class and add the class into our Visual Studio project:

Adding the Form's Proxy Class into the Visual Studio Project

	Action	Result
1	Generate the Form's proxy class of it's schema definition file: A) Start in your InfoPath form, and under File > Publish, choose Export Source Files. Export the source files to a directory you can get at with command prompt. B) Open the Visual Studio command prompt and navigate to the published directory. Run this command: `xsd myschema.xsd /c /l:cs` C) This command will write a new file called myschema.cs. Add this file into your Visual Studio project with the workflow we setup in the previous section.	Your Visual Studio project will have a new class added that is a proxy to the form's data.
2	Name the class file name and provide a namespace for the class: A) Rename the file to whatever the class name is (or more specifically, the name of the root element in the data in the InfoPath form). In this case, rename the file to TicketModificationFields.cs. B) Wrap the TicketModificationFields class with the same namespace that your workflow is running in. By default, TicketModificationFields will not be in a namespace and your workflow code won't be able to instantiate it. C) Add a using statement for the System.IO namespace.	The class name will be unique (rather than the generic "myschema.xsd") and will be in the same namespace as Workflow1.
3	Add code to your EnableWorkflowModification activity: A) Back on your workflow designer surface, right click the EnableWorkflowModification activity click Generate Handlers. B) Add the code in listing 9.6 into this method.	A new method will be created. We want to use this enableWorkflowModification1_MethodInvoking method to instantiate our TicketModificationFields class, and get our data ready for our form to consume

Note: for listing 9.6 to work, a few new fields are required

In section 9.3.3 you created four string fields in the onWorkflowActivated1_Invoked method: tier;, tier2; tier3; and currentTier. These fields were local to this method. Listing 9.6 is referencing these fields, so they will now need to be declared outside the method, and made available to the entire class.

Listing 9.6 enableWorkflowModification1_MethodInvoking

```
TicketModificationFields data = new TicketModificationFields();          1
```

```
data.CurrentOwnerGroup = currentTier;                                    2
data.NextTierGroup = tier2;                                              2

using (StringWriter writer = new StringWriter())
{
    XmlSerializer s = new XmlSerializer(                                 3
        typeof(TicketModificationFields));                               3
    s.Serialize(writer, data);                                           3

    this.enableWorkflowModification1_ContextData1 = writer.ToString();   4
}
```

1) Instantiate form's proxy class
2) Determine support group to escalate to
3) Serialize data into StringWriter
4) Assign ContextData to output string

The first thing this code does is instantiates our proxy class built of the form's XML schema definition **#1**. We then want to **#2** set a couple values in that class that will get passed into the InfoPath form. The form needs to know the current tier, and the next tier the ticket can be escalated to. **#3** Next, serialize this data into a StringWriter that can set its string value to the InfoPath modification form's ContextData property.

enableWorkflowModification1_MethodInvoking only fires once

This method sets the values that will show in the modification form. Since this code only fires once, the current tier will be tier 1, and the next will be tier 2. Another EnableWorkflowModification activity with a different correlation token will be needed to update the modification form with new values after the first modification occurs. If a second EnableWorkflowModification activity is not used, the modification form will have the same field values, when in fact they ought to have been incremented (current = tier 2 and next = tier 3). This second, and possibly third, EnableWorkflowModification activity is not hard to setup, but unnecessarily elongates the scope of this example.

With the ContextData under our belt, we now can focus on the last part of this InfoPath journey, which is to tell our workflow which form to use when the modification link is clicked. All this is done in the workflow's Elements.xml file, we first need to get our form into our project and feature. To do this, simply add the **Published** form inside the workflow container like before. Once inside, right click the form and set the DeploymentType property of the form to ElementFile. Afterwards, your Visual Studio solution explorer should look something like Figure 9.17.

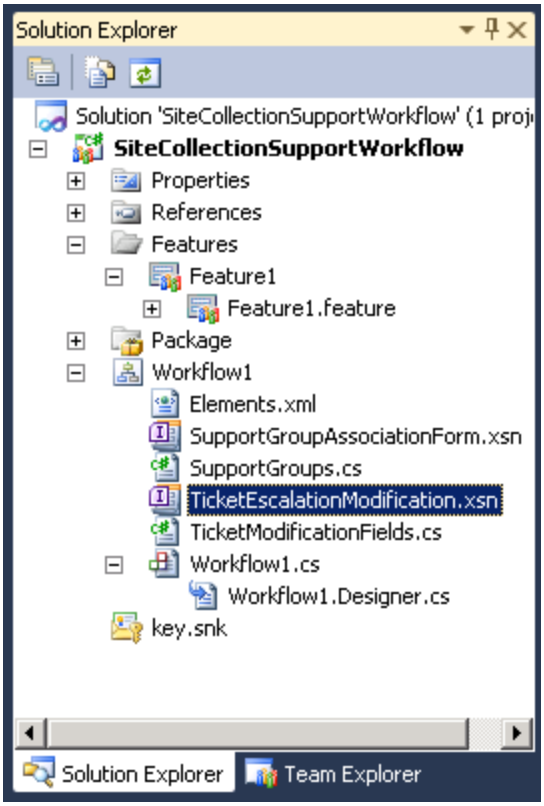


Figure 9.18 Add the modification form into the workflow, and change its DeploymentType to ElementFile to get it packaged up in the feature.

With our modification form inside our workflow, it's finally time to tell our workflow to use this form. Follow these steps to edit the Elements.xml file:

Edit the Feature's Elements.xml file to tell the Workflow to use the Modification Form

	Action	Result
1	In the Workflow element, add a new ModificationUrl attribute: ModificationUrl="_layouts/ModWrkflIP.aspx". Deploy the Visual Studio project by right clicking the project, and choosing Deploy.	This will deploy our new form into Forms Server because of the pluming we setup in section 9.2 for our association form.
2	Within the MetaData element , add a Modification_[UNIQUE GUID]_FormURN element: A) In the MetaData element, add a child element called <Modification_[UNIQUE GUID]_FormURN></Modification_[UNIQUE GUID]_FormURN>. Replace [UNIQUE_GUID] with the ModificationId	A new child element will be added to the MetaData element. This element will tell the workflow what form in Forms

	GUID your EnableWorkflowModification and OnWorkflowModified activities are using. B) Grab the URN from your form in Forms Server (via View Properties), and paste it into this element.	Server is acting as our modification form.
3	Add a Modification_GUID_Name element: A) Similarly, create another element called <Modification_GUID_Name>. B) Inside this Name element, type a name for your modification. This name is what the title of the URL will be on the workflow status page. For example, type "Escalate Support Request"	A new child element will be added to the MetaData element. This element will tell the workflow what the name of the modification will be. This name shows on the Workflow Status page.

Build and Deploy the project again which will ship off the latest Elements.xml updates. Lastly, find a list in SharePoint you can add your workflow to, to test. Note the dependency on the "Status" (Issue Status is the display name) column. You should be using a out-of-the-box Issues list named Support Requests. Add the workflow to the list and specify some values in the association form. Afterwards, create a new item in the list (with Status set to Active) and start the workflow on that item. After you start it the workflow will be "In Progress" waiting for the Issue Status column to be set to Resolved. While it waits a modification is enabled. Click on "In Progress" and that will take you to the workflow status page, where you can start a modification. Click the "Escalate Support Request" modification link and our form should appear with the values sent from the workflow. After you click Yes, or submit, the OnWorkflowModifed activity will fire, and since there is no code currently in that activity, the workflow will simply log "Modification Success!" (Figure 9.19) into the history log via the LogToHistoryListActivity place directly after the OnWorkflowModified activity.

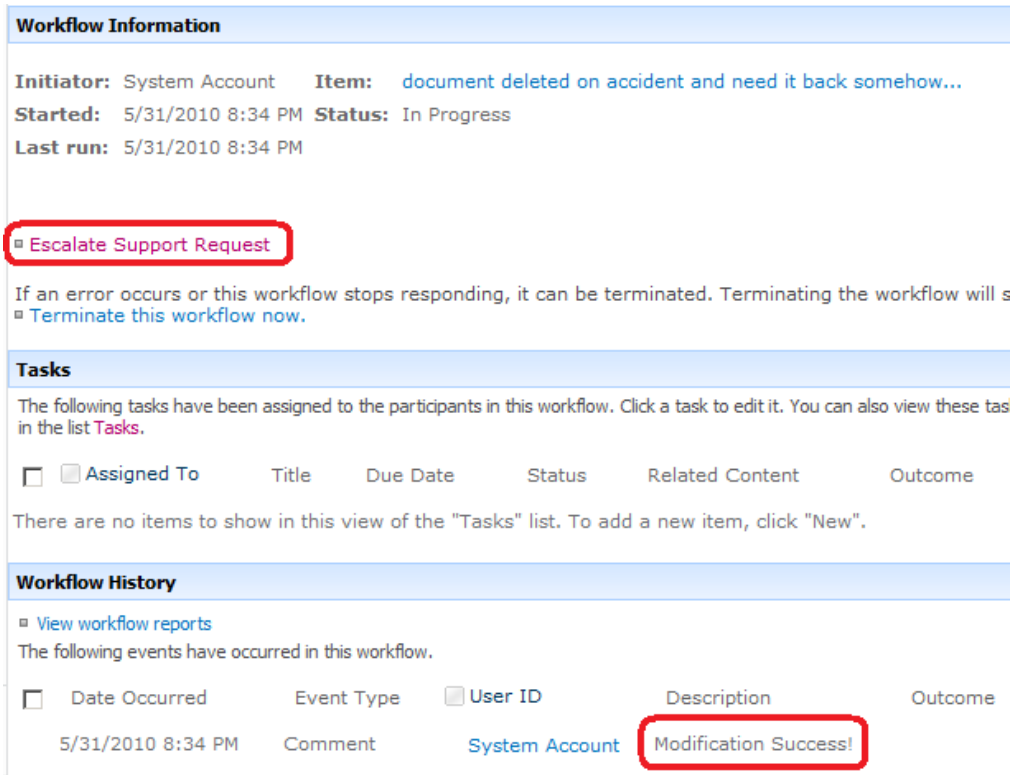


Figure 9.19 After you click the Yes button on your modification form, the onWorkflowModified1 activity will fire followed by the logToHistoryList1 activity which logs that the modification succeeded.

To take this example a step further, in the OnWorkflowModifed activity you could include code that changes the "currentTier" field to the next tier if the user selects the Yes button in the form. If the user cancels, then do nothing and the While loop continues the workflow looping until the request is resolved. If Yes is chosen, and the request is escalated, you could then send an email to the users in that tier's group notifying them that a request is waiting for their attention. To make the example even more complete, you could then assign a task to that tier, that when marked completed changes the Issue Status to resolved, thus ending the workflow. Refer to chapter 10 to learn more about tasks in Visual Studio workflows.

9.4 ASP.NET Forms in Visual Studio Workflows

After that discussion on InfoPath association and initiation forms, you're going to find ASP.NET forms a bit of fresh air. That's because it is much easier to deploy an ASP.NET form into a Visual Studio workflow. While ASP.NET forms are a bit trickier to develop, they are a ton easier to deploy. Part of the reason for this is all you have to do to add a new ASP.NET form into a workflow is simply right clicking the workflow, and choosing Add new item, and then select which form type you want to add. After you

add it, the form will be automatically packaged up into the feature and package, and all that's left is to deploy the project. Then, when you associate your workflow to a list, or initiate a workflow on an item, you'll get your new custom ASP.NET form! That takes only seconds to do! Before we get too excited, you'll notice when you see your form in the browser that it's blank. Deployment was easy, now it's time to put our developer hat on and get some controls on that form, and start passing data between the form and the workflow.

The example we'll be creating in this section is a generic initiation form. Since association and initiation forms are created in almost the identical way, it makes sense to only walk through one of the two. During the walkthrough, differences for association forms will call out, so even if you're building an association form, this walkthrough will be helpful.

Rather than continue working in the project we built for our Support Request workflow in the previous section, let's start from the beginning with a new project. Create a new sequential workflow project called TestASPNETWorkflowForms. Create a list workflow, and bind it to a list of your choice with the project provisioning wizard. After the project has been created, right click on Workflow1, and choose Add, New Item. The new item dialog will appear (Figure 9.18). Select either to add an association or initiation form.

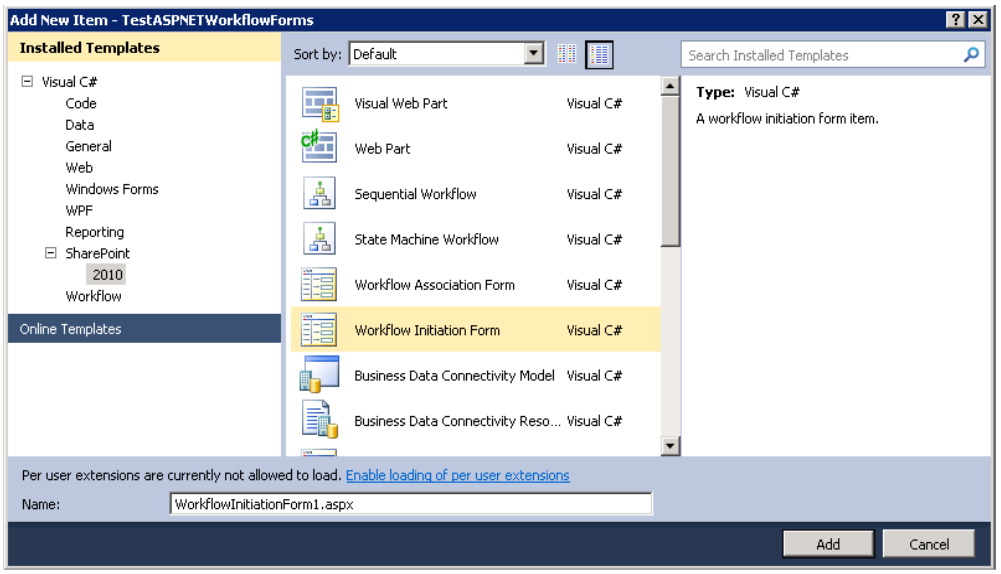


Figure 9.20 ASP.NET forms are extremely easy to integrate into a Visual Studio workflow. Simply right click the workflow and add a new item. Afterward, the ASP.NET form will be automatically packaged up and deployable through the workflow feature.

After you have the form added, you'll notice that you are taken to the ASP.NET HTML view of the form (Figure 9.19). What's so great about this auto generated form is that it already has a button on it that's all wired up and ready to go. If you go to the code behind of the form (right click the form in the solution explorer, and choose View Code) you'll notice four auto generated methods of particular interest. The first is the Page_Load method. You can use this method to set default values for the fields

on your form. Secondly is the GetInitiationData (or conversely, GetAssociationData) method. Our workflow calls this method to retrieve the values the user entered in the form. All we need to do is return a string in this method that contains a serialized class with the form data stored in it. Then, on the workflow side of things, we can deserialize this class and the workflow can do something with the data. The third and fourth methods of interest are the StartWorkflow_Click and Cancel_Click methods. These methods submit or cancel the form respectively. Not much to worry about with these because they've been filled out for you. Typically you won't even need to alter these methods, but depending on your business logic you may want/need to and they're available if necessary.

Going back to our ASP.NET HTML view, what we want to do is add a bunch of ASP.NET controls onto this form that our users can interact with. Inside the PlaceHolderMain content placeholder, add a few text boxes. We will use these text boxes to allow the user to enter in some information. We will later take this information and pass it into our workflow. Figure 9.26 shows how your ASP.NET code may look when complete.

```
<asp:Content ID="Main" ContentPlaceHolderID="PlaceHolderMain" runat="server">

    Please enter the first initialization parameter: 
    <asp:TextBox ID="InitParameter1" runat="server" /><br />

    Please enter the second initialization parameter: 
    <asp:TextBox ID="InitParameter2" runat="server" /><br /><br />

    <asp:Button ID="StartWorkflow" runat="server"
        OnClick="StartWorkflow_Click" Text="Start Workflow" />
          
    <asp:Button ID="Cancel" runat="server" OnClick="Cancel_Click" Text="Cancel" />
</asp:Content>
```

Figure 9.21 You can add ASP.NET controls on the form, and in the code behind you pull out these values and send them back to the workflow through the GetInitiationData method.

Now that we have our form the way we want it, we need to ship the data entered by the user back to the workflow. To do this we'll return a string containing the data returned through the GetInitiationData method (or GetAssociationData). We don't want to ship back just any string, we need a string that represents a serialized class that both the form and the workflow can instantiate and serialize/deserialize. Right click Workflow1, and choose Add, New Item. Select a class file and name the class InitiationFormParameters. Make the class public, and add two public strings, one for each parameter in the initiation form.

```
public class InitiationFormParameters
{
    public string InitiationParameter1;
    public string InitiationParameter2;
}
```

Back in the code behind of the ASP.NET form, find the GetInitiationData method. Add the code in Listing 9.7 into this method.

Listing 9.7 GetInitiationData method

```
string initdata = string.Empty;

InitiationFormParameters data = new InitiationFormParameters();                1
data.InitiationParameter1 = InitParameter1.Text;                               1
data.InitiationParameter2 = InitParameter2.Text;                               1

using (StringWriter writer = new StringWriter())
{
   XmlSerializer s = new XmlSerializer(
      typeof(InitiationFormParameters));
   s.Serialize(writer, data);                                                  2

   initdata = writer.ToString();
}

return initdata;                                                               3
```

1) Create class off form data
2) Serialize class values into string
3) Return string

This code first **#1** creates an instance of our class and assigns its properties to the values stored in the form that the user provided. Afterward, we **#2** serialize that class into a string with a StringWriter, and then **#3** return that string.

