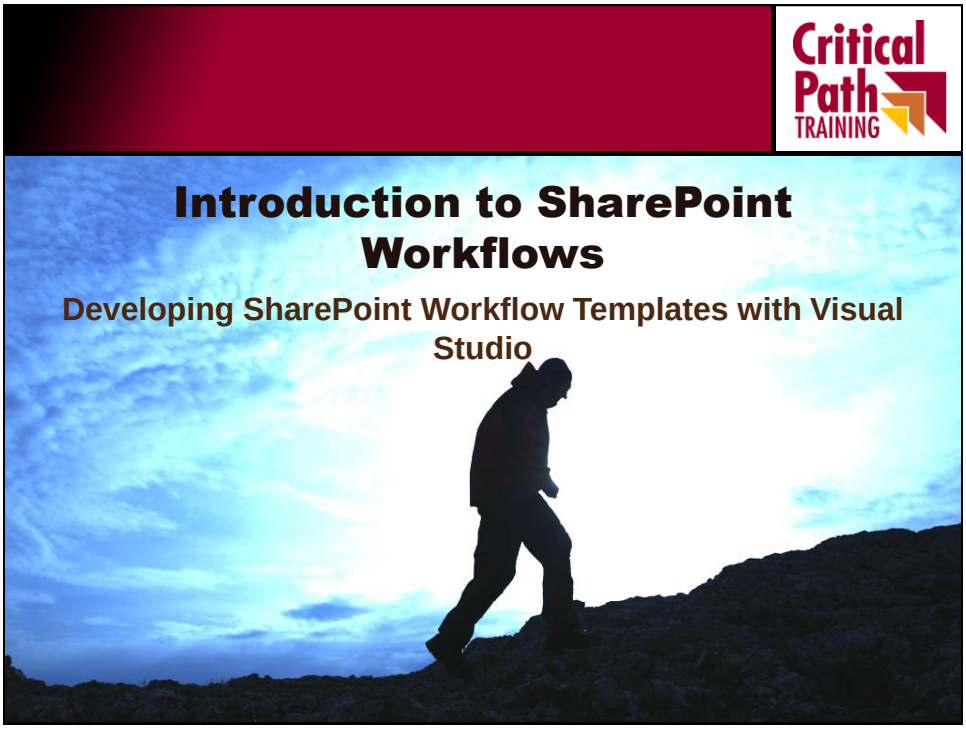
SPW401: Developing SharePoint Workflow Templates with Visual Studio

Schedule of lectures

1. Introduction to SharePoint Workflows
2. Windows Workflow Foundation Primer
3. Understanding the Windows Workflow Foundation Runtime
4. Windows Workflow Foundation Integration with SharePoint 2007
5. Programming with SharePoint Workflow API
6. Developing SharePoint Workflow Templates with Visual Studio 2008
7. Creating and Waiting on SharePoint Tasks
8. Creating Workflow Association Forms
9. Creating Workflow Instantiation and Modification Forms
10. Integrating InfoPath Forms into a SharePoint Workflow
11. Developing Custom Activities for a SharePoint Workflow
12. Extending SharePoint Designer with Custom Activities

Revision: v3.0



Agenda

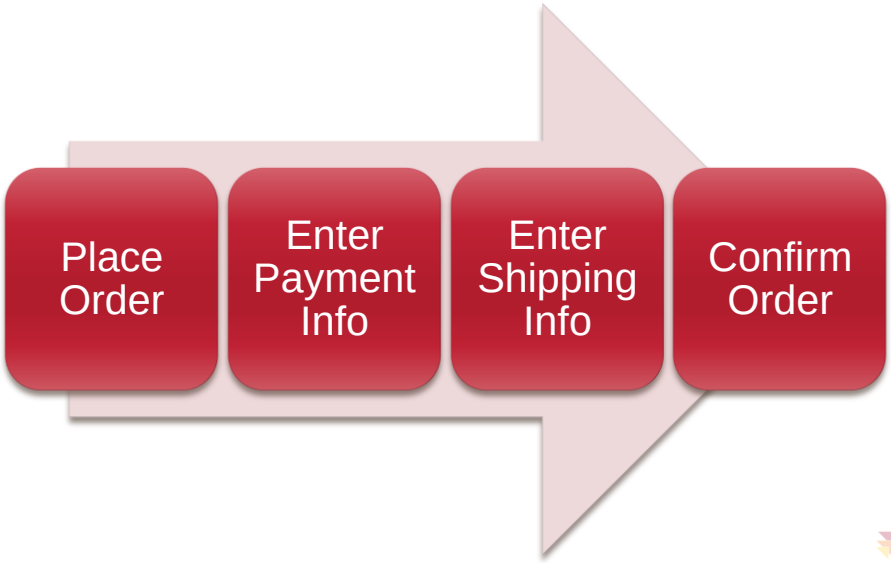
- What is a Reactive Program?
- Why is workflow beneficial
- Introduction to Windows Workflow Foundation
- Introduction to Workflow in SharePoint

Reactive Programming

- Processes requiring human interaction
  - Process defined centrally
  - Asynchronously interact with other systems

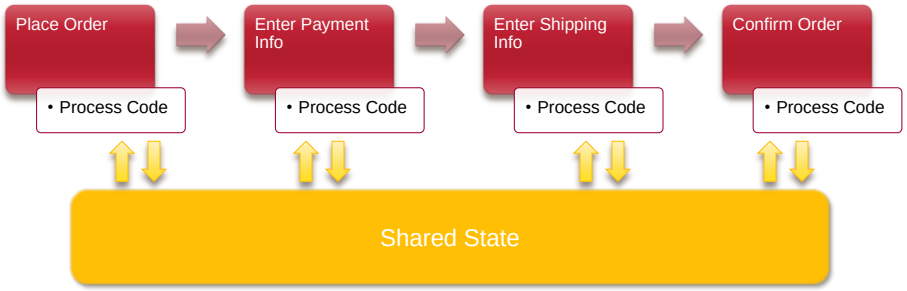
```
Users
Web Services
Other external services
```

Scenario – Online Order Pipeline



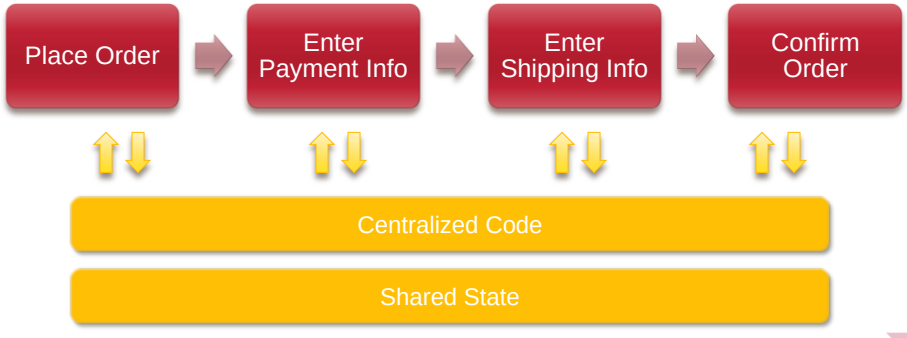
Where does the code live?

- Each transition's code exists in step
  - Decentralized process



A Centralized Model

- Centralize code in central controller
  - Decouples interface from process

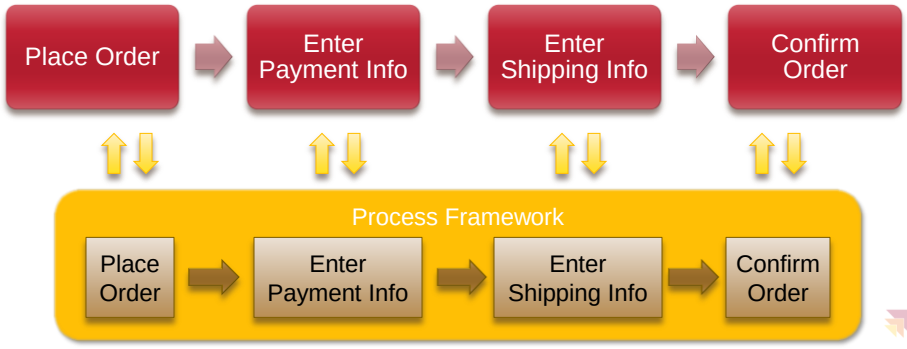


Common Process Functionality

- Centralized process leads to common behaviors
  - Track the state of the process
  - Move between “states”
  - Communicate with the outside world

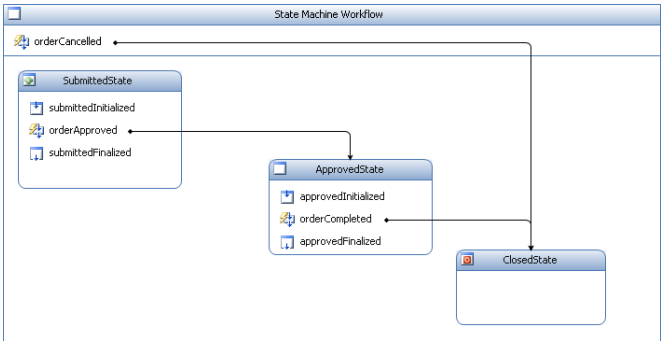
A Process Management Framework

- Decoupling allows generic process framework
  - Stores state as .NET objects
  - Defines process as an interconnected set of steps
  - Define a well known interface for communication



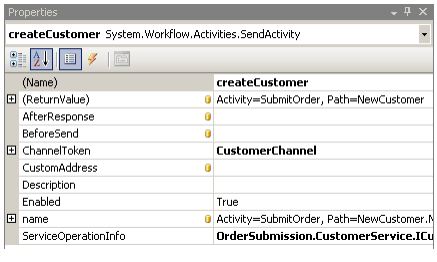
Windows Workflow Foundation

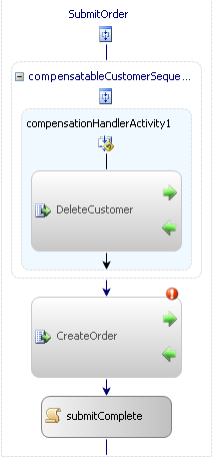
- Provides processing framework
- Shipped with .NET 3.0
  - Design environment in VS2005, VS2008



What is WinWF?

- Workflows are made up of Activities
  - Processes = Workflows
  - Steps = Activities





Hosting the WinWF Runtime

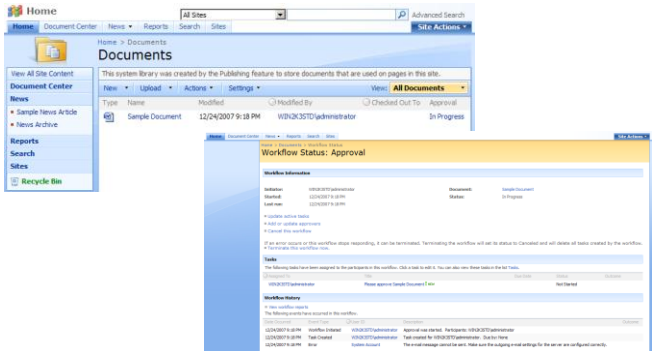
- Can be hosted in any application
  - Windows Forms
  - ASP.NET
  - SharePoint
- Host/Framework interaction via services
  - ExternalDataExchange
  - Persistence
  - Threading
  - Etc...

SharePoint Integration

- SharePoint is another WinWF Host
  - SharePoint provides it's own services
  - Can execute any WinWF workflow
- Ships with several built in workflows
  - Three State
  - Approval – MOSS only
  - Translation – MOSS only

Human Interaction in SharePoint

- Human interaction defined via Tasks
  - Assigned to users
  - Workflow “pauses” while waiting for task completion
  - Think of SharePoint as UI for WinWF



Student Questionnaire

- What’s Your Name?
- What Company are you with?
- How have you evolved as a Developer?
- Do you have experience with…
  - The .NET Framework and Visual Studio
  - WSS 2.0 and SPS 2003
  - WSS 3.0 and MOSS
  - WinWF or BizTalk

Day 1

- Introduction to SharePoint Workflows
- Windows Workflow Foundation Primer
- Understanding the Windows Workflow Foundation Runtime

Day 2

- Windows Workflow Foundation Integration with SharePoint 2007
- Programming with SharePoint Workflow API
- Developing SharePoint Workflow Templates with Visual Studio 2008

Day 3

- Creating and Waiting on SharePoint Tasks
- Creating Workflow Association Forms
- Creating Workflow Instantiation and Modification Forms

Day 4

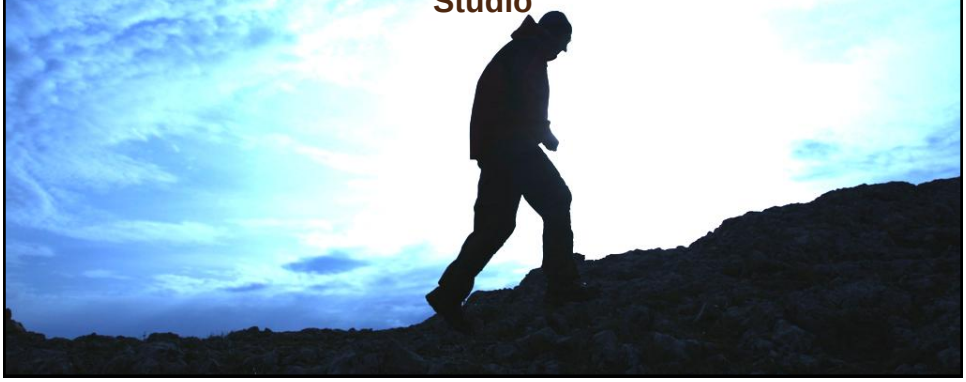
- Integrating InfoPath Forms into a SharePoint Workflow
- Developing Custom Activities for a SharePoint Workflow
- Extending SharePoint Designer with Custom Activities

Summary

- What is a Reactive Program?
- Why is workflow beneficial
- Introduction to Windows Workflow Foundation
- Introduction to Workflow in SharePoint

Windows Workflow Foundation Primer

Developing SharePoint Workflow Templates with Visual Studio



Agenda

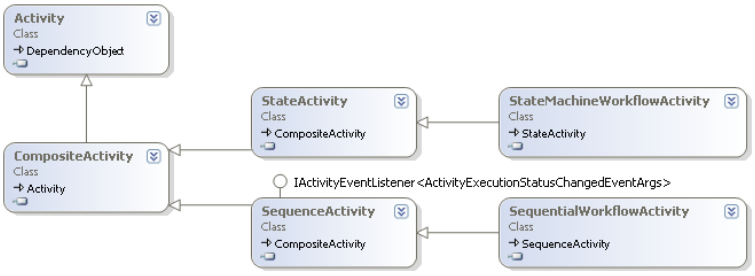
- What is a workflow?
- Running Workflows
- Activities
- Built In Activities

What is a Workflow?

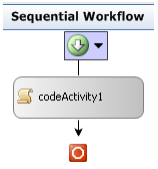
- Collection of activities
  - Activities are just classes

    `Activity`
  - Some activities can contain other activities

    `CompositeActivity`
  - Workflows are also activities

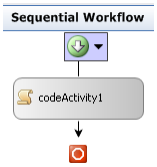


How are workflows defined? (Code)




```
private void InitializeComponent()
{
    this.CanModifyActivities = true;
    this.codeActivity1 = new System.Workflow.Activities.CodeActivity();
    //
    // codeActivity1
    //
    this.codeActivity1.Name = "codeActivity1";
    this.codeActivity1.ExecuteCode += new System.EventHandler(ExecuteCode);
    //
    // HelloWorld
    //
    this.Activities.Add(this.codeActivity1);
    this.Name = "HelloWorld";
    this.CanModifyActivities = false;
}
```

How are workflow's defined? (XOML)




```
<SequentialWorkflowActivity xmlns:x="…" xmlns="…"
                            x:Class="HelloWorldWorkflow.Workflow"
                            x:Name="Workflow" >
    <CodeActivity x:Name="codeActivity1"
                  ExecuteCode="codeActivity1_ExecuteCode" />
</SequentialWorkflowActivity>
```

Xoml

- Xml format used to control object creation
  - Allows dynamic loading of workflows

```
<SequentialWorkflowActivity xmlns:x="…" xmlns="…" x:Name="Workflow"
                            x:Class="HelloWorldWorkflow.Workflow">
    <CodeActivity x:Name="codeActivity1"
                  ExecuteCode="codeActivity1_ExecuteCode" />
</SequentialWorkflowActivity>
```

```
private void InitializeComponent()
{
    this.CanModifyActivities = true;
    this.codeActivity1 = new System.Workflow.Activities.CodeActivity();
    this.codeActivity1.Name = "codeActivity1";
    this.codeActivity1.ExecuteCode += new System.EventHandler(ExecuteCode);
    this.Activities.Add(this.codeActivity1);
    this.Name = "HelloWorld";
    this.CanModifyActivities = false;
}
```

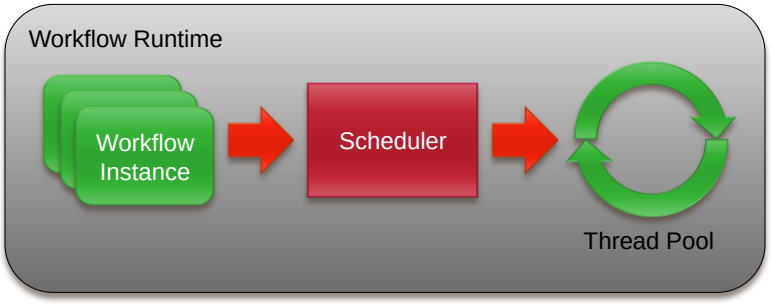
Code Behind

- Implemented using partial classes
  - Xoml and Code workflow classes are marked as partial

```
public sealed partial class HelloWorld : SequentialWorkflowActivity
{
    public HelloWorld() { InitializeComponent(); }
}

partial class HelloWorld
{
    private CodeActivity codeActivity1;

    private void InitializeComponent()
    {
        this.CanModifyActivities = true;
        this.codeActivity1 = new CodeActivity();
        this.codeActivity1.Name = "codeActivity1";
        this.Activities.Add(this.codeActivity1);
        this.Name = "HelloWorld";
        this.CanModifyActivities = false;
    }
}
```

Running a Workflow

- Several classes are involved
  - WorkflowRuntime hosts the execution environment
  - WorkflowInstance represents the executing process
  - Scheduler provides the environment with a thread

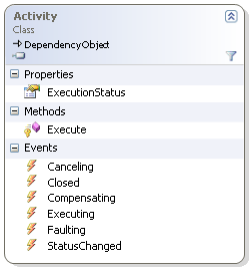


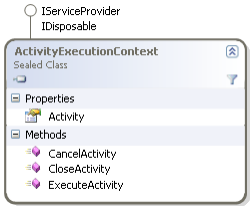
Running a Workflow

```
using (WorkflowRuntime workflowRuntime = new WorkflowRuntime())
{
    AutoResetEvent waitHandle = new AutoResetEvent(false);
    workflowRuntime.WorkflowCompleted +=
        delegate(object sender, WorkflowCompletedEventArgs e)
        {
          waitHandle.Set();
        };
    workflowRuntime.WorkflowTerminated +=
        delegate(object sender, WorkflowTerminatedEventArgs e)
        {
            Console.WriteLine(e.Exception.Message);
            waitHandle.Set();
        };

    WorkflowInstance instance =
        workflowRuntime.CreateWorkflow(typeof(Workflow));
    instance.Start();

    waitHandle.WaitOne();
}
```

Activities

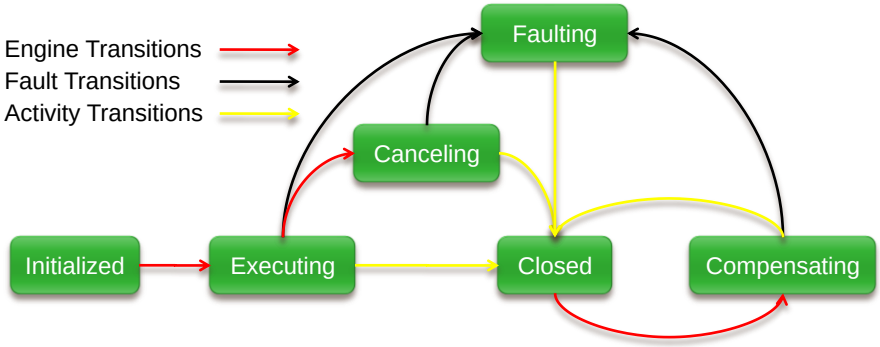
- All activities live in an Activity Execution Context
  - Activity Execution Context (AEC) tracks activity state
  - Optimizes creation and destruction of activity tree
  - Quick to serialize and de-serialize





Activity Lifecycle

- Activities exist in a specific state
  - Activities are completed when in the closed state

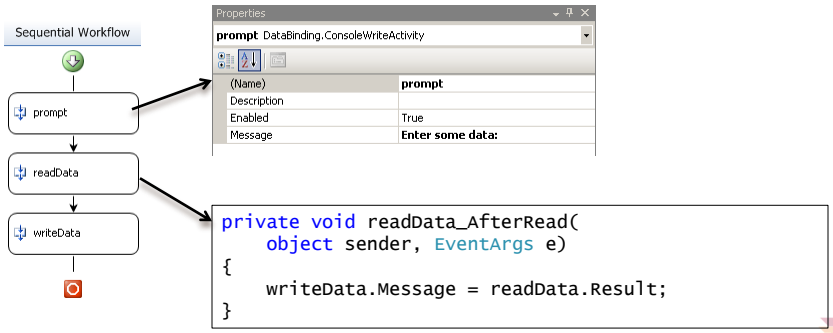


Activity Execution

- Work queues used to detach activity from thread
  - Activity starts async method
  - Execute method returns executing
  - When async operation complete, item placed in queue
  - When the item is de-queued, the activity is resumed

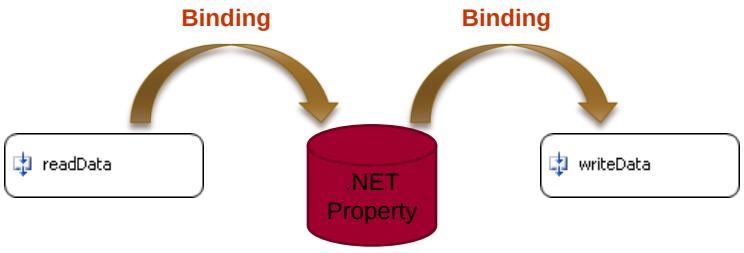
Designing with Activities

- Activities are configured via Properties
  - Can be assigned via code
  - Data flows from activity to activity
  - Mechanism needed to declaratively connect properties



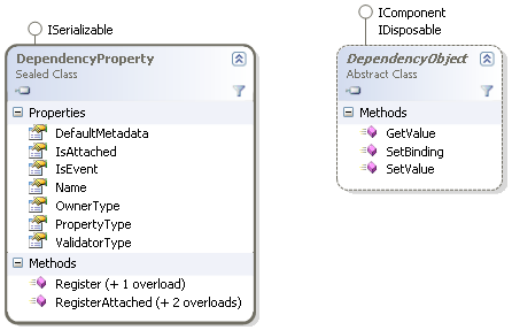
Binding Activity Properties

- Declarative activity relationships are better
  - Minimizes amount of repetitive code
  - Requires centralized storage of bind-able properties



Dependency Properties/Objects

- Designed to allow runtime object extensions
  - Similar to dynamic languages
  - composite activities may use to attach data to children `ConditionedGroupActivity`



Using DependencyProperties

- Requires two parts
  - Object that derives from DependencyObject
  - Static DependencyProperty object
- Often accompanied by helper .NET properties
  - DependencyObject's Get/SetValue used for access

```
public static readonly DependencyProperty ResultProperty =
   DependencyProperty.Register("Result",
       typeof(string), typeof(ConsoleReadActivity));

public string Result
{
    get { return (string)GetValue(ResultProperty); }
    set { SetValue(ResultProperty, value); }
}
```

Using Data Binding

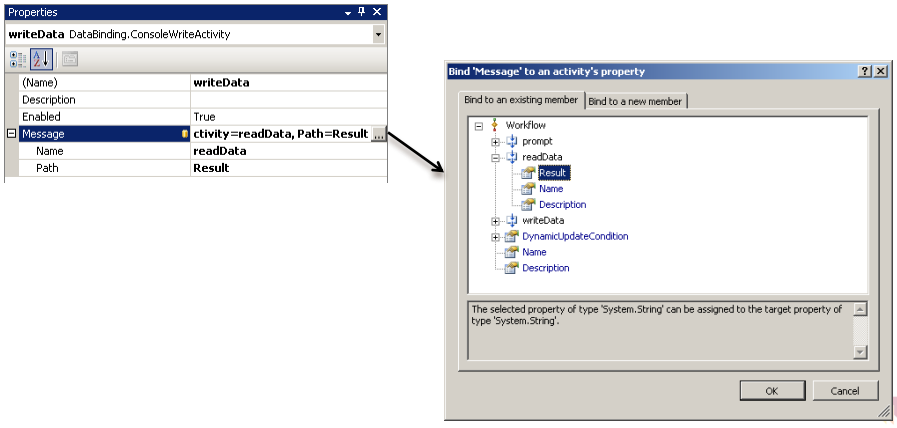
- Done using an ActivityBind object
  - Defines the source object and property
  - Defines target property (must be DependencyProperty)
  - Can be defined via Code or Xoml

```
ActivityBind activitybind1 = new ActivityBind();
activitybind1.Name = "readData";
activitybind1.Path = "Result";
this.writeData.SetBinding(
    DataBinding.ConsoleWriteActivity.MessageProperty,
    activitybind1);
```

```
<SequentialWorkflowActivity xmlns:ns0="…" xmlns:x="…" xmlns="…"
    x:Name="Workflow1" >
  <ns0:ConsoleWriteActivity x:Name="consoleWriteActivity1"
      Message="{ActivityBind Workflow,Path=Prompt}" />
</SequentialWorkflowActivity>
```

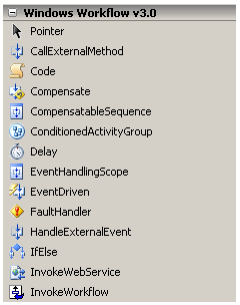
Using Data Binding

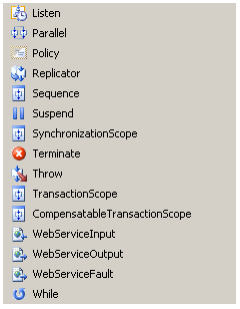
- Or use the VS 2008 user interface
  - Property window links to binding dialog



Built-in Workflow Activities

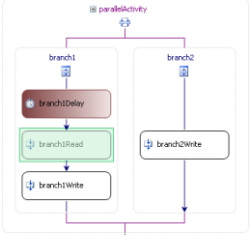
- Located in System.Workflows.Activities Assembly
- Used as a starting point for custom activities

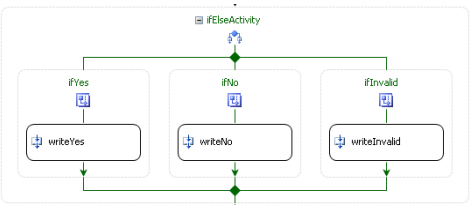




Branching Activities

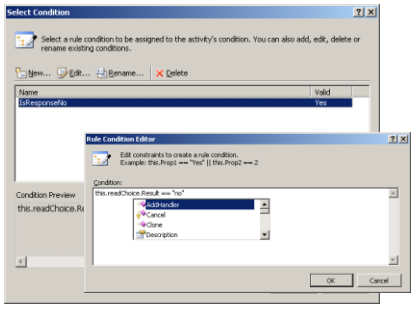
- Controls the flow of activity execution
  - Two determine when activities run
    `Sequential and Parallel`
  - One determines which activities run
    `If/Else`





Declarative Conditions

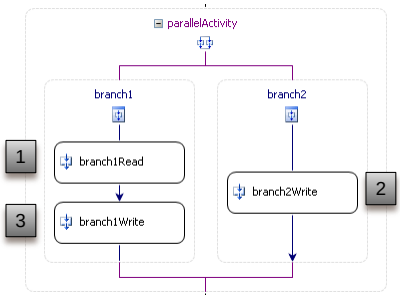
- Allow declarative conditions to be declared
  - Defined using C# like syntax, stored as xml rules
  - Embedded into assembly as a resource
  - Attached to activities properties via named conditions

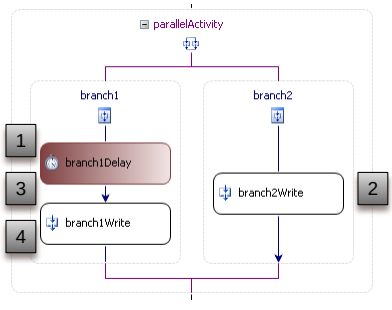



```
<RuleDefinitions xmlns="http://schemas.microsoft.com/...">
  <RuleDefinitions.Conditions>
    <RuleExpressionCondition Name="IsResponseNo">
      <RuleExpressionCondition.Expression>
        <ns0:CodeBinaryOperatorExpression Operator="ValueEquality" xmlns:ns0="...">
          <ns0:CodeBinaryOperatorExpression.Left>
            <ns0:CodePropertyReferenceExpression PropertyName="Result">
              <ns0:CodePropertyReferenceExpression.TargetObject>
                <ns0:CodeFieldReferenceExpression FieldName="readChoice">
                  <ns0:CodeFieldReferenceExpression.TargetObject>
                    <ns0:CodeThisReferenceExpression />
                  </ns0:CodeFieldReferenceExpression.TargetObject>
                </ns0:CodeFieldReferenceExpression>
              </ns0:CodePropertyReferenceExpression.TargetObject>
            </ns0:CodePropertyReferenceExpression>
          </ns0:CodeBinaryOperatorExpression.Left>
          <ns0:CodeBinaryOperatorExpression.Right>
            <ns0:CodePrimitiveExpression>
              <ns0:CodePrimitiveExpression.Value>
                <ns1:String xmlns:ns1="...">no</ns1:String>
              </ns0:CodePrimitiveExpression.Value>
            </ns0:CodePrimitiveExpression>
          </ns0:CodeBinaryOperatorExpression.Right>
        </ns0:CodeBinaryOperatorExpression>
      </RuleExpressionCondition.Expression>
    </RuleExpressionCondition>
  </RuleDefinitions.Conditions>
</RuleDefinitions>
```

Parallel Activity

- Allows two independent execution paths
  - Does not provide multi-threaded
  - Does provide better utilization of single thread





Looping Activities

- Looping done using **for** or **for-each** constructs
  - While loop provides for functionality