Our workflow calls this GetInitiationData method and loads the string into a property of the OnWorkflowActiviated activity called InitializationData (or AssociationData). What we can do is in the event handler of the OnWorkflowActivity activity is deserialize the string stored in the InitializationData property into a class our workflow can use. To do this, on the workflow designer surface, right click on the OnWorkflowActivated activity and choose Generate Handlers. Replace the code found in Listing 9.8 with the onWorkflowActivated1_Invoked method.

Listing 9.8 onWorkflowActivated1_Invoked method

```
string param1;
string param2;

private void onWorkflowActivated1_Invoked(object sender, ExternalDataEventArgs e)
{
   XmlSerializer serializer = new XmlSerializer(typeof(InitiationFormParameters));

   XmlTextReader reader = new XmlTextReader(new System.IO.StringReader(        1
      onWorkflowActivated1.WorkflowProperties.InitiationData));

   InitiationFormParameters initiationFormData =                               2
      (InitiationFormParameters)serializer.Deserialize(reader);

   param1 = initiationFormData.InitiationParameter1;                           3
   param2 = initiationFormData.InitiationParameter2;
}
```

1) List global workflow variables
2) Read InitiationData string into XmlTextReader

3) Deserialize XML into class object
4) Assign global variables

The goal of this code is to retrieve the data that the user entered in the initiation (or association) form. Our workflow can then perform some action on that data. The first thing we want to do is **#1** read the string out of the InitiationData property (or AssociationData) into an XmlTextReader. Then, we want to **#2** deserialize that XML into a new object that is the same type the string was serialized into (in this case the InitiationFormParameters object). After we have our object, we can start **#3** assigning some global variables, or take other actions.

With this last bit of code in, we need to do one last step to make sure everything is working. Let's add a LogToHistoryListActivity activity below the OnWorkflowActivated activity. We can use this history logger to render our form values on the workflow status page. After the LogToHistoryActivity activity is added, right click on it, and choose Generate handlers. In the method that was generated, add the following line of code to write out our form values:

```
logToHistoryListActivity1.HistoryDescription =
    "Param1: " + param1 + " Param2: " + param2;
```

With this activity logging our form parameters, we're finally at a place to do some testing. Build the project, and then deploy it. If you hadn't already associated the workflow to a list, do so now. Otherwise, start the workflow on an item in the list. When you start the workflow you should see our custom ASP.NET initiation form (Figure 9.20).

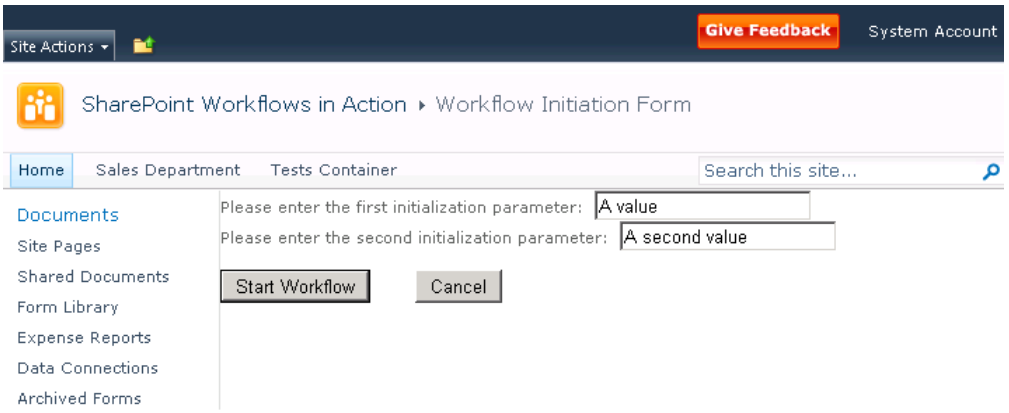


Figure 9.22 Our initiation form will prompt the user for pertinent information when the workflow starts.

After you enter some values and submit the form, you should see a status of completed. Click the status column and this will take you to the workflow status page. At the bottom, you should see the workflow history, with the parameters we entered into the initiation form listed (Figure 9.21).

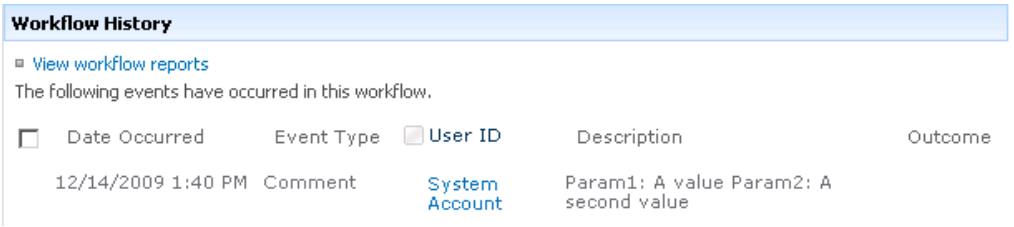


Figure 9.23 You can tell that the code ran correctly because the values entered in the form were written to the workflow history on the workflow status page.

9.5 Summary

It is important to know how to work with the data stored in an InfoPath form. For instance, data can be mapped to columns in the library, and you can add .NET code to alter the form's behavior. After a form is submitted, you can use a workflow to programmatically retrieve data from the form. These development techniques are a good foundation for workflow forms such as custom initiation and association forms.

Workflow forms are a powerful way to enable your end users to provide information to their workflows, as well as edit their workflow's behavior. There are three main types of workflow forms including workflow initiation forms, association forms, and modification forms.

Association forms present themselves when a workflow is first bound to a list, library or site. You can use this form to provide settings for all workflows of the given type that run on that list, library, or site. Initiation forms on the other hand display directly before a workflow executes. By default, this form is blank, but you can easily customize it through either InfoPath in SharePoint Designer or Visual Studio, or with ASP.NET in Visual Studio. If you go with InfoPath, the Form Template is added to the feature and deployed into Forms Server. You also have to tell the workflow to use your form in the workflow's Elements.xml file by specifying the form's unique URN ID. ASP.NET forms on the other hand require much less pluming to work, just right click the workflow, and choose Add, New Item, and select an Association Form! It's easier to add and deploy with ASP.NET, but harder to work with if you're not good at HTML or ASP.NET programming.

Association and Initiation forms allow the user to provide data to the workflow before the workflow starts. Modification forms on the other hand allow the user to provide data to the workflow after it has started. This gives you the ability to alter the behavior of the workflow and provide new data into the workflow during its execution. Often, you do so when reassigning a task or approval to another individual. This modification is accomplished through two main activities in a Visual Studio workflow, the EnableWorkflowModifications activity, and the OnWorkflowModifed activity. With the EnableWorkflowModifications activity you can create a scope in the workflow where modifications are allowed and appropriate. This gives you control when users can and cannot modify the workflow. Then, if the WorkflowModified event is raised in that scope, the OnWorkflowModified activity will fire and execute your change logic.

You may at this point be wondering about task edit forms. Well if you are, check out the next chapter, Task Processing in Visual Studio Workflows. In this chapter we cover these forms in detail, along with many other notable task management related concepts that will add a lot of value into your workflows.

10

Workflows and Task Processes

In Chapter 4, you saw task processing capabilities for SharePoint Designer workflows. Well, Visual Studio also provides similar task processing capabilities for its workflows. There are task related activities for Visual Studio workflows that can create tasks, delete tasks, and complete tasks. There are also event activities that your workflows can use to react to human interaction such as if a task is edited or deleted. By putting these activities to use, you can create and assign tasks to users, and have the workflow go idle until the task is completed, for example.

Beyond these activities, Visual Studio also provides support for full customizations of the task edit forms with custom InfoPath forms. These forms can easily be integrated into your Visual Studio workflows to support complex form requirements, or if you simply want to make the form more intuitive to your end users. When the task's assignee edits the task, by default, they see the out-of-the-box task edit form. This form may not be adequate for what you're trying to do. For example, the out-of-the-box form has a lot of fields on it that you may not require them to toggle. This extra data may be confusing to users, and building a custom InfoPath form to show in place of the out-of-the-box task edit form may be preferable.

10.1 Using Task Related Activities

In addition to workflows created in SharePoint Designer, Visual Studio workflows offer a similar task processing opportunity. There are several SharePoint activities for Visual Studio that help manage tasks in workflows. You can use these activities in your Visual Studio workflows to create tasks, delete tasks, update tasks, etc. Table 10.1 shows the complete listing of task related activities that are available. While this is the complete listing, we'll cover only the most commonly used activities such as activities that create and delete tasks, as well as activities that react to events like when a task is modified or deleted.

Table 10.1 Out-of-the-box Task Related Activities for Visual Studio

CreateTask Activity	Creates a default task
CreateTaskWithContentType Activity	Create a task off a content type
DeleteTask Activity	Deletes a task

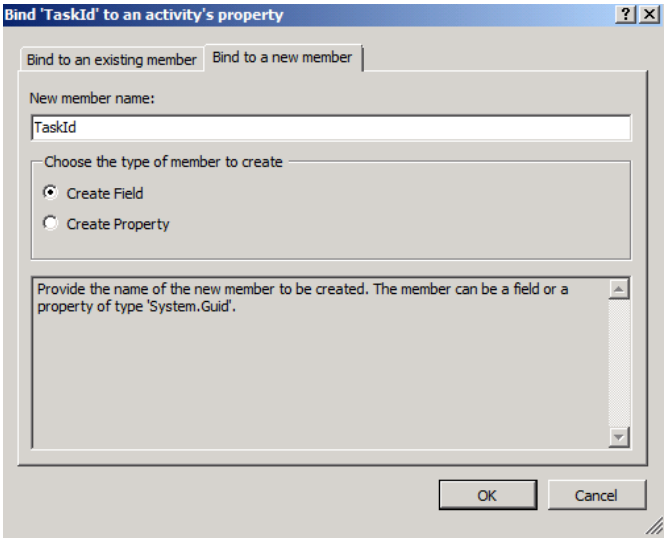
OnTaskChanged Activity	Responds to a task being changed
OnTaskCreated Activity	Responds to a task being created
OnTaskDeleted Activity	Responds to a task being deleted
RollbackTask Activity	Rolls back a task to its last accepted state
UpdateAllTasks Activity	Updates all tasks associated with the workflow that are not completed
UpdateTask Activity	Updates a task's fields via it's TaskProperties property

Now that we have a picture of all the task activities that are available to us, let's build an example that puts them into use. The example we'll go through will be the same Capital Expenditure Request workflow that we build in SharePoint Designer, but this time we'll build it in Visual Studio. This will help you see how the two workflow tools compare, when building the same example. No need to create a new requests list, we'll simply add a second workflow onto the list we used in the last section. To get started, create a new Visual Studio sequential workflow project titled "CapitalExpenditureRequests". In the project creation wizard, enter the URL to our capital expenditure requests list, and create a new List Workflow with the name "Expenditure Requests Approval - VS". Lastly, associate the new workflow to the capital expenditure requests list.

If you remember, in the SharePoint Designer requests example we had two actions that audited whether the workflow item was changed or deleted. In either case, we wanted the workflow to stop processing. To setup this same functionality in our Visual Studio workflow, we can use an EventHandlingScope activity paired with the OnTaskDeleted and OnWorkflowItemChanged activities. The EventHandlingScope activity allows you to react to events that fire inside that activities scope. We'll put the core of our workflow's business logic inside this activity, so anytime a task is deleted, for example, our EventHandlingScope activity will capture the event and execute our code. Follow these steps to configure these events:

Stopping the workflow when the Task is deleted, or the workflow item was changed

	Action	Result
1	Setup the EventHandlingScope and two EventDriven activities: A) Drop the EventHandlingScope activity onto the workflow Designer surface B) Navigate to the View Event Handlers view of the EventHandlingScope activity by clicking the activity name, and on the drop down, click the View Event Handlers link C) Drop two EventDriven activities inside the EventHandlers activity that is automatically added for you	
2	Configure the OnWorkflowItemChanged and SetState activities in the first EventDriven activity: A) Drop an OnWorkflowItemChanged activity and set the correlation token property of the activity to "workflowToken". B) Below the OnWorkflowItemChanged activity, add a SetState activity	The OnWorkflowItemChanged activity will be the first activity inside the EventDriven activity. This will tell the EventDriven

	and set the correlation token of this activity to "workflowToken" as well as rename the activity to "setCanceledState1". Note: We can use this activity to set our custom workflow states rather than the default "In Progress" and "Completed". C) Right click the setCanceledState1 activity and choose Generate Handlers. We'll fill out the code for this activity later.	activity to wait for the workflow item to change, and if it does, we'll terminate the workflow.
3	Configure the DeleteTask activity so we won't have any orphaned tasks: A) Drop the DeleteTask activity below the SetState activity, and set the token to a new token such as "TaskToken". B) After you set the token to a new token, you'll need to set the OwnerActivityName below the correlation token to Workflow1 by clicking the plus sign next to the CorrelationToken property. C) Our DeleteTask property needs to know what task to delete. We can tell it what to delete by setting the TaskId property of the activity. Set the property to a new field named TaskId by binding to a new member:  D) Assign the TaskId to a new Guid by going into the code behind and changing the TaskId field to as follows: `public Guid TaskId = Guid.NewGuid();` **Setting the TaskId field properly…** Note, when you first create the TaskId field, it will be set	The DeleteTask activity will now sit below the SetState activity. With this activity in place, our workflow will delete any orphaned tasks when the workflow terminates.

	to something like:
	`public Guid TaskId =`
	`default(System.Guid);`
	This will end up setting the TaskId to an empty guid, eg {00000000-0000-0000-0000-000000000000}. This will cause our workflow to fail because our activities need to be assigned to an actual guid.	

4	Lastly, add the Terminate activity to stop the workflow.	After you've completed these steps, your first EventDriven activity should look like Figure 10.1.

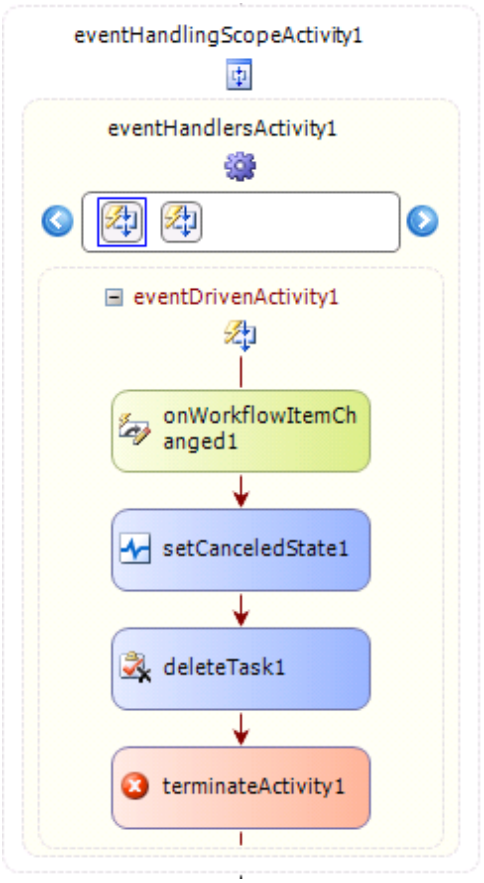


Figure 10.1 The OnWorkflowItemChanged activity will fire anytime the workflow item is changed while the workflow is waiting for request approval

Now that our workflow is listening for the request to be updated, and canceling the workflow and deleting tasks if that happens, we're ready to do the same if the task itself is deleted. If the workflow is waiting on the submission of a task, and that task is deleted, we want to ensure our workflow doesn't end up in an orphaned state. So, in the second EventDriven activity, add the OnTaskDeleted activity as the first activity.

TaskId Property of the OnTaskDeleted activity

Make sure that the TaskId property of the OnTaskDeleted activity is set to the same field as the DeleteTask activity was set to earlier. All these activities in this capital expenditure request workflow are all working off the same activity and their TaskId properties should all be the same.

Add and configure the SetState and Terminate activities as you just did for the first EventDriven activity. There's no point in adding the DeleteActivity since by this point it will have already been deleted. Afterwards, your second EventDriven activity should look like Figure 10.2.

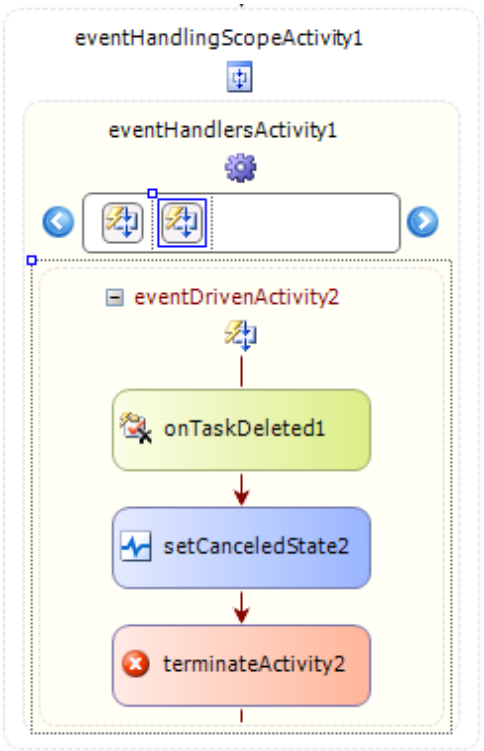


Figure 10.2 Similarly to the OnWorkflowItemChanged activity, the OnTaskDeleted activity will fire if the task is deleted before it can be approved.