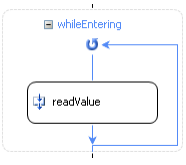
    Based on exit condition
  - Replicator provides for-each functionality

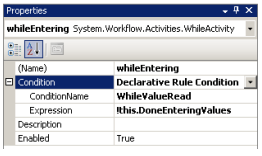
    Based on list data and optional exit condition

While Activity

- Continues executing until a condition is false
  - Condition can be declarative or code
  - Can only contain a single child activity

    Use a sequence activity if you need multiple





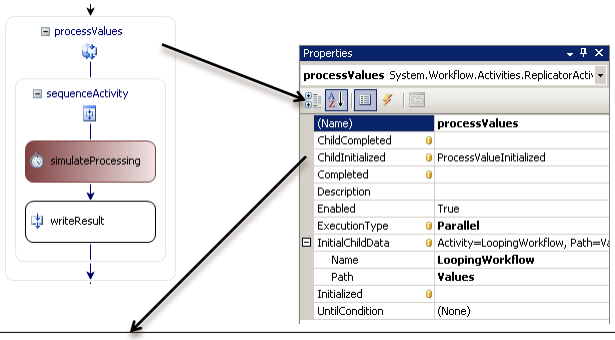

```
private void ReadValueComplete(object sender, EventArgs e)
{
    string result = (sender as ConsoleReadActivity).Result;
    this.DoneEnteringValues = string.IsNullOrEmpty(result);
    if (!this.DoneEnteringValues)
        Values.Add(result);
}
```

Replicator Activity

- Creates on instance of contained activity per item
  - Can only contain a single child activity
  - Instances can be run in sequence or parallel
  - No way to bind current element to activity

    Must be done in code

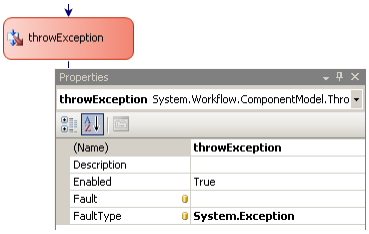
Replicator Activity

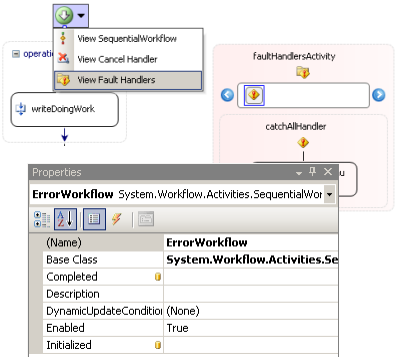



```
private void ProcessValueInitialized(
  object sender, ReplicatorChildEventArgs e)
{
    ConsoleWriteActivity writeResult =
        e.Activity.GetActivityByName("writeResult")
        as ConsoleWriteActivity;
    writeResult.Message =
        string.Format("  {0}", e.InstanceData);
}
```

Error Handling

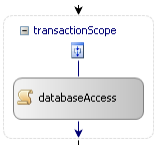
- Exception handling in workflow similary to .NET
  - Exceptions thrown with ThrowFault activity
  - Exceptions caught with FaultHandler activity

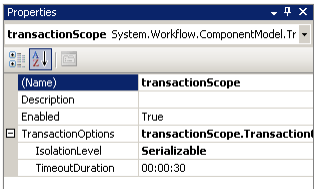




Transactions

- Wraps activities into a distributed transaction
  - Workflow can't sleep while in a transaction
    `No Delay, Web Service calls, etc…`
  - Transactions implemented by external plug-in services
    `Default implementations use TransactionScope`



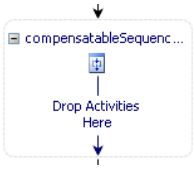


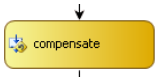
Compensation

- Can't hold transactions and wait for user input
  - Humans are slow; would lead to resource contention
  - Better way is to manually 'undo' on failure

    `Ex. Undo create customer with delete customer`

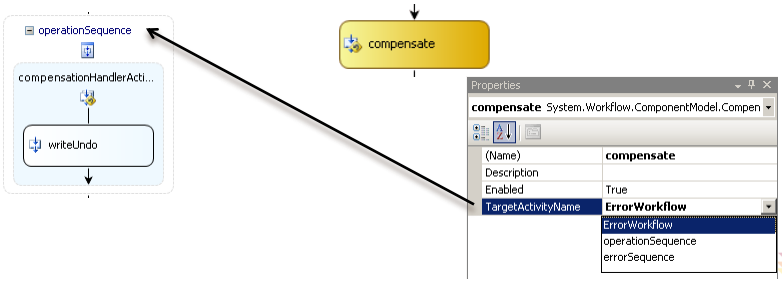
- Done using CompensatableScope and Compensate activities





Using Compensation

- CompensatableSequence activities hold actions
  - Has Compensation Handler holding 'undo' activities
- Compensate activity initiates 'undo' activities
  - Runs Compensation Handler for target and children
  - Handler executed if sequence completed successfully

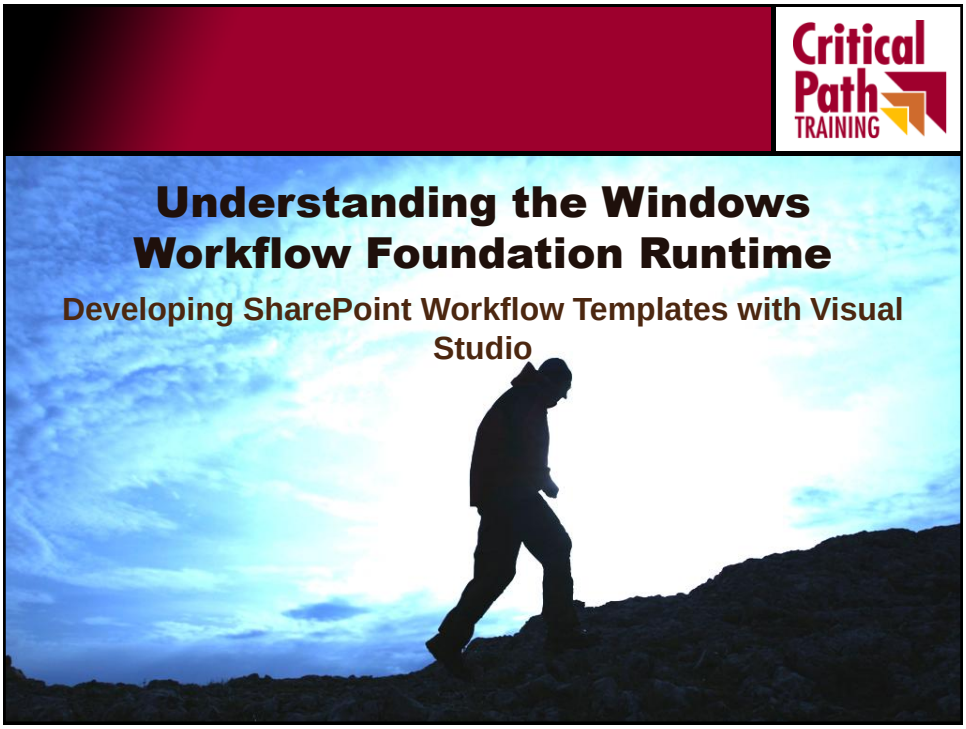


Other Activities

- Code Activity
  - Allows execution of custom code
- Delay Activity
  - Forces the workflow to sleep for a specified time
- CompensatableTransactionScope Activity
  - Provides compensation and transactions together
- Web Service Activities
  - Allow invocation of web services inside the workflow
- Suspend and Terminate Activities
  - Allow suspension and termination inside the workflow

Summary

- What is a workflow?
- Running Workflows
- Activities
- Built In Activities

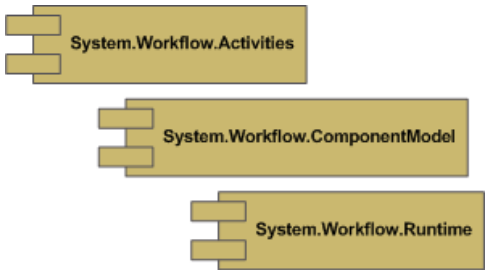


Agenda

- Workflow Runtime
- WorkflowRuntimeServices
- ExternalDataExchangeService
- StateMachine Workflow
- WorkflowPersistenceService

Workflow Runtime

- Part of .NET 3.0
  - System.Workflow.Activities
  - System.Workflow.ComponentModel
  - System.Workflow.Runtime



Workflow Runtime Services

- Allows host to choose and extend the runtime
  - Services derive from WorkflowRuntimeService
- Key WorkflowRuntimeServices
  - WorkflowPersistence Service
  - ExternalDataExchangeService

ExternalDataExchangeService

- Allows direct communication to host
  - Hosts one or more “services” inside it

ExternalDataExchange Interfaces

- Hosted services derive from special interfaces
  - Interfaces have the ExternalDataExchange attribute
  - Interfaces contain methods and events

```
[ExternalDataExchange]
public interface ITimerService
{
    event EventHandler<TimerEventArgs> Complete;
    void StartTimer(string timerId, int delay);
}
```

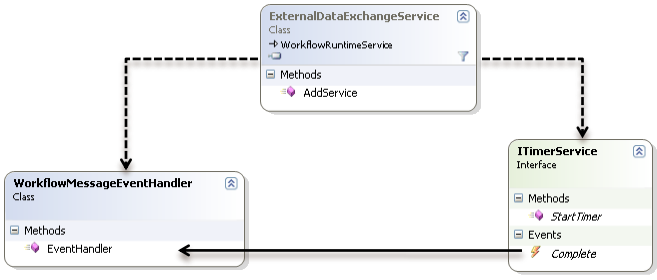
ExternalDataExchange "Services"

- Implements the ExternalDataExchange interface
  - Registered with ExternalDataExchangeService
  - Optionally derives from WorkflowRuntimeService

```
public class TimerService : ITimerService
{
    public event EventHandler<TimerEventArgs> Complete;

    public void StartTimer(string timerId, int delay)
    {
        Guid instanceId = WorkflowEnvironment.WorkflowInstanceId;

        /* Implementation omitted for clarity */
    }
}
```

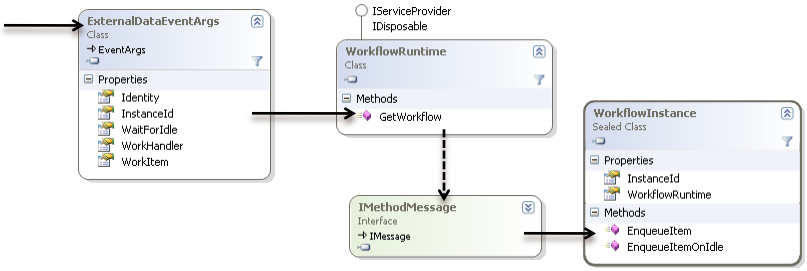

How does it work?

- ExternalDataExchangeService registers "service"
  - Registers "service" with workflow runtime
  - Handles all events in "service" interfaces



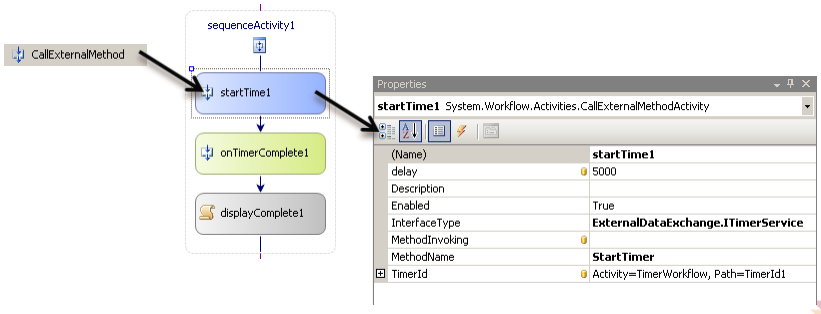
How does it work?

- Interface event routed back to WorkflowInstance
  - Event is raised with ExternalDataExchangeEventArgs
  - WorkflowInstance located using event args InstanceId
  - Internal handler converts event to IMethodMessage
  - Message sent using WorkflowInstance.EnqueueItem



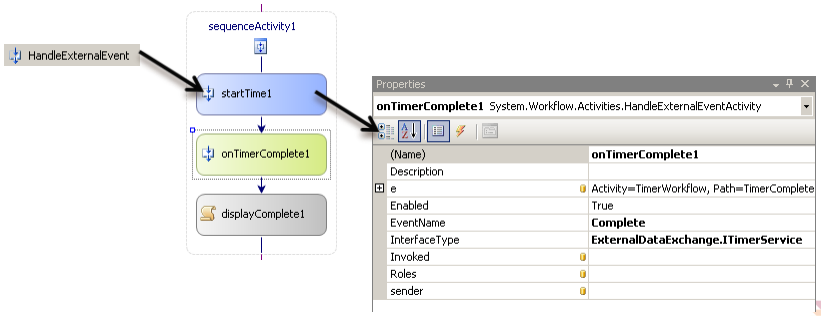
CallExternalMethodActivity

- Calls external method by defining
  - The interface to use – InterfaceType
  - The method to call – MethodName
  - Any properties defined by the method



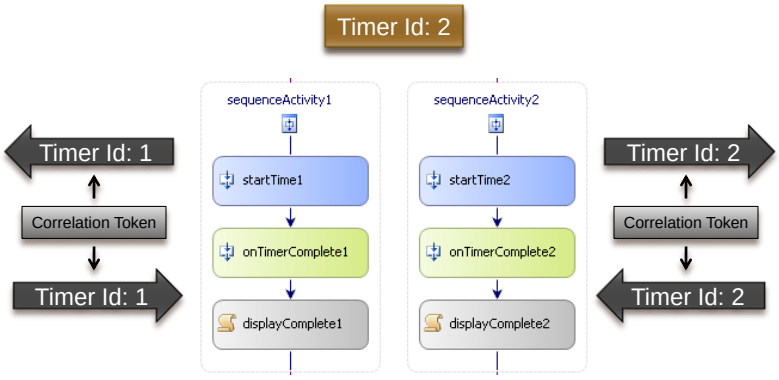
HandleExternalEventActivity

- ***Waits*** for an external event by defining:
  - The interface to use - InterfaceType
  - The event to wait for - EventName
  - Any parameters provided with the event



Correlation

- What if the workflow receives multiple events
  - Which waiting activity receives the event?
  - Some way is needed to relate an event to an activity



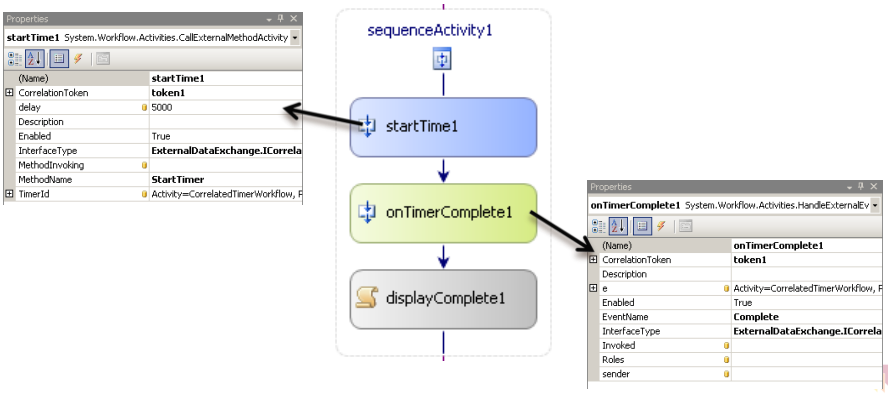
Correlation Attributes

- Attributes applied to ExternalDataExchange interface
  - Define the parameter used for correlation
  - Defines how a method initializes the correlation
  - Defines how an event relates to another method

```
[ExternalDataExchange]
[CorrelationParameter("timerId")]
public interface ICorrelatedTimerService
{
    [CorrelationAlias("timerId", "e.TimerId")]
    event EventHandler<TimerEventArgs> Complete;

    [CorrelationInitializer]
    void StartTimer(string timerId, int delay);
}
```

Using CorrelatedTokens

- Correlated activities need a way to connect
  - CorrelationInitializers create or assign tokens
  - Token used on event activities to define relationship

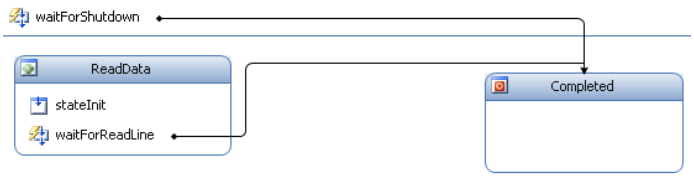


Are Sequential Workflows best?

- Human processes often:
  - Stop while waiting for user interaction
  - Start again on receipt of external events
  - Skip steps when appropriate
  - Go back to previous steps
- Failings of sequential workflow
  - Difficult to skip steps
  - Difficult to go back to previous steps

State Machine Workflow

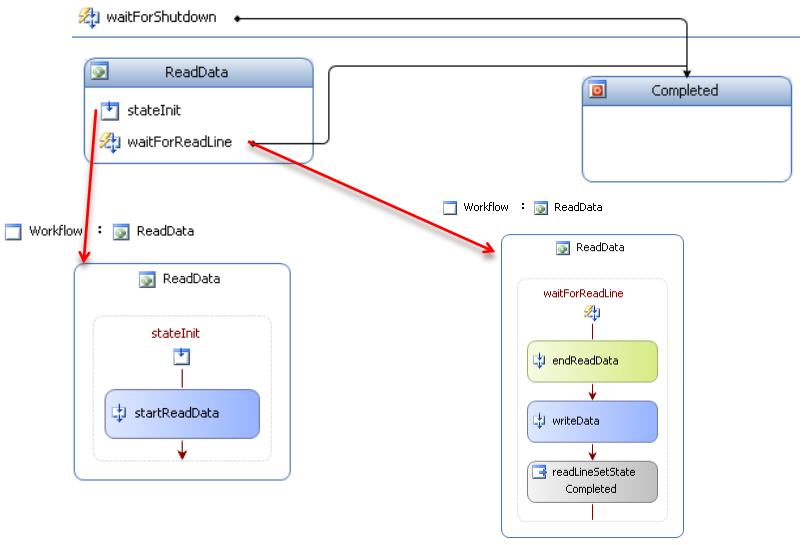
- Break processes into discrete states
  - Define events that initiate state changes
  - Execute activities on state enter and exit
  - Execute activities on transitions



Sequential vs. State Machine

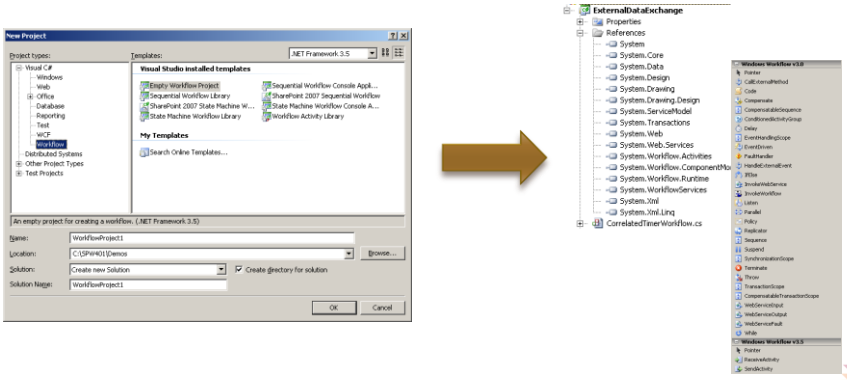
- Sequential more common for
  - Orchestration processes
  - Simple human interaction
- State machines more common for
  - Complex human interaction
  - UI Flow
- Sequential is just a special case state machine
  - Single state workflow with initiation activities

Example State Machine Workflow



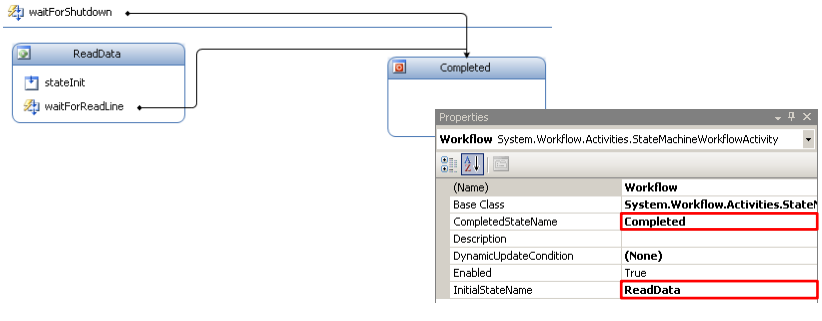
Building State Machine Workflows

- New State Machine workflow project
  - Adds a reference to the workflow assemblies
  - Adds workflow activities to the toolbox



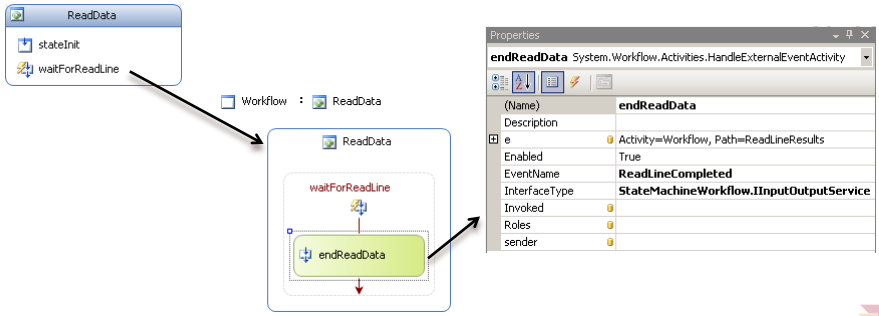
Defining the States

- Workflow composed of multiple State activities
  - State activities are available in the toolbox
  - The workflow starts in a specific start state
  - When the workflow enters the end state, it completes



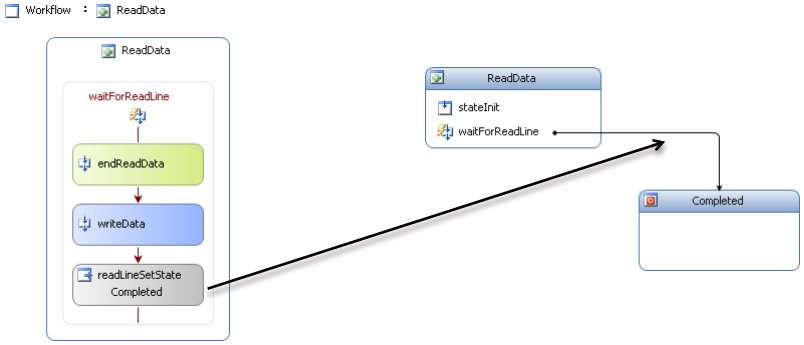
Responding to External Events

- External stimuli causes activities to execute
  - Indicated using EventDriven activities in a state
  - EventDriven activities are mini sequential workflows
  - Must begin with HandleExternalEvent activity



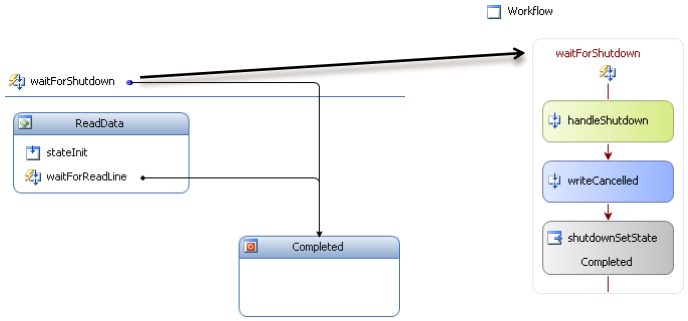
Changing States

- EventDriven activities don't change current state
  - SetState activity is used to change the state
  - Often placed in EventDriven activities
  - The state diagram will add arrows indicating transitions



Responding to Global Events

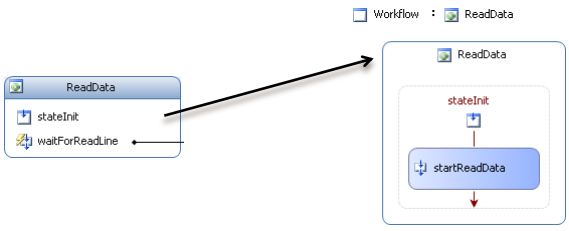
- Events are ignored if not currently listening
  - What if we want to listen for an event across all states?
- EventDriven activities can be workflow scoped
  - Event will be handled regardless of current state



State Initialization and Finalization

- State Init and Finalization tasks are tied to state
  - Both init and finalize are mini sequential workflows
  - Initialization execute on state entry
  - Finalization execute on state exit

  `Finalization will always execute on state changes`



Resource Utilization

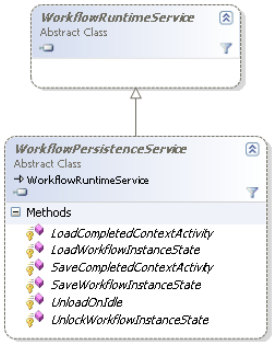
- State Machine workflows often are waiting
  - Waste of memory
  - Lack of scalability
- Memory needs to be reclaimed while waiting

Workflow Serialization

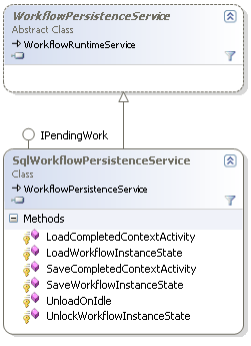
- Workflows and Activities are .NET objects
  - Marked as Serializable
  - Can be serialized using IFormatter
- Idle workflows can be serialized and unloaded
  - Workflows are idle when waiting for external event
  - Serialized workflow needs a location to be stored

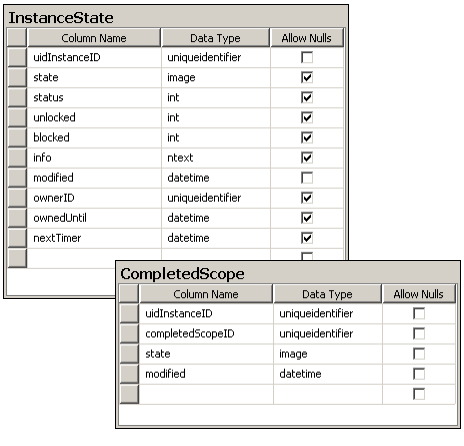
Persistence Service

- WinWF provides WorkflowPersistenceService
  - Derives from WorkflowRuntimeService
  - Provides methods to load and save workflow instances

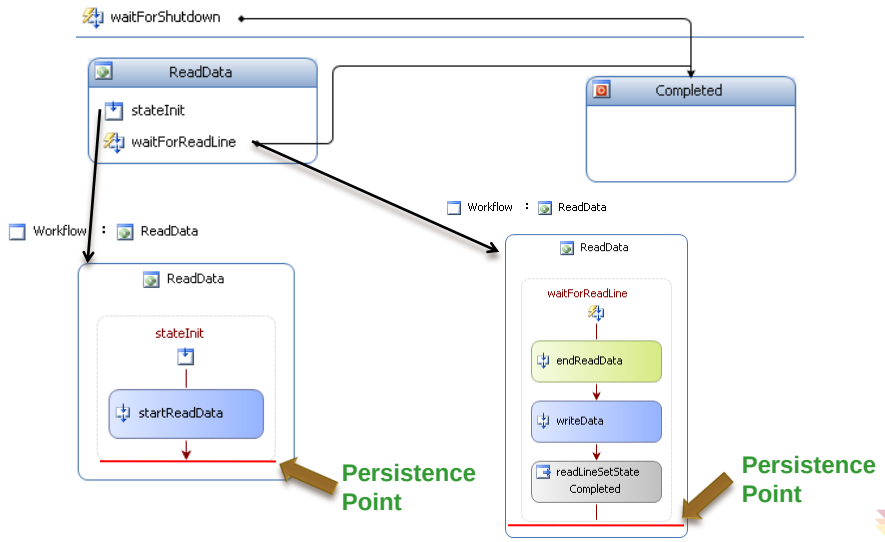


SqlWorkflowPersistenceService





When does a workflow persist?