Now that we have our changed and deleted events all wired up, it's time to add the main business logic into our workflow. Similarly, to what we did in the SharePoint Designer example, we want a task to

be created when the workflow is first started. After the task has been created, we want to wait for the approver to approve or reject the request. After the approver has gone in and done that, we'll then set the workflow's status to the corresponding approval. Let's start with simply the creation of the task, and waiting for its approval. Follow these steps to configure this part of the workflow:

Configure the workflow to create a new task and wait for its approval

	Action	Result
1	Drop a Sequence and CreateTask activities inside the EventHandlingScope activity: A) Navigate back to the View EventHandling Scope view of the EventHandlingScope activity. B) Drop a Sequence activity inside the EvenHandlingScope activity. C) Drop a CreateTask activity inside the sequence activity.	The main area of the EventHandlingScope activity will be configured to have a Sequence and a CreateTask activities.
2	Configure the CreateTask activity: A) Set the correlation token to the same token that was used in the DeleteTask and OnTaskDeleted activities, in this case, set it to "TaskToken" (with owner set to Workflow1). B) Set the TaskId property of the activity to the same ID that was used in the DeleteTask and OnTaskDeleted activities. C) Bind the TaskProperties property to a new member called TaskProperties. We can use this field to set values of the task like Title, Assigned To, Due Date, etc. D) Right click the CreateTask activity and choose Generate Handlers. Inside the createTask1_MethodInvoking method, enter the following code to set a few of the task's properties before it's created (Listing 10.1)	The create task activity will now be configured to create the same task that the DeleteTask activity will delete if the workflow item is changed.

Listing 10.1 createTask1_MethodInvoking

```
private void createTask1_MethodInvoking(object sender, EventArgs e)
{
   TaskProperties.Title =
     onWorkflowActivated1.WorkflowProperties.Item.Title;
   TaskProperties.Description =
     onWorkflowActivated1.WorkflowProperties.
     Item["RequestDescription"].ToString();
  TaskProperties.AssignedTo = "philsdevdomain\\pwicklund";
}
```

	Action	Result
3	Add a While activity below the CreateTask activity, making the workflow to keep looping until the task is completed: A) Below the CreateTask activity, drop in a While activity. B) Change the Condition property of this activity to Code Condition.	The While activity will be placed below the CreateTask activity. It will be configured to keep looping until the

C) Click the plus sign next to the Condition property, and enter a condition method name of WhileTaskNotCompleted. This will kick you over to the code view of the workflow. D) Enter the following code into the WhileTaskNotComplete method (Listing 10.2).	taskComplete field is set to true.

Listing 10.2 WhileTaskNotComplete

```
bool taskComplete = false;
private void WhileTaskNotComplete(object sender, ConditionalEventArgs e)
{
  if (taskComplete)
    e.Result = false; // similar to while(0)
  else
    e.Result = true;  // similar to while(1)
}
```

	Action	Result
4	Add a OnTaskChanged activity inside the While activity. A) Back on the designer view of the workflow, drop the OnTaskChanged activity inside the while activity. Essentially, this activity will execute each time our task is updated, and the while loop will check if the task has been completed or not. If it has completed, the while loop will finish looping B) Set the correlation token to "TaskToken", and set the TaskId property to the TaskId field that has been previously created. C) Similar to how you bound the TaskId property, bind the BeforeProperties and AfterProperties properties to new members called BeforePropertiers and AfterProperties respectively. D) Right click the OnTaskChanged activity, and click Generate Handlers. E) In the code behind of this activity we need to set the taskComplete boolean that was reference in step 3D. Enter the following code into the onTaskChanged1_Invoked method (Listing 10.3).	The OnTaskChanged activity will be inside the While activity. It will be configured to wait for the task that was created in step 3 to be updated. When it is updated, it checks to see if the Status column is set to something other than Not Started.

Listing 10.3 onTaskChanged1_Invoked

```
private Guid statusColumnId =
  new Guid("{c15b34c3-ce7d-490a-b133-3f4de8801b76}");
private string status = "";
private void onTaskChanged1_Invoked(object sender, ExternalDataEventArgs e)
{
  status = AfterProperties.ExtendedProperties[statusColumnId].ToString();
  if (status == "Not Started")
    taskComplete = false;
```

```
    else
      taskComplete = true;
  }
```

How the ExtendedProperties property works…

The TaskProperties property contains a lot of the task's common fields such as Title, DueDate, and Description. For a few default columns like Status, and any custom columns, ExtendedProperties must be used. ExtendedProperties is a hash that contains the rest of the default columns such as Status, as well as any custom columns you may have added to the list yourself, or through a content type. The interesting thing about this is the default columns are entered in the hash through a unique id, their guid, whereas custom columns are entered with their column name. Figure 10.3 shows how to get the field Id with PowerShell if you want to retrieve a default column out of the extended properties hash.

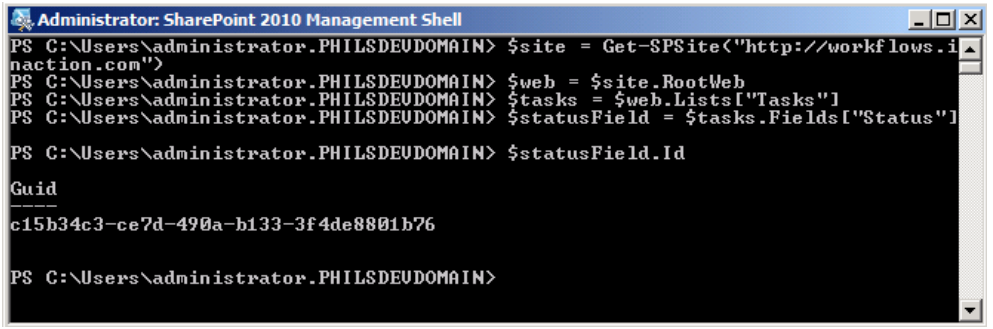


Figure 10.3 The default ExtendedProperties can be retrieved out of the has via their unique ID. You can use PowerShell to get this ID.

With these steps in place, our while activity will now stop looping when the Status column is changed to something other than "Not Started". Now what we want to do is add an IfElse activity to determine what to set the workflow's status to. If the status in the task is set to Approved, set the workflow's status to Approved. If the status in the task is set to Deferred, set the workflow's status to deferred, etc. Note: By default, the Approved and Rejected statuses are not in the status column. In the tasks list, edit the Status column and enter the statuses shown in Figure 10.4.

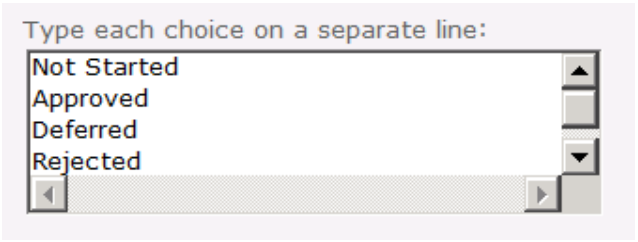


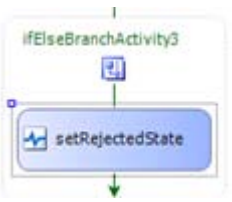
Figure 10.4 By default, a task doesn't have our statuses in the Status column and you'll need to modify the column to include our statuses.

Configure an If-Else activity to determine whether the status is Approved or Deferred

	Action	Result
1	Add an IfElse activity with three branches: A) Drop the IfElse activity below the EventHandlingScope activity. B) Right click the IfElse activity and click Add Branch to add a third branch into the activity.	A new IfElse activity will be below the EventHandlingScope activity.

2	Add a code condition to the first branch to determine if the status was Approved. If it was approved, set the state of the workflow to be Approved:
	
	A) For the first branch, edit the Condition property and specify Code Condition.
	
	B) Click the plus sign and type the condition method as "IsApproved".
	
	C) In the IsApproved method that was just generated for you, enter the following code:
	`If (status == "Approved")`
	`  e.Result = true;`
	`else`
	`  e.Result = false;`
	
	D) Add a SetState activity into the first branch and rename it to setApprovedState.
		The first branch in the IfElse activity will check to see if the status is set to Approved. If so, the first branch will execute, otherwise it will move on to the second branch.

3	Add a code condition to the second branch to determine if the status was Deferred. If it was deferred, set the state of the workflow to be Deferred:
	
	A) For the second branch, edit the Condition property and specify Code Condition.
	
	B) Click the plus sign and type the condition method as "IsDeferred".
	
	C) In the IsDeferred method that was just generated for you, enter the following code:
	`If (status == "Deferred")`
	`  e.Result = true;`
	`else`
	`  e.Result = false;`
	
	D) Add a SetState activity into the second branch and rename it to setDeferredState.
		The second branch in the IfElse activity will check to see if the status is set to Deferred. If so, the second branch will execute, otherwise it will move on to the third branch.

4	Add a SetState activity into the third branch and rename it to setRejectedState. Note: If the first branch returns false and the second branch returns false, the activities in the third branch will execute	
5	Give each of the three SetState activities a correlation token of workflowToken, and right click each and choose Generate Handlers.	Afterwards, the red exclamation point on each

	of the Set State activities will go away and each will have a method that can be used to set the state.

At this point, our workflow is nearly complete. All we have left to do is fill out the handlers for the 5 SetState activities. Before you do that, confirm that your workflow designer surface looks something like Figure 10.5.

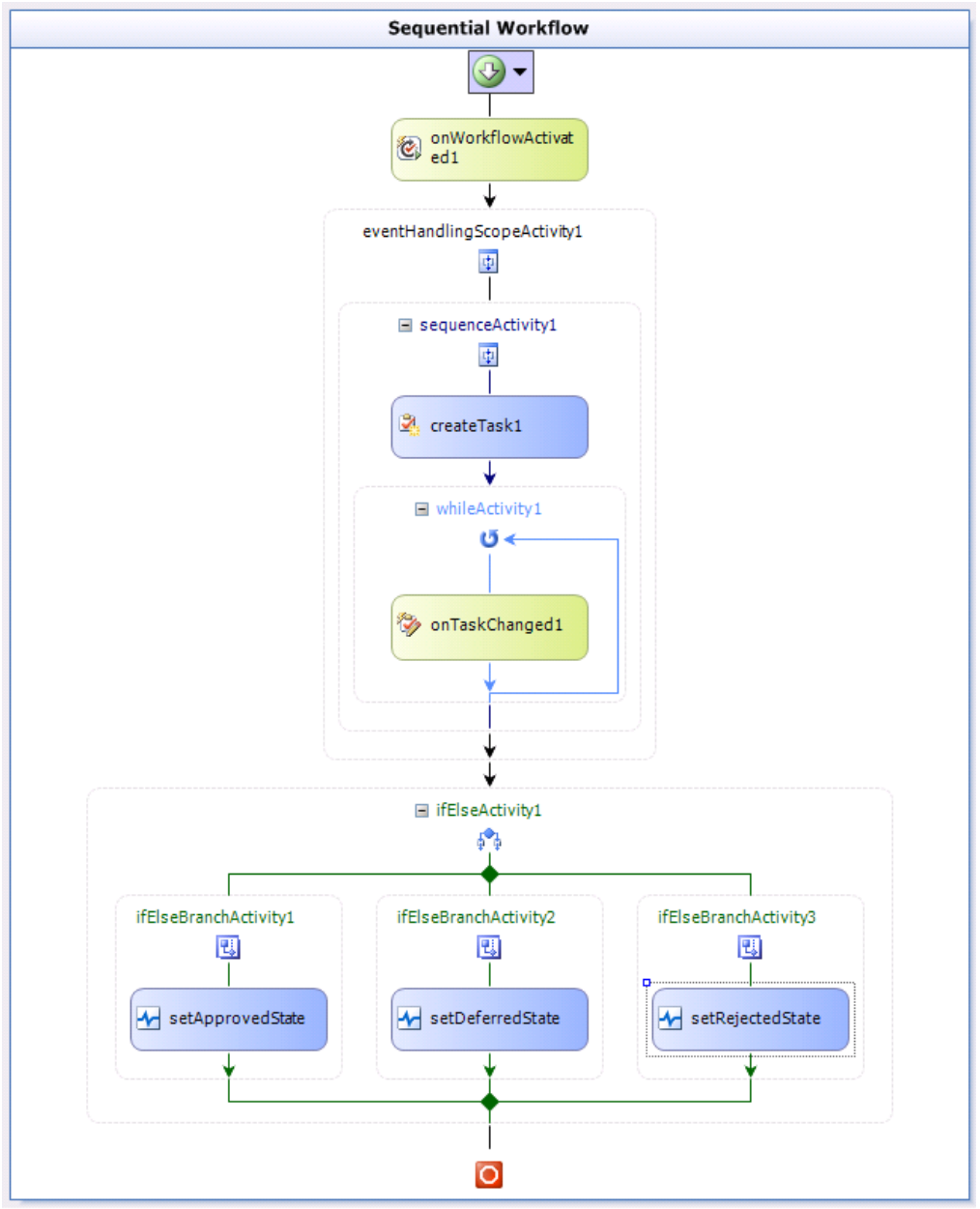


Figure 10.5 Your workflow designer surface should look like this after you have all the activities configured.

Now to be more consistent with the SharePoint Designer workflow we created in the last section, all we would need to do is add a few SendEmail activities to send out the notifications. That activity is covered in chapter 7 where we introduce Visual Studio workflows, so we'll skip is here for brevity's sake.

The first step to setting up our custom workflow statuses is to define those statuses in the workflow's Elements.xml file. Any custom statuses need to be defined in this XML file, inside the MetaData element. The statuses live inside child element called ExtnededStatusColumnValues, with each status inside a StatusColumnValue tag. Open the Workflow1's Elements.xml file and define your extended status values similar to Figure 10.6.

```
<MetaData>
  <AssociationCategories>List</AssociationCategories>
  <StatusPageUrl>_layouts/WrkStat.aspx</StatusPageUrl>
  <ExtendedStatusColumnValues>
    <StatusColumnValue>Approved</StatusColumnValue>
    <StatusColumnValue>Rejected</StatusColumnValue>
    <StatusColumnValue>Deferred</StatusColumnValue>
    <StatusColumnValue>Canceled</StatusColumnValue>
  </ExtendedStatusColumnValues>
</MetaData>
```

Figure 10.6 Custom workflow statuses first need to be defined in the workflow's Elements.xml file.

Each SetState activity has a property called State. This corresponds to a integer value that determines what state to set the workflow's status to. There are 15 states by default, so you can set this property to a number from 1 to 15, and you'll set your workflow's status to one of those out-of-the-box states. The table below shows the object model's current values for these states:

Table 10.1 There are 10 workflow states defined in the SPWorkflowStatus enum, with a Max value set to 15.

State (SPWorkflowStatus enum)	Value
Completed	5
ErrorOccurred	3
ErrorOccurredRetrying	7
FailedOnStart	1
FailedOnStartRetrying	6
InProgress	2
Max	15
NotStarted	0
StoppedByUser	4
ViewQueryOverflow	8

To set the workflow to one of our custom states, we need to set it to something 16 or higher. For example, if we set it to 16, we'd set the workflow's status to "Approved". If we set it to 19, the workflow's status would be "Canceled". A better way to do this is through code, rather than hard coding in a value in the State property. Through code, we can use the SPWorkflowStatus.Max property, which will protect us if Microsoft ever changes the state max from 15, to say 16, which would through our status values off and may even break our workflow. Configure your SetState handlers similarly to Listing 10.4 to do this.

Listing 10.4 Handlers for the SetState activities

```
enum CustomStates                                                    1
{
    Approved = 0,
    Rejected,
    Deferred,
    Canceled
};

private void setCanceledState_MethodInvoking(object sender, EventArgs e)
{
    ((SetState)sender).State = (Int32)SPWorkflowStatus.Max +          2
        (Int32)CustomStates.Canceled;
}

private void setApprovedState_MethodInvoking(object sender, EventArgs e)
{
    ((SetState)sender).State = (Int32)SPWorkflowStatus.Max +
        (Int32)CustomStates.Approved;
}

private void setDeferredState_MethodInvoking(object sender, EventArgs e)
{
    ((SetState)sender).State = (Int32)SPWorkflowStatus.Max +
        (Int32)CustomStates.Deferred;
}

private void setRejectedState_MethodInvoking(object sender, EventArgs e)
{
    ((SetState)sender).State = (Int32)SPWorkflowStatus.Max +
        (Int32)CustomStates.Rejected;
}
```

1) Refactor status integer into enum
2) Add Max value to custom status

As you can see, we're using an enum **#1** to set our custom state integers. Then in each handler, we're taking the max value and adding it to our enum's integer value **#2**, thus calculating the state to set the workflow's status to.

After you have this code entered, build your project, and deploy into SharePoint. Next, create a new expenditure request, and run this new workflow on that request. You should notice the workflow sitting in the "In Progress" state, and a new task created in the tasks list. Edit the Status column on that task by changing it to a state other than Not Started. After which, back on the Capital Expenditure Requests list, the Visual Studio workflow's status column should correspond to the status that was selected in the task (Figure 10.42).

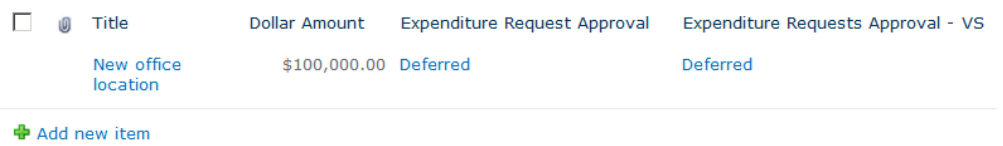


Figure 10.7 After the task was deferred, the workflow's status column was updated just like how the SharePoint Designer workflow did it.