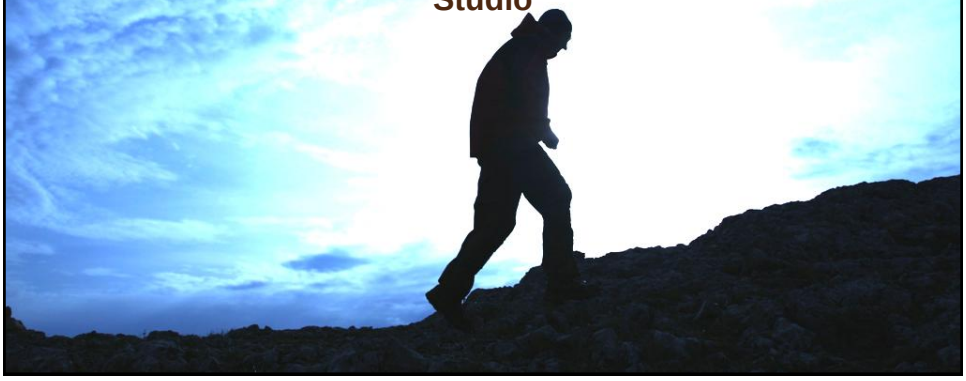


Summary

- Workflow Runtime
- WorkflowRuntimeServices
- ExternalDataExchangeService
- StateMachine Workflow
- WorkflowPersistenceService

Windows Workflow Foundation Integration with SharePoint 2007

Developing SharePoint Workflow Templates with Visual Studio

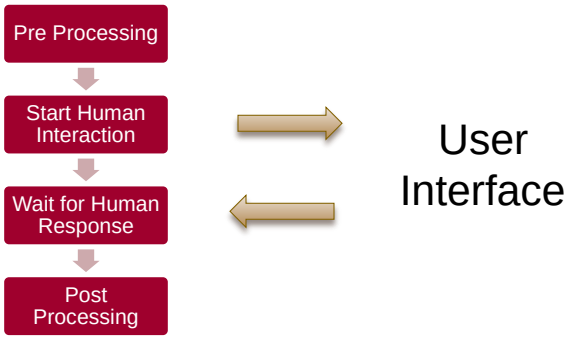


Agenda

- WSS as a Workflow UI
- Association Workflow to Lists and Content Types
- Executing Workflow in WSS
- WSS/WinWF Communication
- Integration with Office 2007

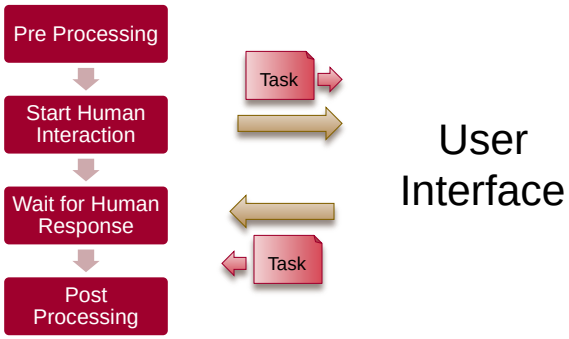
Workflow UI

- Human workflow requires process and UI
  - WinWF provides process, but no UI
  - WinWF provides external communication mechanism
  - Traditionally it's the developers job to provide UI



Workflow UI Patterns

- UI in human workflow centers around tasks
  - Process defines tasks and waits for result
  - Humans complete tasks and provide result
  - Process continues to execute

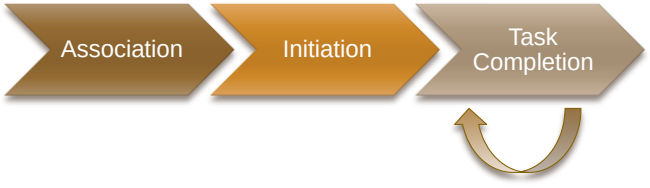


WSS as Workflow UI

- Tasks are very similar to list items
  - In fact WSS 3.0 has a list type called tasks
  - Could the tasks list be our Workflow UI?
  - Can the tasks list hold multiple types of tasks with custom UIs?
- What useful functionality do lists have in WSS
  - Custom display and edit forms
  - Potential for multiple content types with custom forms
  - Filtering to allow display of a single type of task

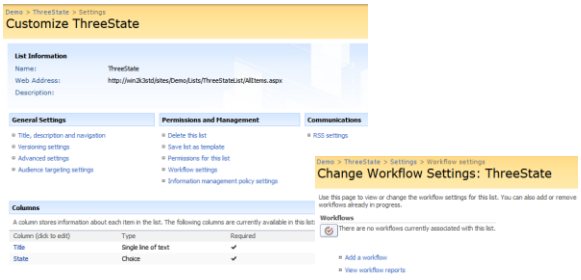
Using Workflow in WSS

- There are three primary interaction points
  - Workflow Association
  - Workflow Initiation
  - Task Completion

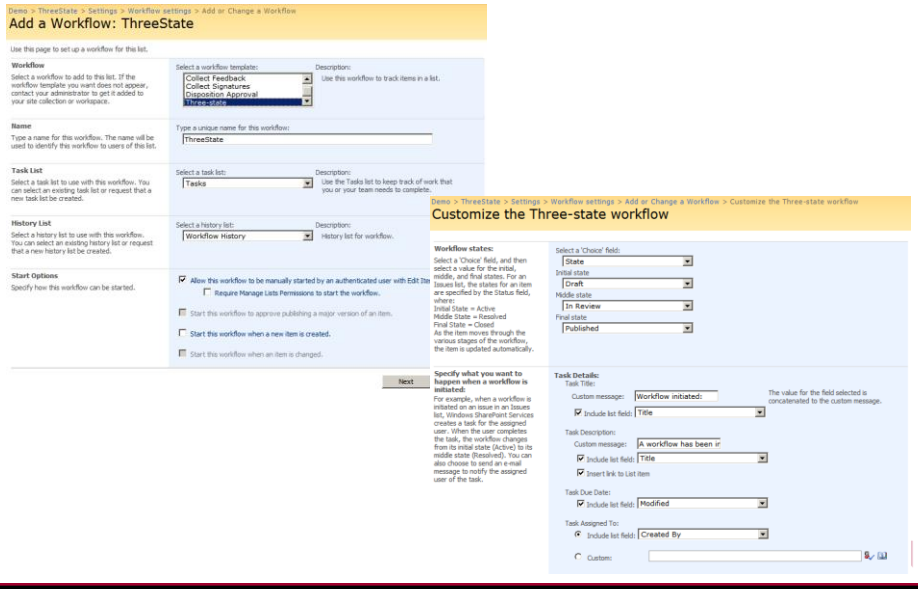


WSS Workflow Association

- Workflows must be associated with containers
  - Can be associated with lists or content types
  - Association ties a type of workflow to a set of items
  - Requires entry of workflow type specific data

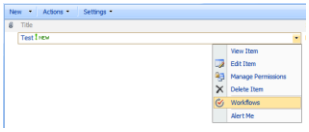


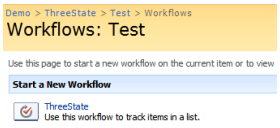
WSS Workflow Association



WSS Workflow Initiation

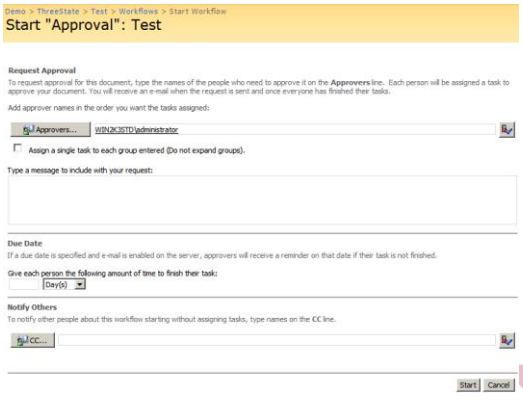
- Initiation occurs at the list item level
  - Workflow types associated with its container allowed
    `Ex. WF Types associated with content type or list`
  - Initiation can be automatic (when item created edited)
  - Initiation can be manual (anyone or only editors)





WSS Workflow Initiation UI

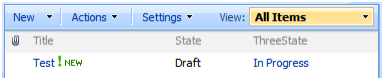
- When a workflow instance is initiated
  - User has a chance to override instance settings
  - Done using UI specified by workflow type

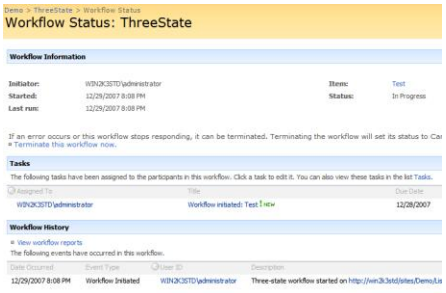


WSS Workflow Status

- Once workflow is running, status accessed via list
  - List column added showing WF instance current state
  - Links to a special status page showing

    `Tasks related to workflow instance`
    `History information logged by workflow instance`



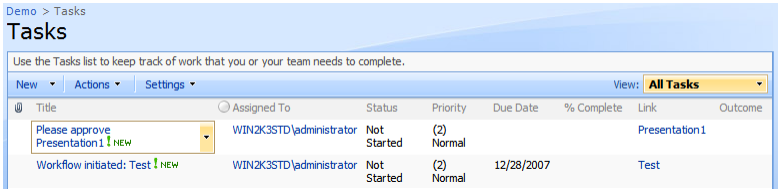


WSS Workflow Modification

- What if a workflow needs to be changed?
  - The list of approvers changes
  - The approval process becomes more urgent?

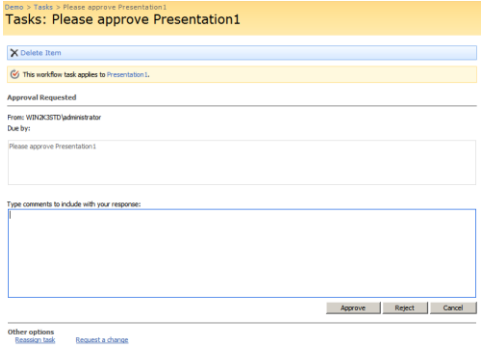
Using WSS Tasks

- Tasks are the key WF/UI interaction point
  - Process defines tasks that the user can complete
  - Tasks are stored in a standard list with custom forms
  - The status UI displays a filtered view of the tasks



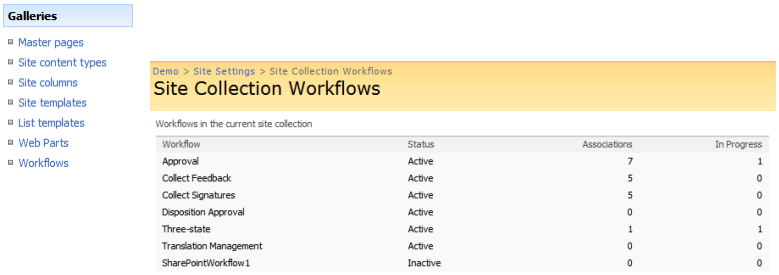
Completing WSS Tasks

- Content types allow multiple forms per list
  - Each workflow can have its own task content type
  - Each content type has it's own custom forms
  - Custom forms have 1 to 1 relationship with aspx pages



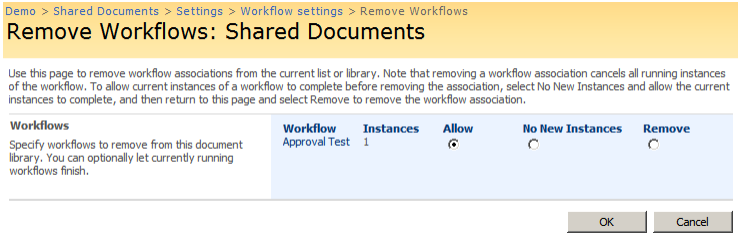
Viewing Current Workflow Tasks

- Report page displays workflow types
  - Located in site settings at site collection level
  - Displays list of all workflow types
  - Displays number of associations and instances



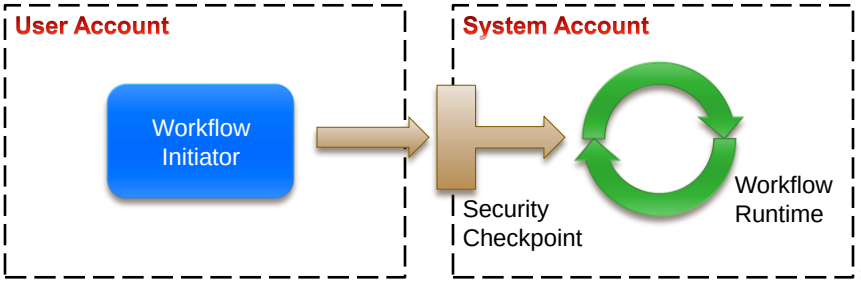
Un-associating a Workflow Type

- Removal of a workflow type done in three steps
  - Remove the ability to start new instances
  - Wait until all instances are complete
  - Remove the workflow association
- It is possible to terminate current instances



WSS Workflow Security

- Workflow runs under worker process identity
  - Instances may have more permissions than initiator
    `Initiator may also be an event, not a user`
  - Security is necessary on the workflow initiation
    `Tight controls on who can start a workflow`

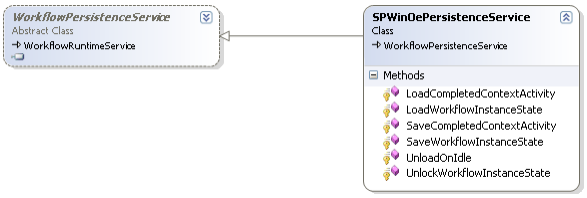


How does this all work?

- WSS integrates with WinWF two ways
  - WSS hosts the WorkflowRuntime class
  - WSS provides custom services for the runtime
    `SPWinOePersistenceService`
    `SPWinOETaskService`
    `SPWinOEWSSService`

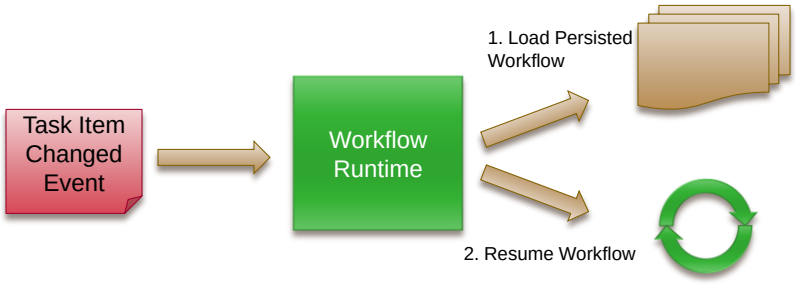
Dehydration of workflow instances

- Workflow persisted while waiting on tasks
  - Running workflow is serialized into content database
  - Persistence service chosen in code
  - Can't be overridden in config file



Rehydration of workflow instances

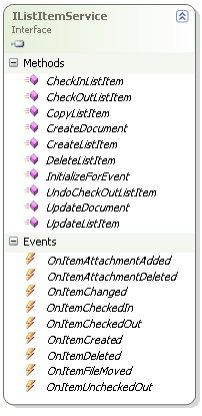
- Workflow deserialized when tasks change
  - Tasks are just list items
  - SPItemEventReceiver fires the onUpdated event
  - Workflow event receivers deserialize and start workflow

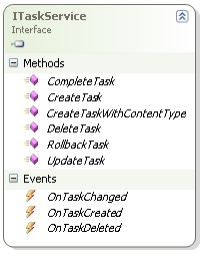


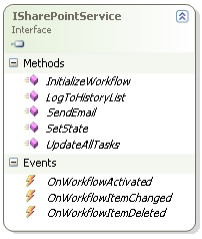
How are events delivered?

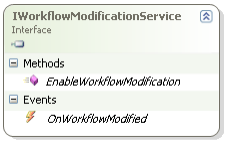
- Done using ExternalDataExchange
  - WSS Defines a set of ExternalDataExchange interfaces
  - Implements the interfaces in special service
  - WSS provides a set of activities to use interfaces

ExternalDataExchange Interfaces



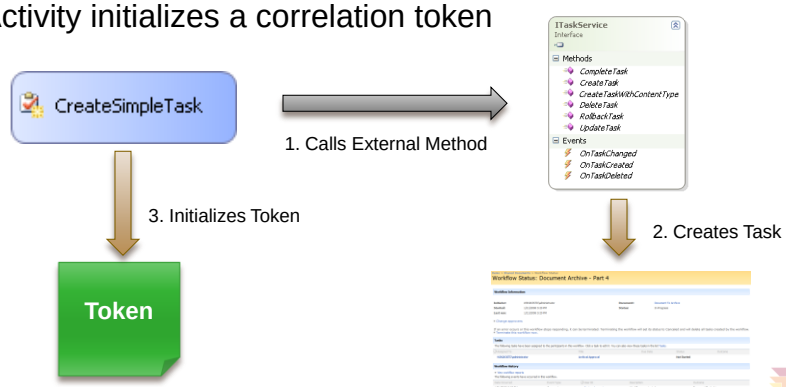






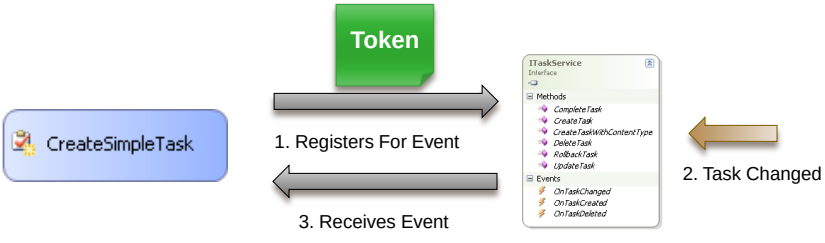
Task Creation Scenario

- Workflow instance uses CreateTask activity
  - Workflow calls method in ITaskService interface
  - Task is created in the Tasks list
  - Activity initializes a correlation token



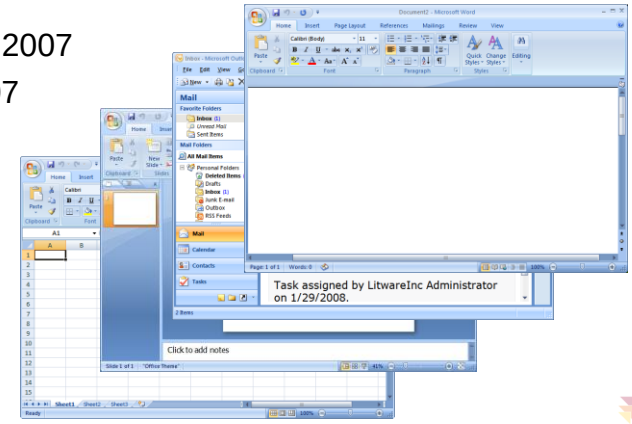
Task Completion Scenario

- WSS Event handler sends event to workflow
  - Event is converted into an ITaskService event
  - Message is correlated to a previous CreateTask call
  - Event is routed back to the correct activity and workflow



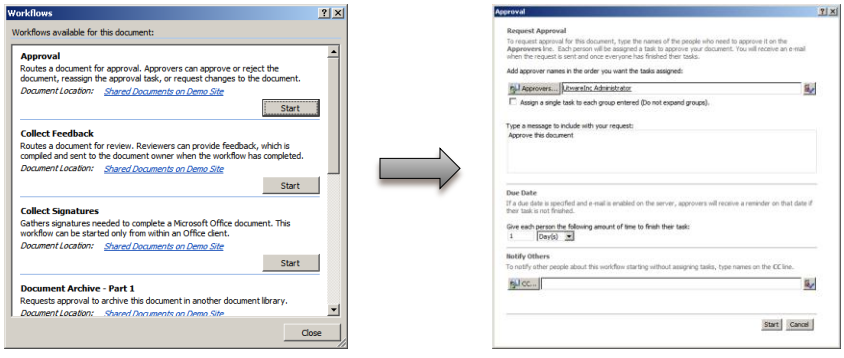
Office 2007 Integration

- Office Applications that support workflow
  - Word 2007
  - Excel 2007
  - PowerPoint 2007
  - Outlook 2007



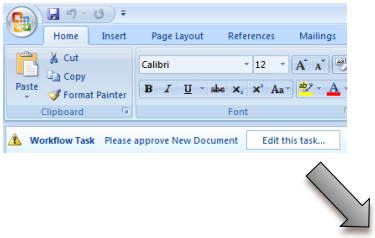
Starting Workflow via Office 2007

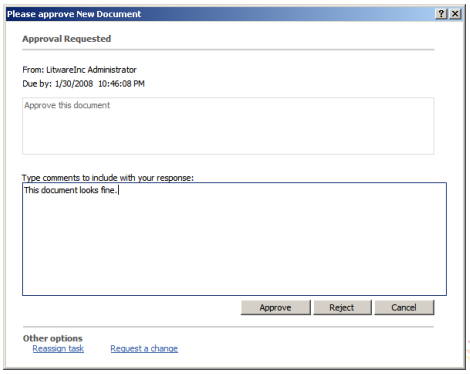
- Office applications allow workflow to be started
  - Can only start workflows previous associated
  - Office hosts initiation page in embedded browser



Acting on Tasks via Office 2007

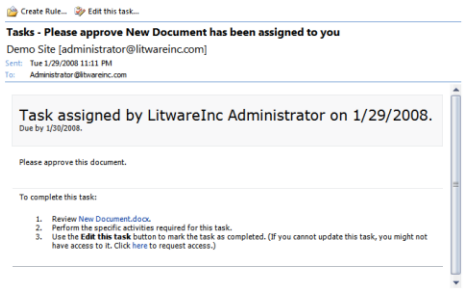
- Office can find any tasks for open item
  - Display notification for the user
  - Can host task edit form in embedded host

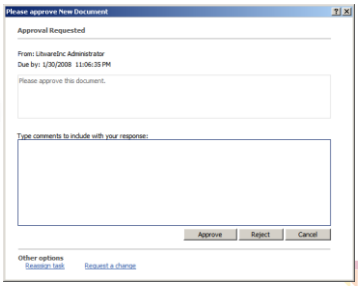




Receiving Tasks via Outlook 2007

- Workflow status received via Email
  - Tasks sent as an email to all recipients
  - Hosts task edit form in embedded host





Summary

- WSS as a Workflow UI
- Association Workflow to Lists and Content Types
- Executing Workflow in WSS
- WSS/WinWF Communication
- Integration with Office 2007

Agenda

- Overview of Workflow API
- Associating workflow templates
- Initiating workflows
- Accessing running instances using the API

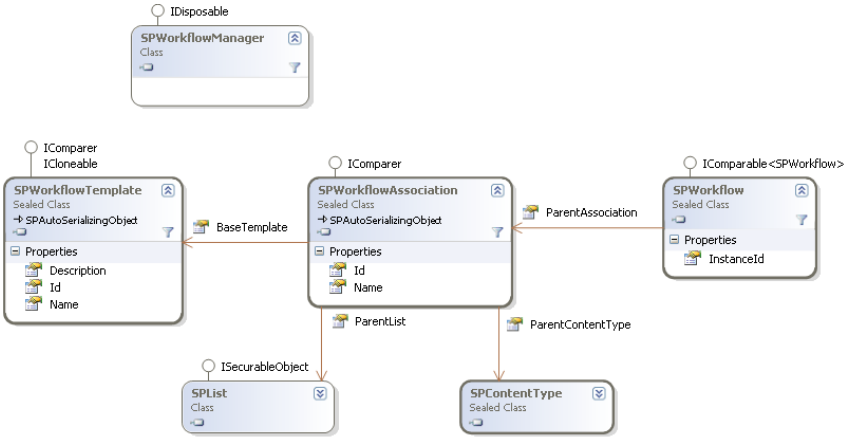
Why do we need Worflow APIs?

- SharePoint allows custom workflows and forms
  - This means developers must create custom UIs
  - Custom UIs require access to workflow concepts
- APIs used by SharePoint UIs are available to developers

Managing Workflow via Code

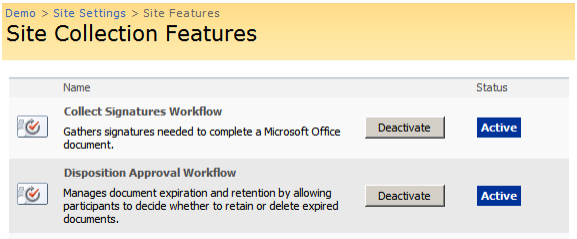
- SharePoint's workflow UIs all use standard API
  - Necessary to allow third party extensions to workflows
  - Used in custom management forms and web parts
  - Used in custom workflow forms
- These APIs allow:
  - Manual workflow template association
  - Manual workflow initiation
  - Manual workflow status
  - Manual workflow modification

Workflow API Objects



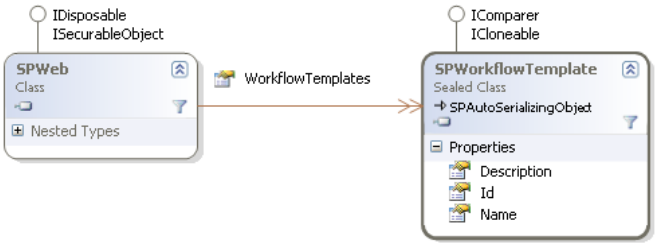
Workflow Templates

- Define a type of workflow with no relationships
  - Installed via features and associated with site collection
  - Used as the template when associating workflows
  - Can be attached to lists or content types



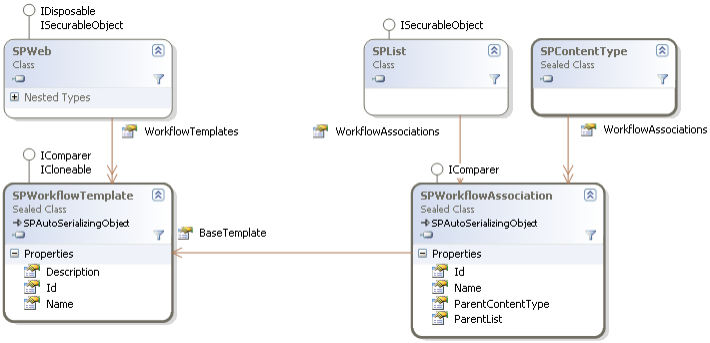
Workflow Templates

- Can be referenced from the SPSite object
  - Can be listed using SPWeb.WorkflowTemplates
  - Can be filtered using SPWorkflowManager
  - No ability to add or remove



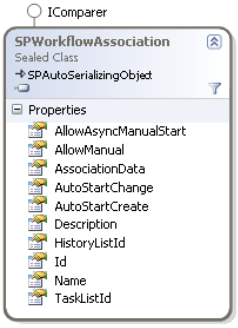
Associating Workflow Templates

- Workflows can't be started until associated
  - Association ties SPWorkflowTemplate to container
  - Association connects SPList or SPContentType
  - Holds association specific data



SPWorkflowAssociation

- Association defined by SPWorkflowAssociation
  - Defines workflow association name
  - Defines startup conditions
    `Auto/Manual Startup`
  - Defines association data
  - Defines the tasks and history lists



Association Data

- Associated Workflows need parameters
  - Differentiates one association to another
    `Ex. Who is the default approver for this process?`
  - Set using a custom string called association data
    `Often formatted as XML`
  - SPWorkflowAssociation.AssociationData stores string

Association Data Example

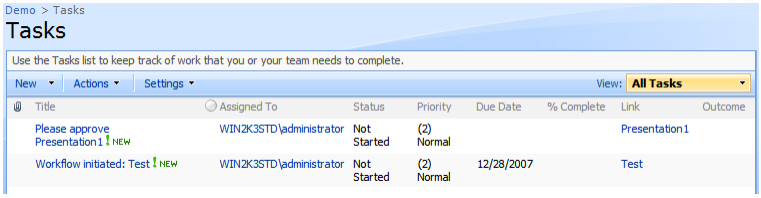
```
<my:myFields xml:lang="en-us" xmlns:xsi="…" xmlns:my="…">
  <my:Reviewers>
    <my:Person>
      <my:DisplayName></my:DisplayName>
      <my:AccountId></my:AccountId>
      <my:AccountType></my:AccountType>
    </my:Person>
  </my:Reviewers>
  <my:CC></my:CC>
  <my:DueDate xsi:nil="true"></my:DueDate>
  <my:Description></my:Description>
  <my:Title></my:Title>
  <my:DefaultTaskType>1</my:DefaultTaskType>
  <my:CreateTasksInSerial>true</my:CreateTasksInSerial>
  <my:AllowDelegation>true</my:AllowDelegation>
  <my:AllowChangeRequests>true</my:AllowChangeRequests>
  <my:StopOnAnyReject xsi:nil="true"></my:StopOnAnyReject>
  <my:WantedTasks xsi:nil="true"></my:WantedTasks>
  <my:SetMetadataOnSuccess>false</my:SetMetadataOnSuccess>
  <my:MetadataSuccessField></my:MetadataSuccessField>
  <my:MetadataSuccessValue></my:MetadataSuccessValue>
  <my:ApproveWhenComplete>false</my:ApproveWhenComplete>
  <my:TimePerTaskVal xsi:nil="true"></my:TimePerTaskVal>
  <my:TimePerTaskType xsi:nil="true"></my:TimePerTaskType>
  <my:Voting>false</my:Voting>
  <my:MetadataTriggerField></my:MetadataTriggerField>
  <my:MetadataTriggerValue></my:MetadataTriggerValue>
  <my:InitLock>false</my:InitLock>
  <my:MetadataStop>false</my:MetadataStop>
  <my:ItemChangeStop>false</my:ItemChangeStop>
  <my:GroupTasks>false</my:GroupTasks>
</my:myFields>
```

Tasks and History lists

- Running workflow needs to store information
  - Tasks to define user interaction
  - History to log key events
  - All stored in special SharePoint lists

Task Lists

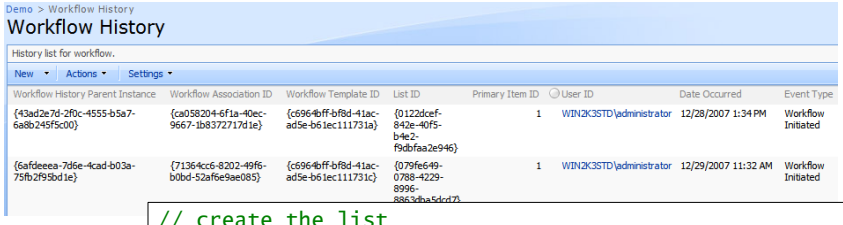
- List defined using the Tasks list template
  - List type of SPListTemplateType.Tasks
  - List template Id of 107




```
// create the list
Guid listId = m_web.Lists.Add(
    name, string.Empty, SPListTemplateType.Tasks);

// commit the changes
m_web.Update();
```

History Lists

- List defined using the WorkflowHistory template
  - List type of SPListTemplateType.WorkflowHistory
  - List template Id of 140




```
// create the list
Guid listId = m_web.Lists.Add(
    name, string.Empty, SPListTemplateType. WorkflowHistory);

// commit the changes
m_web.Update();
```


Creating Workflow Associations

- Done using SPList.AddWorkflowAssociation
  - Verify tasks and history lists are available
  - Find the workflow template to use
  - Create a new SPWorkflowAssociation

    Optionally assign new association data

```
// create the association
SPWorkflowAssociation result =
    SPWorkflowAssociation.CreateListAssociation(template,
txtName.Text,
        taskList, historyList);
result.AssociationData = associationData;

// add the association to the list
list.AddWorkflowAssociation(result);
list.Update();
```


Starting workflow instances

- All running workflows are tied to a list item
  - Workflows available tied to item's list or content type
  - Start using SPWorkflowManager.StartWorkflow

    Requires list item, association, and init data

```
SPListItem listItem = ...;
SPWorkflowAssociation association =
    list.WorkflowAssociations.
        GetAssociationByName("Approval",
            CultureInfo.CurrentCulture);
SPWorkflow workflow =
    site.WorkflowManager.StartWorkflow(
        listItem, association, initiationData);
```

Initiation Data

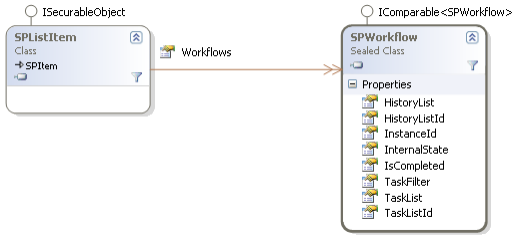
- Workflow instances require parameters as well
  - Just like association data in concept
  - Custom string passed directly to the workflow
    `Often formatted XML`