10.3.2 Custom Task Edit Forms

The core of our Capital Expenditure Request workflow is complete, but one problem still exists. The task edit form the approver uses is not very intuitive. The problem is all they're supposed to edit is the status column, there's nine columns showing on that form. It could be very easy for the approver to lose track of what it is they're supposed to do.

> **Forms Server Dependency**
>
> Note: This section requires the SharePoint Server edition of SharePoint. SharePoint Foundation does not allow for the deployment of InfoPath forms into Forms Server.

The solution involves creating a custom InfoPath task edit form. This custom form will have the request's title and description, and three buttons. A button to approve, defer, and reject the form, and that's it! This custom InfoPath form (Figure 10.8) is more intuitive for handling expenditure requests.

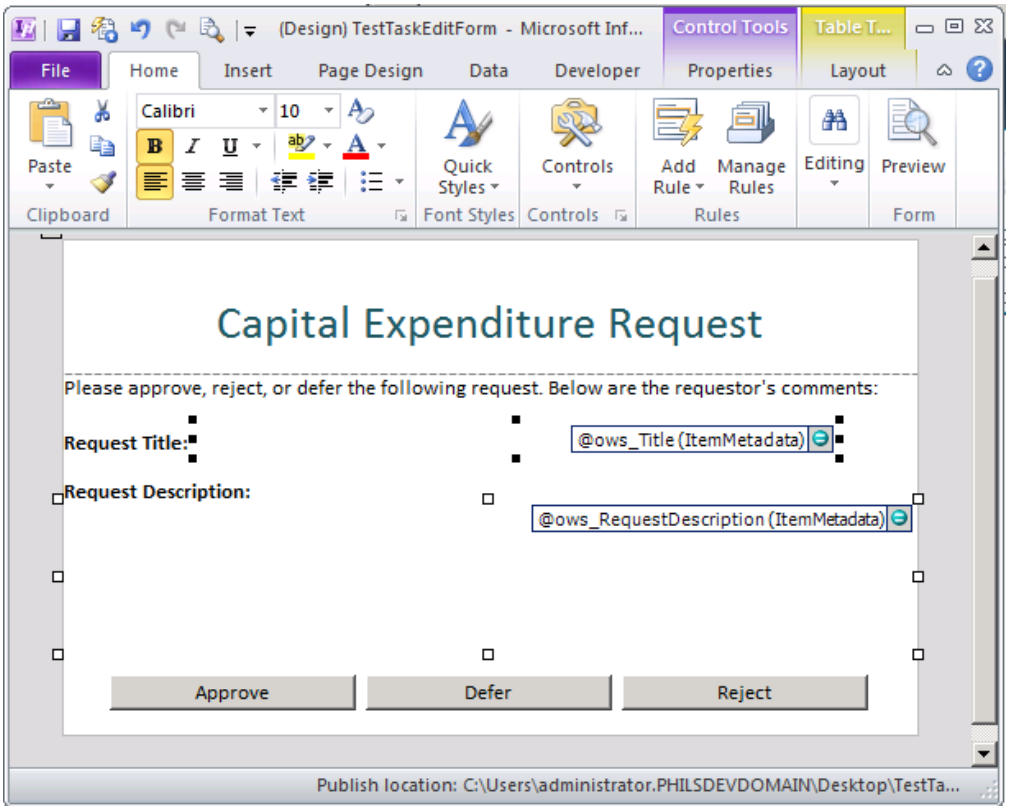


Figure 10.8 Custom InfoPath task edit forms are useful when you need a very intuitive and customized form for your workflow.

Now, at this point you may be thinking why not simply customize the out-of-the-box form as was described in chapter 8? While this is an option, the trouble with it is most workflows are reusable, in that they can be deployed across larger scopes than simply one list. Rather than customize the out-of-box task edit form time and time again, we'll build it once, and our workflow will deploy that custom InfoPath form on our behalf.

Before jumping into the Visual Studio project, there are a few special things you need to do to make this InfoPath form work. Notice in Figure 10.8 there are two main requirements of the form, firstly to show the capital expenditure request's title and description to the approver, and secondly, to show three buttons, one for each acceptable status.

Since at design time, InfoPath does not know what task list we are associating the form with, we can't added SharePoint data directly to the form. SharePoint automatically sends the list item data to the InfoPath form on form load. So, all we need to do is tell the InfoPath form what data to grab. To do this, we need to setup a secondary data source pointing to a XML file titled ItemsMetadata.xml, which references the schema that SharePoint is passing. Follow these steps to set this up:

Configure a secondary data source pointing to our task data

	Action	Result

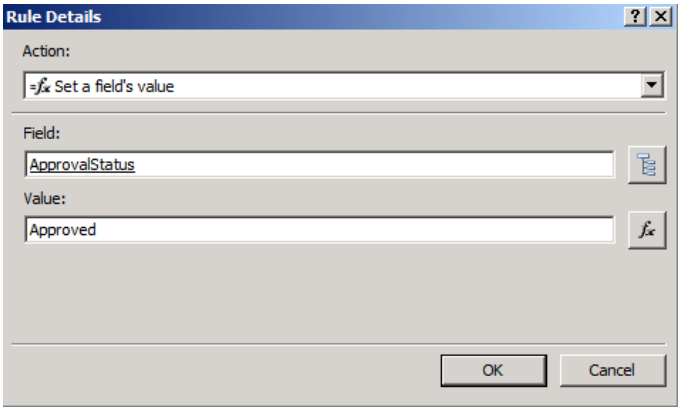
1	Create an ItemsMetadata.xml file which references the schema that SharePoint is passing:
	A) Open Notepad, and enter the following XML:
	`<z:row xmlns:z="#RowsetSchema"`
	`ows_Title=""`
	`ows_Body="" />`
	B) Save the file as ItemMetadata.xml.
	Note: The filename and XML therein is case sensitive!
		A new file will be created called ItemsMetadata.xml. This file will contain the XML schema for our task list data.

2	Configure the secondary data source for the task edit form in InfoPath: A) In the task edit form in InfoPath designer, in the Ribbon under the Data tab, click Data Connections. B) Click Add to add a new data connection and leave “create a new data connection to receive data”, and then click Next. C) Choose XML Document, and select the ItemMetadata.xml file on the file system where you saved it, and then click next. D) Select to include the file as a resource, click Next, and then click Finish.	Our task edit form will now be configured with a secondary data source to our SharePoint data via the ItemsMetadata.xml file.

Now with our secondary data source created, we can add controls to the form that read data out of this source. Add two new controls to the form, one titled Title, and the other titled Description. For the title text box, right click and choose Change Binding. Select ItemMetadata from the data source drop down, and select the ows_Title field. Do the same for the Description text box, except select the ows_Description field. With these two text boxes in place, when our form loads it will pull the title and description out of the task list item and place it in these fields. This will help the approver know whether they should approve the request or not. Lastly, we need to configure the three buttons and afterwards we'll be ready to publish. Follow these steps to configure the submit buttons:

Configure the Approve, Defer, and Reject submit buttons

	Action	Result
1	First, setup a data field to store the approval status . Our workflow will look at this field to determine if the request was approved or not. A) Under the Data tab, click Show Fields. B) In the Main data source, add a new field called ApprovalStatus and leave the data type set to text.	A new data field will be configured to store which of the three buttons was clicked.
2	Drop three button controls on the form and change the Label property of each button to	Three buttons will be

	Approve, Defer, and Reject respectively.	on the form all with unique names. (similar to Figure 10.8)
3	Configure the Approve button to set the ApprovalStatus field, submit the form, and close the form: A) Click the Approve button, and under the Home tab, click Manage rules. Click the New drop down, and choose Action. Title the action Approve. B) Under the **Run these actions** drop down, select **Set a field's value**. Select the ApprovalStatus field, and set it to Approved:  C) Under the **Run these actions** drop down again, add a second action, Submit Data, to submit the form. A dialog will appear. Click Add to add a new Data Connection and choose to submit to the hosting environment. Title the data connection Submit. D) Add a third action under the **Run these actions** drop down. This time, choose close the form action.	The approval button now will be configured to set the ApprovalStatus field, and submit and close the form.
4	Repeat step 3 for the Defer and Reject buttons, except this time set the ApprovalStatus to Deferred and Rejected respectively. Note: You also won't need to create a new Data Connection. Just use the connection created for the Approve button.	Now all three buttons will be configured to set the ApprovalStatus field, and submit and close the form.

Now we should be good to go to publish our form. Save, and publish the form to a network location. Ensure that the security level is set to Domain when you publish. Now let's go ahead and integrate this published form into our Visual Studio workflow. Note, the following steps will walk you through this process, but for a more detailed walkthrough please reference section three in chapter 9. Follow these steps to tell the workflow to use your custom task edit form, rather than the out-of-the-box one:

Configure your workflow to use your custom task edit form, rather than the default, out-of-the-box form

	Action	Result
1	Add the task edit form into the Visual Studio project: A) Back in the Capital Expenditure Requests workflow, right click the Workflow1 workflow, and choose Add, Existing Item. Select the published version of the task edit form you just got done creating. B) After the form has been loaded into the project, right click the form and choose properties, and change the Deployment Type property to be ElementFile.	The task edit InfoPath form that was created earlier will now be in the Visual Studio project.

2	Next ,edit the Feature's manifest to include an event receiver that will deploy this form into Forms Server.
	
	A) Double click the feature, and under Manifest, click Edit options. Enter the following into the Feature element as attributes:
	`ReceiverAssembly="Microsoft.Office.Workflow.Feature, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c"`
	`ReceiverClass="Microsoft.Office.Workflow.Feature.WorkflowFeatureReceiver"`
		The feature will now be set to deploy the form to Forms Server when the feature is first activated.

3	Register the form with the Feature: A) Enter the following property into the Properties element: `<Property Key="RegisterForms" Value="Workflow1\*.xsn" />`	The feature will now know what forms need to be deployed to Forms Server.
4	Now that the form is deployable to Form Server, deploy the solution and check Forms Server to ensure it worked. A) Right click the project and click Deploy. B) Navigate to SharePoint Central Administration, and find the form in Forms Server. C) View the properties on the form and note its URN.	The form's unique ID (URN) is retrieved out of Forms Server.

5	With the form deployed, tell the workflow to actually use the form.
	
	A) Open the workflow's Elements.xml file
	
	B) Enter the following as attributes in the Workflow element:
	`TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F980F93160"`
	
	C) Enter the following element under the MetaData element:
	`<Task0_FormURN></Task0_FormURN>`
	
	D) Inside the Task0_FormURN element, paste the form's URN copied from Form
		Afterwards, the workflow's Elements.xml file will look something like Figure 10.9.

	Server.	
6	Lastly, back on the workflow's designer surface, double click the CreateTask activity. In the createTask1_MethodInvoking method, add another property setting to tell the task which form to use: `TaskProperties.TaskType = 0;`	The workflow will now know which form to render when the user goes to edit the task.

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Workflow
     Name="Expenditure Requests Approval - VS"
     Description="My SharePoint Workflow"
     Id="65bdb960-e5b4-4e07-b7e5-8ac2ad5205de"
     CodeBesideClass="CapitalExpenditureRequests.Workflow1.Workflow1"
     CodeBesideAssembly="$assemblyname$"
     TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F980F93160">
    <Categories/>
    <MetaData>
      <Task0_FormURN>
        urn:schemas-microsoft-com:office:infopath:CapitalRequestEditForm:-m
      </Task0_FormURN>
      <AssociationCategories>List</AssociationCategories>
      <StatusPageUrl>_layouts/WrkStat.aspx</StatusPageUrl>
      <ExtendedStatusColumnValues>
        <StatusColumnValue>Approved</StatusColumnValue>
        <StatusColumnValue>Rejected</StatusColumnValue>
        <StatusColumnValue>Deferred</StatusColumnValue>
        <StatusColumnValue>Canceled</StatusColumnValue>
      </ExtendedStatusColumnValues>
    </MetaData>
  </Workflow>
</Elements>
```

Figure 10.9 The Elements.xml file is updated with two main things, the TaskListContentTypeId and the task's form's URN.

TaskProperties.TaskType

The TaskType property lets you create multiple "types" of tasks, and you can associate each type of task with a different form. Notice the zero in the `Task0_FormURN` element in the workflow's Elements.xml file? The zero in the element name corresponded to the TaskType property of the task. The TaskType property was set to zero in step 10, informing the workflow to use form zero when the task is edited.

This would be helpful if we were creating two tasks, one for a regular approver, and one for the CEO. The CEO should see a different form when they go in to approve the request than the regular approver would see. In this case, the regular approver could get a task type of zero, the CEO could get a task type of one, and both could have their own FromURN in the elements file.

Before we can deploy we need to do one last thing, which is to update our OnTaskChanged activity. If you remember, that activity is currently looking at the Status column on the task. Now with our custom task edit form we are no longer going to be looking at that status column, but rather want to look at the ApprovalStatus field in the InfoPath form's data when the form is submitted. Go into the event handler of the OnTaskChanged activity and enter code similar to Listing 10.5.

Listing 10.5 onTaskChanged1_Invoked