Initiation Data Example

```
<my:myFields xml:lang="en-us" xmlns:xsi="…" xmlns:my="…">
  <my:Reviewers>
    <my:Person>
      <my:DisplayName>WIN2K3STD\administrator</my:DisplayName>
      <my:AccountId>WIN2K3STD\administrator</my:AccountId>
      <my:AccountType>User</my:AccountType>
    </my:Person>
  </my:Reviewers>
  <my:CC></my:CC>
  <my:DueDate xsi:nil="true"></my:DueDate>
  <my:Description></my:Description>
  <my:Title></my:Title>
  <my:DefaultTaskType>1</my:DefaultTaskType>
  <my:CreateTasksInSerial>true</my:CreateTasksInSerial>
  <my:AllowDelegation>true</my:AllowDelegation>
  <my:AllowChangeRequests>true</my:AllowChangeRequests>
  <my:StopOnAnyReject xsi:nil="true"></my:StopOnAnyReject>
  <my:WantedTasks xsi:nil="true"></my:WantedTasks>
  <my:SetMetadataOnSuccess>false</my:SetMetadataOnSuccess>
  <my:MetadataSuccessField></my:MetadataSuccessField>
  <my:MetadataSuccessValue></my:MetadataSuccessValue>
  <my:ApproveWhenComplete>false</my:ApproveWhenComplete>
  <my:TimePerTaskVal xsi:nil="true"></my:TimePerTaskVal>
  <my:TimePerTaskType xsi:nil="true"></my:TimePerTaskType>
  <my:Voting>false</my:Voting>
  <my:MetadataTriggerField></my:MetadataTriggerField>
  <my:MetadataTriggerValue></my:MetadataTriggerValue>
  <my:InitLock>false</my:InitLock>
  <my:MetadataStop>false</my:MetadataStop>
  <my:ItemChangeStop>false</my:ItemChangeStop>
  <my:GroupTasks>false</my:GroupTasks>
</my:myFields>
```

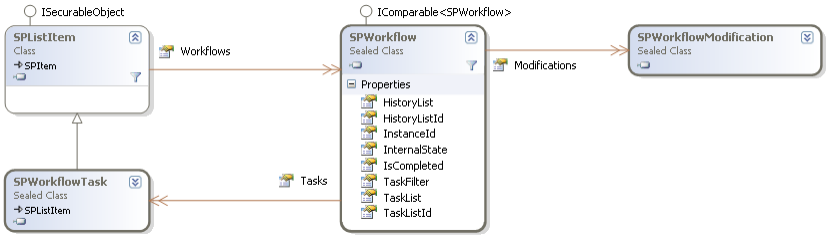
SPWorkflow Object

- Running workflows represented by SPWorkflow
  - Can be found in SPListItem.Workflows list
  - Can also be found using SPWorkflowManager
  - Provides access to state, tasks, and other details



Workflow Status

- Once the workflow is running, status is available
  - SPWorkflow object allows status access
  - Can provide basic workflow stats
  - Can provide a list of tasks and history events
  - Can provide access to available modifications

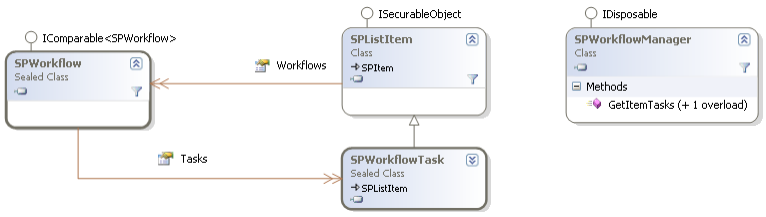


Current Tasks

- Tasks related to the workflow are provided
  - Accessible using tasks list and filter
  - List of tasks available via

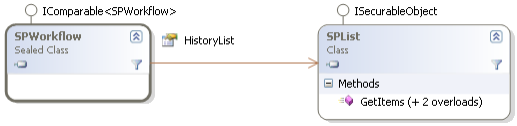
    `SPWorkflow.Tasks`

    `SPWorkflowManager.GetItemTasks`



Current History

- History related to the workflow is provided
  - No simple property that filters the history list
  - Accessible using history list and query




```
<Where>
  <Eq>
    <FieldRef Name="WorkflowInstance“ />
    <Value Type="Text">
     {00000000-0000-0000-0000-000000000000}
    </Value>
  </Eq>
</Where>
```


Current History Query

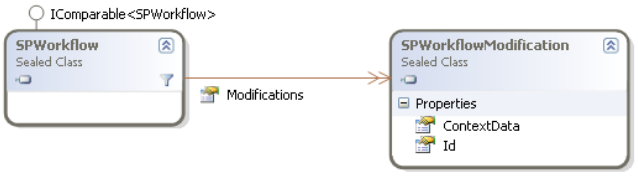
- Get history for workflow using SPList.GetItems

```
SPQuery query = new SPQuery();
query.Query =
    "<Where>" +
      "<Eq>" +
        "<FieldRef Name=\"WorkflowInstance\" />" +
        "<Value Type=\"Text\">" +
          workflow.InstanceId.ToString() +
        "</Value> " +
      "</Eq>" +
    "</Where>";
SPListItemCollection history =
    workflow.HistoryList.GetItems(query);
```

Workflow Modifications

- Available modifications are stored in SPWorkflow
  - Accessed via SPWorkflow.Modifications
  - Stored in SPWorkflowModification class

    Only contains ID and ContextData



Displaying Available Modifications

- SPWorkflowModification only contains an Id
  - Id represents a predefined modification point
  - Metadata describing modification stored in template
  - Accessed using specially formatted property names

    `Name – Modification_`*`ModID`*`_Name`

```
SPWorkflow workflow = …;
SPWorkflowTemplate template =
    workflow.ParentAssociation.BaseTemplate;

SPWorkflowModification modification;
foreach (modification in workflow.Modifications)
    Console.WriteLine(
        template[
            "Modification_" + modification.Id.ToString() + "_Name"]);
```


SPWorkflowModification ContextData

- Once a modification is made changes are stored
  - SPWorkflowModification.ContextData stores changes
  - Up to the form implementer to store changes
  - Up to the workflow implementer to use changes

Updating Workflow Associations

- Done using UpdateWorkflowAssociation method
  - Available in SPList and SPContentType

```
SPWorkflowAssociation association = …;

// update the association
association.ParentList.
  UpdateWorkflowAssociation(association);
```


Removing Workflow Association

- Done using RemoveWorkflowAssociation method
  - Available in SPList and SPContentType

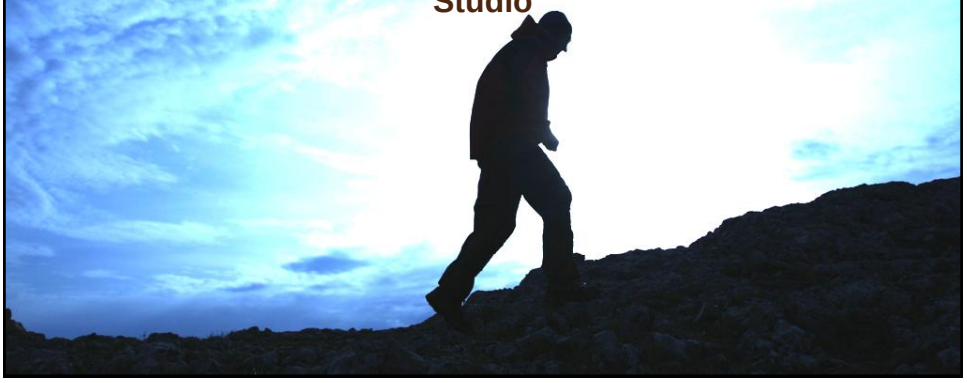
```
SPWorkflowAssociation association = …;

// remove the association
association.ParentList.RemoveWorkflowAssociation(association);
association.ParentList.Update();
```

Summary

- Overview of Workflow API
- Associating workflow templates
- Initiating workflows
- Accessing running instances using the API

Developing SharePoint Workflow Templates with Visual Studio 2008

Developing SharePoint Workflow Templates with Visual Studio

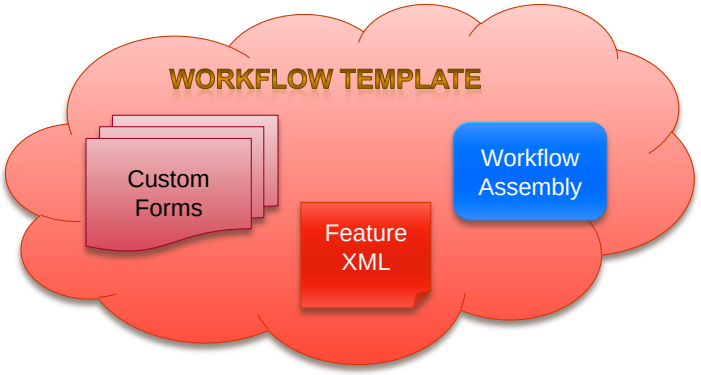


Agenda

- Create a simple workflow in VS 2008
  - Installing workflow using features
- Pass startup parameters to workflows
- Build state machine workflows in WSS
- Handle item change events in workflow

WSS Workflow Templates

- Made up of several parts
  - Workflow assembly
  - Feature definition
  - Workflow forms



What makes up a workflow assembly?

- Workflow assembly contains all used by workflow
  - Workflow classes
  - Feature activation handlers
  - Code behind for forms
  - Any other supporting code
- Workflow assembly must be deployed in GAC

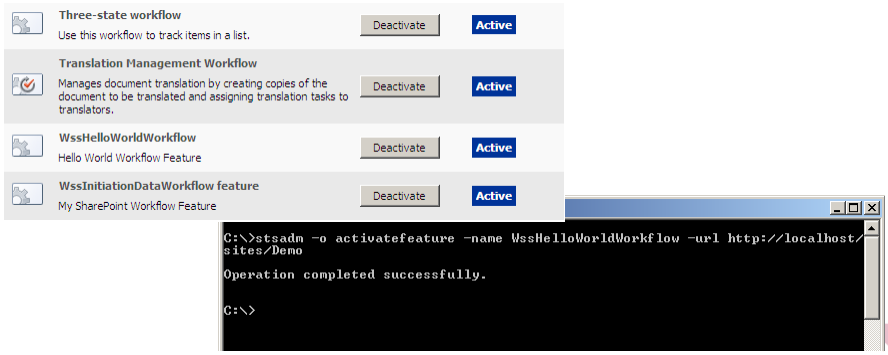
Packaging Workflow Templates

- Feature definition packages assembly and forms
  - Feature scoped at Site level
  - Workflow element ties a name to workflow class
  - Also Defines any form overrides

```
<Workflow
   Name="Hello World Workflow"
   Description="My Hello World Workflow"
   Id="59dd0352-c8be-4e08-9e0e-3c6a91fa18ec"
   CodeBesideClass="WssHelloWorldWorkflow.HelloWorld"
   CodeBesideAssembly="WssHelloWorldworkflow, ...">
  <Categories />
  <MetaData />
</Workflow>
```

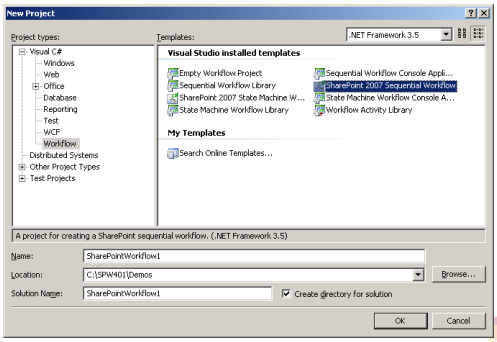
Installing Workflow Template

- Feature activation adds workflow templates
  - Activated via UI or stsadm
  - Represented as SPWorkflowTemplate
  - Related to SPWebs within the site feature activated in



Visual Studio 2008 WSS Workflow

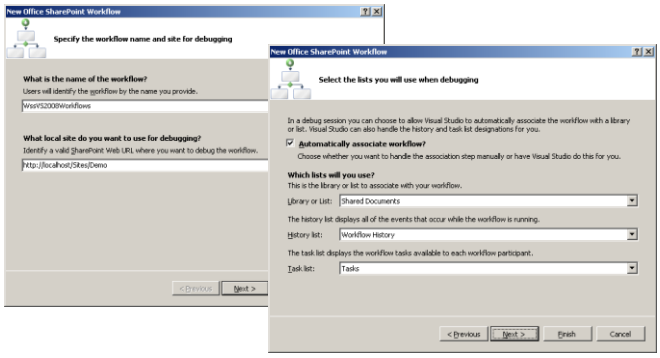
- Visual Studio 2008 adds WSS Workflow Project
  - Provides feature related files
  - Provides facilities to automatically deploy workflows
  - Automatically associates workflow to a list



Creating VS2008 Workflow Project

- Requires key information about workflow
  - Workflow name and site path required
  - Allows automatic association to an existing list

    `Requires list name, task list, history list`



VS2008 Workflow Project

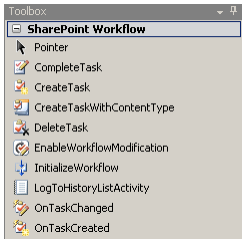
- Completed project is WSS Specific
  - Contains Feature.xml and Workflow.xml
  - Contains user properties in Project.csproj.user

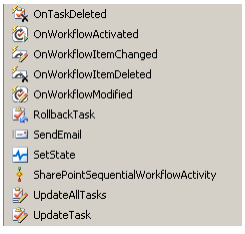
    Used to deploy and associate the workflow

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ProjectExtensions>
    <VisualStudio>
      <FlavorProperties GUID="{…}" xmlns="">
        <DisplayName>WSSHelloWorldWorkflow</DisplayName>
        <SiteURL>http://…/Docs</SiteURL>
        <ListURL>http://…/Docs/Documents/Forms/AllItems.aspx</ListURL>
        <TargetList Id="{…}">Documents</TargetList>
        <HistoryList Id="{…}">Workflow History</HistoryList>
        <TaskList Id="{…}">Tasks</TaskList>
        ...
      </FlavorProperties>
    </VisualStudio>
  </ProjectExtensions>
</Project>
```

SharePoint Workflow Activities

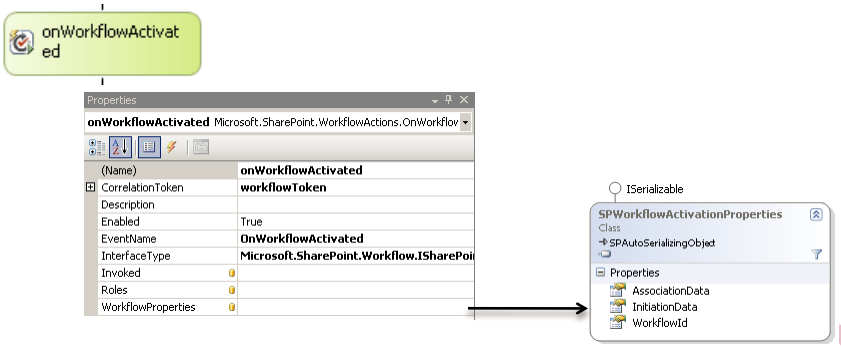
- Integration with SharePoint done via activities
  - Communicate with host via ExternalDataExchange
  - Focused around workflow, task, and workflow item





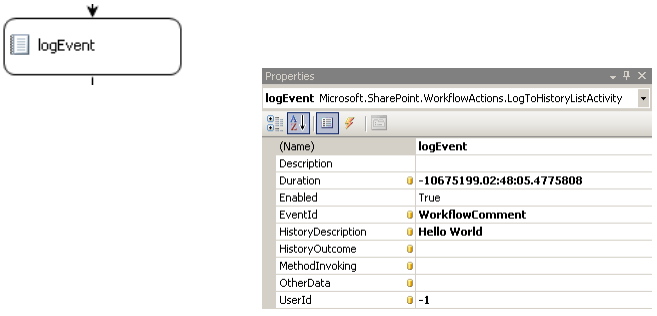
OnWorkflowActiviated Activity

- All SharePoint workflows start with this activity
  - Derived from HandleExternalEventActivity
  - Used to receive SPWorkflowActivationProperties



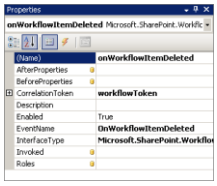
LogToHistoryList Activity

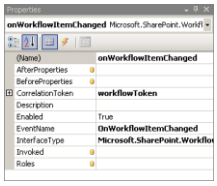
- Used to log status information to the host
  - Implemented like CallExternalMethod
  - Used to log an entry to the workflow's history list



Other Workflow Item Events

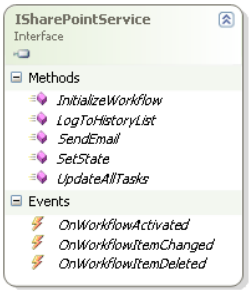
- Workflow can also wait for the item to change
  - OnWorkflowItemChanged received on item change
  - OnWorkflowItemDeleted received on item delete
  - Both contain a Hashtable of before and after properties





How Workflow Communication Works

- Communication done using ISharePointService
  - Implemented by SPWinOEWSSService
  - Provides events needed for workflow and item activities
  - Also provides methods for other communication



Passing Data to Workflow Instances

- Done using AssociationData and InitiationData
  - AssociationData assigned at workflow association
  - InitiationData assigned at workflow start
  - Both strings, often passed as xml

```
<my:myFields xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:my="http://schemas.microsoft.com/office/infopath/2003/myXSD">
  <my:Reviewers>
    <my:Person>
      <my:DisplayName>WIN2K3STD\administrator</my:DisplayName>
      <my:AccountId>WIN2K3STD\administrator</my:AccountId>
      <my:AccountType>User</my:AccountType>
    </my:Person>
  </my:Reviewers>
  <my:CC></my:CC>
  <my:DueDate xsi:nil="true"></my:DueDate>
  ...
</my:myFields>
```

Building InitiationData

- Often formatted using serialized classes
  - Define classes storing initiation data
  - Use XmlSerializer to convert classes to XML

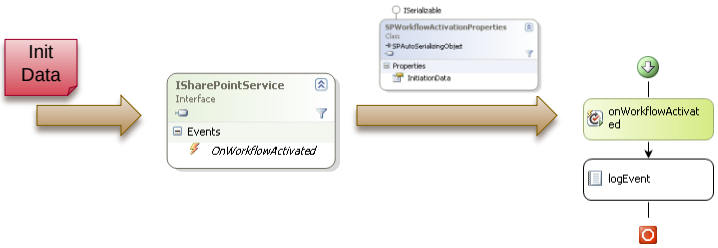
```
[Serializable]
public class StartupData
{
    public string Message { get; set; }
    public static string Serialize(StartupData value)
    {
        XmlSerializer serializer =
            new XmlSerializer(typeof(StartupData));
        using (StringWriter writer = new StringWriter())
        {
            serializer.Serialize(writer, value);
            return writer.ToString();
        }
    }
}
```

Sending InitiationData

- Sent using SPWorkflowManager.StartWorkflow
  - String representing serialized initiation data sent
  - Starts workflow instance and sends data via event

Receiving InitiationData

- Received via OnWorkflowActivated activity
  - First activity in workflow is OnWorkflowActivated
  - Receives event from the host containing initiation data



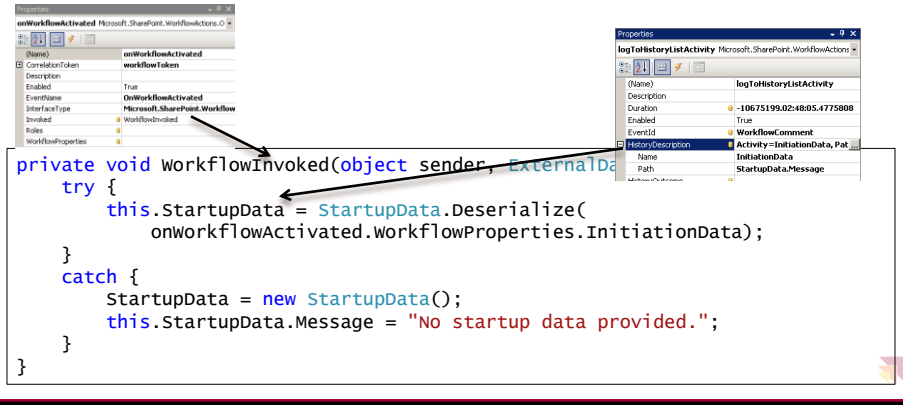
Deserializing InitiationData

- InitiationData received as string and deserialized
  - String data deserialized via XmlSerializer
  - Converts string xml into classes

```
[Serializable]
public class StartupData
{
    public string Message { get; set; }

    public static StartupData Deserialize(string value)
    {
        XmlSerializer serializer =
            new XmlSerializer(typeof(StartupData));
        using (StringReader reader = new StringReader(value))
            return serializer.Deserialize(reader) as StartupData;
    }
}
```

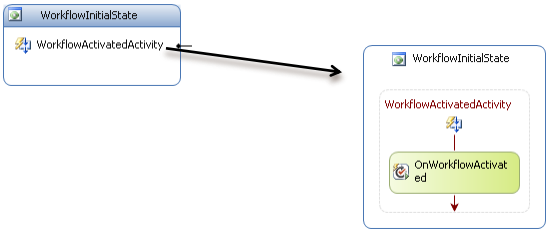
Verifying InitiationData

- Write initiation parameters to the event log
  - Receive and deserialize initiation data
  - Write data to the event log



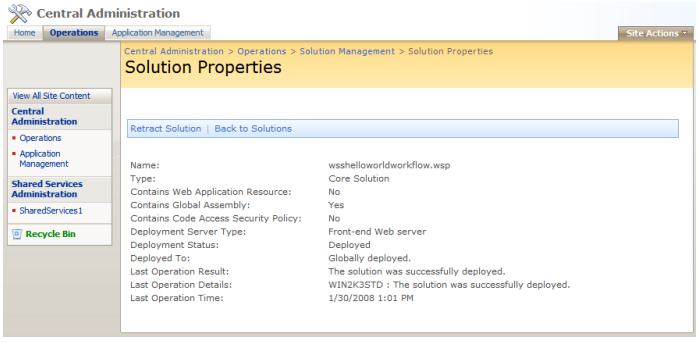
Creating WSS State Machines

- State Machines can be used in WSS
  - Same requirements as SequentialWorkflows exists
  - Workflows start with OnWorkflowActivated activity
  - Usually done with an EventDriven activity in start state



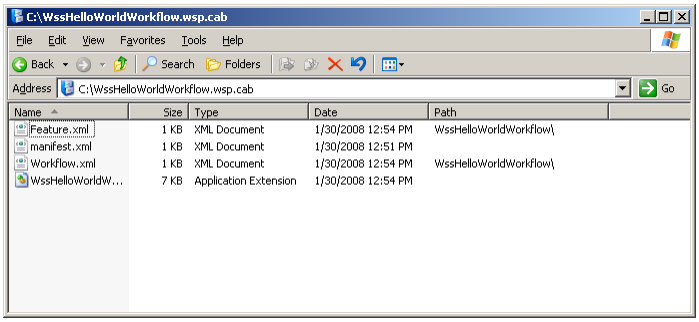
Deploying Workflows

- Features don't handle entire deployment
  - Need a way to copy files, register files in GAC
  - Solution packages are the solution
  - Installed using stsadm



Solution Package structure

- Solution packages are single .wsp files
  - Just renamed .cab files
  - Cab files contain all needed files and a manifest file
  - Manifest file contains instructions to install package



Workflow Solution Package

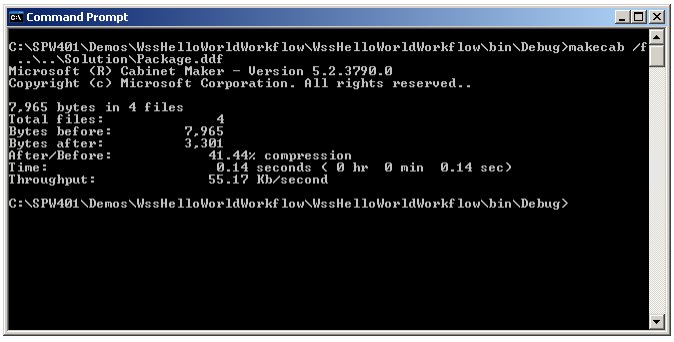
- Manifest contains references to all files
  - Installs DLL in the GAC
  - Copies .aspx files to Layouts folder
  - Copies features into Features folder
  - Installs features

Workflow Solution Manifest

```
<?xml version="1.0" encoding="utf-8" ?>
<Solution SolutionId="189c9a36-a1a1-4f76-a36c-fbba6cb99fdb"
          xmlns="http://schemas.microsoft.com/sharepoint/">
  <FeatureManifests>
    <FeatureManifest Location="WssHelloWorldWorkflow\Feature.xml" />
  </FeatureManifests>
  <Assemblies>
    <Assembly Location="WssHelloWorldWorkflow.dll“
              DeploymentTarget="GlobalAssemblyCache"/>
  </Assemblies>
</Solution>
```


Creating the solution Package

- Packaged into cab using makecab.exe
  - Requires DDF file
  - DDF file contains list of files to package
  - Each file contains a source and destination path



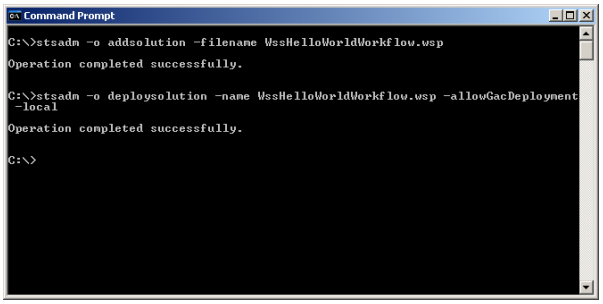
Workflow Solution DDF

```
.OPTION EXPLICIT     ; Generate errors
.Set CabinetNameTemplate=WssHelloWorldWorkflow.wsp
.set DiskDirectoryTemplate=CDROM ; All cabinets go in a single directory
.Set CompressionType=MSZIP;** All files are compressed in cabinet files
.Set UniqueFiles="ON"
.Set Cabinet=on
.Set DiskDirectory1=Package

..\..\Solution\Manifest.xml manifest.xml

..\..\Template\Features\WssHelloWorldWorkflow\Feature.xml
   WssHelloWorldWorkflow\Feature.xml
..\..\Template\Features\WssHelloWorldWorkflow\Workflow.xml
   WssHelloWorldWorkflow\Workflow.xml

WssHelloWorldWorkflow.dll WssHelloWorldWorkflow.dll
```

Installing the solution package

- Installation is made up of two steps
  - stsadm –o addsolution –filename Package.wsp
  - stsadm –o deploysolution –name Package.wsp

    Use -allowgacdeployment to authorize GAC install
    Use -local or -immediate to deploy now

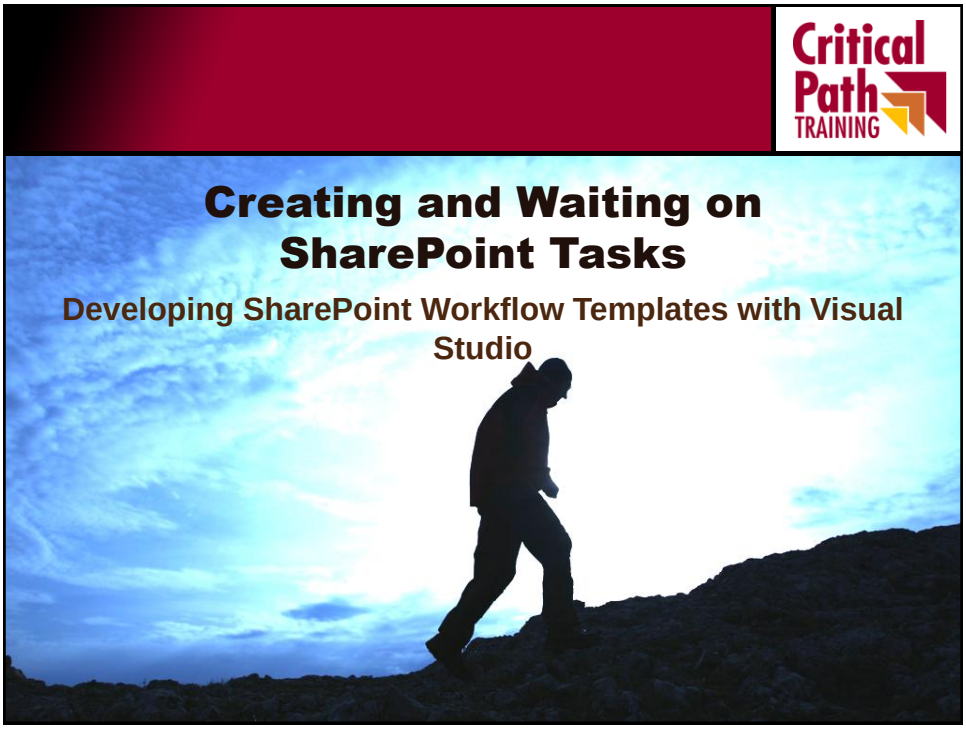


Integrating Solution into Build

- Often integrated into post build steps
  - Post build runs makecab.exe
  - Deploy configuration runs stsadm to install and deploy
- Some open source packages under development
  - WSPBuilder - http://www.codeplex.com/wspbuilder
    `Generates solution based on files in folders`
  - WSP Proj - http://www.codeplex.com/wspprojecttemplate
    `Project template that generates manifest and ddf`
  - Stsdev - http://www.codeplex.com/stsdev
    `Generates simple VS Project files`

Summary

- Create a simple workflow in VS 2008
  - Installing workflow using features
- Pass startup parameters to workflows
- Build state machine workflows in WSS
- Handle item change events in workflow

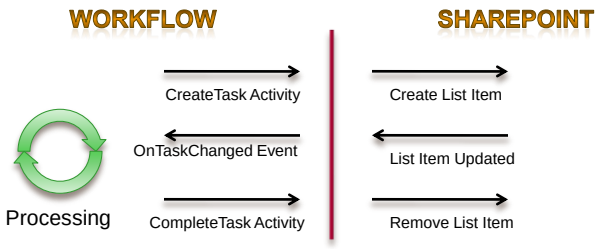


Agenda

- Discuss workflow/task interaction
- Create a simple workflow using tasks
- Add a custom form a task

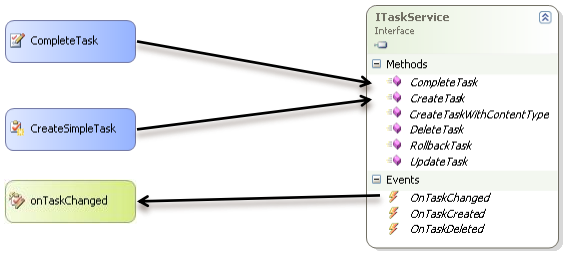
Task Responsibilities

- Tasks are SharePoint's way of user interaction
  - WSS responsible for managing tasks
  - Workflow responsible for passing data to tasks
  - Workflow responsible for reacting to task changes



Workflow/WSS Task Interaction

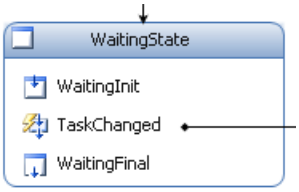
- WF\WSS interaction done via activities
  - CreateTask and CompleteTask used to manage tasks
  - OnTaskChanged used to wait for task changes
  - All methods use ITaskService interface



Common Task Scenario

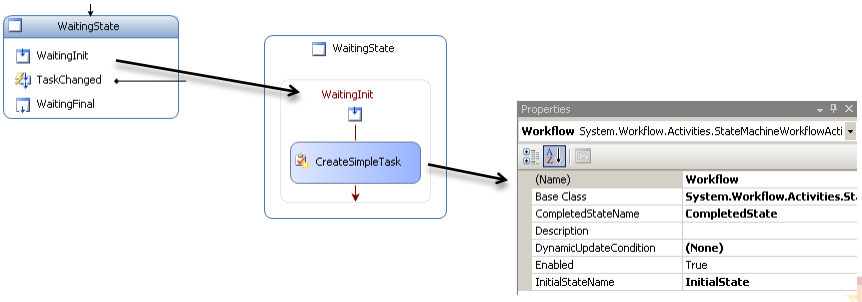
- Waiting State
  - State init creates a task
  - Waits for task changed event

    `Transitions out of state if task was completed`
  - State finalization completes or deletes a task



Creating the Task

- Done using CreateTask activity
  - Requires correlation token
  - Requires task ID
  - Requires task specific properties



CreateTask Correlation Token

- Correlation token defines activity relationships
  - Token initialized in CreateTask activity
  - Used later to wait or act on the task created
- Often scoped at the state
  - Can't be used outside the state
  - Allows state to be re-enterable

(Name)	**CreateSimpleTask**
⊟ CorrelationToken	**taskToken**
OwnerActivityName	**WaitingState**

CreateTask TaskId

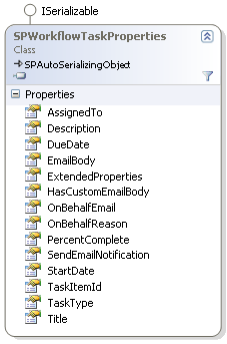
- TaskId required for task related activities
  - Stored as a Guid
  - Used to initialize the correlation token
- Often stored as a workflow property
  - Initialized prior to CreateTask execution
  - CreateTask activity bound to the property

⊟ TaskId	**Activity=Workflow, Path=Ta**
Name	**Workflow**
Path	**TaskId**

```
public Guid TaskId { get; set; }

this.TaskId = Guid.NewGuid();
```

CreateTask TaskProperties

- Defines the details of an activity
  - Stored as SPWorkflowTaskProperties
  - Contains title, description, assigned to, etc…
  - Allows addition of extra data using ExtendedPropertes
  - Only needed while activity executes



Assigning TaskProperties Explicitly

- Storing task properties in workflow wasteful
  - Why not create them only when needed?
  - Can't we just assign the properties in constructor?

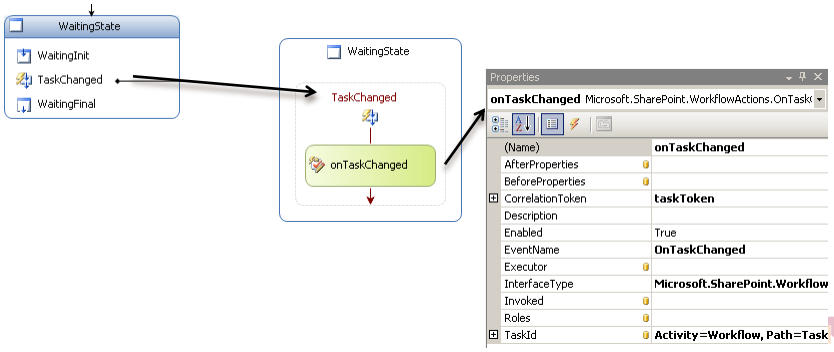
    Not always. Sometimes the activities are copied.
  - Solution; handle MethodInvoking event and use sender

MethodInvoking | CreateSimpleTask_Invoking

```
private void CreateSimpleTask_Invoking(object sender, EventArgs e) {
  TaskId = Guid.NewGuid();

  CreateTask activity = sender as CreateTask;
  activity.TaskProperties = new SPWorkflowTaskProperties();
  activity.TaskProperties.AssignedTo = "Administrator";
  activity.TaskProperties.Title = "Simple Task";
  activity.TaskProperties.Description = "Complete this simple task.";
}
```

Waiting for task changes

- Done using OnTaskChanged event activity
  - Waits for ITaskService.OnTaskChanged event
  - Initialized using correlation token
  - Doesn't flag task as complete, that's up to the workflow



Accessing Task Properties

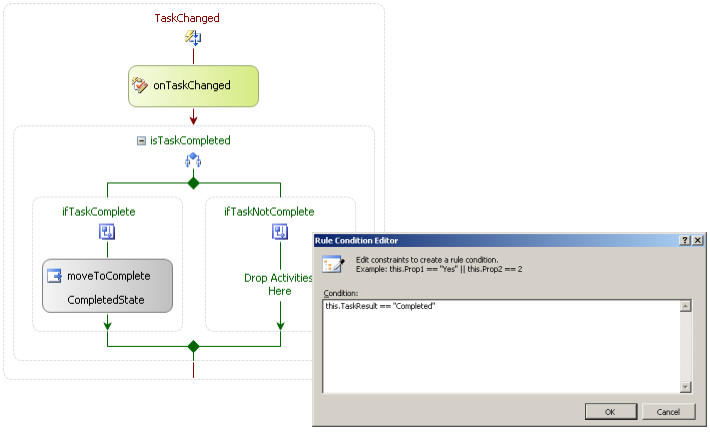
- Properties available in OnTaskChanged activity
  - BeforeProperties contain properties before changes
  - AfterProperties contain properties after changes
  - Both exposed as SPWorkflowTaskProperties
  - Can be bound or accessed directly using Invoked event

Invoked	TaskChanged_Invoked

```
private void TaskChanged_Invoked(object sender,
  ExternalDataEventArgs e)
{
    SPTaskServiceEventArgs args = e as SPTaskServiceEventArgs;
    string state = args.afterProperties.ExtendedProperties["..."];

    /* Code operating on task state */
}
```

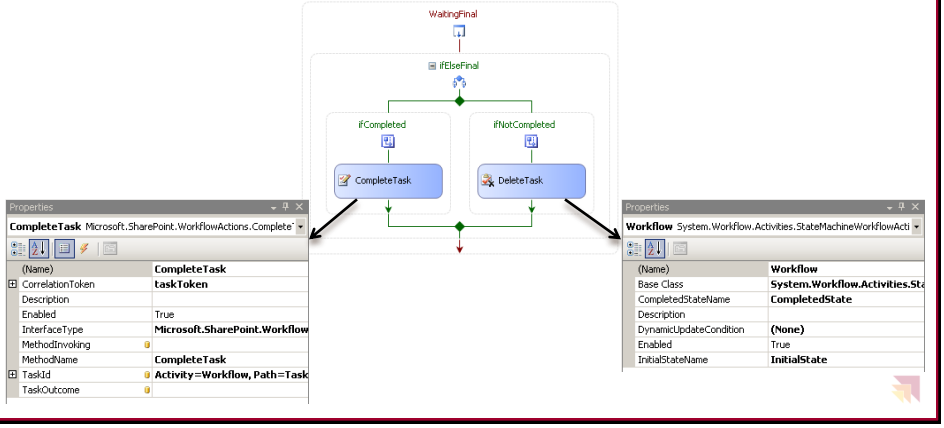
Using Task Properties

- Decisions made by workflow based on properties
  - "Status" property often used to determine next steps



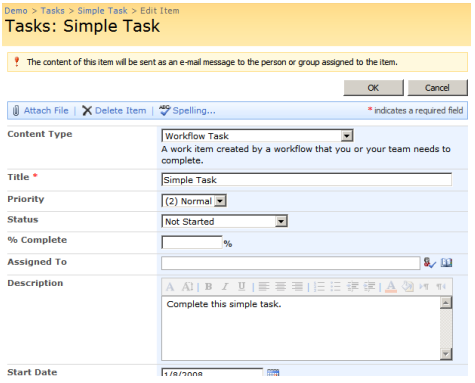
Completing/Deleting Tasks

- Done using CompleteTask or DeleteTask activity
  - Both require CorrelationToken and TaskId
  - Often used in state finalization activity



Custom Task Forms

- Default task forms aren't enough
  - Very generic, no task specific information
  - No way to use custom extended properties



Content Type Forms

- Content types used to differentiate task types
  - Each task created has a specific content type
  - Custom task content types have custom forms
  - Content type forms defined in feature
    `http://schemas.microsoft.com/sharepoint/v3/contenttype/forms/url`

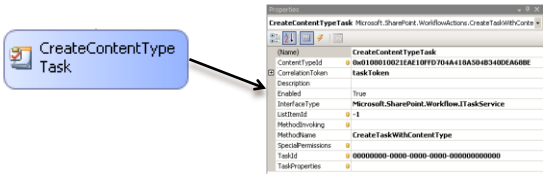
```
<ContentType ID="0x0108010021EAE10FFD704A418A504B340DEA68BE" Name="…"
             Group="…" Description="…" Version="0" Hidden="False">
  <FieldRefs />
  <XmlDocuments>
    <XmlDocument NamespaceURI ="http://...">
      <FormUrls xmlns="http://...">
        <Edit>_layouts/WssDemo/CustomFormsTaskForm.aspx</Edit>
        <Display>_layouts/WssDemo/CustomFormsTaskForm.aspx</Display>
      </FormUrls>
    </XmlDocument>
  </XmlDocuments>
</ContentType>
```

Using Task Content Types

- Two ways of defining task content type
  - Default content type id defined in workflow feature
  - Content type set in CreateTaskWithContentType

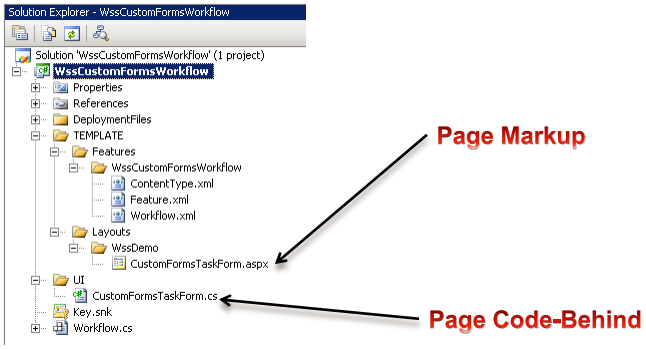
    Done with ContentTypeId property

```
<Workflow ...
    TaskListContentTypeId="0x0108010021EAE10FFD704A418A504B340DEA68BE">
  ...
</Workflow>
```



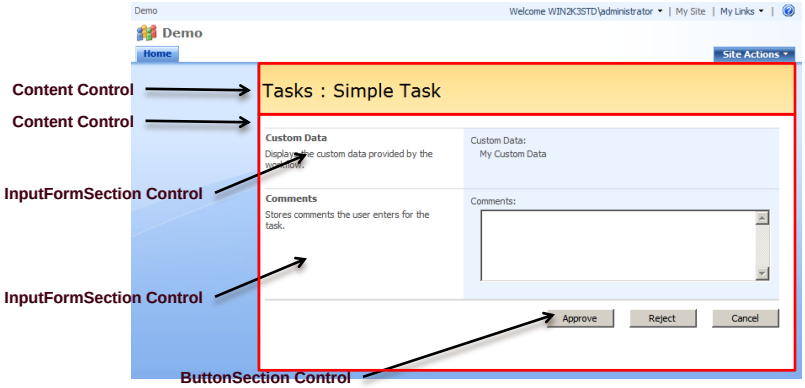
Building Custom Task Forms

- Task forms are aspx pages
  - Visual Studio doesn't support forms directly
  - Markup and Code-Behind connected manually



Page Visual Design

- WSS Provides support for common look and feel
  - Application.master used to provide standard “chrome”
  - Special UserControls used to provide layout



Master Page and Code Behind

- ASPX Page directive defines page properties
  - Defines master page
  - Defines code behind class
- Assembly directive references workflow assembly

```
<%@ Assembly Name="WssCustomFormsWorkflow, ..." %>
<%@ Page Language="C#" MasterPageFile="~/_layouts/application.master“
        Inherits="WssCustomFormsWorkflow.UI.CustomFormsTaskForm" %>
```

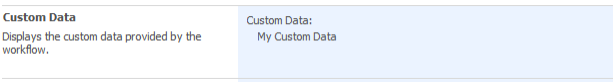
WSS Layout User Controls

- Controls used by WSS to define look and feel
  - Provides simple standard structure
  - Rendered HTML decorated with standard CSS classes
  - Registered on each page using Register directive

```
<%@ Register TagPrefix="wssuc" TagName="InputFormSection"
             Src="/_controltemplates/InputFormSection.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="InputFormControl"
             Src="/_controltemplates/InputFormControl.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="ButtonSection"
             Src="/_controltemplates/ButtonSection.ascx" %>
```

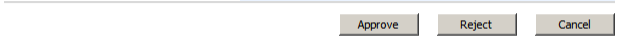

WSS Layout User Controls

```
<wssuc:InputFormSection Title="Custom Data" Description="..." runat="server">
  <template_inputformcontrols>
    <wssuc:InputFormControl LabelText="Custom Data:" runat="server">
      <Template_Control>
        <asp:Label ID="lblCustomData" runat="server" />
      </Template_Control>
    </wssuc:InputFormControl>
  </template_inputformcontrols>
</wssuc:InputFormSection>
```



WSS Layout User Controls

```
<wssuc:ButtonSection runat="server" ShowStandardCancelButton="false">
  <template_buttons>
    <asp:PlaceHolder runat="server">
      <asp:Button ID="btnApprove" Text="Approve“ UseSubmitBehavior="false"
                  class="ms-ButtonHeightWidth" OnClick="Approve_Click"
                  runat="server" />
       
      <asp:Button ID="btnReject" Text="Reject" UseSubmitBehavior="false"
                  class="ms-ButtonHeightWidth" OnClick="Reject_Click"
                  runat="server" />
       
      <asp:Button ID="btnCancel" Text="Cancel" UseSubmitBehavior="false"
                  class="ms-ButtonHeightWidth" OnClick="Cancel_Click"
                  runat="server" />
    </asp:PlaceHolder>
  </template_buttons>
</wssuc:ButtonSection>
```



Form Code-Behind

- Task forms are provided a set of URL parameters
  - Parameters determine the list and id of the task item
  - List and task item used to find the workflow
  - Lookup often done in OnLoad event

```
protected override void OnLoad(EventArgs e)
{
    _taskList = Web.Lists[new Guid(Request.Params["List"])];
    _taskItem = _taskList.GetItemById(int.Parse(Request.Params["ID"]));
    _workflow = new SPWorkflow(Web,
                  new Guid(_taskItem["WorkflowInstanceID"] as string));
    _task = _workflow.Tasks[0];
    _list = Web.Lists[_workflow.ListId];

    ...
}
```

Populating the Form

- The Task SPListItem used to populate data
  - SPListItem fields can be accessed direction
  - Extended properties accessed with SPWorkflowTask

```
protected override void OnLoad(EventArgs e)
{
  ...

  if (!IsPostBack)
  {
    Hashtable extendedProperties =
      SPWorkflowTask.GetExtendedPropertiesAsHashtable(_taskItem);
    lblCustomData.Text =
      extendedProperties["CustomData"] as string;
  }

  ...
}
```

Closing the Form

- User can close form in multiple ways
  - All ways of closing the form are task specific

    `Ex. Approve, Reject, Delegate, Cancel`
  - Code-behind sends properties back to the workflow
  - Page is closed by redirecting to another page

```
protected void Cancel_Click(object sender, EventArgs e)
{
    SPUtility.Redirect(_list.DefaultViewUrl,
        SPRedirectFlags.Default, this.Context);
}
```

Sending Properties to Workflow

- Response sent to workflow using ExtendedProps
  - SPWorkflowTask.AlterTask accepts a Hashtable of data
  - Data contains one or more properties used by workflow

```
protected void Approve_Click(object sender, EventArgs e)
{
    // build properties to send to workflow
    Hashtable data = new Hashtable();
    data.Add("Result", "Approved");
    data.Add("Comments", txtComments.Text);

    // update the task and redirect to list's default view
    SPWorkflowTask.AlterTask(_taskItem, data, true);
    SPUtility.Redirect(_list.DefaultViewUrl,
        SPRedirectFlags.Default, this.Context);
}
```

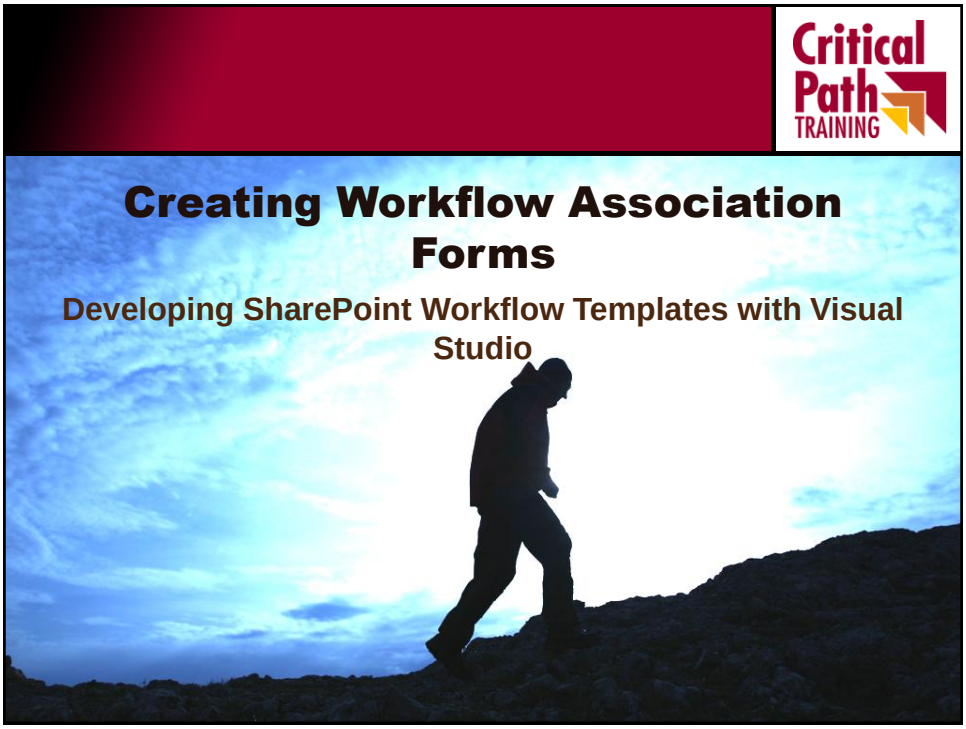
Using Task Results in Workflow

- Task extended properties accessible in workflow
  - Accessible via afterProperties
  - Contains a hashtable named ExtendedProperites
  - ExtendedProperties also contains task item columns
    Accessible by list item field IDs

```
private void TaskChanged_Invoked(object sender, ExternalDataEventArgs e)
{
    SPTaskServiceEventArgs args = e as SPTaskServiceEventArgs;

    // access a named extended property
    TaskResult =
        args.afterProperties.ExtendedProperties["Result"] as string;

    // access a list item field by id
    Guid commentsId = new Guid("{52578FC3-1F01-4f4d-B016-94CCBCF428CF}");
    TaskComments =
        args.afterProperties.ExtendedProperties[commentsId] as string;
}
```

Summary

- Discuss workflow/task interaction
- Create a simple workflow using tasks
- Add a custom form a task

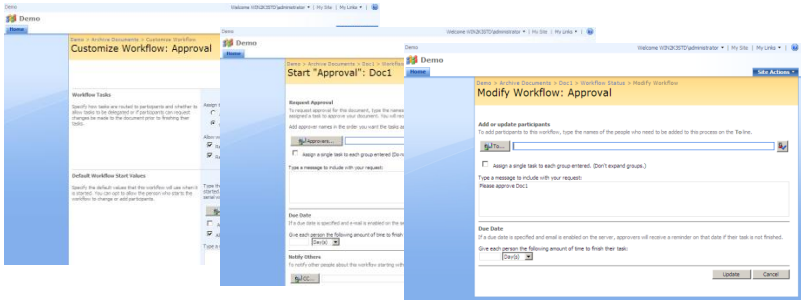


Agenda

- Custom Workflow Forms in SharePoint
- Workflow Form Data Flow
- Building & Registering Custom Association Forms
- Creating/Updating Workflow Associations

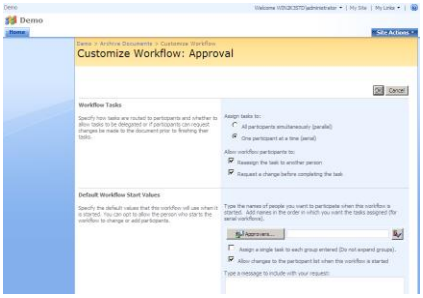
Workflow Forms

- WSS Workflows reference three types of forms
  - Association forms for attachment to list/content type
  - Instantiation forms for starting workflow on a list item
  - Modification forms for changing settings while running
  - All types of forms are optional



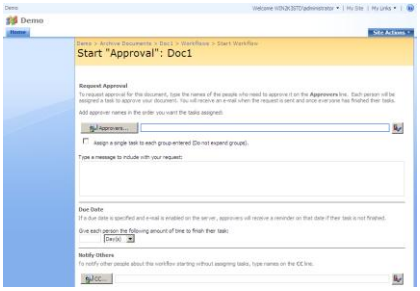
Association Forms

- Displayed when associating a workflow template
  - Displayed after default association page
  - Specific to each workflow template
  - Often used to setup “default” parameters
  - Responsible for creating the SPWorkflowAssociation



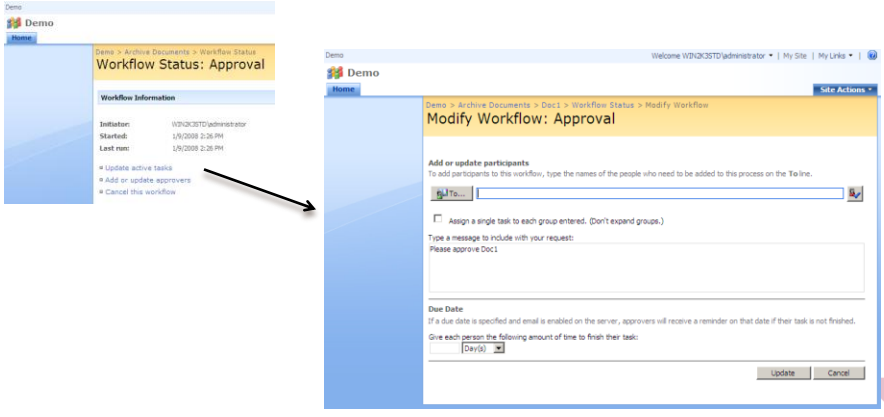
Instantiation Forms

- Displayed whenever a workflow instance started
  - Displayed immediately when workflow started
  - Specific to each workflow template
  - Often initialized using the association form's data
  - Responsible for starting the workflow using



Modification Forms

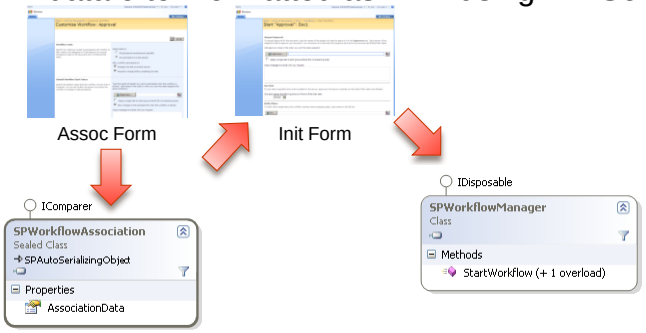
- Accessible from the workflow status page
  - Modification points registered by workflow
  - Links to forms displayed on status page



Form Data Flow

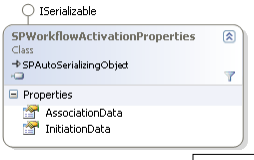
- Data flows from Assoc form to Instantiation form
  - Assoc form data stored in SPWorkflowAssociation
  - Init data passed from form to SPWorkflowManager

    `Data passed via parameter in StartWorkflow method`
  - Form data often formatted as XML using XmlSerializer



Using Form Data in Workflow

- Workflow accesses Assoc and Initiation Data
  - Provided to workflow in OnWorkflowActivated activity
  - Up to the workflow developer to use data appropriately




```
private void onWorkflowActivated_Invoked(
    object sender, ExternalDataEventArgs e)
{
    SPActivationEventArgs args = e as
        SPActivationEventArgs;

    AssociationData assocData = AssociationData.Deserialize(
        args.properties.AssociationData);
    this.CustomData = assocData.DefaultCustomData;
}
```

Registering Custom Forms

- Registration of custom forms done in feature
  - Workflow element provides form URL attributes
  - Form URL is used to load a custom .aspx page
  - If no URL is registered, no form is displayed

```
<Workflow
   Name="Wss Custom Forms Workflow"
   Description="…
   Id="3768e1f1-560e-4b95-8d60-dcbb53c37a56"
   CodeBesideClass="WssCustomFormsWorkflow.Workflow"
   CodeBesideAssembly="WssCustomFormsWorkflow, ..."
   TaskListContentTypeId="0x0108010021EAE10FFD704A418A504B340DEA68BE"
   AssociationUrl ="_layouts/WssDemo/CustomFormsAssocForm.aspx"
   InstantiationUrl="_layouts/WssDemo/CustomFormsInitForm.aspx"
   ModificationUrl="_layouts/WssDemo/CustomFormsModForm.aspx">
  ...
</Workflow>
```

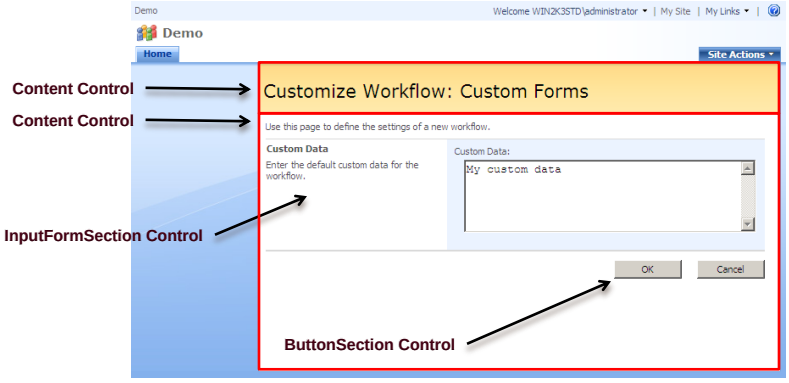

Building Custom Association Form

- Steps for building association forms:
  - Design aspx page
  - Create code-behind class

    `Initialize the UI elements`
    `Load any existing association data`
    `Create or Update the Workflow Association`
  - Register the new custom form

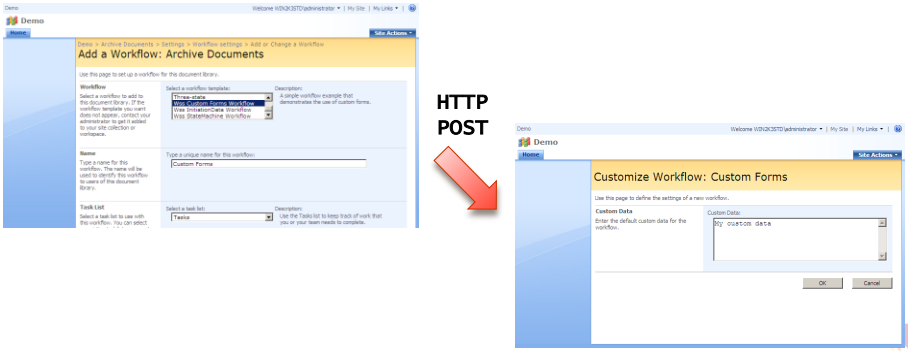
Designing Association Form ASPX

- Uses the same layout concepts as task form
  - Uses same master page
  - Uses same WSS layout UserControls



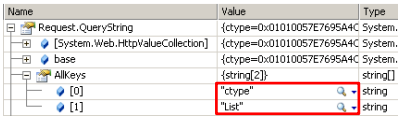
Association Forms are unique?

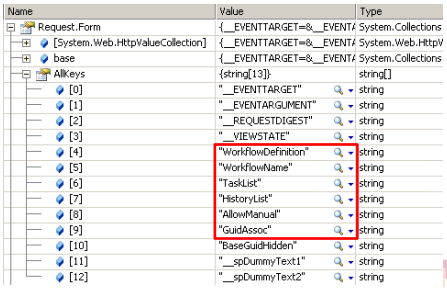
- Association form is accessed via post back
  - Previous page posts to the association form
  - Passes key workflow information via post form
  - Requires some special handling by form developer



Association Form Post and URL data

- Data contained in URL and Post form defines
  - Is this a new association or an existing association?
  - Is this association attached to a list or a content type?
  - What are the names of the tasks and history lists?
  - What permissions and startup handlers are used?





Why a POST?

- Why use a HTTP POST? Why not a GET?
  - First step of association shouldn't create association
    `All data collected on first steps must be sent`
  - Get doesn't allow data transmission beyond URL
  - Post allows more data to be transmitted easily

How is handling a POST different?

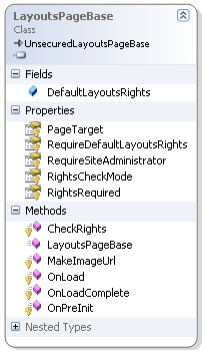
- ASP.NET uses a postback to perform processing
  - Data on the page's form posted back to current page
  - If not careful, the initial post's data is lost
  - Solution; put the data from initial post into page's form

```
<input type="hidden" name="WorkflowDefinition"
       value=<%= Request.Form["WorkflowDefinition"]; %>>
<input type="hidden" name="WorkflowName"
       value=<%= Request.Form["WorkflowName"]; %>>
<input type="hidden" name="AddToStatusMenu"
       value=<%= Request.Form["AddToStatusMenu"]; %>>
<input type="hidden" name="AllowManual" value=<%= Request.Form["AllowManual"]; %>>
<input type="hidden" name="RoleSelect" value=<%= Request.Form["RoleSelect"]; %>>
<input type="hidden" name="AutoStartCreate"
       value=<%= Request.Form["AutoStartCreate"]; %>>
<input type="hidden" name="AutoStartChange"
       value=<%= Request.Form["AutoStartChange"]; %>>
<input type="hidden" name="GuidAssoc" value=<%= Request.Form["GuidAssoc"]; %>>
```

Creating Code-Behind Class

- Code behind class derives from LayoutPageBase
  - Provides base functionality of pages in layouts folder

```
public class CustomFormsAssocForm :
  LayoutsPageBase
{
}
```



Processing Page Parameters

- Page parameters come from URL and form
  - Immediate focus finding association's parent
  - Can be list, content type, or both
  - Determined with List and ctype URL parameters