```
private string status = "";
private void onTaskChanged1_Invoked(object sender, ExternalDataEventArgs e)
{
  status = ExtendedProperties["ApprovalStatus"].ToString();
}
```

Another important thing to note is with a custom task edit form, our task cannot be modified if it is not submitted. What this means is we do not need the while activity anymore. Whereas before, someone could easily edit the task's Title column, for example, and never change the Status column which our workflow was dependent upon. Feel free to move the OnTaskChanged activity out of the while activity, and directly after the CreateTask activity. Afterwards you can delete the while activity all together.

We're finally ready to test. Go ahead an build and deploy the Visual Studio solution. Thereafter, create a new Capital Expenditure Request, and start the workflow on that request. Click the task that is created, and you should see our custom InfoPath task edit form (Figure 10.10). Within the form, you'll notice the title and description of the expenditure request, and after you click one of the three approval buttons, the workflow will complete and its status will be updated.

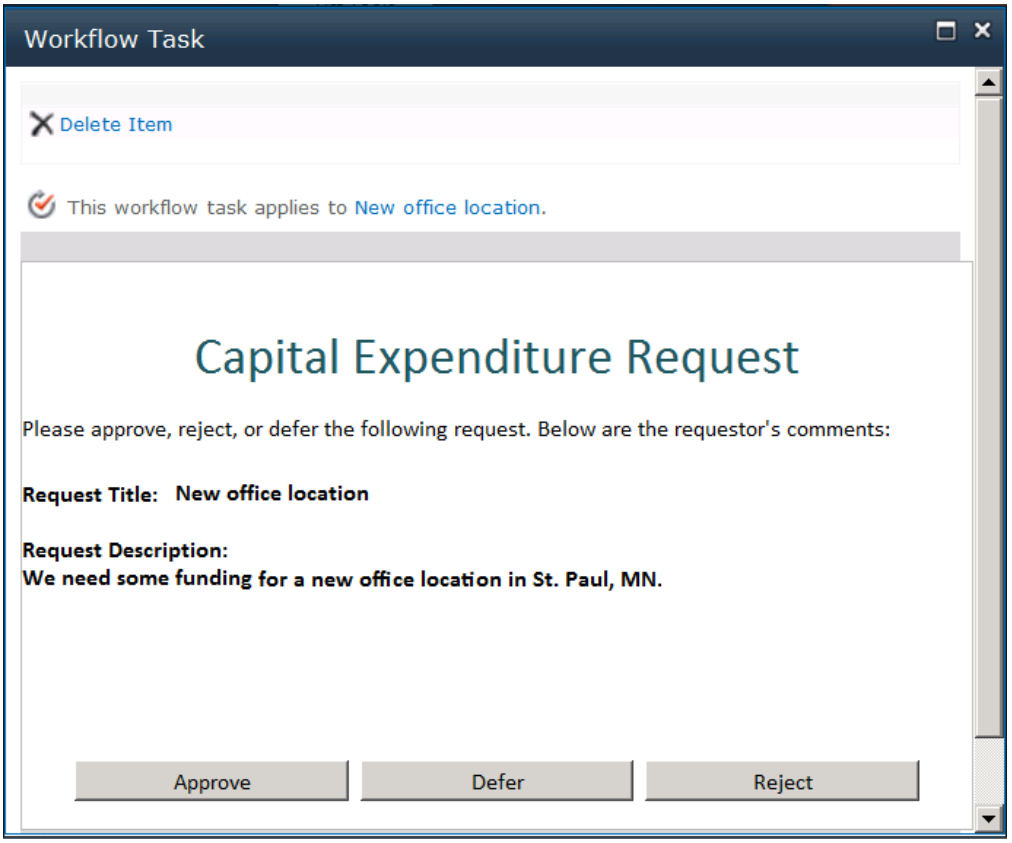


Figure 10.10 When it's all said and done, our new custom task edit form has a much slimmed down, and easier interface for our end users.

10.4 Summary

When building Visual Studio workflows that need to perform task processing, you'll find a host of task related activities you can use. Activities such as the CreateTask and OnTaskChanged activities allow you to create a new task and wait for that task to be completed. Other tasks such as the DeleteTask and OnTaskDeleted are commonly used as well.

What's also great about Visual Studio workflows over SharePoint Designer workflows is the task edit forms can be more complex. In SharePoint Designer, all you could do is add or remove fields from the task edit form. In Visual Studio, you can design an InfoPath form from scratch that your workflow can use to meet even the most complicated of form and task management requirements. These InfoPath task edit forms can be integrated and deployed through your Visual Studio project.

11

Custom Workflow Activities and Conditions

Activities in the most basic sense are simply object oriented classes. In object oriented programming, you would never have one class that does everything. Instead you would have many classes that when all put together make up the application. This analogy is true for workflows and activities as well. You would never want a large, complex Visual Studio workflow to have, for example, ten thousand lines of code in its code view. It's a better practice to extract out discrete pieces of code into custom activities that are managed outside the workflow's template. The workflow template should simply be used to model the workflow's activities, and pictorially show the "flow of work". The vast majority of the custom code should be managed outside the template in custom activities.

By using custom activities you will greatly improve the readability and maintainability of your workflows. From a readability perspective, consider the "code" activity. If you remember, a code activity is simply used to add code into a workflow template. It is a very often an abused activity. For example, if you have a code activity with 500 lines of code in it, you're defeating the value of the workflow template. You can't see pictorially what that activity is doing. If you took the logic out of that activity and extracted it into several custom activities, you'll not only gain the readability benefits by seeing those activities modeled on the template, but the code will also be a lot more maintainable.

There are two basic types of custom activities you can build: leaf activities and composite activities. While composite activities are akin to "branches" with many "leaves" (child activities), leaf activities only contain themselves and their own code. A composite activity contains within itself many leaf activities, and is in some sense a mini workflow within the parent workflow template. Examples of leaf activities include Delay, CreateTask, and SetState. Examples of composites include While, IfElse, and Sequence.

Activities can be packaged up and published into SharePoint Designer, as well as used in Visual Studio workflows. Deploying to SharePoint Designer involves creating an ACTIONS file and deploying it to the file system. SharePoint Designer will download that file when it remotely connects to your SharePoint site, and the file will tell SharePoint Designer what custom actions to load and make available to the user.

Custom conditions can also be published into SharePoint Designer. Conditions used to determine if a block of actions should execute or not. To create a condition you simply need a .NET class and a method that returns a Boolean. You then point to your method in the same ACTIONS file, and SharePoint Designer will treat the method as a custom condition for use in SharePoint Designer workflows.

11.1 Building Custom Leaf Activities

Building custom leaf activities is relatively easy. The basic procedure is to create a class and extend the Activity base class. Then, simply override the Execute method and start coding! To add your activity to the workflow, simply build the project and the activity will appear in the toolbox for use. Beyond this there are a few other techniques you can incorporate to make your custom activity a bit more robust. This includes creating custom properties that will be loaded in the properties interface. With these properties you can also write custom validation and specify defaults settings. You can even "theme" your activity by changing the background color of the activity when it is dropped on the workflow template.

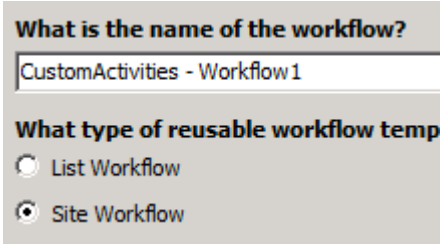
11.1.1 Custom Activity Fundamentals

Before we get into anything that's at all considered advanced, let's build a very generic custom leaf activity. To do this we're simply going to create a new project and a new class that extends the base Activity class. We're then going to override the Execute method, build, and drop our activity onto a workflow template. Follow these steps to do this:

Setting the stage for our example

This sub-section is intentionally generic, to keep things simple. You're going to create an activity named CreateSubSites that you'll extend to provision sub sites form within a workflow, in later sections. This is a simple example of where a custom activity is beneficial because the creating of sub sites is something that can be highly reusable across many workflows.

Building a custom leaf activity and using it in a workflow

	Action	Result
1	Create a new workflow project: A) Create a new Sequential Workflow project. B) Name the project *CustomActivities* and associate the workflow with as a Site Workflow. 	A new Visual Studio project will be created and bound to a SharePoint site.

2	Create a new class and extend the Activity base class:
	
	A) Right click the project, and choose Add, Class…
	
	B) Name the class CreateSubSite
	
	C) in the class's code file, add the following using statements at the top:
	
	`using System.Workflow.ComponentModel;`
	`using Microsoft.SharePoint;`
	
	D) Make the CreateSubSite class public and extend it to the Activity base:
	
	`public class CreateSubSite: Activity`
		You project will have a new class in it that extends Activity.

3	Override the Execute method: A) Within the class, override the protected Execute method B) change the return value to be ActivityExecutionStatus.Closed	The class will have a single method, and afterward the code in its entirety will look similar to Listing 11.1.

Listing 11.1 CreateSubSite activity class file contents

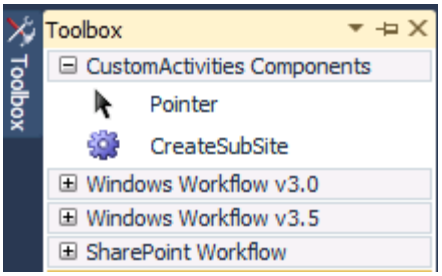
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Workflow.ComponentModel;
using Microsoft.SharePoint;

namespace CustomActivities
{
    public class CreateSubSite: Activity
    {
        protected override ActivityExecutionStatus Execute(
            ActivityExecutionContext executionContext)
        {
            return ActivityExecutionStatus.Closed;
        }
    }
}
```

Action	Result

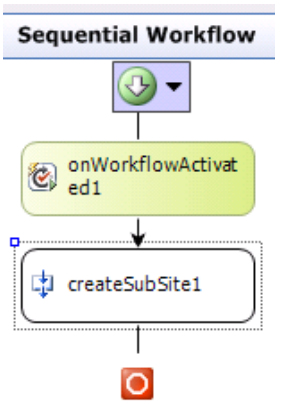
4 Drag and drop the custom activity onto Workflow1's designer surface:

A) Build the project.

B) Double click Workflow1, and notice our custom activity in the tool box:



C) Drag and drop the custom activity onto the workflow template.



Pretty easy huh? Now before we can move onto the next section, there are still a few things that need to be said. Namely, we need to discuss the return value of the Execute method, and since Execute isn't the only method you can override, we need to discuss some of the other methods that are available and what they can be used for.

The reason ActivityExecutionStatus is the return type on the Execute method is that the runtime needs to know the status of the activity after execute was called. Table 11.1 shows a list of possible statuses and when they are useful:

Table 11.1 ActivityExecutionStatus enumeration options

Canceling	Canceling will tell the runtime that something in the activity is causing it to cancel what it was trying to do.
Closed	Closed is what is used under normal conditions. This tells the runtime the activity has executed successfully and the runtime can move to the next activity.
Compensating	Compensating tells the runtime that the activity is cancelling and is trying to roll back its changes.
Executing	Executing tells the workflow that the activity is still executing. This is most commonly used in composite activities, where a parent activity still has child activities that are still executing.
Faulting	Faulting tells the run time that an exception or error has occurred while the activity was executing.
Initialized	Initialized tells the runtime that the activity has initialize but has not yet executed. This is typically not used by developers.

11.1.2 Adding Dependency Properties and Validation

By adding custom properties into your activities, you're giving the person building the workflow to specify values at design time. Additionally, with custom properties activities will be able to pass values back and forth among each other at run time. For our CreateSubSite activity we want to add properties for the SiteName, SiteUrl, SiteTemplate, and the workflow context. After our properties are added, the person building the workflow will be able to specify values in the property editor (Figure 11.1):

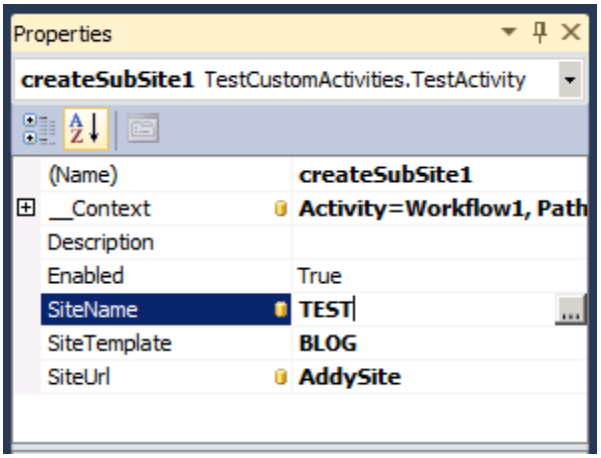


Figure 11.11 You can add properties into your activity to allow the user building the workflow to at design time set values in the property menu.

There are two types of properties we'll want to consider, the first being standard .NET properties, and the second being Dependency Properties. Standard .NET properties work nicely if all we want to do is make a property available in the property pane (Figure 11.1) so the person building the workflow can specify a value at design time. Dependency properties come into play if the value isn't known until run time. For instance, you may want to have the output of one activity pass a value into the input of another activity. That value won't be known to the receiving activity until at runtime. In this case, dependency properties are needed.

Figure 11.2 shows how if your properties are dependency properties, they will show up underneath the activity in the property binding dialog. You can then bind the property of one activity to a property of another:

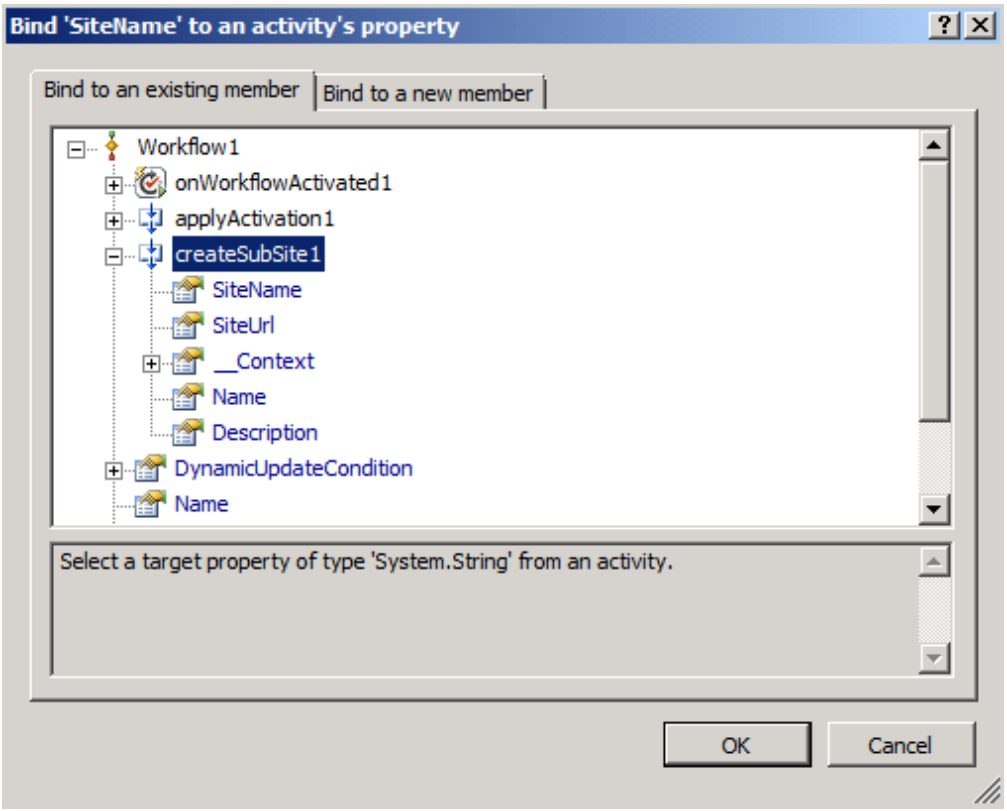


Figure 11.12 Dependency properties are needed when you won't know a value until run time. An example of this is when you want to set the value of a property to the value of a property in a separate activity. Only dependency properties show up in the bind dialog under the corresponding activity.

Dependency properties store their values in a hash controlled by the workflow run time, rather than local variables like most .NET property values are stored in. Since a .NET property could be bound to any variable anywhere, there is a dependency there that makes them less reusable. Dependency properties on the other hand are not dependent because they always reference the same hash. Thus, dependency properties are more often than not the best practice, and all the properties we build in this section will be dependency properties.

Dependency properties are coded very similarly to standard .NET properties, with a few small exceptions. Notice a sample dependency property in Listing 11.2:

Listing 11.2 Sample Dependency property

```
public static DependencyProperty SiteUrlProperty =                          1
   System.Workflow.ComponentModel.DependencyProperty.
   Register("SiteUrl", typeof(string), typeof(TestActivity));

[Description("Enter a relative URL for the new site")]
[Browsable(true)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
```

```
public string SiteUrl                                                    2
{
   get                                                                   3
   {
      return ((string)(base.GetValue(TestActivity.SiteUrlProperty)));
   }
   set
   {
      base.SetValue(TestActivity.SiteUrlProperty, value);
   }
}
```

1) use a standard .NET property
2) reference dep prop in get/set
3) register dep prop with runtime hash

The first thing you'll notice is our dependency property. SiteUrlProperty is "registered" with the workflow runtime's hash through the DependencyProperty's Register method **(#1)**. In addition, notice how there's a standard .NET property titled SiteUrl **(#2)**, but its getter and setter **(#3)** is referencing the SiteUrlProperty dependency property above, rather than a generic string object for example.

Now, getting back to our custom CreateSubSite activity, we want to add four custom dependency properties that will help us in the provisioning of the sub site. First we want to add two simple string properties for the site's name and URL. Thirdly we need a property so to specify what site template to use (eg., blog, wiki, team site, etc.). Lastly we need a property that will contain the workflow's context. The workflow context will have valuable information in it like the context SPSite and SPWeb objects. This will allow us to not have to hard code any URLs and our activity can be entirely generic. To add these properties, enter the code found in Listing 11.3 inside the CreateSubSite class, above the Execute method:

Note: two more using statements needed

After you enter the code in Listing 11.3, add a using statement to System.ComponentModel and Microsoft.SharePoint.WorkflowActions, so it will build successfully.

Listing 11.3 SiteName, SiteUrl, SiteTemplate, and Context dependency properties

```
public enum SiteTemplates
{
    BLOG,
    MPS,
    STS,
    WIKI
};

public static DependencyProperty SiteNameProperty =
    System.Workflow.ComponentModel.DependencyProperty.Register(
    "SiteName", typeof(string), typeof(CreateSubSite));

public static DependencyProperty SiteUrlProperty =
    System.Workflow.ComponentModel.DependencyProperty.Register(
    "SiteUrl", typeof(string), typeof(CreateSubSite));

public static DependencyProperty SiteTemplateProperty =
    System.Workflow.ComponentModel.DependencyProperty.Register
    ("SiteTemplate", typeof(SiteTemplates), typeof(CreateSubSite));
```

```
public static DependencyProperty __ContextProperty =
    System.Workflow.ComponentModel.DependencyProperty.Register(
    "__Context", typeof(WorkflowContext), typeof(CreateSubSite));

[Description("Enter a name for the new site")]
[Browsable(true)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
public string SiteName
{
    get
    {
        return ((string)(base.GetValue(CreateSubSite.SiteNameProperty)));
    }
    set
    {
        base.SetValue(CreateSubSite.SiteNameProperty, value);
    }
}


[Description("Enter a relative URL for the new site")]
[Browsable(true)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
public string SiteUrl
{
    get
    {
        return ((string)(base.GetValue(CreateSubSite.SiteUrlProperty)));
    }
    set
    {
        base.SetValue(CreateSubSite.SiteUrlProperty, value);
    }
}

[Description("Specify the site template that will be used.")]
[Browsable(true)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
public SiteTemplates SiteTemplate
{
    get
    {
        return ((SiteTemplates)(base.GetValue(
            CreateSubSite.SiteTemplateProperty)));
    }
    set
    {
        base.SetValue(CreateSubSite.SiteTemplateProperty, value);
    }
}

[Browsable(true)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
public WorkflowContext __Context
{
    get
    {
        return ((WorkflowContext)(base.GetValue(
            CreateSubSite.__ContextProperty)));
```

```
        }
        set
        {
            base.SetValue(CreateSubSite.__ContextProperty, value);
        }
    }
```

Note! Property naming rules can't be broken

The name of the context property is preceded by two underscores, "_ _Context". You may be thinking this is a bit goofy, but it is necessary because later in this chapter we're going to publish this activity to SharePoint Designer. SharePoint Designer looks for a few properties with very specific names, and _ _Context happens to be one of them. When Designer sees this property it will know to save the WorkflowContext in that property.

You may also notice how all the static dependency properties have "Property" at the end of the name. This is a requirement. The name of the dependency property must start with the property name "eg, SiteUrl" and end with "Property", eg, SiteUrlProperty.

Also notice the first parameter in the Register method. This is the key in the runtime property hash. This key must be the property name, eg "SiteUrl".

Failure to comply with these naming rules will result in runtime errors!

When you have gotten the listing entered into your activity, go ahead and build the project again. Afterwards, go back to Workflow1 and click on the CreateSubSite activity. Notice our four properties will show up in the property menu (Figure 11.3). Feel free to specify a site name, URL, and pick your favorite template.

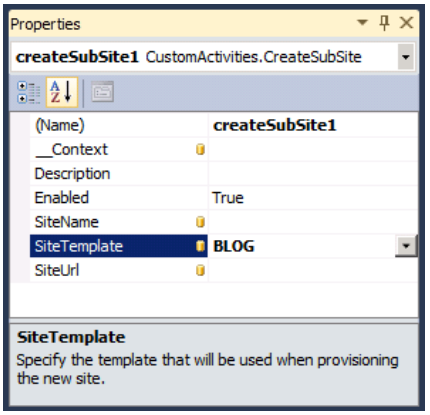
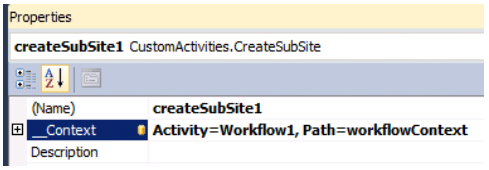


Figure 11.13 After you build, properties will show up in the property menu of the activity.