```
protected override void OnLoad(EventArgs e) {
    // read the form level parameters
    string listId = this.Request.Params["List"];
    string ctypeId = this.Request.Params["ctype"];

    // determine the type of association
    if (!string.IsNullOrEmpty(listId) && !string.IsNullOrEmpty(ctypeId))
        this._associationType = AssociationType.ListContentType;
    else if (!string.IsNullOrEmpty(listId) && string.IsNullOrEmpty(ctypeId))
        this._associationType = AssociationType.List;
    else if (string.IsNullOrEmpty(listId) && !string.IsNullOrEmpty(ctypeId))
        this._associationType = AssociationType.ContentType;

    ...
}
```

Create or Update Association

- GuidAssoc param determines create or update
  - If form contains GuidAssoc, association exists
  - GuidAssoc used to lookup the association object

```
// read the form level parameters
string guidAssocId = Request.Params["GuidAssoc"];

// validate the workflow association id
Guid? workflowAssociationId = new Guid?();
if (!string.IsNullOrEmpty(guidAssocId))
    workflowAssociationId = new Guid(guidAssocId);
```

Lookup existing Association

- Location of association based on parent

```
case AssociationType.ContentType:
    _contentType = Web.AvailableContentTypes[new SPContentTypeId(ctypeId)];
    _workflowAssociation =
        _contentType.WorkflowAssociations[workflowAssociationId.Value];
    break;

case AssociationType.List:
    _list = Web.Lists[new Guid(listId)];
    _workflowAssociation =
        _list.WorkflowAssociations[workflowAssociationId.Value];
    break;

case AssociationType.ListContentType:
    _list = Web.Lists[new Guid(listId)];
    _contentType = _list.ContentTypes[new SPContentTypeId(ctypeId)];
    _workflowAssociation =
        _contentType.WorkflowAssociations[workflowAssociationId.Value];
    break;
```


Storing Form parameters for later

- URL and Form parameters used to load data
  - Form parameters won't be available in postback
  - Form parameters written back into hidden fields

```
protected override void OnPreRender(EventArgs e) {
    // register parameters in the hidden field
    ClientScript.RegisterHiddenField("WorkflowName", Request.Params["WorkflowName"]);
    ClientScript.RegisterHiddenField("WorkflowDefinition",
        Request.Params["WorkflowDefinition"]);
    ClientScript.RegisterHiddenField("AddToStatusMenu",
        Request.Params["AddToStatusMenu"]);
    ClientScript.RegisterHiddenField("AllowManual", Request.Params["AllowManual"]);
    ClientScript.RegisterHiddenField("RoleSelect", Request.Params["RoleSelect"]);
    ClientScript.RegisterHiddenField("GuidAssoc", Request.Params["GuidAssoc"]);
    ClientScript.RegisterHiddenField("SetDefault", Request.Params["SetDefault"]);
    ClientScript.RegisterHiddenField("HistoryList", Request.Params["HistoryList"]);
    ClientScript.RegisterHiddenField("TaskList", Request.Params["TaskList"]);
    ClientScript.RegisterHiddenField("UpdateLists", Request.Params["UpdateLists"]);
    ClientScript.RegisterHiddenField("AutoStartCreate",
        Request.Params["AutoStartCreate"]);
    ClientScript.RegisterHiddenField("AutoStartChange",
        Request.Params["AutoStartChange"]);

    ...
```

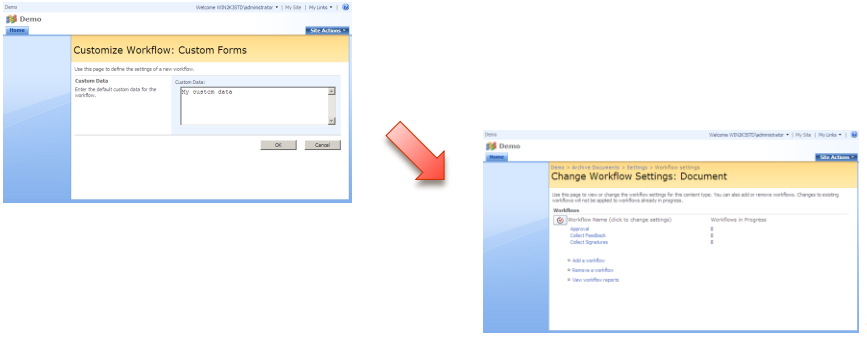
Displaying existing Association

- If association exists the UI needs to be populated
  - UI population usually done in OnLoad if !IsPostBack
  - Initial assoc form load is a postback
  - Solution is to initialize in OnPreRender

```
protected override void OnPreRender(EventArgs e)
{
    // bind the association data (if any)
    if (_workflowAssociation != null)
    {
        // deserialize the association data
        AssociationData assocData = AssociationData.Deserialize(
            _workflowAssociation.AssociationData);

        // bind the controls to the association data
        txtCustomData.Text = assocData.DefaultCustomData;
    }
```

Handling Button Events

- The user can click OK or Cancel to submit form
  - OK causes the SPWorkflowAssociation object creation
  - Cancel causes no object creation
  - Both end up redirecting back to workflow settings page



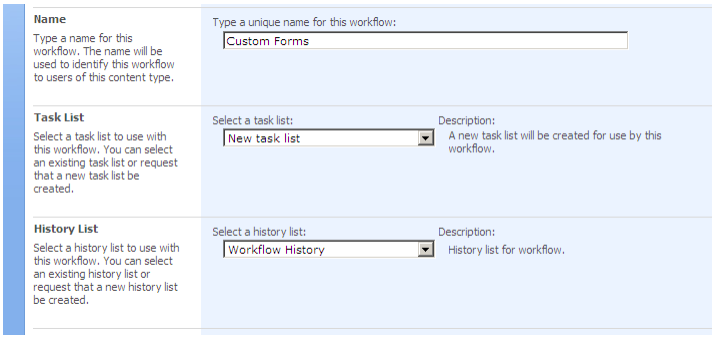
Handling OK Button Click Events

- Create and update actions require assoc data
  - First step of handler is to serialize assoc data to string
  - Often done using a custom class and XmlSerializer

```
AssociationData associationData = new AssociationData();
associationData.DefaultCustomData = txtCustomData.Text;
string serializedData = associationData.Serialize();


public class AssociationData
{
    public string DefaultCustomData { get; set; }
    public string Serialize()
    {
        XmlSerializer serializer = new XmlSerializer(typeof(AssociationData));
        using (StringWriter writer = new StringWriter())
        {
            serializer.Serialize(writer, this);
            return writer.ToString();
        }
    }
```

Finding/Creating Supporting Lists

- Workflow associations depend on special lists
  - Lists store tasks and history for workflow instances
  - Previous page allows choice of existing or new lists
  - Association form responsible for finding or creating lists



Finding/Creating Supporting Lists

- List name or ID encoded in Form parameters
  - If name starts with 'z', it's the name of a new list
  - Otherwise the name is the list name or ID

  `If parent is list, it's a name or ID`

```
SPList LookupOrCreateList(string paramList, SPListTemplateType listType) {
    if (paramList.StartsWith("z")) {
        Guid newListId =
            Web.Lists.Add(paramList.Substring(1), "Workflow Tasks",
            listType);
        return Web.Lists[newListId];
    }
    else {
        return Web.Lists[new Guid(paramList)];
    }
}
```

Creating or Updating Association?

- Create or update based on lookup in OnLoad
  - If a workflow association was found, it's an update

```
protected void Submit_Click(object sender, EventArgs e) {
    ...


    // create or find the lists
    SPList taskList = LookupOrCreateList(Request.Params["TaskList"],
        SPListTemplateType.Tasks);
    SPList historyList = LookupOrCreateList(Request.Params["HistoryList"],
        SPListTemplateType.WorkflowHistory);

    // create or update the workflow association
    if (_workflowAssociation != null)
        UpdateWorkflowAssociation(taskList, historyList,
            associationData.Serialize());
    else
        CreateWorkflowAssociation(taskList, historyList,
            associationData.Serialize());
```

Updating Existing Association

- Existing SPWorkflowAssociation values updated
  - Values from HTTP Form and URL updated first
  - AssociationData from custom form updated next

```
// assign base workflow association information
_workflowAssociation.Name = Request.Params["WorkflowName"];
_workflowAssociation.AutoStartCreate =
    (Request.Params["AutoStartCreate"] == "ON");
_workflowAssociation.AutoStartChange =
    (Request.Params["AutoStartChange"] == "ON");
_workflowAssociation.AllowManual = (Request.Params["AllowManual"] == "ON");
_workflowAssociation.AssociationData = associationData;

// assign the chosen task list to the association
if (_workflowAssociation.TaskListId != taskList.ID)
    _workflowAssociation.SetTaskList(taskList);

// assign the chosen history list to the association
if (_workflowAssociation.HistoryListId != historyList.ID)
    _workflowAssociation.SetHistoryList(historyList);
```

Commiting Association Updates

- Once changes made, they must be committed
  - Done with parent's UpdateWorkflowAssociation method

```
switch (_associationType)
{
    case AssociationType.ContentType:
        // commit the changes to the existing workflow association
        _contentType.UpdateWorkflowAssociation(_workflowAssociation);
        break;

    case AssociationType.List:
        // commit the changes to the existing workflow association
        _list.UpdateWorkflowAssociation(_workflowAssociation);
        break;

    case AssociationType.ListContentType:
        // commit the changes to the existing workflow association
        _contentType.UpdateWorkflowAssociation(_workflowAssociation);
        break;
}
```

Creating New Associations

- Done with SPWorkflowAssociation static methods
  - One method for each type of parent
  - All take similar parameters

```
case AssociationType.ContentType:
    _workflowAssociation =
        SPWorkflowAssociation.CreateSiteContentTypeAssociation(
            workflowTemplate, workflowName,
            taskList.Title, historyList.Title);
    break;

case AssociationType.List:
    _workflowAssociation = SPWorkflowAssociation.CreateListAssociation(
        workflowTemplate, workflowName, taskList, historyList);
    break;

case AssociationType.ListContentType:
    _workflowAssociation =
        SPWorkflowAssociation.CreateListContentTypeAssociation(
            workflowTemplate, workflowName, taskList, historyList);
break;
```

Adding the new Association to Parent

- Once new association is populated, it's added
  - Each parent type has AddWorkflowAssociation method

```
case AssociationType.ContentType:
    PopulateWorkflowAssociation(taskList, historyList, assocData);
    _contentType.AddWorkflowAssociation(_workflowAssociation);
    break;

case AssociationType.List:
    PopulateWorkflowAssociation(taskList, historyList, assocData);
    _list.AddWorkflowAssociation(_workflowAssociation);
    break;

case AssociationType.ListContentType:
    PopulateWorkflowAssociation(taskList, historyList, assocData);
    _contentType.AddWorkflowAssociation(_workflowAssociation);
    break;
```

Redirecting to Workflow Settings

- On OK or Cancel handler complete, form “closes”
  - Closes by redirecting to WorkflowSettings page
  - URL parameters differ based on association parent

```
string url = null;
string list = Request.Params["List"];
string ctype = Request.Params["ctype"];

switch (_associationType)
{
    case AssociationType.ContentType:
        url = "WrkSetng.aspx?ctype=" + paramCtype;
    case AssociationType.List:
        url = "WrkSetng.aspx?List=" + paramList;
    case AssociationType.ListContentType:
        url = "WrkSetng.aspx?List=" + paramList + "&ctype=" + paramCtype;
}

SPUtility.Redirect(url, SPRedirectFlags.RelativeToLayoutsPage, this.Context);
```

ContentTypes and WF Associations

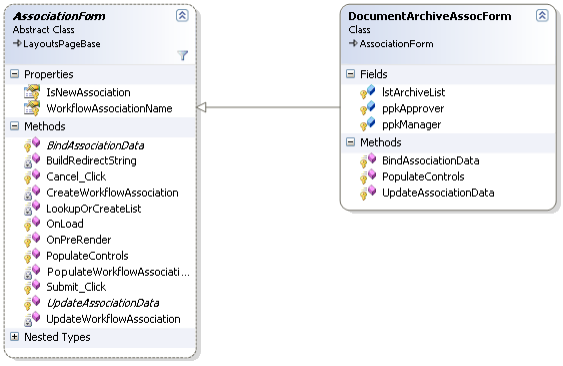
- Are content type association changes inherited?
  - When UpdateWorkflowAssociationsOnChildren called
  - Association form should call if UpdateLists is true

    Should be called after Add and Update call

```
// check whether updates to content types cascade to their lists
_updateContentTypeLists = (Request.Params["UpdateLists"] == "TRUE");

...
// update the association and update child associations
_contentType.UpdateWorkflowAssociation(_workflowAssociation);
if (_updateContentTypeLists)
    _contentType.UpdateWorkflowAssociationsOnChildren(true, true, true);

...
// add the association and update child associations
_contentType.AddWorkflowAssociation(_workflowAssociation);
if (_updateContentTypeLists)
    _contentType.UpdateWorkflowAssociationsOnChildren(true, true, true);
```

Making it Simpler

- Use a standard base class
  - Most of the code in an association form never changes
  - Code that varies displays and saves form data

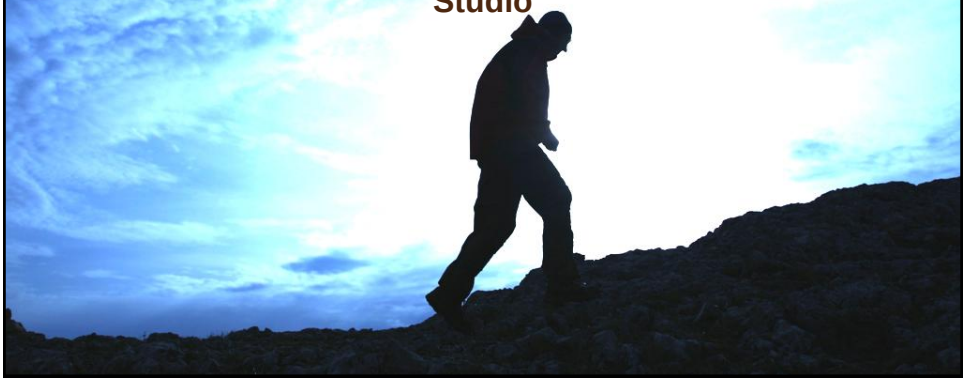


Summary

- Custom Workflow Forms in SharePoint
- Workflow Form Data Flow
- Building & Registering Custom Association Forms
- Creating/Updating Workflow Associations

Creating Workflow Instantiation and Modification Forms

Developing SharePoint Workflow Templates with Visual Studio



Agenda

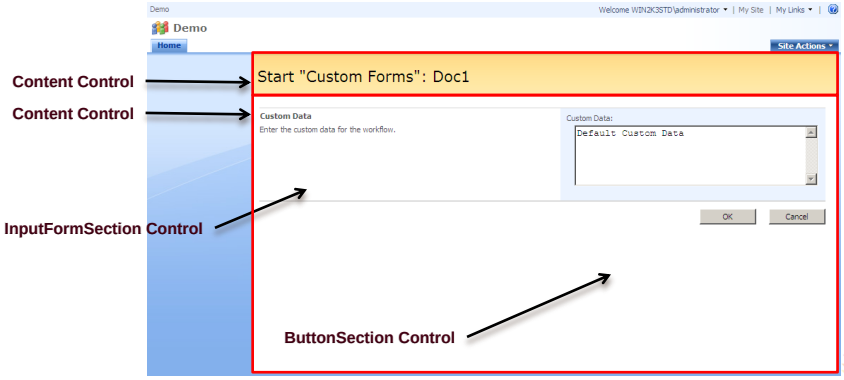
- Create and Register Initiation Forms
- Create Modification Forms
- Register and Enable Modification Forms

Building Custom Initiation Forms

- Steps for building initiation forms:
  - Design aspx page
  - Create code-behind class
    - **Initialize the UI elements**
    - **Load any existing association data**
    - **Start the workflow instance**
  - Register the new custom form

Designing Initiation Form ASPX

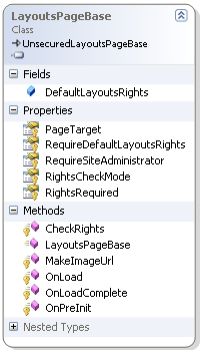
- Uses same layout concepts as association form
  - Uses same master page
  - Uses same WSS layout UserControls



Creating Code-Behind Class

- Code behind pages derive from LayoutPageBase
  - Provides base application page functionality

```
public class CustomFormsInitForm :
  LayoutsPageBase
{
}
```



Processing Page Parameters

- Page parameters come from URL
  - List and ctype define the parent of the association
  - ID identifies the list item the workflow instance is for
  - TemplateId identifies the workflow association

```
protected override void OnLoad(EventArgs e)
{
    // read the form level parameters
    string listId = Request.Params["List"];
    string listItemId = Request.Params["ID"];
    string ctypeId = Request.Params["ctype"];
    string templateId = Request.Params["TemplateId"];

    // find the list and list item
    _list = Web.Lists[new Guid(listId)];
    _listItem = _list.GetItemById(int.Parse(listItemId));
```

Locating the Workflow Association

- Workflow association may be in two locations
  - Could be associated with the list
  - Could be associated with the content type
  - Easiest way to find it is check both locations

```
// check for the workflow association in the list
_workflowAssociation = _list.WorkflowAssociations[
    new Guid(templateId)];

// if the association wasn't found, check the content type
if (_workflowAssociation == null)
{
    SPContentType _contentType = _list.ContentTypes[
        new SPContentTypeId(ctypeId)];
    _workflowAssociation = _contentType.WorkflowAssociations[
        new Guid(templateId)];
}
```

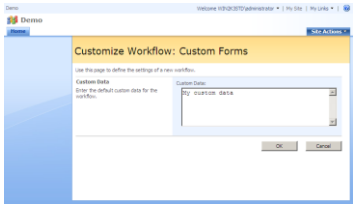
Initializing the UI

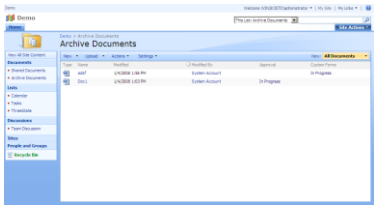
- The initial page load is not caused by a postback
  - Much simpler than Association form
  - Override OnLoad event and load when !IsPostBack

```
protected override void OnLoad(EventArgs e)
{
    ...

    // bind the association data (if any)
    if (!IsPostBack && (_workflowAssociation != null))
    {
        // bind the controls to the association data
        AssociationData assocData =
            AssociationData.Deserialize(_workflowAssociation.AssociationData);
        txtCustomData.Text = assocData.DefaultCustomData;
    }
```

Handling Button Events

- The user can click Start or Cancel to submit form
  - Start causes the workflow instance to start
  - Cancel causes no workflows to start
  - Both end up redirecting back to the list's default view





Handling Start Button Click

- Starting the workflow requires Initiation Data
  - Initiation data is gathered from the UI and serialized
  - Workflow is started using the site's workflow manager

  Requires the workflow association and list item

```
protected void Start_Click(object sender, EventArgs e)
{
    // serialize the initiation data
    InitiationData initiationData = new InitiationData();
    initiationData.CustomData = txtCustomData.Text;

    // start the new workflow instance
    Web.Site.WorkflowManager.StartWorkflow(
        _listItem, _workflowAssociation,
        initiationData.Serialize());
```

Redirecting to List Default View

- Start and Cancel buttons both redirect to list
  - Redirects to list's default view url

```
protected void Cancel_Click(object sender, EventArgs e)
{
    // redirect to the list default view
    SPUtility.Redirect(_list.DefaultViewUrl,
        SPRedirectFlags.Default, this.Context);
}
```


Registering Instantiation Forms

- Registration of custom forms done in feature
  - Workflow element provides form URL attributes
  - Form URL is used to load a custom .aspx page
  - If no URL is registered, no form is displayed

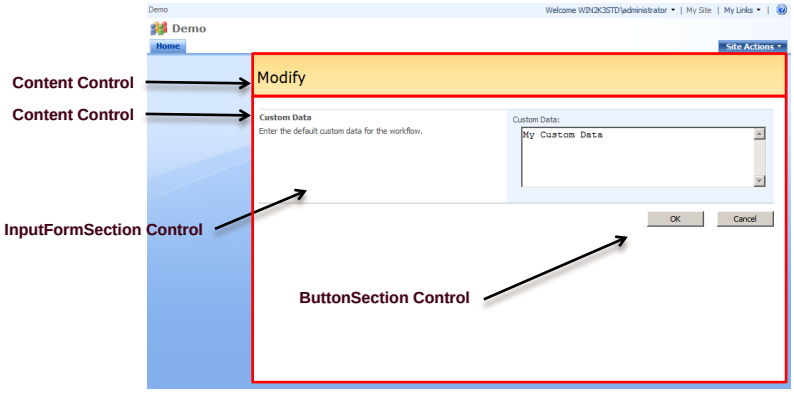
```
<Workflow
   Name="Wss Custom Forms Workflow"
   Description="…
   Id="3768e1f1-560e-4b95-8d60-dcbb53c37a56"
   CodeBesideClass="WssCustomFormsWorkflow.Workflow"
   CodeBesideAssembly="WssCustomFormsWorkflow, ..."
   TaskListContentTypeId="0x0108010021EAE10FFD704A418A504B340DEA68BE"
   AssociationUrl ="_layouts/WssDemo/CustomFormsAssocForm.aspx"
   InstantiationUrl="_layouts/WssDemo/CustomFormsInitForm.aspx"
   ModificationUrl="_layouts/WssDemo/CustomFormsModForm.aspx">
  ...
</Workflow>
```

Building Custom Modification Forms

- Steps for building modification forms:
- Design aspx page
- Create code-behind class
  - **Initialize the UI elements**
  - **Load any existing association data**
  - **Start the workflow instance**
- Register the new custom form
- Add metadata defining link
- Enable modification in workflow

Designing Modification Form ASPX

- Uses same layout concepts as association form
  - Uses same master page
  - Uses same WSS layout UserControls



Processing Page Parameters

- Page parameters come from URL
  - List and ID identify the list item related to the workflow
  - WorkflowInstanceID identifies the workflow instance
  - ModificationID identifies a SPWorkflowModification

```
protected override void OnLoad(EventArgs e)
{
    // read the form level parameters
    string listId = Request.Params["List"];
    string listItemId = Request.Params["ID"];
    string workflowId = Request.Params["WorkflowInstanceID"];
    string modificationId = Request.Params["ModificationID"];

    // find the list, list item, workflow, and modification
    _list = Web.Lists[new Guid(listId)];
    _listItem = _list.GetItemById(Convert.ToInt32(listItemId));
    _workflow = _listItem.Workflows[new Guid(workflowId)];
    _modification = _workflow.Modifications[new Guid(modificationId)];
```

Initializing the UI

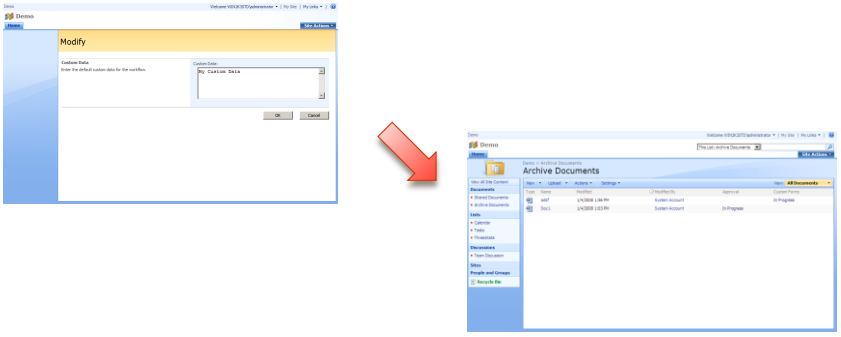
- The initial page load is not caused by a postback
  - Override OnLoad event and load when !IsPostBack
  - Use SPWorkflowModification.ContextData to load UI

    Often ContextData is XML created by XmlSerializer

```
protected override void OnLoad(EventArgs e)
{
    ...

    // populate the controls
    if (!IsPostBack)
    {
        // deserialize the association data
        ModificationData modData =
            ModificationData.Deserialize(_modification.ContextData);

        // bind the controls to the association data
        txtCustomData.Text = modData.CustomData;
    }
```

Handling Button Events

- The user can click OK or Cancel to submit form
  - OK applies modifications to workflow
  - Cancel causes no modifications to be made
  - Both end up redirecting back to the list's default view



Handling OK Button Click

- Changes applied using ModifyWorkflow method
  - Modification data gathered from UI and serialized
  - Workflow is modified using the site's workflow manager

  Requires SPWorkflow and SPWorkflowModification

```
protected void Start_Click(object sender, EventArgs e)
{
    // populate the modification data using the UI
    ModificationData modData = new ModificationData();
    modData.CustomData = txtCustomData.Text;

    // modify the workflow
    Web.Site.WorkflowManager.ModifyWorkflow(
        _workflow, _modification, modData.Serialize());
```

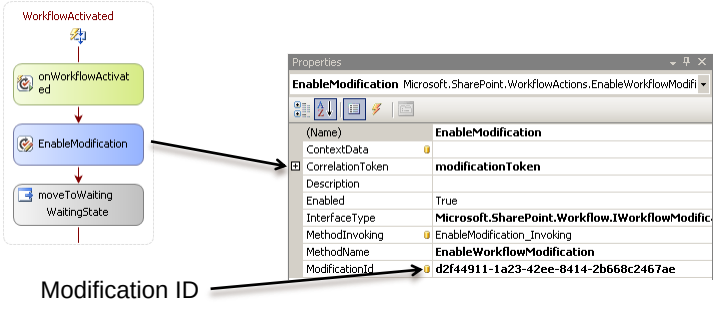
Redirecting to List Default View

- Just like instantiation form, close redirects to list
  - Redirects to list's default view url

```
protected void Cancel_Click(object sender, EventArgs e)
{
    // redirect to the list default view
    SPUtility.Redirect(_list.DefaultViewUrl,
        SPRedirectFlags.Default, this.Context);
}
```


Using the Modification Form

- Each type of modification identified by an ID
  - Used when modifications are enabled in the workflow
  - Specified in the EnableWorkflowModification activity
  - Related to a new modification correlation token



Enabling Workflow Modifications

- Modification form needs data to populate form
  - Association or Initiation data may be out dated
  - Better solution is to store data in modification object

    Use SPWorkflowModification.ContextData property

```
void EnableModification_Invoking(object sender, EventArgs e)
{
    ModificationData modData = new ModificationData();
    modData.CustomData = this.CustomData;

    EnableWorkflowModification enableModification =
        sender as EnableWorkflowModification;
    enableModification.ContextData = modData.Serialize();
}
```

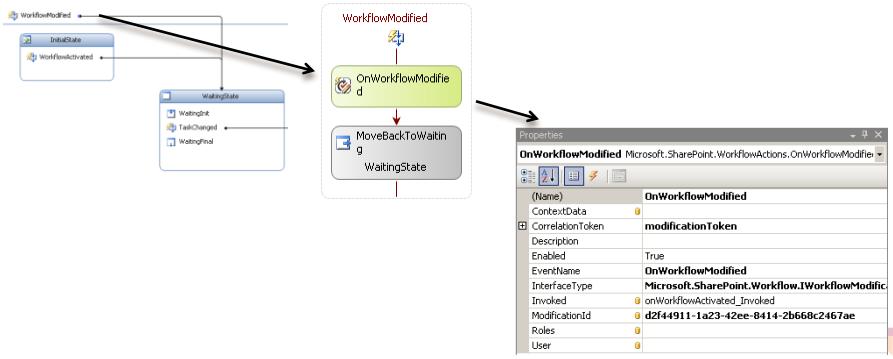

Registering Workflow Modifications

- Registration of custom forms done in feature
  - Form registration isn't only change required
  - The name of the modification is needed as well

    Used when displaying the modification link
  - Name identified by Modification ID

Responding to Modifications

- Modifications are received via event
  - Received using OnWorkflowModified activity
  - Uses modification correlation token and modification ID
  - Provides the context data returned from the form



Integrating Modified Data

- OnWorkflowModified event returns form data
  - Available in the SPModificationEventArgs.data property
  - Can be processed in the MethodInvoked handler

```
private void OnWorkflowModified_Invoked(
    object sender, ExternalDataEventArgs e)
{
    SPModificationEventArgs args = e as SPModificationEventArgs;

    ModificationData modData =
        ModificationData.Deserialize(args.data);
    this.CustomData = modData.CustomData;
}
```

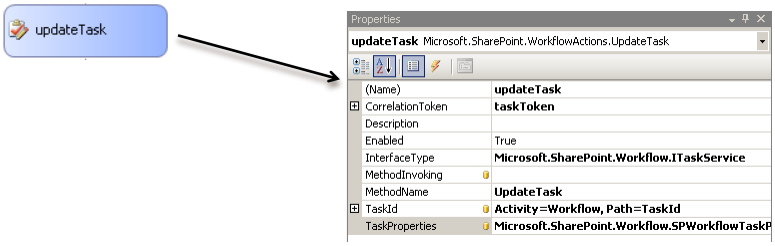
Resetting Workflow State

- Modifications often made as workflow is waiting
  - Tasks will most likely need to be updated
  - Two options

    `Update Task`
    `Delete and Create Task`

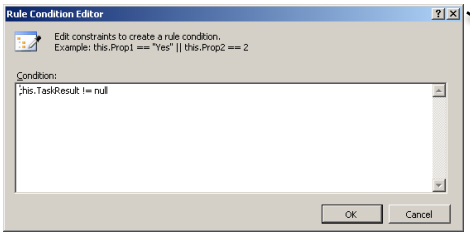
Updating Tasks

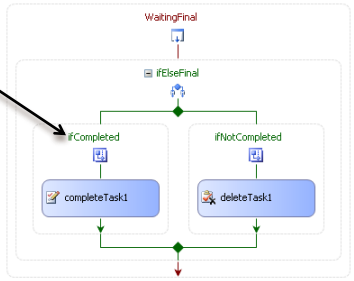
- Done using the UpdateTask activities
  - Requires task correlation token
  - Can't be done outside the token's scope
  - Task updated with SPWorkflowTaskProperties object



Delete and Re-create Task

- Can use state initializer and finalizer
  - Create task in initializer
  - Complete or delete task in finalizer based on result
    `Store result when task is complete`
  - State transition causes initializer and finalizer execution





Summary

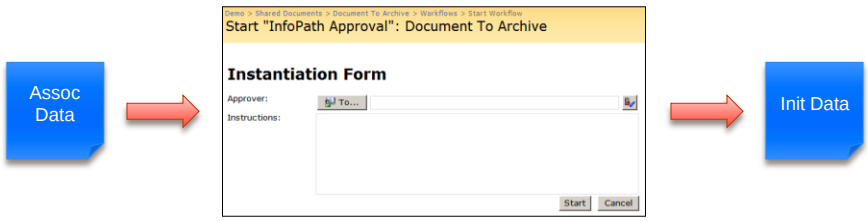
- Create and Register Initiation Forms
- Create Modification Forms
- Register and Enable Modification Forms

Agenda

- Why use InfoPath?
- How it works
- Creating workflow forms with InfoPath
- Interacting with hosting environment
- Custom .NET code

Workflow Forms have commonalities

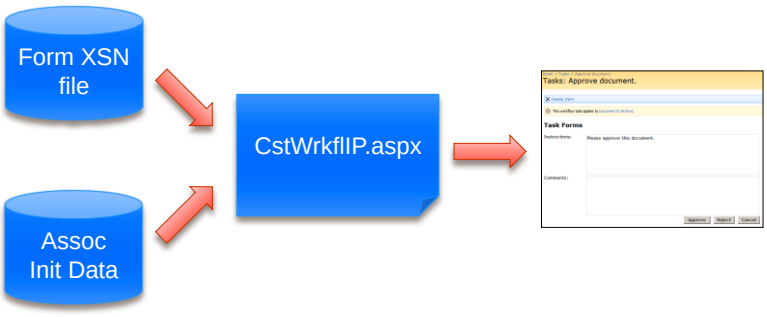
- Every workflow form type has many similarities
  - Essentially accept data, modify it, and pass it on
  - Accepting and passing on data is identical across forms
  - The modification of data is the unique part



Common Forms Framework

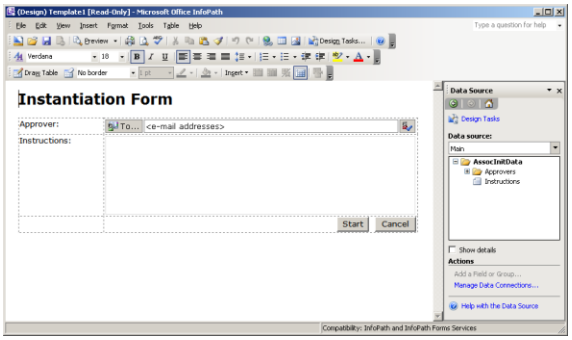
- Abstracting out the modification simplifies forms
  - One solution is to create a standard base class
  - Another solution is to define forms as metadata

    `Layout as well as behavior are needed`



What is InfoPath?

- InfoPath is a tool that stores forms as metadata
  - Designer stores layout and behavior in a XSN file
  - Viewer opens XSN file and loads/saves data as XML
  - Designer intended to be useable by non-developers

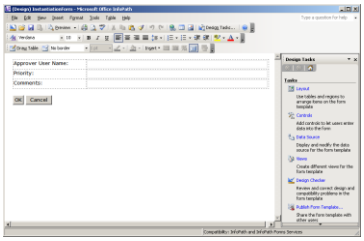


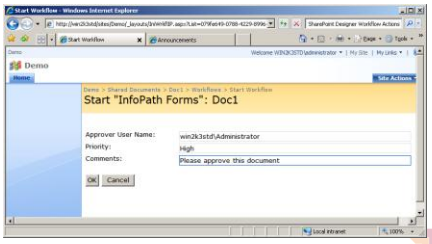
Difficulties Deploying InfoPath 2003

- InfoPath 2003 is a client side only application
  - Requires InfoPath on the end users machine
  - Trust issues required complex signing process
  - A better way was needed to deploy InfoPath forms

InfoPath Forms Services

- SharePoint 2007 introduced Forms Services
  - Ships as part of MOSS
  - Interprets InfoPath files and displays as web page
  - InfoPath forms now accessible to anyone via browser

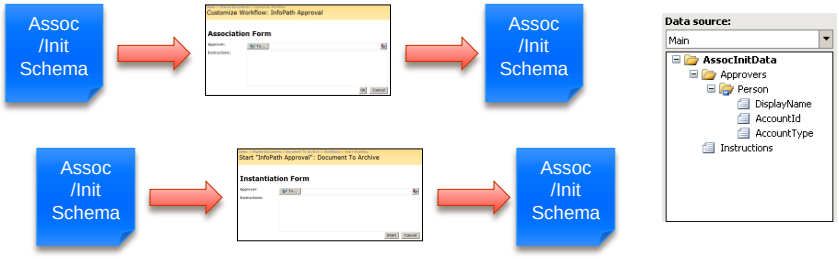




Using InfoPath Forms in Workflow

- InfoPath data model imposes some restrictions
  - InfoPath forms have same input and output schema
  - Requires Association and Initiation data share schema

  `Rarely is this a problem`

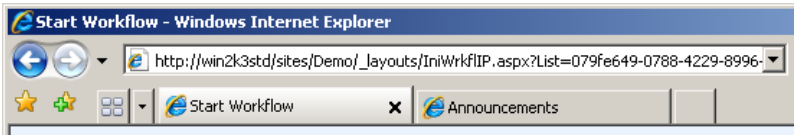


Forms Services and Workflow

- MOSS defines standard workflow ASPX pages
  - One ASPX page for each type of form

Association Form	_layouts/CstWrkflIP.aspx
Instantiation Form	_layouts/IniWrkflIP.aspx
Modification Form	_layouts/ModWrkflIP.aspx
Task Form	_layouts/WrkTaskIP.aspx

  - Pages automatically load the correct InfoPath form



InfoPath Forms ASPX Pages

- Page markup contains a special web control
  - XmlFormView control renders the InfoPath form
  - Control requires two pieces of information

    The InfoPath form to host
    The XML data used to initialize the form

```
<%@ Register Tagprefix="InfoPath"
             Namespace="Microsoft.Office.InfoPath.Server.Controls"
             Assembly="Microsoft.Office.InfoPath.Server, ..." %>

<asp:Content ContentPlaceHolderId="PlaceHolderMain" runat="server">
    <InfoPath:XmlFormView id="XmlFormView" runat="server" />
    ...
</asp:Content>
```

InfoPath and Workflow Metadata

- WSS Workflow Features can have metadata
  - This metadata is accessible via an API
  - Forms Services pages access standard metadata tags
  - These tags define a URN for each type of form

```
<Workflow Id="66dd3439-b412-423e-8e15-cc972c9eb36a"
  ...
  AssociationUrl ="_layouts/CstWrkflIP.aspx"
  InstantiationUrl="_layouts/IniWrkflIP.aspx">
  <MetaData>
    <Association_FormURN>
      urn:...:AssociationForm:urn-WssInfoPathFormsWorkflow
    </Association_FormURN>
    <Instantiation_FormURN>
        urn:...:InstantiationForm:urn-WssInfoPathFormsWorkflow
    </Instantiation_FormURN>
  </MetaData>
</Workflow>
```

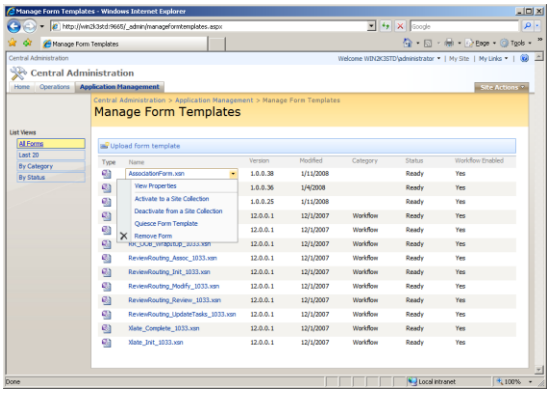
Registration of InfoPath Forms

- Locating forms by URN requires form registration
  - On feature activation event receiver registers forms
    `Microsoft.Office.Workflow.Feature.WorkflowFeatureReceiver`
  - Event receiver uses path in feature properties element

```
<Feature Id="695fb738-da73-438d-aa10-d50669dc3457"
  ReceiverAssembly="Microsoft.Office.Workflow.Feature, ..."
  ReceiverClass="Microsoft.Office.Workflow.Feature.WorkflowFeatureReceiver"
  xmlns="http://schemas.microsoft.com/sharepoint/">
  ...
  <Properties>
    ...
    <Property Key="RegisterForms" Value="Forms\*.xsn" />
  </Properties>
</Feature>
```

Form Registration with Central Admin

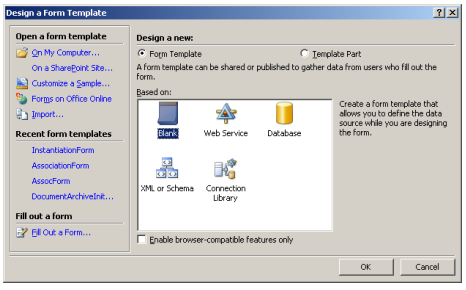
- Central Admin allows form management
  - Accessed from Application Management tab
  - Allows upload and installation of custom forms

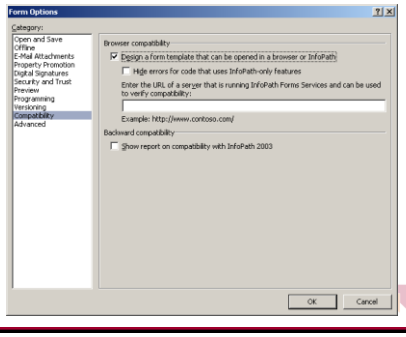


Creating custom InfoPath Forms

- Create a new form by designing a blank form
  - Some actions are not supported in Form Services
  - Use Tools -> Form Options to choose web form

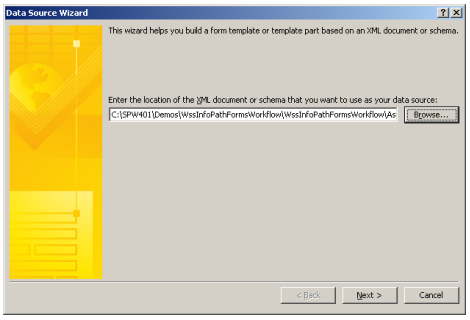
  `Raises validation error if form not supported`

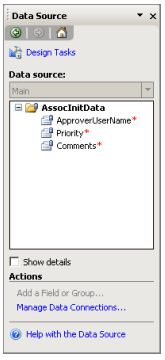




Importing Data Schema

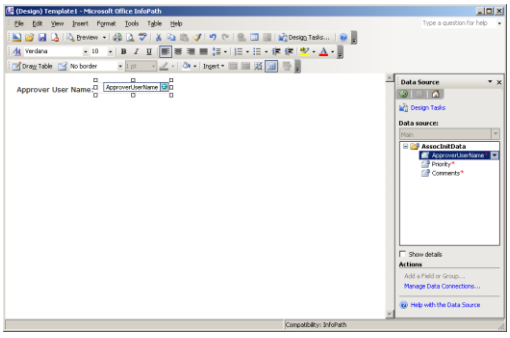
- Form schema can be imported or created
  - Created XML schemas use generated namespace
    Complications when sharing schema between forms
  - Imported using Tools -> Convert Main Data Source





Adding Input Controls

- Data has a natural relationship to controls
  - Controls bound to data elements
  - Data can be “dragged” onto design service as controls
    Control type and label inferred from data

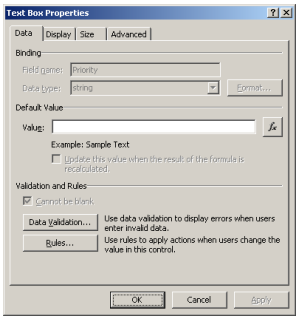


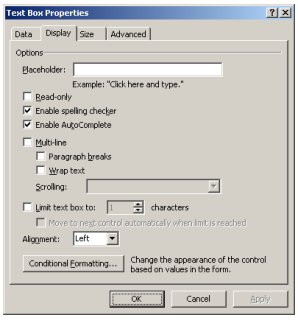
Layout Tables

- InfoPath provides formatting tools to designer
  - Layout tables provides structure
  - Text can have Word like layout and font applied
  - Experience intended to be similar to Word

Control Properties

- Each InfoPath control has specific properties
  - Allows restrictions such as read only
  - Allow sizing and formatting options
  - Provides custom data validation rules

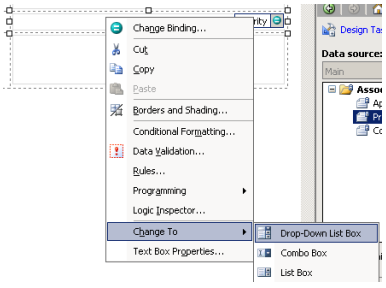


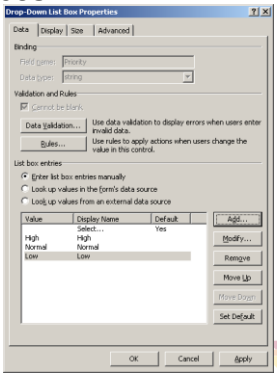


Using Drop Down Lists

- Drop Down Lists allow choosing from a list
  - TextBox converted using Change To context menu
  - Selected value bound to primary data source
  - Options can come from multiple sources

    `Fixed list, external data source`

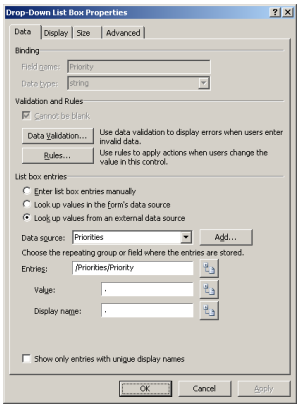


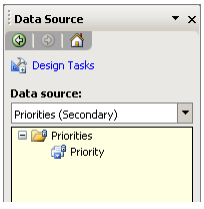


Binding to Secondary Data Sources

- External data can be accessed for list population
  - Data can come from multiple locations

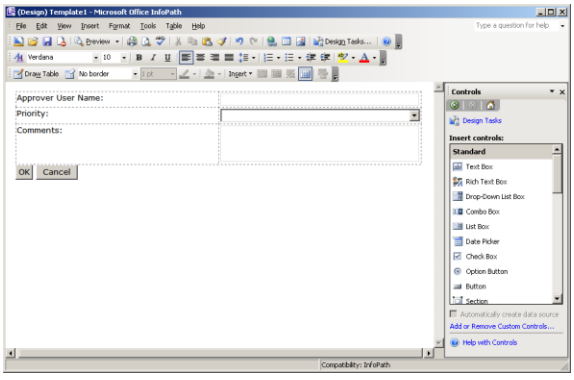
    `Xml, WebService, SharePoint lists, etc…`





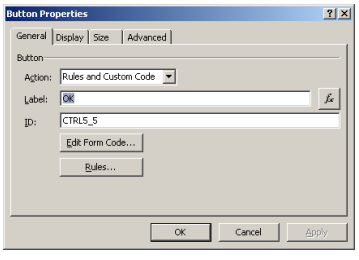
Submitting the Data

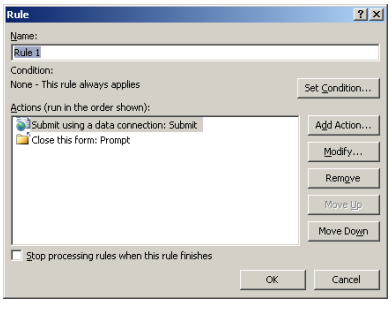
- Data submission initiated by button clicks
  - Controls can be added without data binding
    `Common for controls with no display`
  - Button controls often used to submit or close



Applying Rules to Buttons

- Data submission done using InfoPath rules
  - Rules are executed when button is clicked
  - Forms submitted using submit to host
    `Special external data source`
  - Forms closed using Close Form action

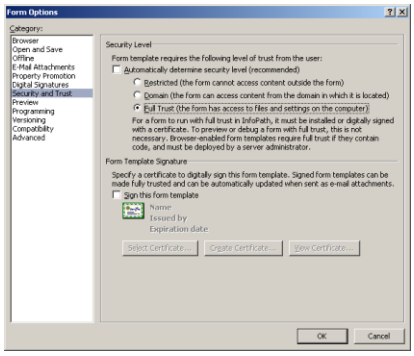




Choosing Trust Level

- InfoPath forms request a level of trust
  - InfoPath defaults to Restricted

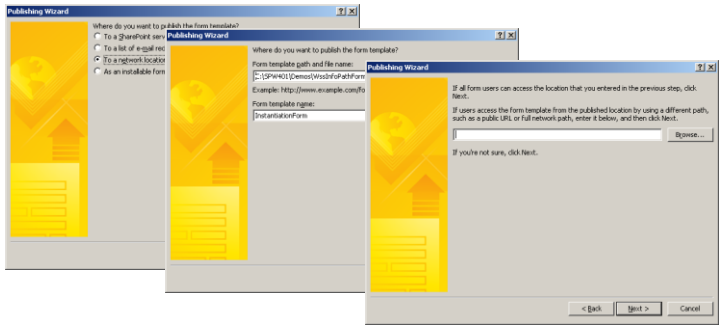
    `Causes InfoPath to fail while installing form`
  - Most WSS applications require Domain or Full Trust



Publishing InfoPath Forms

- Publishing is required to create a usable form
  - Signs form if a certificate is attached to the form
  - Attaches a publish location

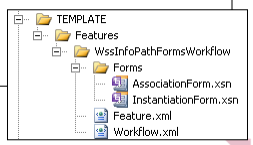
    `Publish location must be empty for SharePoint`



Deploying InfoPath Forms

- Published forms deployed with feature
  - Often located in Forms sub folder
  - Copied to the FEATURES folder along with feature xml
  - Registered by the feature activation event receiver

```
<Feature Id="695fb738-da73-438d-aa10-d50669dc3457"
    ...
    ReceiverAssembly="Microsoft.Office.Workflow.Feature, ..."
    ReceiverClass="Microsoft.Office.Workflow.Feature.WorkflowFeatureReceiver"
    xmlns="http://schemas.microsoft.com/sharepoint/">
  ...
  <Properties>
    <Property Key="RegisterForms" Value="Forms\*.xsn" />
  </Properties>
</Feature>
```



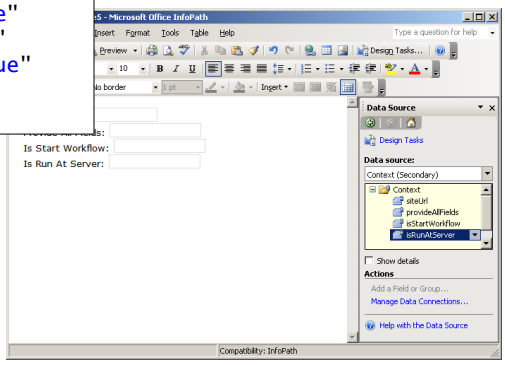
Forms Services Host Interaction

- InfoPath Forms have no connection to WSS
  - ASPX Forms can access WSS Object Model
  - InfoPath Forms are isolated from the Object Model
  - InfoPath does have access to Context data source
  - Task Forms can request task fields via ItemMetadata

Context Data Source

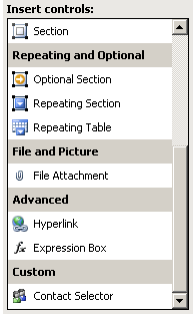
- Context data source based on XML file
  - Data automatically populated by ASPX host page
  - Allows InfoPath form access to critical context data

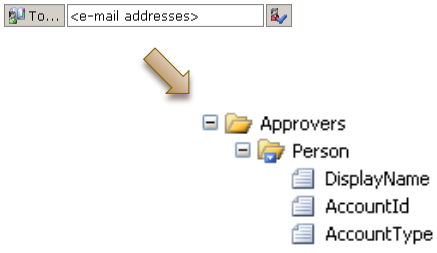
```
<Context
  isStartWorkflow="true"
  isRunAtServer="false"
  provideAllFields="true"
  siteUrl="" />
```



Using People Picker

- External ActiveX control used to pick users
  - Added to the Controls list in InfoPath
  - Bound to a special data structure

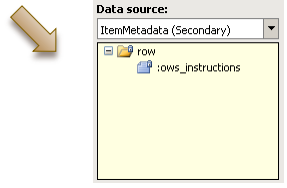




Accesing Task Extended Properties

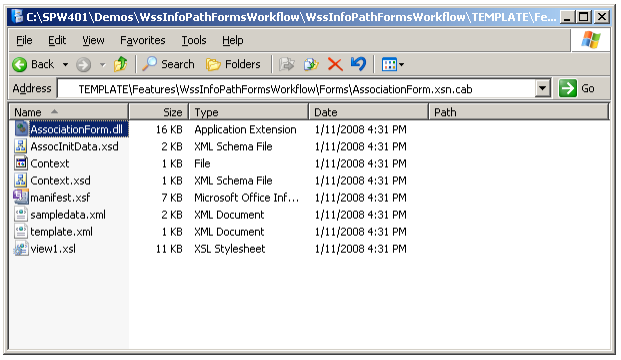
- Task properties provided via ItemMetadata
  - ItemMetadata.xml provided by task form designer
  - Defines which properties should be provided
    ows_ prefix required
  - Accessed as secondary data source by form designer

```
<z:row
xmlns:z="#RowsetSchema"
  ows_Instructions="" />
```



InfoPath Code Behind

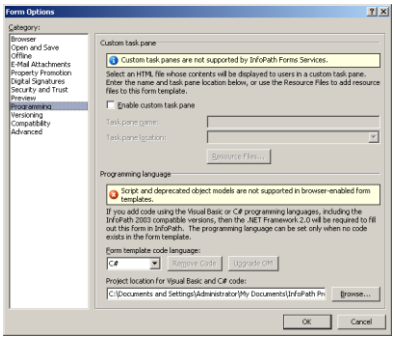
- InfoPath 2007 Supports .NET Code Behind
  - Key events can be handled using C# or VB Code
  - Generates an assembly that is embedded in template
    Template is just a CAB file



Editing Code Behind

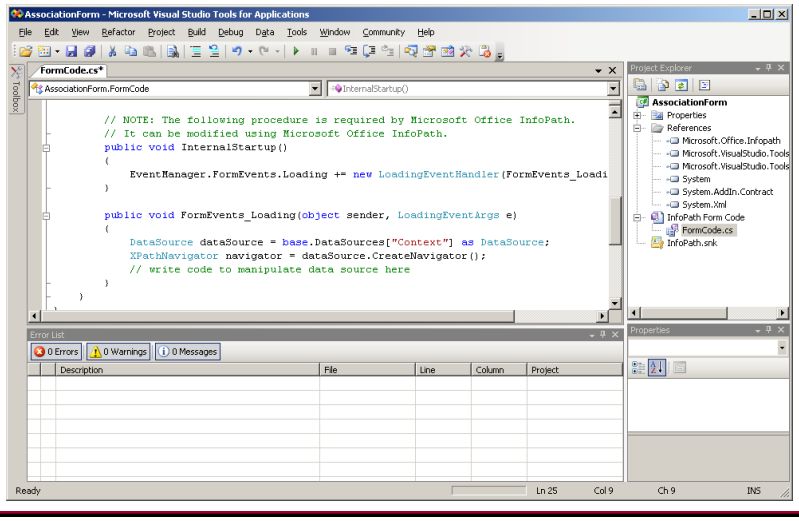
- Visual Studio Tools for Applications (VSTA) used
  - Deployed as part of InfoPath 2007
  - Allows full editing and compilation of C# and VB Code
  - Project created automatically when handling events

Language chosen in Form Options



Building Simple Initialization Code

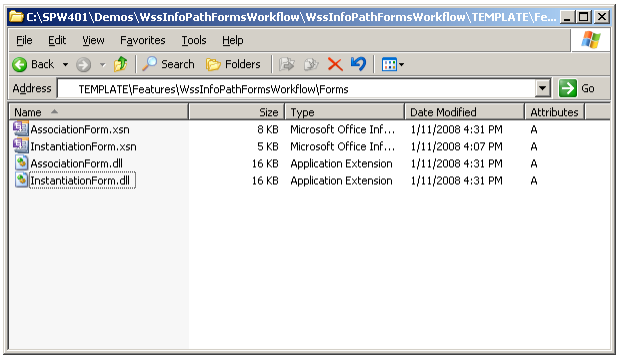
- Code Behind has direct access to form data



Forms Services and Code Behind

- Forms Services has unique requirements
  - Won't access assembly inside of the XSN
  - Requires assembly be placed in same folder as XSN

Currently no automated way to perform



Common Code Behind Problems

- Deploying code behind can be difficult
  - Make sure the form's security level is correct
  - Make sure the assembly is outside the XSN
  - Don't forget to check LOGS folder

Summary

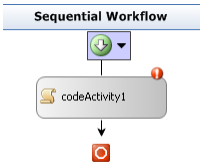
- Why use InfoPath?
- How it works
- Creating workflow forms with InfoPath
- Interacting with hosting environment
- Custom .NET code

Agenda

- Create activities
  - Events and Dependency Properties
  - Custom Activity Validators
- Understanding Activity Lifecycle
- Creating Composite Activities
- Custom Activity Designers

Why Custom Activities?

- Activities provide functionality encapsulation
  - Similar in concept to methods in C# or VB
  - Activity developers provide libraries to workflow builder
  - Custom designers communicate intent
  - Custom validators prevent simple mistakes



Hello World Activity

- All activities derive from Activity class
  - Key activity method is Execute
  - Execute is where the activity starts
  - Returning status of closed indicates the activity is done

```
public class HelloWorldActivity : Activity
{
    protected override ActivityExecutionStatus Execute(
                       ActivityExecutionContext executionContext)
    {
        Console.WriteLine("Hello World");
        return ActivityExecutionStatus.Closed;
    }
}
```

Adding Parameters to Activities

- Parameters allow activities to be reused
  - Activity properties are just .NET properties
  - Should wrap DependencyProperty to allow binding

```
public static DependencyProperty MessageProperty =
   DependencyProperty.RegisterAttached("Message",
      typeof(string), typeof(WriteConsoleActivity));

public string Message
{
    get { return base.GetValue(MessageProperty) as string; }
    set { base.SetValue(MessageProperty, value); }
}
```


Adding Events to Activities

- Events provide callback capabilities
  - Activity events are just .NET events
  - Should wrap DependencyProperty to allow binding
  - Event implementation delegates to Activity members

```
public static DependencyProperty InvokingEvent =
    DependencyProperty.RegisterAttached("Invoking",
      typeof(EventHandler), typeof(WriteConsoleActivity));

public event EventHandler Invoking
{
    add { base.AddHandler(InvokingEvent, value); }
    remove { base.RemoveHandler(InvokingEvent, value); }
}
```

Raising Events in Activities

- Events raised using Activity.RaiseEvent method
  - Raises the event registered using AddHandler
  - Automatically deals with any bindings

```
protected override ActivityExecutionStatus Execute(
    ActivityExecutionContext executionContext)
{
    // raise the bound event
    base.RaiseEvent(InvokingEvent, this, EventArgs.Empty);

    // write the value stored in the dependency property
    Console.WriteLine(this.Message);

    // tell the framework that this activity is done
    return ActivityExecutionStatus.Closed;
}
```

Validating Activities

- Activities can have an associated validator
  - Must derive from ActivityValidator
  - Validated at design time and at compile time
  - Validate method performs the validation

    Receives activity, returns an error collection

```
internal class WriteConsoleActivityValidator : ActivityValidator
{
    public override ValidationErrorCollection Validate(
        ValidationManager manager, object obj)
    {
        ...

        // return the errors
        return errors;
    }
}
```

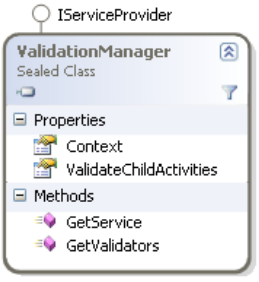
Validating Activities

- Validation fails when contained by an activity type

```
public override ValidationErrorCollection Validate(
    ValidationManager manager, object obj) {
    WriteConsoleActivity activity = obj as WriteConsoleActivity;
    if (activity == null)
        throw new InvalidOperationException();

    // check if any parent actions are a transaction scope
    ValidationErrorCollection errors = new ValidationErrorCollection();
    Activity parent = activity.Parent;
    while (parent != null) {
        if (parent is TransactionScopeActivity)
            errors.Add(new ValidationError(
                "Error Message", 100));
        parent = parent.Parent;
    }

    // call the base validation method
    errors.AddRange(base.Validate(manager, obj));
    return errors;
}
```

ValidationManager

- Allows communication between validators
  - Context allows data to be passed to child validators
  - ValidateChildActivities determines scope of validation
  - GetValidators and GetService allow context access



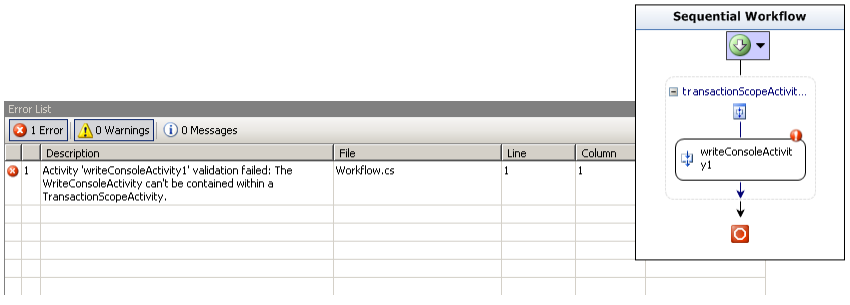
Attaching the Validator

- Validator attached to activity by .NET attribute
  - ActivityValidatorAttribute defines the validator type
  - Workflow designer and compiler reference attribute

```
[ActivityValidator(typeof(WriteConsoleActivityValidator))]
public class WriteConsoleActivity : Activity
{
    ...
}
```

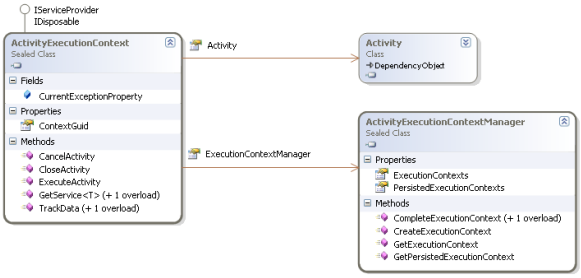
Validators In Action

- Validators fire at design time and compile time
  - Design time validator errors are displayed in designer
  - Compile time validator errors displayed in error list
  - Multiple validator errors can be displayed at once



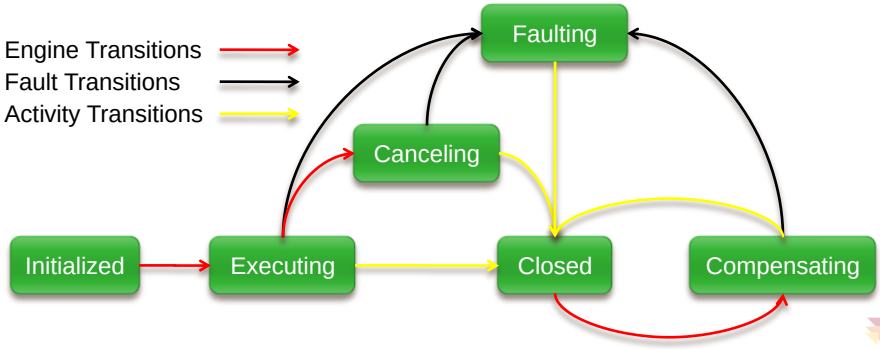
Activity Execution Context

- Created automatically for each activity
  - Manages the activities lifecycle
    `Allows activities to close or cancel themselves`
  - Can create new AECs using ExecutionContextManager
    `Used for looping activities; While or Replicator`



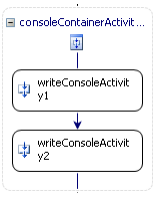
Activity States

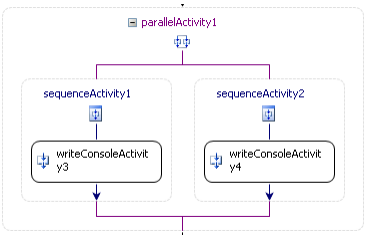
- Activity developers must be aware of states
  - State transitions are often influenced by custom code
    `Ex. If execute completes but activity not done`
  - Valid state transitions defined by Workflow Runtime



Composite Activities

- Some activities contain multiple child elements
  - These activities execute child activities in different ways
  - Sequence executes on child at a time
  - Parallel executes all simultaneously
  - If/Else executes based on condition





Sequence Activity

- Execute method isn't always synchronous
  - Execute method returns the state of the activity
  - Activity can wait for events from the runtime

  Often used to initiate activity state transition

```
public class ConsoleContainerActivity : SequenceActivity,
{
    protected override ActivityExecutionStatus Execute(
        ActivityExecutionContext executionContext)
    {
        ...