Now you may be wondering what you're supposed to do with the __Context property. This property is one of those properties that must be a dependency property because we won't know the value of the property until run time. We get the WorkflowContext from the output of the ApplyActivation activity. Follow these steps to configure the ApplyActivation activity to set our context for us:

Configure the ApplyActivation activity to set the WorkflowContext property

Action	Result
1 Configure the ApplyActivation activity: A) Drag and drop the ApplyActivation activity onto Workflow1's template. B) Bind the __Context property of the ApplyActivation activity to a new member field called "workflowContext". C) On the CreateSubSite activity, bind the __Context property to the workflowContext field.	

After completing these steps, the CreateSubSite activity will have a WorkflowContext object at runtime. If we didn't first use the ApplyActivation activity, the __Context property would have been null. You may have noticed on the ApplyActivation activity that there was another underscore property titled __WorkflowProperties. "_ _WorkflowProperties" is another one of those tokens that SharePoint Designer looks for. Actually, __Context and __WorkflowProperties have very similar objects like the SPSite and SPWeb. We could've just used the workflowProperties object that the OnWorkflowActivated activity initializes. If you ever need a WorkflowContext object, rather than a WorkflowProperties object, you now know how to configure the ApplyActivation activity to initialize that WorkflowContext property.

Before we move onto the next section, let's add some code into our Execute method. With our properties in place we can go ahead and add the code that will provision the sub site. Enter the code found in Listing 11.4 inside the Execute method:

Listing 11.4 Site provisioning code within the Execute method

```
SPSecurity.RunWithElevatedPrivileges(delegate()                          1
    {

        using (SPSite site = new SPSite(__Context.Site.ID))
        {
            using (SPWeb web = site.AllWebs[__Context.Web.ID])           2
            {
                if (web.Webs.Names.Contains(SiteUrl))                    3
                    throw new UrlAlreadyInUseException();
                else
                {
                    web.Webs.Add(                                        4
                        SiteUrl,
                        SiteName,
                        "desc.",
                        1033,
                        SiteTemplate.ToString(),
                        false,
                        false);
                }
            }
```

```
            }
        });
    return ActivityExecutionStatus.Closed;
```

1) elevate user to service account
2) get site and web from context
3) confirm URL is unique
4) provision the sub site

The first thing that needs to happen in the Execute method is the user's privileges need to be elevated to that of the service account **(#1)**. This is because we can't guarantee that the user will have the rights necessary to create sub sites in SharePoint. Thereafter we need to get the SPSite and SPWeb objects from our context **(#2)**. The SPWeb object will be the parent site to the sub site that is to be created. Before we create the sub site we want to confirm that the URL is unique **(#3)**. You can't have two sites with the same URL. It would be proper to use a FaultHandlingActivity to handle a uniquely named exception like the UrlAlreadyInUseExepection shown in this listing (Figure 11.4). With this route you can then put the burden on the workflow to know how to handle the exception. After we confirm the URL is available, the site is provisioned **(#4)**.

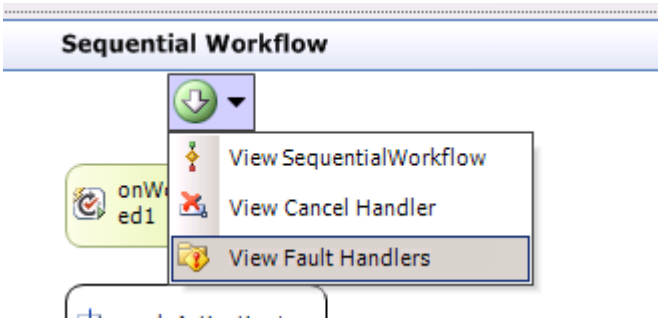


Figure 11.14 You can specify the FaultType property of a FaultHandling activity to handle a unique exception such as a UrlAlreadyInUse exception.

If you've never created a custom exception, it's really easy. Create a new class and use the following code as an example on how to create a custom exception:

```
public class UrlAlreadyInUseExeception : Exception
{
    public UrlAlreadyInUseException()
    {
    }
}
```

11.1.3 Property Validation

Wouldn't it be nice now to add some validation around our newly created properties? Take the SiteUrl property for example. The URL is a very sensitive property because if it is entered incorrectly the Add method on the Webs collection will throw and exception. An example would be if you entered special characters in the URL, or if you didn't make the URL relative, but rather put in a full URL such as

http://intranet/someurl. When you call Webs. Add the URL must be relative and must not contain any forward slashes or special characters. Let's enhance our activity with a custom validator that will raise a compile time error if the URL is given a value that does not meet our requirements.

To create a custom validator, you need to create a new class that extends ActivityValidator. The ActivityValidator class has a method called Validate that we will then need to override. It's within this method that we can put whatever code we want to determine if the property(s) were configured property. The return type on the method is ValidationErrorCollection, a collection of ValidationError objects. You can fill this collection up with ValidationErrors and Visual Studio will show those errors in the error screen when you build the project and the build fails (Figure 11.5).

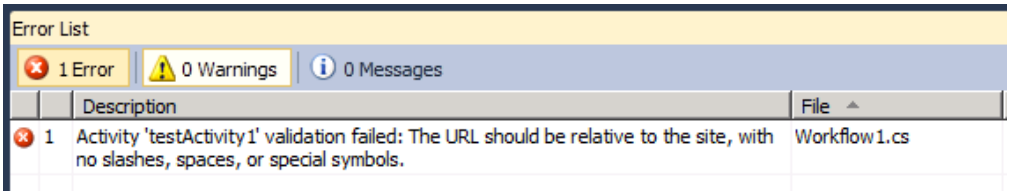


Figure 11.15 By creating a custom validator, you can render compile time errors when properties are set incorrectly.

The Validate method again is programmed as you want. If you have five properties you want to validate, you may write code that validates all five in the one Validate method. On the other hand, you could write five separate validators. Your choice. Use the following steps to create a validator for our CreateSubSite activity:

Creating a custom activity validator

	Action	Result
1	Create a new class called UrlValidator A) Right click the project, choose Add, Class… and name the class UrlValidator. B) At the top of the class add the following using statements: using System.Workflow.ComponentModel.Compiler; using System.Text.RegularExpressions; C) make the class public and extend ActivityValidator: public class UrlValidator: ActivityValidator D) Override the Validate method.	After you override the Validate method the class in its entirety will look similar to Listing 11.5.

Listing 11.5 UrlValidator class shell

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

```
using System.Workflow.ComponentModel.Compiler;
using System.Text.RegularExpressions;

namespace CustomActivities
{
    public class UrlValidator: ActivityValidator
    {
        public override ValidationErrorCollection Validate(
            ValidationManager manager, object obj)
        {
            return base.Validate(manager, obj);
        }
    }
}
```

	Action	Result
2	Enter the code found in Listing 11.7 into the Validate method.	The Validate method will contain the code found in Listing 11.6.

Listing 11.6 Validate method code

```
CreateSubSite createSubSite = obj as CreateSubSite;
ValidationErrorCollection errCol = new ValidationErrorCollection();

if (createSubSite != null)                                          1
{
    string url = createSubSite.SiteUrl;
    if (url != null && url.Trim() != string.Empty)
    {
        Regex noSpecialChars = new Regex(@"\W");                    2
        if (noSpecialChars.IsMatch(url))
        {
            errCol.Add(new ValidationError(                         3
                "The URL should be relative to the site, " +
                "with no slashes, spaces, or special symbols.", 0001));
        }
    }

    if (url == string.Empty)                                        4
    {
        errCol.Add(new ValidationError(
                "The SiteUrl property cannot be null", 0002));
    }
    if (createSubSite.SiteName == string.Empty)
    {
        errCol.Add(new ValidationError(
                "The SiteName property cannot be null", 0003));
    }
}
return errCol;
```

1) ensure activity and url are not null
2) check url for special characters
3) add error message into collection

4) confirm neither the Url nor Name is empty

The Validate code first gets a reference to the CreateSubSite activity. The properties can be pulled out of this object. Next we check to ensure that the activity isn't null **(#1)**, and if not we run a regular expression against the Url checking to see if any special characters are present **(#2)**. If any special characters are found, an error message is added to the collection **(#3)**. Next we confirm that both the SiteUrl and SiteName properties have a value **(#4)**. Since the SiteTemplate is an enumeration and the Context is built at run time, it isn't necessary to validate those properties.

	Action	Result
3	Add an ActivityValidation attribute onto the CreateSubSite class: A) Back in your CreateSubSite activity add the following attribute onto the class: `[ActivityValidator(typeof(UrlValidator))]` B) add a using statement for System.Workflow.ComponentModel.Compiler	The activity will now be instructed to run the UrlValidator Validate method when properties change and when the project is built.
4	Build the project and test the validator: A) Build the project B) On Workflow1, change the SiteUrl property to be "/blog".	You will see a red exclamation point next to the activity (Figure 11.6) and after you build you you'll receive a build error.

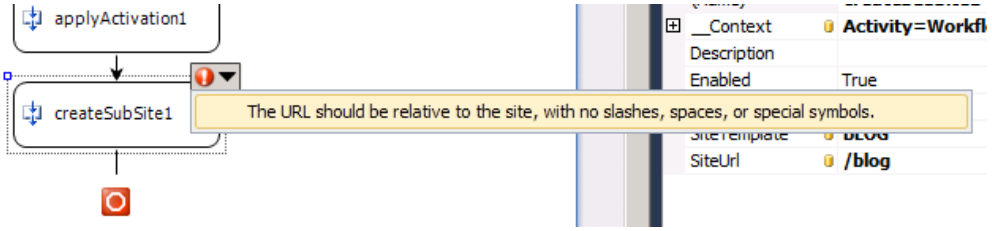


Figure 11.16 In addition to compile time errors, users will also get red exclamation points next to the activity when a property is set incorrectly.

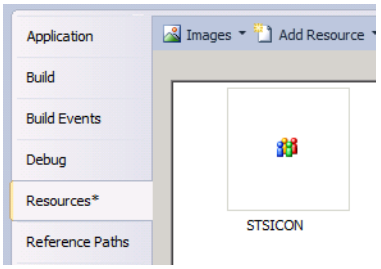
We just got done adding some valuable custom properties and property validation into our custom activity. Nevertheless, our activity is lacking quite a bit in the "cosmetic" department and we're not setting any default values to our properties. The next section will walk you through how to customize the toolbox item, as well as set default values to our parameters when the user drops the activity onto the workflow template.

11.1.4 Activity Tool Box Items

By creating a class that extends ActivityToolboxItem we can easily customize the way our activity looks in the toolbox. What's more is we can also assign default values to our custom properties when the user drop the activity onto the workflow's template. This is rather simple and only requires a few lines of code to implement.

The ActivityToolboxItem we build in this section is rather simple, but when we get to composite activities, this class becomes much more prevalent. Follow these steps to add a custom ActivityToolboxItem to your custom activity:

Add a custom ToolboxItem to the CreateSubSite activity

	Action	Result
1	All proper tool box items should have a custom icon. Add a 16x16 pixel icon as a project resource that will be referenced in code later: A) Right click the project, and choose properties and then select the Resource tab. B) Click the blue link to create a new resource file. C) Under the Add Resource menu item, select Add Existing Item and browse out and select the STSICON.GIF file under "c:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\IMAGES"	

2	Create a new class file for the custom ActivityToolboxItem:
	
	A) Right click the project and choose Add, Class… and name the class CreateSubSiteToolboxItem.
	
	B) Add the using statements to the top of the class for the following namespaces:
	
	`System.Workflow.ComponentModel.Design`
	
	`System.Runtime.Serialization;`
	
	`System.ComponentModel;`
	
	`System.ComponentModel.Design;`
	
	C) Replace the class with the code found in Listing 11.7.
		The ActivityToolboxItem class will be completed.

Listing 11.7 CreateSubSiteToolboxItem class

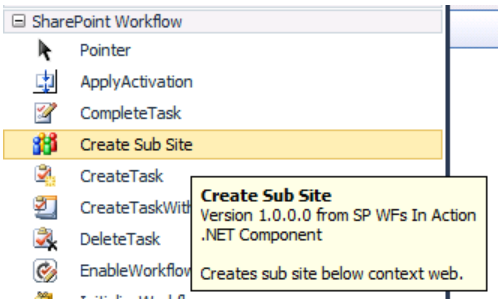
```
[Serializable]                                                          1
internal class CreateSubSiteToolboxItem : ActivityToolboxItem
{
    public CreateSubSiteToolboxItem(Type type)
        : base(type)
    {
    }
```

```
    private CreateSubSiteToolboxItem(SerializationInfo serializeinfo,
        StreamingContext context)
    {
        this.Deserialize(serializeinfo, context);                          2
        this.Description = "Creates sub site below context web.";
        this.Company = "SP WFs in Action";
        this.DisplayName = "Create Sub Site";
        this.Bitmap = new System.Drawing.Bitmap(
            CustomActivities.Properties.Resources.STSICON);
    }

    protected override IComponent[] CreateComponentsCore(          3
        IDesignerHost host)
    {
        CreateSubSite css = new CreateSubSite();
        css.SiteName = "Contoso Team Site";

        return new IComponent[]                                        4
        {
            css
        };
    }
}
```

1) Declare the class as serializeable
2) Specify toolbox meta data
3) Specify default properties
4) Return custom activity

Notice how the first thing our code does is marks the class as serializable **(#1)**. Next follows two required constructors. In the second constructor we deserialize the class and assign the item's metadata values such as description, company and an icon **(#2)**. The following overridden method allows us to specify default values to our properties **(#3)**. It's not required to override this method, but it can be helpful when you want preset values. This method becomes more important when we get into composite activity because we'll be programmatically adding child activities into the activity. Right now we're only returning the activity itself **(#4)**, with no child activities.

At this point you can build your project and test out the tool box item. The first thing you'll notice when you get to the tool box is that neither your item metadata nor icon is showing. On the plus side, if you go ahead and drop the activity onto the workflow template the SiteName property will have been set to "Contoso Team Site". So what gives? Why is the code only half working? Well, Visual Studio looks at activities it finds within the current project differently that the activities that are manually added into the toolbox. Therefore, what we need to do is browse out and add our custom activity to the toolbox manually. This is one reason why many people keep their workflows and their activities in separate projects. Follow this last step to manually add your activity to the tool box:

	Action	Result
3	Add the CreateSubSite activity manually into the tool box: A) Right click inside the toolbox and select Choose Items B) Click the Browse button and browse out to the CustomActivities.dll assembly in the Bin\Debug directory of your project. C) Click OK and take another look at the tool box!	

Now with our toolbox item showing custom metadata and an icon, it's time to spruce up the activity look and feel once it gets onto the workflow template. We can change the background color of the activity, as well as provide a custom icon to render.

11.1.5 Theming your Activity

By theming your activities you can customize the background colors, font types, font colors, and icons. Obviously it's not the most critical of techniques but it certainly adds a bit of spice and uniqueness to your home-grown activities. This is done with the combination of two classes, ActivityDesigner and ActivityDesignerTheme.

When you create a custom ActivityDesigner class you can do several things to your activities. For example you can change the size of the activity or change the placement of the text in the activity itself. A neater trick is you can add custom verbs into your activity. When you right click on the activity you get a menu with many buttons, one of which is a Properties button. Some other familiar verbs are Generate Handlers, Copy, and Disable. You can add custom verbs into this menu with an event handler attached to the verb to run custom code.

The reason ActivityDesigner is needed for themes, and not just the ActivityDesignerTheme class, is that an activity can have a "Design" attribute pointing to the ActivityDesigner. Then, the ActivityDesigner has an "ActivityDesignerTheme" attribute pointing to your custom theme. So in essence, if all you want to do is theme your activity, the ActivityDesigner class will be empty. Inside the ActivityDesignerTheme class's contructor is where you'll set all your properties such as BorderStyle, BackgroundStyle, and ForeColor. Follow these steps to configure your activity's custom theme:

Create a custom ActivityDesigner and ActivityDesignerTheme for the Create Sub Site activity

	Action	Result
1	Create a new class file for the ActivityDesigner: A) Right click the project, and choose Add, Class… and name the class CreateSubSiteActivityDesigner. B) Add using statements for the following namespaces:	A new class file will be created called CreateSubSiteActivityDesinger.cs with appropriate using statements added and a public class extended from Activity Designer.

```
System.Workflow.ComponentModel.Design
System.Drawing
```

C) Make the class public and extend the ActivityDesigner class:

```
public class
CreateSubSiteActivityDesigner:
ActivityDesigner
```

2 Create a second class either in the same file or a new file for the ActivityDesignerTheme:

A) Underneath the first class, add a second public class called CreateSubSiteActivityDesignerTheme and extends ActivityDesignerTheme..

B) Create a public constructor that takes a WorkflowTheme as a parameter and sets to color properties:

```
public CreateSubSiteActivityDesignerTheme(
    WorkflowTheme theme)
    : base(theme)
{
    this.BackColorStart = Color.Orange;
    this.BackColorEnd = Color.OrangeRed;
    this.BackgroundStyle =
        System.Drawing.Drawing2D.
        LinearGradientMode.Horizontal;
}
```

In the same cs file will be added a second public class that extends ActivityDesignerTheme.