        // start the first child activity and register for closed event
        base.EnabledActivities[0].RegisterForStatusChange(
            Activity.ClosedEvent, this);
        executionContext.ExecuteActivity(base.EnabledActivities[0]);

        // tell the runtime this activity is still executing
        return ActivityExecutionStatus.Executing;
    }
```

IActivityEventListener Interface

- Activities send events when their state changes
  - Event received via IActivityEventListener
  - Parent activities monitor state of children using events

    Register for events using RegisterForStatusChange
- Receive events in IActivityEventListener.OnEvent

```
base.EnabledActivities[0].RegisterForStatusChange(
    Activity.ClosedEvent, this);

void OnEvent(object sender,
    ActivityExecutionStatusChangedEventArgs e) {
    ...

    // unregister the handler for status events
    e.Activity.UnregisterForStatusChange(
        Activity.ClosedEvent, this);
}
```

Handling Child Closed Event

- Next activity executed when previous closed
  - Execute method registers for child activity closed event
  - When event received, next activity started
  - When no activities remain, sequence activity closes

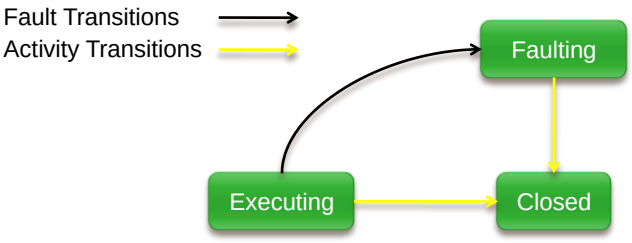
```
public void OnEvent(object sender,
     ActivityExecutionStatusChangedEventArgs e)
{
    ActivityExecutionContext executionContext =
        sender as ActivityExecutionContext;
    ...
    if (activity.ExecutionStatus == ActivityExecutionStatus.Executing)
    {
        // if no more children exist
        if (!this.TryScheduleNextChild(executionContext))
            executionContext.CloseActivity();
    }
}
```

Handling Faults

- Exceptions during execution lead to faulting state
  - Runtime transitions activity to faulting state
  - Executes Activity.HandleFault and enters closed state

```
This is where activities can clean up
```

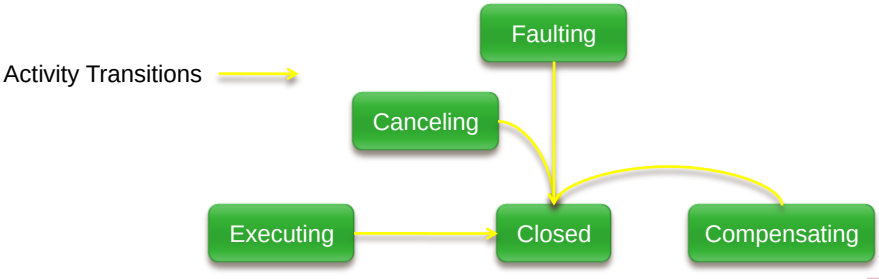
  - Faults in HandleFault cause transition to faulting state



Activity Completion

- When the activity completes it is closed
  - Completed by ActivityExecutionContext.CloseActivity
  - Activities can override OnClosed to perform cleanup

```
Called any time the closed state is entered
Can be called multiple times
```



Composite Activity Validators

- Derives from CompositeActivityValidators
  - Adds support for fault and cancel views
  - Often used to restrict types of activities

    Ex. Don't allow any EventDriven activities
    Can check immediate children and descendents

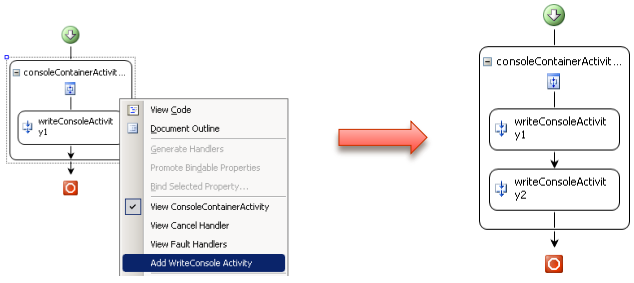
```
internal class ConsoleContainerActivityValidator : CompositeActivityValidator {
    public override ValidationErrorCollection Validate(
        ValidationManager manager, object obj) {
        ...

        ValidationErrorCollection errors = new ValidationErrorCollection();
        foreach (Activity childActivity in activity.EnabledActivities)
            if (!(childActivity is WriteConsoleActivity))
                errors.Add(new ValidationError("...", 100));

        ...
    }
}
```

Activity Designers

- Designers control Activity usage in designer
  - Designers can restrict an activity's parent and children

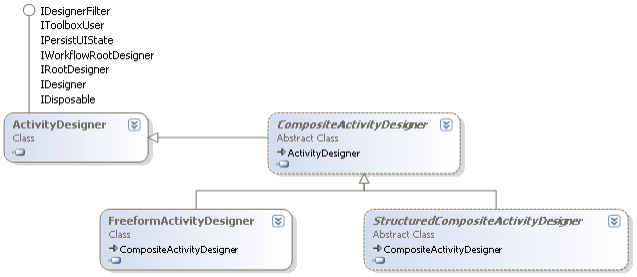
    Insert, Delete events can be controled
  - Designers can control the look and feel of an activity

    Colors, border styles, etc…



Creating Designers

- Custom designers derive from ActivityDesigner
  - More advanced designers can derive from others
  - Methods are overridden to control designer behavior



Restricting Parent Activities

- Designers can restrict an activity's parent activity
  - Done by override CanBeParentedTo
  - Decision made based on the parent designer
  - Often used to keep activity in specific parent activities

```
internal class WriteConsoleActivityDesigner : ActivityDesigner
{
    public override bool CanBeParentedTo(
        CompositeActivityDesigner parentActivityDesigner)
    {
        return (parentActivityDesigner.Activity is
                  ConsoleContainerActivity);
    }
}
```

Restricting Child Activities

- Designers can restrict an activities children
  - Done by overriding several methods

    `CanInsertActivities`
    `CanMoveActivities`
    `CanDeleteActivities`

```
public class ConsoleContainerActivityDesigner :
    SequentialActivityDesigner {
    public override bool CanInsertActivities(
        HitTestInfo insertLocation,
        ReadOnlyCollection<Activity> activitiesToInsert) {
        foreach (Activity activity in activitiesToInsert)
            if (!(activity is WriteConsoleActivity))
                return false;
        return true;
    }
}
```

Custom Designer Verbs

- Activity designers support custom verbs
  - If/Else Activity has Add Branch verb
  - Verbs perform a variety of actions

    `Add new branches`
    `Create new child activities`
    `Etc…`

Adding Custom Verbs

- Verbs defined by the activity's designer
  - Verbs represented by ActivityDesignerVerb
  - List of verbs provided by designer's Verbs property

```
protected override ActivityDesignerVerbCollection Verbs {
    get {
        // create the list containing the verbs
        ActivityDesignerVerbCollection verbs =
                  new ActivityDesignerVerbCollection(base.Verbs);

        // add the new verb
        verbs.Add(
            new ActivityDesignerVerb(this, DesignerVerbGroup.Actions,
                "Add WriteConsole Activity",
                new EventHandler(AddWriteConsole_Click)));

        // return the list of verbs
        return verbs;
    }
}
```

Handling Verb Click Events

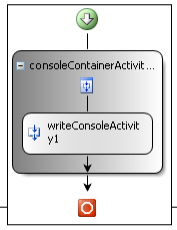
- Verbs have event handler code attached
  - EventHandler delegate attached to verb object
  - When the verb is clicked, the event handler executed

```
private void AddWriteConsole_Click(object sender, EventArgs e)
{
    CompositeActivity activity = base.Activity as CompositeActivity;

    // create the list of new activites
    List<Activity> activities = new List<Activity>();
    activities.Add(new WriteConsoleActivity());

    // determine where to add the activities
    ConnectorHitTestInfo location =
        new ConnectorHitTestInfo(this,
            HitTestLocations.Designer, activity.Activities.Count);

    // add the activities to the designer
    this.InsertActivities(location, activities.AsReadOnly());
}
```

Activity Designer Themes

- Defines visual display characteristics of activity
  - Theme class derives from ActivityDesignerTheme
  - Added to designer with ActivityDesignerThemeAttribute
  - Defines fore and back colors, activity connectors, etc…

```
[ActivityDesignerTheme(typeof(ConsoleContainerActivityTheme))]
public class ConsoleContainerActivityDesigner

internal class ConsoleContainerActivityTheme : CompositeDesignerTheme
{
    public ConsoleContainerActivityTheme(WorkflowTheme theme)
      : base(theme)
    {
        this.ShowDropShadow = true;
        this.BackColorStart = Color.Gray;
        this.BackColorEnd = Color.White;
    }
}
```



Activity Toolbox Items

- Allows definition of activity “package” in toolbox
  - Class derives from ActivityToolboxItem
  - Attached to activity using ToolboxItemAttribute
  - Override CreateComponentsCore to Create activities

    Default implementation creates one activity

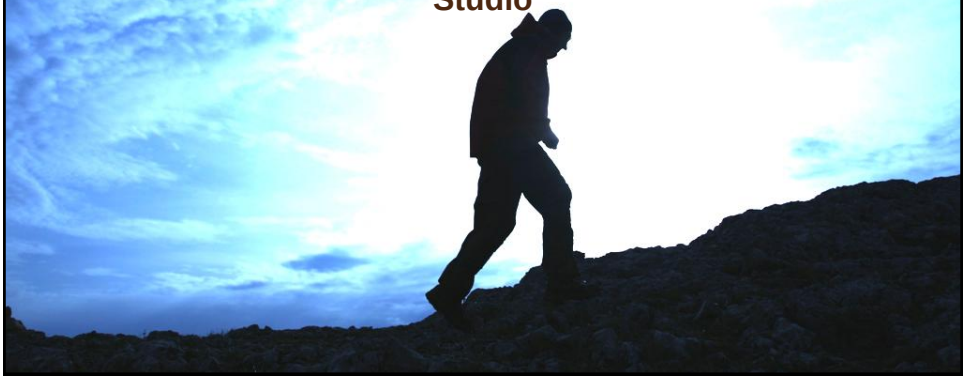
```
[ToolboxItem(typeof(ConsoleContainerActivityToolboxItem))]
public class ConsoleContainerActivity : SequenceActivity

public class ConsoleContainerActivityToolboxItem : ActivityToolboxItem
{
    protected override IComponent[] CreateComponentsCore(
        IDesignerHost host) {
        ConsoleContainerActivity container =
            new ConsoleContainerActivity();
        container.Activities.Add(new WriteConsoleActivity());
        return new IComponent[] { container };
    }
}
```

Summary

- Create activities
  - Events and Dependency Properties
  - Custom Activity Validators
  - Custom Activity Designers
- Understanding Activity Lifecycle
- Creating Composite Activities

Extending SharePoint Designer with Custom Activities

Developing SharePoint Workflow Templates with Visual Studio

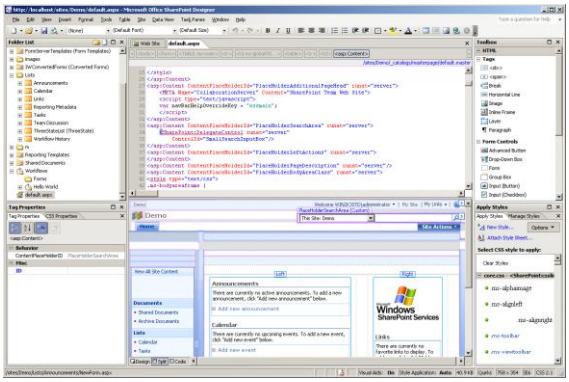


Agenda

- SharePoint Designer vs. VS 2008
- No Code Workflows
- Building Workflows in SharePoint Designer
- Extending SharePoint Designer with Custom Activities

What is the SharePoint Designer?

- SPD is designed for non developers
  - Allows WYSIWYG page management
  - Allows rule based workflow development

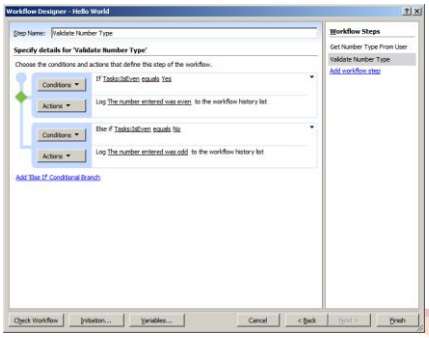


Which is better? VS 2008 or SPD

- SPD is primarily a non-developer tool
  - Allows tasks to be done easily, not efficiently
  - Modifications to content not easily redeployed
  - Useful for per-site modifications
- Rule of thumb
  - SPD for end user and non deployable solutions
  - VS 2008 for deployable solution packages

No Code Workflows

- Declarative definition made up of multiple files
  - Wfconfig.xml referencing all needed iles
  - XOML file containing the workflow
  - ASPX files defining the workflow forms



Wfconfig.xml file

- Defines the structure of the workflow
  - Defines workflow template and association
  - Defines the location and data layout of all forms
  - Always named <workflow file>>.wfconfig.xml

```
<WorkflowConfig>
  <Template
      BaseID="{8F5D9FAF-6F2A-4EF9-9A70-5E7CD49C294D}"
      DocLibID="{1891A50F-275C-4180-B338-6B589BDD04F6}"
      XomlHref="Workflows/Hello World/Hello World.xoml“
      RulesHref="Workflows/Hello World/Hello World.xoml.rules"
      ... >
  </Template>
  <Association
      ListID="{079FE649-0788-4229-8996-8863DBA5DCD7}"
      TaskListID="{0B84F02E-031D-48ED-BAAC-D8A878CFADBC}"
      StartManually="true”>
  </Association>
```

Wfconfig.xml file

```
<ContentTypes>
  <ContentType Name="Get Number Type" ContentTypeID="0x01080100…"
    Form="/Workflows/Hello World/Get Number Type.aspx" ...>
    <Fields>
      <Field DisplayName="IsEven" ...>
        <Default>1</Default>
      </Field>
    </Fields>
  </ContentType>
</ContentTypes>
<Initiation URL="Workflows/Hello World/Hello World.aspx">
  <Fields>
    <Field Name="Value" DisplayName="Value" Type="Number" ...>
      <Default>0</Default>
    </Field>
  </Fields>
  <Parameters>
    <Parameter Name="Value" Type="System.Double" />
  </Parameters>
</Initiation>
</WorkflowConfig>
```

Deploying No Code Workflows

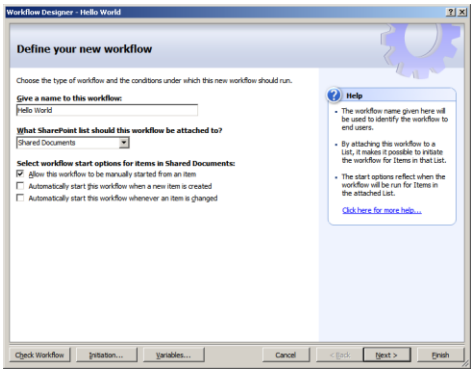
- Deployment handled via web service
  - websvcWebPartPages exposes workflow methods

    `FetchLevelWorkflowActions`
    `ValidateWorkflowMarkupAndCreateSupportObjects`
    `AssociationWorkflowMarkup`

- Used by SharePoint Designer to create workflow

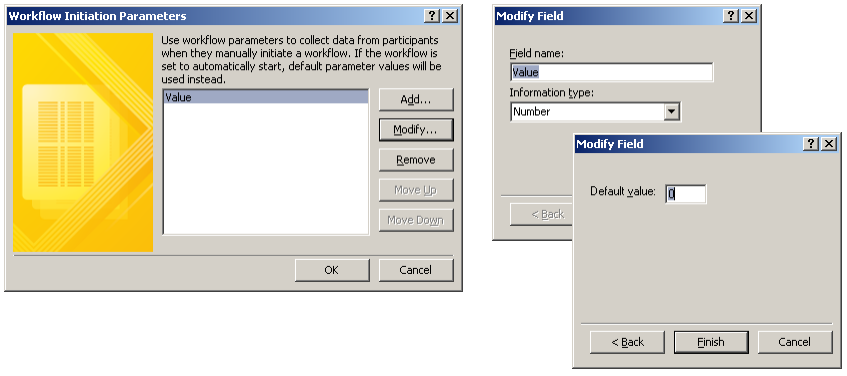
Creating SPD Workflow

- Association created automatically
  - List and startup parameters chosen at design time



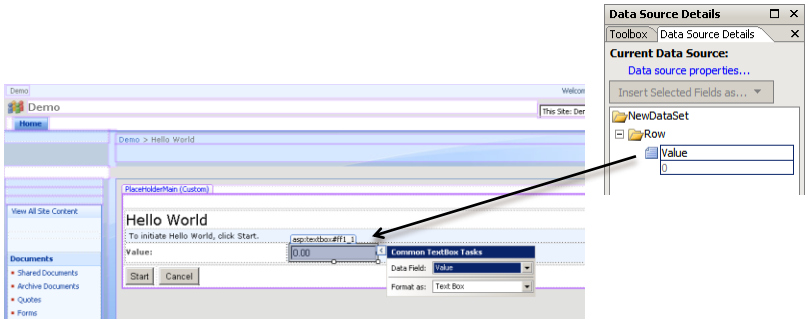
Workflow InstantiationForm Definition

- Instantiation form generated automatically
  - Initiation form made up of a set of fields
  - Each field has a name and type



Customizing Instantiation Form

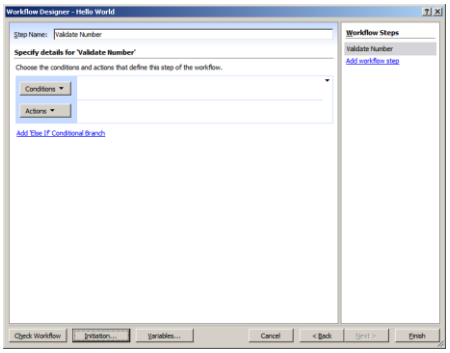
- Generated form designed as an .ASPX page.
  - Initiation form fields available in Data Source pane
  - Controls displaying fields can be formatted
  - Page is designed just like any other .ASPX page



Building Workflow in SPD

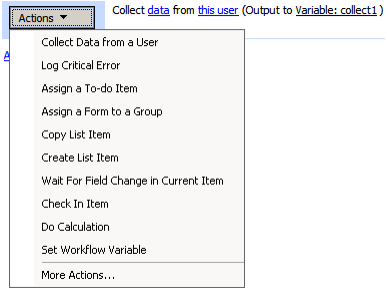
- SharePoint Designer provides workflow builder
  - Build workflow based on basic rules
  - Rules made up of conditions and actions

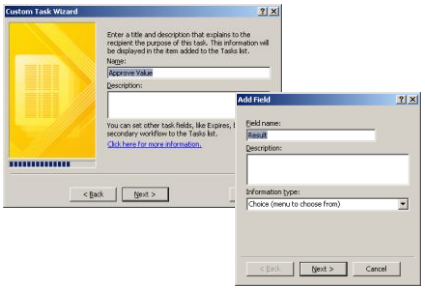
    `Similar to Outlook rules`



Workflow Task Form Definitions

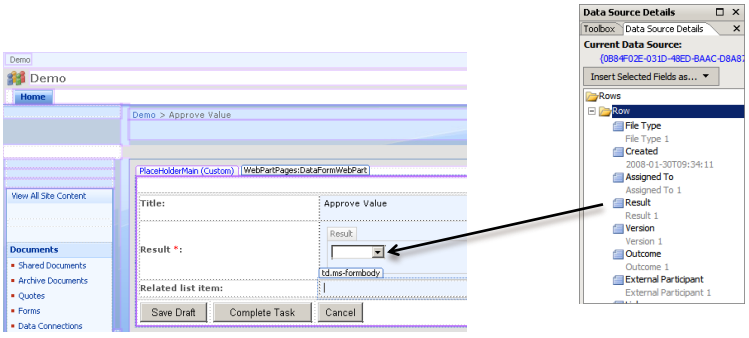
- Tasks handled automatically by designer
  - Single action creates task and waits for completion
    `Entire task process in one activity`
  - Form is generated automatically based on fields





Customizing Task Form

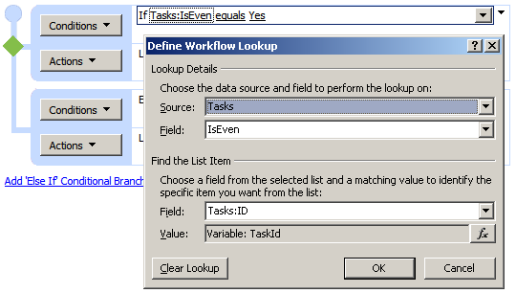
- Generated by workflow wizard as .ASPX page
  - Same process as editing Instantiation form
    `Different data source`
  - Task form fields available in Data Source pane



Adding Conditions

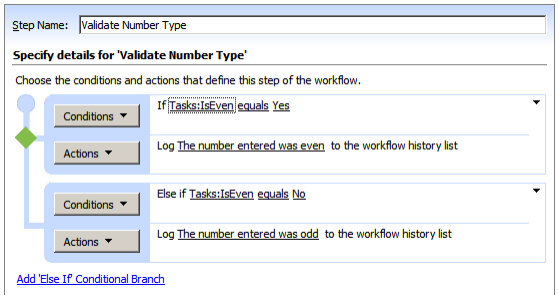
- Conditions determine if related actions execute
  - Conditions can compare fields or constant values
  - Fields defined as fields from forms or current item
  - Other lists can be accessed using simple “queries”

```
Task data is stored in Tasks list
```



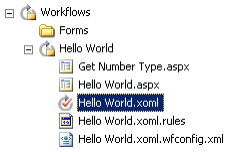
Multiple Conditions

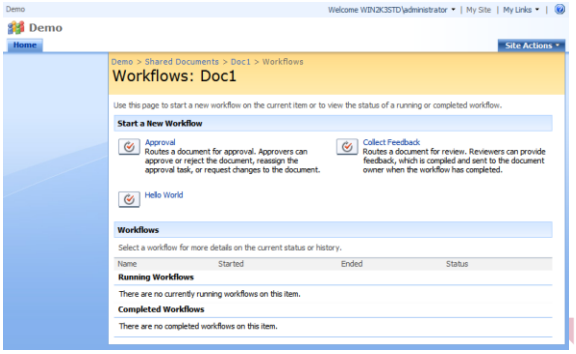
- Steps can contain multiple condition/actions
  - Each condition executed separately
  - Allows if/else structure in workflow rules



Completed Workflow

- Workflow stored in SharePoint site
  - All files needed are generated automatically
    `Hard coded lists make deployment difficult`
  - Workflow initiated same as any other workflow





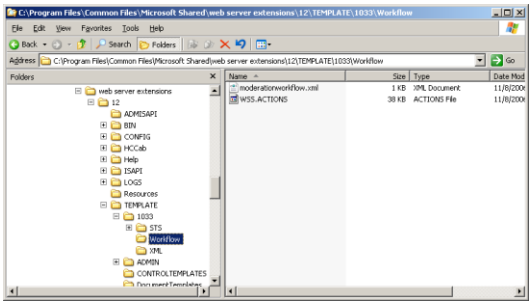
What are Conditions and Actions?

- Conditions and Actions are shortcuts
  - Groups smaller actions into larger functional pieces
  - Allows non-developers to avoid fine grained activities
  - Developers build actions for non-developers to use

Developers	Non-Developers
Build Custom Actions Build Custom Conditions	Build Workflows • Use Custom Actions • Use Custom Conditions

WSS.ACTIONS

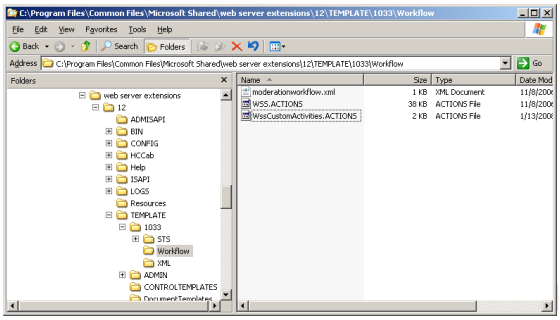
- Contains definitions for conditions and actions
  - Maps conditions to methods
  - Maps actions to custom activities
  - Parameters are mapped to user friendly options
  - Resides in TEMPLATE\1033\Workflow folder



Custom Conditions and Actions

- Add custom items by creating second .actions file
  - Must have the .actions extension
  - Made up of a conditions and actions section
  - Must exist in the TEMPLATE\1033\Workflow folder

    `1033 is US English, for others use appropriate ID`



Defining Custom Conditions

- Custom conditions are static methods
  - Method must have specific signature

    Must return bool

    Must accept context parameters

```
public class WssWorkflowConditions
{
    public static bool IsEven(WorkflowContext context,
        string listId, int itemId, double data)
    {
        return ((int)data % 2) == 0;
    }
```


Custom Conditions in .ACTIONS

- .ACTIONS maps methods to sentences
  - Sentences define placeholders for parameters
  - Placeholders are mapped to method parameters

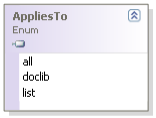
```
<Condition Name="Is even" FunctionName="IsEven" AppliesTo="all"
    ClassName="WssCustomActivities.WssWorkflowConditions"
    Assembly="WssCustomActivities, ...">
      <RuleDesigner Sentence="%1 is even">
        <FieldBind Id="1" Field="_1_" Text="value" />
      </RuleDesigner>
      <Parameters>
        <Parameter Name="_1_" Direction="In"
                   Type="System.Double, mscorlib" />
      </Parameters>
    </Condition>
```

Scoping Conditions

- Most conditions and activities apply globally
  - What if a they only work on a list item?
  - .ACTIONS files allow scoping to specific types
  - Defined in the AppliesTo attribute

    Applies to conditions and activities

```
<Action ... AppliesTo="all" >
```



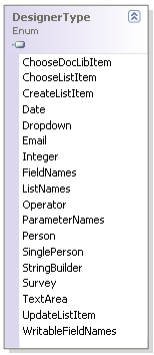
Rule Designers

- Designer made up of sentence and fields
  - Sentence defines what the user sees
  - Placeholders replaced by parameters in designer

    Click placeholder to select parameter
  - Dialog displayed depends on DesignerType

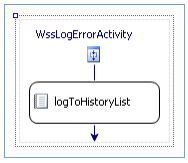
    Defaults to field chooser

```
<RuleDesigner Sentence="Log '%1' as critical error">
  <FieldBind Field="ErrorMessage" Id="1"
             DesignerType="TextArea" />
</RuleDesigner>
```



Defining Custom Actions

- Custom Activities are defined as WinWF Activities
  - Activities are often designer based Sequential activities
    Although any activity will work
  - Exposes parameters as dependency properties




```
public static DependencyProperty ErrorMessageProperty =
    DependencyProperty.Register("ErrorMessage",
    typeof(System.String),
    typeof(WssLogErrorActivity));

public String ErrorMessage
{
    get { ... }
    set { ... }
}
```

Custom Activities in .ACTIONS

- .ACTIONS maps Activities to sentences
  - Sentences define placeholders for parameters
  - Placeholders are mapped to method parameters
  - All parameters must be mapped to dependency properties

```
<Action Name="Log Critical Error" AppliesTo="all" Category="Core Actions"
        ClassName="WssCustomActivities.WssLogErrorActivity"
        Assembly="WssCustomActivities, ..." >
  <RuleDesigner Sentence="Log '%1' as critical error">
    <FieldBind Field="MyErrorMessage" DesignerType="TextArea" Id="1" />
  </RuleDesigner>
  <Parameters>
    <Parameter Name="MyErrorMessage" Direction="In"
               Type="System.String, mscorlib" />
  </Parameters>
</Action>
```

Special Activity Parameters

- Special parameters are available to activities
  - Defines the workflow context, current list or item
  - Special names required in the .ACTIONS file

```
<Parameter Name="__Context"
  Type="Microsoft.SharePoint.WorkflowActions.WorkflowContext, ..."
  Direction="In"/>
<Parameter Name="__ListId" Type="System.String, ..."
  Direction="In" />
<Parameter Name="__ListItem" Type="System.Int32, ..."
  Direction="In" />
```


Working with .ACTIONS files

- SP Designers accessed data via web services
  - Changes to .ACTIONS requires AppPool recycle
  - Changes do not require SP Designer restart

- Common Problems
  - Can't load custom conditions and activities

    `Make sure Method, Class, Assembly Names correct`
  - Can see custom items, but can't use them

    `Make sure parameter names and types match`

Summary

- SharePoint Designer vs. VS 2008
- No Code Workflows
- Building Workflows in SharePoint Designer
- Extending SharePoint Designer with Custom Activities
-
-
-
-