3 Add the ActivityDesignerTheme attribute onto the CreateSubSiteActivityDesigner class:

A) Above the CreateSubSiteActivityDesigner class, add the following attribute:

```
[ActivityDesignerTheme(typeof(
    CreateSubSiteActivityDesignerTheme))]
```

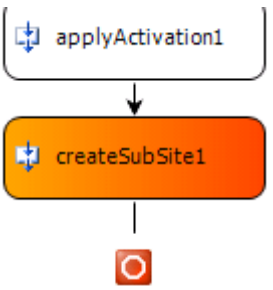
The ActivityDesignerTheme attribute will be placed on the ActivityDesigner class.

4 Add a ActivityDesigner attribute onto the custom activity and test the theme:

A) Navigate to the CreateSubSite activity and add the following attribute onto the class:

```
[Designer(typeof(
    CreateSubSiteActivityDesigner))]
```

B) Build the project and navigate back to the workflow template. Notice how the activity's background is now orange.



Cache Note: Visual Studio likes to cache the display properties of these actions. To clear the cache, close down Visual Studio, remove the DLL from the GAC (c:\windows\system32\ assembly, if necessary), and re-launch Visual Studio.

5 The final step is to update the activity's icon to be the same icon as what we configured in the toolbox in the previous section. Add a ToolboxBitmap attribute to the activity's class to configure the icon:

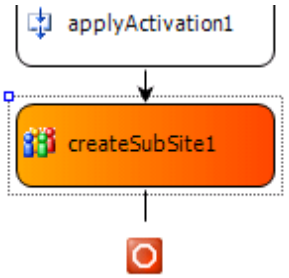
A) If your icon is a GIF file, it needs to be a PNG instead. Save the icon as a PNG and add it as a resource like we did for the ToolboxItem.

B) Change the build action of the PNG to be Embedded Resource via the properties of the PNG.

C) Add the ToolboxBitmap attribute to the CreateSubSite class, build the project, and navigate back to the workflow template to see if the icon renders:

```
[ToolboxBitmap(typeof(CreateSubSite),
   "Resources.STSICON.png")]
```

Note: when you add a file as a resource it places it in a folder called Resources. Since CreateSubSite is at the root namespace of the project, we can use our activity in the typeof. If CreateSubSite isn't in the root namespace, reference a different class that is in the root. Many people create a class at the root namespace called ResourceFinder for this purpose.



When you're done, your CreateSubSiteActivityDesign.cs file will look similar to Listing 11.8.

Listing 11.8 CreateSubSiteActivityDesigner.cs in its entirety

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Workflow.ComponentModel.Design;
using System.Drawing;

namespace CustomActivities
{
    [ActivityDesignerTheme(typeof(CreateSubSiteActivityDesignerTheme))]
```

```
    public class CreateSubSiteActivityDesigner : ActivityDesigner
    {
    }

    public class CreateSubSiteActivityDesignerTheme : ActivityDesignerTheme
    {
        public CreateSubSiteActivityDesignerTheme(WorkflowTheme theme)
            : base(theme)
        {
            this.BackColorStart = Color.Orange;
            this.BackColorEnd = Color.OrangeRed;
            this.BackgroundStyle = System.Drawing.Drawing2D.
                LinearGradientMode.Horizontal;
        }
    }
}
```

Now that we've conquered leaf activities, it's time to take things to the next level. Composite activities allow you to create activities that contain child activities. Most of the info covered in this leaf activities section is foundational to composites, and with it under our belt we're ready to tackle something more advanced.

11.2 Building Custom Composite Activities

Composite activities are most helpful when you want to create reusable groups of activities. For instance, you may have 5 or six activities all working together, and you want a parent activity that can "package up" those child activities, thus making them reusable in other workflows. This is the first and most common use for composite activities.

A second use is to create control flow activities. Activities that control the flow of work are called control flow composite activities. This includes activities that are iterative, such as the While activity, and those that are not iterative, such as the IfElse and Parallel activities. You could create a custom composite that served a similar purpose as a "For Loop" (rather than "while loop"). The subject of creating custom control flow activities is a bit beyond the scope of this book. A book focused solely on Windows Workflow Foundation techniques may be a good option for researching these more advanced techniques. Rather, let's focus on composites that are more common, such as parent/child composites for reuse purposes.

Composite activities extend the CompositeActivity class. The While, IfElse, Sequence, and Parallel activities all extend CompositeActivity. When you want to create a custom composite, you have the option of extending CompositeActivity directly, or to extend a composite activity such as the Sequence activity to serve as your base. The latter is the most common approach.

As Figure 11.7 shows, when you extend a Sequential activity, you're left with an activity that looks identical to the Sequence activity when on the activity's designer surface. It then becomes a simple process of dragging and dropping your "child" activities inside your custom sequential activity, and thereafter building and re-using that activity in as many workflows as you wish.

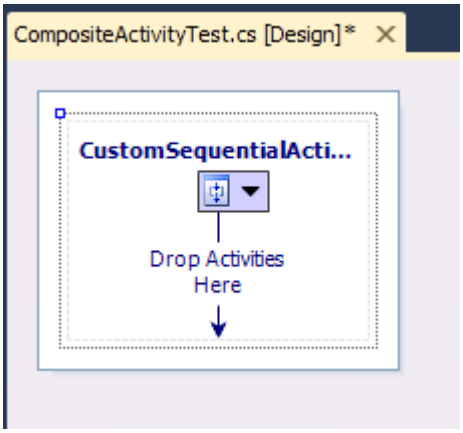


Figure 11.17 You can create a composite activity that extends another composite activity. As is in this case, you can extend the Sequence activity to give quick access to being able to drop child activities in that execute sequence.

Dependency properties also play a role in composite activities. Notice how in Figure 11.8 all child activities inside the custom sequential composite activity have padlock icons next to them. This is informing you that you cannot edit those activities. Rather, you must edit the composite activity that is packaging up those child activities. In regards to properties, what if those locked child activities have properties that need to be set? What you can do is "promote" those properties to the parent composite activity, and they will then show up in the composite's property window. You can only promote dependency properties, not the standard .NET properties.

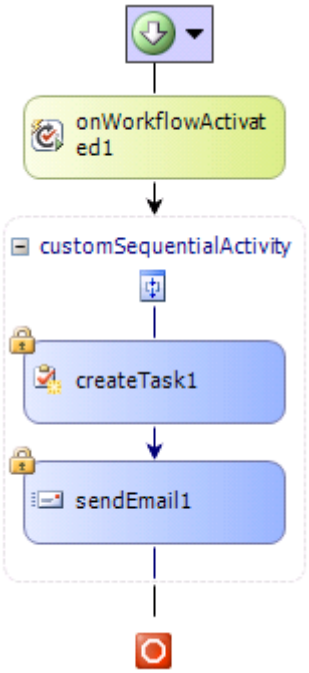


Figure 11.18 When you drop a composite activity onto a workflow template, the child activities will be locked and you won't be able to edit them or their properties.

With the introduction out of the way, let's go ahead and build a custom composite activity. Our composite activity is going to be wrapper for the CreateSubSite activity we built in the previous section. In addition to creating a sub site, it will also email the site collection administrator notifying them that the site was provisioned. Follow these steps to create this composite activity:

Creating a custom composite activity that contains child activities

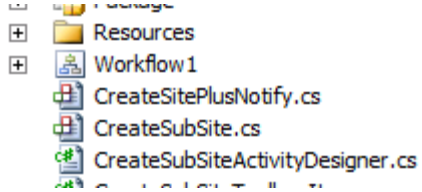
	Action	Result

1	Within the CustomActivities project, create a new class that extends Sequence activity:
	
	A) Right click the project, and select New, Class… and give the class a name of CreateSitePlusNotify
	
	B) Add a using statement each of the following namespaces:
	`System.Workflow.Activities`
	`System.Workflow.ComponentModel`
	`System.ComponentModel`
	
	C) Make the class public and have it extend the
		

SequenceActivity base:

```
Public class CreateSitePlusNotify:
    SequenceActivity
```

D) Build the project and the icon of the activity file should change, and after it does, double click the activity to open the activity's template:

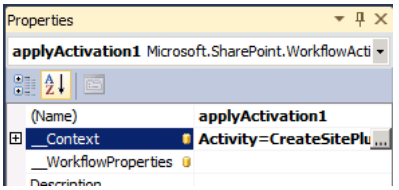


Note: You may have to close and reopen Visual Studio to get the icon to update.

2 Configure the ApplyActivation activity:

A) Drag and drop the ApplyActivation activity onto CreateSitePlusNotify's template.

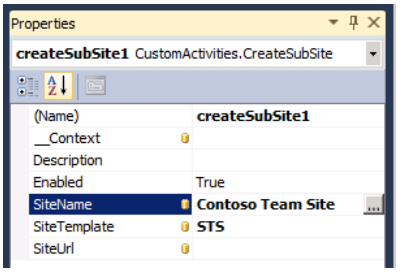
B) Bind the __Context property of the ApplyActivation activity to a new member field called "workflowContext".



3 Configure the CreateSubSite activity:

A) Below the ApplyActivation activity, drop the CreateSubSite activity.

B) Provide a SiteName and SiteTemplate for the activity

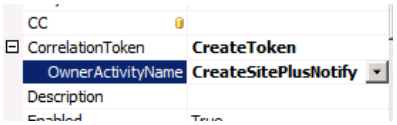


4 Bind the SiteUrl to a new Dependency Property:

A) Click on the "…" ellipsis inside the SiteUrl property.

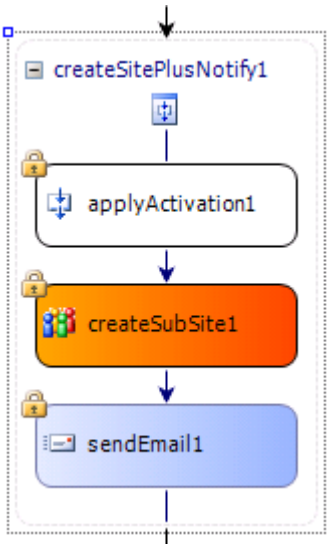
B) Bind the property to a new member (property, NOT field) named "SiteUrl".

Note: by binding to a new property, this will "promote" the property and make it available on the parent's property editor.

5	Configure the SendEmail activity: A) Drop a SendEmail activity under the CreateSubSite property. B) Edit the correlation token property of the activity to have a token named something similar to "CreateToken" and under the plus symbol next to the correlation token properly, specify the owner as CreateSubSitePlusNotify:  C) Enter some text into the Body and Subject properties, and promote the To property to a new dependency property named SCAdminEmail, similar to how you did it in step 4.	A SendEmail activity will be configured and the To property will be promoted to the parent composite activity.

6	Since we created the activity off the "Class" new item template, rather than the "Activity" template, we need to do a few things that ought to have been done for us. Add a constructor to the activity, and two attributes to the InitializeComponent method:
	
	A) Right click the CreateSitePlusNotify activity and choose View code. Notice how a method named InitializeComponent was created for you with all the child activities configured.
	
	B) Add the following attributes to the InitializeComponent method:
	`[System.Diagnostics.`
	`DebuggerNonUserCode]`
	`[System.CodeDom.Compiler.`
	`GeneratedCode("","")]`
	
	Note: you're not supposed to directly edit the InitializeComponent method!
	
	C) Add a constructor to the activity:
	
	`public CreateSitePlusNotify()`
	`{`
	`InitializeComponent();`
	`}`
		A constructor is added to the activity and two attributes added to the InitializeComponent method.
		
		Note: In Visual Studio 2010 RC, the "Activity" new item template wasn't available for SharePoint Workflow projects. If it were, this step wouldn't have been necessary because it would've been done on your behalf automatically.

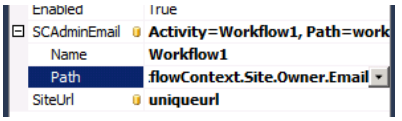
7 Build the project and drop the CreateSitePlusNotify activity onto Workflow1's template.

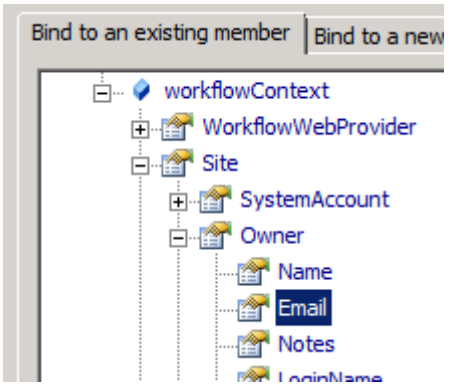


8 The final thing to do is set the dependency properties we promoted from the activity's child activities:

A) Click on the createSitePlusNotify1 activity, and type in a SiteUrl (this could alternatively be set by code).

B) Bind the SCAdminEmail property to the workflow's context site's owner's email address by clicking the "..." ellipsis, and drilling into the workflowContext property:





And that's it folks! It's easy to see that building reusable composite activities are not too difficult. Listing 11.9 shows the CreateSitePlusNotify activity code in its entirety for your reference:

Listing 11.9 CreateSitePlusNotify activity code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Workflow.Activities;
using System.Workflow.ComponentModel;
using System.ComponentModel;

namespace CustomActivities
{
    public class CreateSitePlusNotify : SequenceActivity
    {
        private Microsoft.SharePoint.WorkflowActions.
            ApplyActivation applyActivation1;
        private CreateSubSite createSubSite1;
        public Microsoft.SharePoint.WorkflowActions.
            WorkflowContext workflowContext = new
            Microsoft.SharePoint.WorkflowActions.WorkflowContext();
        private Microsoft.SharePoint.WorkflowActions.SendEmail sendEmail1;
        public static System.Workflow.ComponentModel.DependencyProperty
            SiteUrlProperty = DependencyProperty.Register(
                "SiteUrl", typeof(System.String),
                typeof(CustomActivities.CreateSitePlusNotify));

        public static System.Workflow.ComponentModel.DependencyProperty
            SCAdminEmailProperty = DependencyProperty.Register(
                "SCAdminEmail", typeof(System.String),
                typeof(CustomActivities.CreateSitePlusNotify));

        public CreateSitePlusNotify()
        {
            InitializeComponent();
        }

        private void InitializeComponent()
        {
            this.CanModifyActivities = true;
            System.Workflow.Runtime.CorrelationToken correlationtoken1 =
                new System.Workflow.Runtime.CorrelationToken();
            System.Workflow.ComponentModel.ActivityBind activitybind1 =
                new System.Workflow.ComponentModel.ActivityBind();
            System.Workflow.ComponentModel.ActivityBind activitybind2 =
                new System.Workflow.ComponentModel.ActivityBind();
            System.Workflow.ComponentModel.ActivityBind activitybind3 =
                new System.Workflow.ComponentModel.ActivityBind();
            this.sendEmail1 =
                new Microsoft.SharePoint.WorkflowActions.SendEmail();
            this.createSubSite1 = new CustomActivities.CreateSubSite();
            this.applyActivation1 =
                new Microsoft.SharePoint.WorkflowActions.ApplyActivation();
            //
            // sendEmail1
            //
            this.sendEmail1.BCC = null;
            this.sendEmail1.Body = "Body";
            this.sendEmail1.CC = null;
            correlationtoken1.Name = "CreateToken";
            correlationtoken1.OwnerActivityName = "CreateSitePlusNotify";
```

```
            this.sendEmail1.CorrelationToken = correlationtoken1;
            this.sendEmail1.From = null;
            this.sendEmail1.Headers = null;
            this.sendEmail1.IncludeStatus = false;
            this.sendEmail1.Name = "sendEmail1";
            this.sendEmail1.Subject = "Subject";
            activitybind1.Name = "CreateSitePlusNotify";
            activitybind1.Path = "SCAdminEmail";
            this.sendEmail1.SetBinding(
                Microsoft.SharePoint.WorkflowActions.SendEmail.ToProperty,
                ((System.Workflow.ComponentModel.ActivityBind)
                    (activitybind1)));
            //
            // createSubSite1
            //
            this.createSubSite1.@__Context = null;
            this.createSubSite1.Name = "createSubSite1";
            this.createSubSite1.SiteName = "Contoso Team Site";
            this.createSubSite1.SiteTemplate =
                CustomActivities.CreateSubSite.SiteTemplates.STS;
            activitybind2.Name = "CreateSitePlusNotify";
            activitybind2.Path = "SiteUrl";
            this.createSubSite1.SetBinding(
                CustomActivities.CreateSubSite.SiteUrlProperty,
                ((System.Workflow.ComponentModel.ActivityBind)
                     (activitybind2)));
            //
            // applyActivation1
            //
            activitybind3.Name = "CreateSitePlusNotify";
            activitybind3.Path = "workflowContext";
            this.applyActivation1.@__WorkflowProperties = null;
            this.applyActivation1.Name = "applyActivation1";
            this.applyActivation1.SetBinding(
                Microsoft.SharePoint.WorkflowActions.
                    ApplyActivation.@__ContextProperty,
                ((System.Workflow.ComponentModel.ActivityBind)
                    (activitybind3)));
            //
            // CreateSitePlusNotify
            //
            this.Activities.Add(this.applyActivation1);
            this.Activities.Add(this.createSubSite1);
            this.Activities.Add(this.sendEmail1);
            this.Name = "CreateSitePlusNotify";
            this.CanModifyActivities = false;
        }

        [System.ComponentModel.DesignerSerializationVisibilityAttribute(
            DesignerSerializationVisibility.Visible)]
        [System.ComponentModel.BrowsableAttribute(true)]
        [System.ComponentModel.CategoryAttribute("Misc")]
        public string SiteUrl
        {
            get
            {
                return ((string)(base.GetValue(CustomActivities.
                    CreateSitePlusNotify.SiteUrlProperty)));
            }
            set
```

```
            {
                base.SetValue(CustomActivities.
                    CreateSitePlusNotify.SiteUrlProperty, value);
            }
        }

        [System.ComponentModel.DesignerSerializationVisibilityAttribute(
            DesignerSerializationVisibility.Visible)]
        [System.ComponentModel.BrowsableAttribute(true)]
        [System.ComponentModel.CategoryAttribute("Misc")]
        public string SCAdminEmail
        {
            get
            {
                return ((string)(base.GetValue(CustomActivities.
                    CreateSitePlusNotify.SCAdminEmailProperty)));
            }
            set
            {
                base.SetValue(CustomActivities.
                    CreateSitePlusNotify.SCAdminEmailProperty, value);
            }
        }
    }
}
```

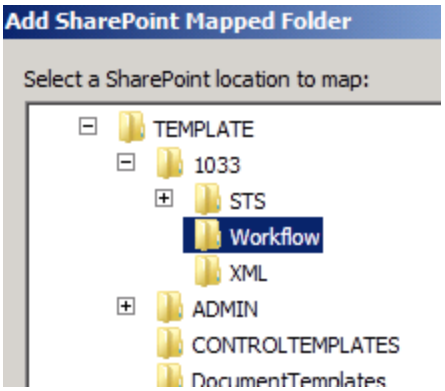
11.4 Publishing Activities to SharePoint Designer

It's no doubt that a lot of people that will be building SharePoint Workflows won't be programmers, and will use tools like SharePoint Designer instead. The trouble is they often will still need to meet very complex business requirements that typically can only be met with Visual Studio workflows. How does someone narrow the chasm between SharePoint Designer and Visual Studio? Publishing your custom activities into SharePoint Designer as custom actions is the easiest way to do this.

When SharePoint Designer connects to a SharePoint site, it first goes to the server and pulls down files on the server with a ACTIONS extension. It's within this file, or files, that SharePoint Designer knows what actions and conditions it should make available to its users when they're developing workflows. All we (the programmers) have to do is create our own ACTIONS file to get our custom activities deployed to SharePoint Designer. The only other task is we need to mark our activity as safe in the web.config of the web application. Follow these steps to deploy our custom CreateSubSite activity into SharePoint Designer:

Publish the CreateSubSite activity to SharePoint Designer

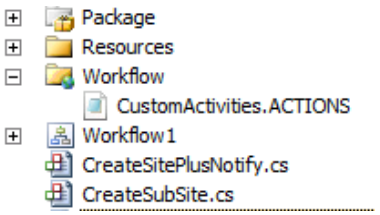
	Action	Result
1	Add a mapped folder in the CustomActivities project: A) Right click the project, and choose Add > SharePoint Mapped Folder… B) Select the Workflow folder under TEMPLATE > 1033:	A folder will be added to the project that is mapped to the SharePoint file system. Any files in the folder will automatically be deployed to SharePoint.



2 Create a new XML file in the new Workflow folder in the project:

A) Right click the workflow folder, and choose Add > New Item.

B) Under the Data tab, select XML file and name the file CustomActivities.ACTIONS, and then click Add.



3 Enter Listing 11.10 into the ACTIONS file.

The ACTIONS file will be populated with the needed XML that tells SharePoint Designer to load our custom action.

4 Add an authorized type in your web application's web.config file:

A) Open your web.config file (usually under c:\inetpub\wwwroot\wss\virual directories\[your web app])

B) Find the authorizedTypes section within System.Workflow.ComponentModel.WorkflowCompiler.

C) Add a new authorized type:

```
<authorizedType
Assembly="CustomActivities,
Version=1.0.0.0, Culture=neutral,
PublicKeyToken=60cb170cc12e2631"
Namespace="CustomActivities"
TypeName="*" Authorized="True" />
```

Note: Make sure you enter your own token!

A authorized type is added into the web.config telling SharePoint that we trust our assembly and namespace.

Listing 11.10 CustomActivities.ACTIONS

```
<?xml version="1.0" encoding="utf-8"?>
<WorkflowInfo Language="en-us">
  <Actions Sequential="then" Parallel="and">
    <Action Name="Create Sub Site"
        ClassName="CustomActivities.CreateSubSite"                        1
        Assembly="CustomActivities, Version=1.0.0.0, Culture=neutral,
            PublicKeyToken=929a2d1b8d7f5b6c"
        AppliesTo="all"
        Category="Custom Actions">                                        2
      <RuleDesigner Sentence="Create sub-site named %1                    3
            with the URL of %2 and using the %3 site template.">
        <FieldBind Field="SiteName" Text="name" Id="1" />                  4
        <FieldBind Field="SiteUrl" Text="url" Id="2" />
        <FieldBind Field="SiteTemplate" DesignerType="Dropdown"           5
              Text="template" Id="3">
          <Option Name="Blog" Value="BLOG"/>
          <Option Name="Meeting Workspace" Value="MPS"/>
          <Option Name="Team Site" Value="STS"/>
          <Option Name="Wiki" Value="WIKI"/>
        </FieldBind>
      </RuleDesigner>
      <Parameters>                                                        6
        <Parameter Name="__Context"                                       7
            Type="Microsoft.SharePoint.WorkflowActions.WorkflowContext,
            Microsoft.SharePoint.WorkflowActions" Direction="In"/>
        <Parameter Name="SiteTemplate" Type="System.Enum, mscorlib"
            Direction="In" />
        <Parameter Name="SiteName" Type="System.String, mscorlib"
            Direction="In" />
        <Parameter Name="SiteUrl" Type="System.String, mscorlib"
            Direction="In" />
      </Parameters>
    </Action>
  </Actions>
</WorkflowInfo>
```

1) Assembly reference
2) Category in Actions dropdown in SPD
3-5) User specified field values
6) Values mapped back to dependency parameters
7) WorkflowContext token

You'll notice in this listing that an action has three main elements, an assembly reference, a RuleDesigner, and Properties. The attributes on the Action element make up the assembly reference that tells SharePoint Designer what assembly our custom activities is contained in **(#1)**. Notice how it's referencing our CustomActivities assembly and default namespace. Also notice the PublickKeyToken setting in the assembly reference. Make sure to specify your own token here. The easiest way to get your token is by browsing to C:\Windows\system32\assembly, find your assembly, right click and choose properties and then copy your token out of the properties window.

Another interesting attribute is the Category attribute **(#2)**. This attribute tells SharePoint Designer what category of actions to stick your custom activity in (Figure 11.9). The next main category of elements in this ACTIONS file is the RuleDesigner element. The RuleDesigner element has a "Sentence" attribute **(#3)** that renders to the user. You can use this sentence to gather information from the user. Within the sentence, you can add tags in the format of percent/id (eg. %1). This tag is mapped to the

field with ID equal to 1, take SiteName for example **(#4)**. The SiteName field has a text attribute set to "name" which will be rendered as a blue link, and when the user clicks the link they'll be prompted to specify some value for the site's name. That value is mapped to one of the parameters in the Parameters element **(#6)**. Those parameters are in turn mapped to dependency properties in the .NET activity itself. This is how values specified by the SharePoint Designer user are passed into the .NET activity.

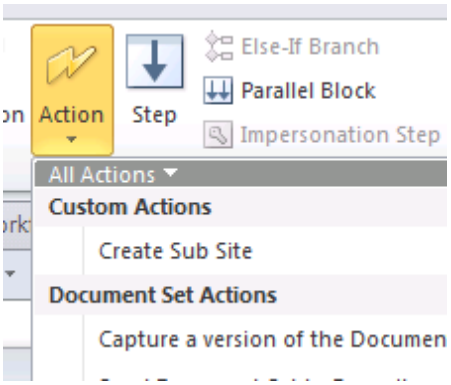


Figure 11.19 When you deploy your custom activity into SharePoint Designer, you can specify a category name to place your activity in, such as "Custom Actions".

There are a couple more points that we need to hit quick. Notice how the SiteTemplate field has a DesignerType attribute **(#5)**. You can use this designer type to control how you want the user to specify the value. In the case of SiteTemplate, it's being rendered with a dropdown populated by some hardcoded options. Reference table 11.2 for more DesignerType settings.

Table 10.2 DesignerType options

Boolean	Dropdown shown with true and false as options.
ChooseDoclibItem	Document library selector
ChooseListItem	List item selector
CreateListItem	Default pop-up
Date	Date/Time selector
Dropdown	Dropdown box with Option elements to populate dropdown
Email	Advanced Email control pop-up
FieldNames	Dropdown with options for all the fields in the current list or library.
Float	Text box allowing for numbers with decimals.
Hyperlink	URL browser pop-up allowing to you browse to a local or remote URL.
Integer	Text box allowing for only integer numbers.

ListNames	Dropdown showing options for all the lists in the current web.
Operator	Dropdown showing operator options, eg. < > !=, etc. Operators must be statically defined as Option elements.
ParameterNames	Dropdown showing list of all variables defined in the workflow.
Person	Person or group selector.
SinglePerson	Person or group selector but you an only select one person or group.
Stringbuilder	Advanced text box.
Survey	Default pop-up
Text	Default pop-up
TextArea	Default pop-up
UpdateListItem	Default pop-up
writablefieldNames	Dropdown showing either fields in current list, or list of document libraries that are editable by the running user.

Lastly, remember the "_ _Context" dependency property we created for our CreateSubSite activity? We gave it a goofy name because SharePoint Designer looks for a dependency property with that name and is smart enough to take the workflow's WorkflowContext object and assign it to the __Context parameter **(#7)**. Other options for this are found in table 11.3 below.

Table 10.3 Parameter Tokens

__ActivationProperties	SPWorkflowActivationProperties: used to get workflow information such as when the workflow started.
__ListId	String: gets the list name the workflow is running on.
__ListItem	Integer: gets the id of the list item the workflow is running on.
__Context	WorkflowContext: gets helpful objects such as SPSite and SPWeb.

With our ACTIONS file in place, we're now ready to publish our activity. Build and deploy the project. This will update the assembly, as well as publish our ACTIONS file into the Workflows folder on the server. Open up SharePoint Designer and connect to a site. Create a new workflow (any type) and SharePoint Designer will then look at your custom ACTIONS file to determine what actions to make available to you. You should notice our custom action in the Custom Actions category. Figure 11.10 shows what our action looks like after it has been added to the workflow template, and the parameters have been specified:

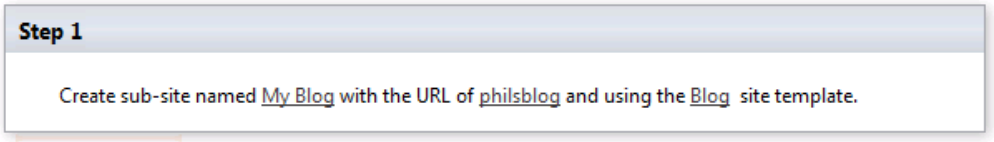


Figure 11.20 The sentence in the ACTIONS file was rendered with all the fields and parameters as editable by the user.

11.3 Building Custom Conditions for SharePoint Designer

We just got done building a custom action for SharePoint Designer, but how about custom conditions? Building custom conditions for SharePoint Designer can introduce a lot of power into your SharePoint Designer workflows. For instance, you could write a custom condition that checks an external data source or system before executing on a block of activities. The best part is, it's very easy to do! Essentially all you have to do is write a .NET method that returns a Boolean, and make that method available in SharePoint Designer as a condition by adding an element in a .ACTIONS file.

To build on our CreateSubSite example, we're going to write a custom condition that we can use to check for URL availability. It's nice that we can create a sub site from a SharePoint Designer workflow, but how will we know that the URL we've specified is available? A sub site might already exist that's using that URL. Our custom activity will take the proposed URL as a parameter, and confirm that a site is not using that URL. If not, the condition will return true, otherwise it will return false. Follow these steps to create and publish our condition to SharePoint Designer:

Create a custom condition and deploy it into SharePoint Designer

	Action	Result

1	Create a new class file named SubSiteExistsCondition:
	
	A) Add a new class into the CustomActivities project named SubSiteExistsCondition.cs.
	
	B) Make the class public and add using statements for the following namespaces:
	
	C) Create a public static method named SubSiteExists with the following code:
	
	`public static bool SubSiteExists(`
	`  WorkflowContext context, string listId,`
	`  int itemId, string url)`
	`{`
	`  if (context.Web.Webs.Names.Contains(url))`
	`  return true;`
	`  else`
	`  return false;`
	`}`	

The first thing you'll see in the SubSiteExists method are three parameters you didn't think you would need. The first three parameters are required parameters. The fourth is optional, but needed for our example. We simply check to see if the current site has a sub site with a url (Name) that is the same as the URL that is passed into the method in the fourth parameter. If yet, return true. If no, return false. Next, all we need to do is add a few elements into our ACTIONS file and we'll be good to go!

	Action	Result
2	Add a Conditions element into the ACTIONS file: A) Above the Actions element, add the Conditions element found in Listing 11.11.	In addition to our custom action, the ACTIONS file will also be publishing a custom condition pointing to the SubSiteExists method.

Listing 11.11 Conditions Element

```
<Conditions>
  <Condition Name="If Sub Site Exists"
             FunctionName="SubSiteExists"                                1
             ClassName="CustomActivities.SubSiteExistsCondition"
             Assembly="CustomActivities, Version=1.0.0.0, Culture=neutral,
                 PublicKeyToken=60cb170cc12e2631"
             AppliesTo="all"
             Category="Custom Conditions">
    <RuleDesigner Sentence="Sub Site exists with a relative url of %1">
      <FieldBind Id="1" Field="_1_" Text="url" DesignerType="TextArea" /> 2
    </RuleDesigner>
    <Parameters>
      <Parameter Name="_1_" Type="System.String, mscorlib" Direction="In"/>
    </Parameters>
  </Condition>
</Conditions>
```

1) Specify method name
2) Use token instead of parameter name

The XML for Conditions is nearly identical to the XML for Actions. There's only a few small differences. Firstly in addition to declaring the assembly and namespace, you also must declare the method name that will execute your condition **(#1)**. Secondly, notice how we're no longer referencing the field name, or what would you'd expect to see "url" **(#2)**. Rather, we're using an _1_ token to tell Designer to pass the string into the first optional parameter. Even though url is the fourth parameter in the method, it is still the first optional parameter. Additional parameters would take the liking of _2_, _3_, etc.

That should be the last step. Build and deploy your project and create a new workflow in SharePoint Designer (or edit existing). You will need to close and reopen the connection to get the latest ACTIONS file. When you done that, add the If Sub Site Exists condition and add an Else branch onto it. Inside the If branch, add a Logger to log that the URL has already been taken. In the else branch, add a Create

Sub Site action along with a Logger to log that the site was created successfully. When you're done, your template should look something like Figure 11.11. Go ahead and publish and test your workflow.

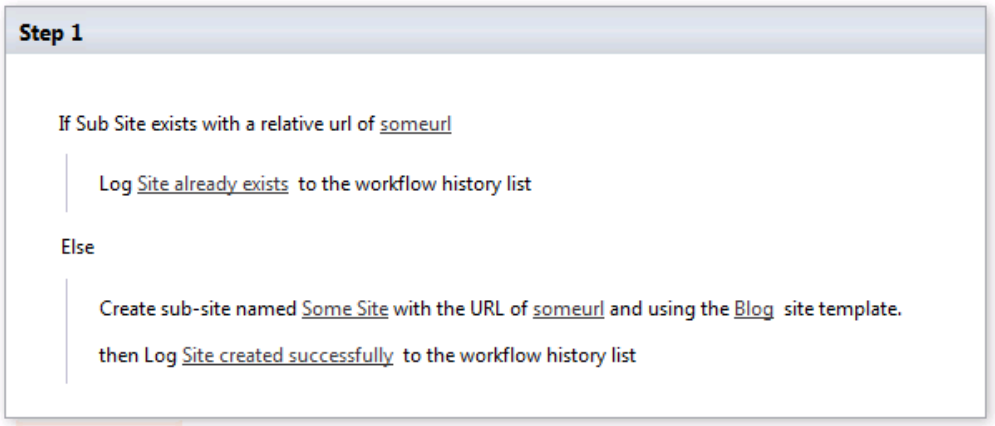


Figure 11.21 Custom conditions can also be deployed into SharePoint Designer. This condition checks to see if a URL has already been taken by another sub site.

11.5 Summary

Knowing how to build custom activities is a fundamental skill for any workflow developer… even SharePoint workflow developers. By building custom activities you can gain the benefits of improved code maintainability and reuse. You'll have a better maintained solution because you're less likely to have a workflow template with 2000+ lines of nasty looking code. Rather, you have your code split out into the various activities (classes) instead of one big template (class). You'll have better reuse of code as well because now you can start dropping your custom activities onto other workflows, rather than recreate them from scratch each time.

As far as types of activities go, you have your simplest activities called leaf activities, and more complex activates called composite activities. Examples of leaf activities are activities such as CreateTask, Delay, SetState, or OnWorkflowItemChanged. These activities do one thing, and they do it well. Composite activities on the other hand are activities that contain child activities. Some examples are the While, IfElse, Sequence, and Parallel activities. These activates are great for packaging up larger chunks of reusable business processes that other workflows can use.

Both types of activities can be published into SharePoint Designer to help add more robust functionality for workflows create from that tool. This is done by publishing an ACTIONS file into the file system on each server in the farm. When SharePoint Designer connects to a site, it first downloads this ACTIONS file so it can know what actions to make available to the user building the workflow. Custom conditions also can be deployed into SharePoint Designer. A custom condition is simply a public static method that returns a Boolean. Simply add the condition into the ACTIONS file and point it to your method, and you're rolling.