

The SOA Magazine

Feature Article



Workflow-Enabled Services with Windows Workflow Foundation

by Nitin Gandhi

Published: May 30, 2009 (SOA Magazine Issue XXIX: May/June 2009)

[Download PDF](#)

[Digg This](#) • [De.licio.us](#) • [Slashdot](#) • [Technorati](#) • [StumbleUpon](#) • [Google Bookmark](#)

Abstract: A workflow is a collection of activities that describes a business process. Modeling business processes is a key requirement for developing service compositions. Workflows provide a way of describing business process logic by expressing the order of execution and relationships between various activities. This article, comprised of introductory pre-release content from the upcoming SOA with .NET book [REF-1], introduces the Windows Workflow Foundation (WF) and explores how it can be used to model and implement workflow logic in support of carrying out that logic via composed services.

Introduction

In some cases the business process is automated, while in other cases it may involve human intervention. Very often it is possible to describe a process as a discrete series of steps that may involve people and software. A workflow is a set of activities that co-ordinate people and software.

The duration of the business process is an important consideration when modeling a workflow; some business processes may involve systems and human intervention and potentially run for several hours, days, or even weeks. Windows Workflow Foundation (WF), introduced in .NET 3.0, was created to meet these requirements, as shown in Figure 1.

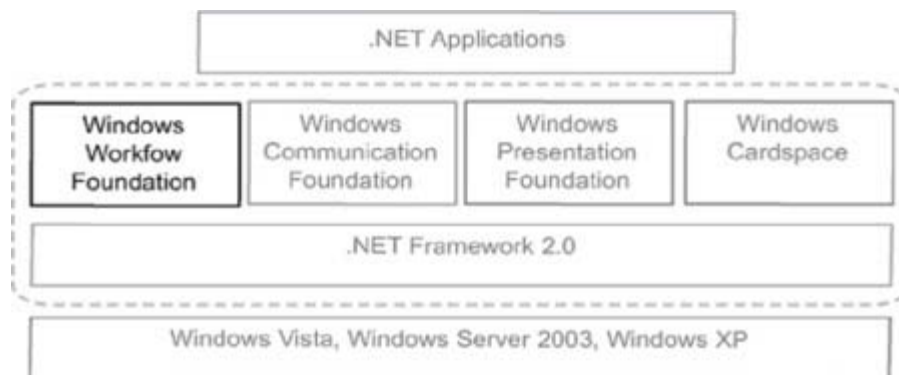


Figure 1: Windows Workflow Foundation is a part of .NET 3.0.

Windows Workflow Foundation (WF) is the single workflow technology for Windows. At the core, WF enables execution of steps required to complete a business process and is used to construct workflow-enabled applications on the Microsoft platform. Workflow solutions are becoming increasingly important as the process-oriented view of software grows.

The WF platform contains a programming model, a workflow engine, and designers for Visual Studio. The automation logic in WF extends to several scenarios, including document-centric workflows, human workflows, business rules driven workflows, and others. It is used to enable both human and system workflows and to wire up opaque blocks of functionality called activities that represent discrete steps in business process. WF has an extensible programming model for building custom activities that encapsulate

workflow functionality.

Windows Workflow Foundation architecture

Major components of Windows Workflow Foundation (WF) are shown in Figure 2 and Table 1. WF includes workflow, activities, WF base activity library, WF runtime engine, and WF runtime services.

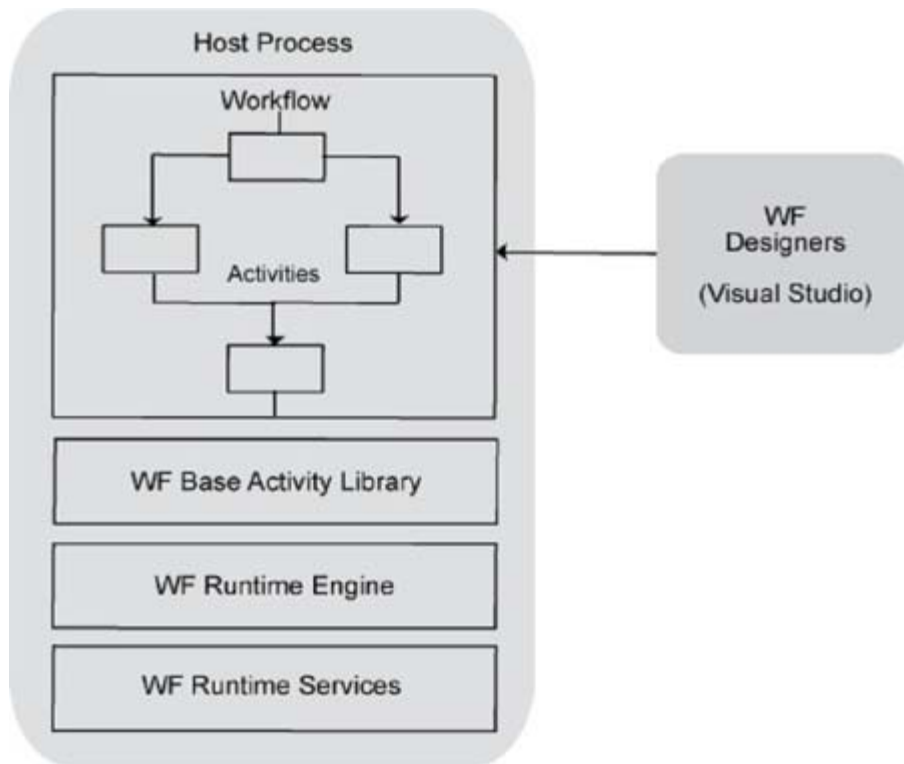


Figure 2: Major components that constitute Windows Workflow Foundation (WF). Each component of WF is explained in Table 1.

Activity	A unit of work, or a discrete step in a business process.
Workflow	A sequence of activities used to implement a business process.
Workflow runtime engine	A workflow instance is created and executed by the WF runtime engine. The runtime engine also manages the state and the communication with the host process.
WF designers	UI tools used to implement a workflow using shapes. Visual Studio includes a WF designer.
WF runtime services	Services that provide hosting flexibility and communication.
Host process	An application that hosts the WF runtime engine which executes the workflow. The host process provides support for runtime services such as persisting the workflow's state.

Table 1: Various components of Windows Workflow Foundation (WF).

WF runtime services are architectural plug points for plug-in resource providers. For example, the default persistence behavior provided by the runtime engine can be changed by providing a runtime service.

WF supports compensating work for a given activity or transaction. It uses the notion of compensation to undo an activity. Within the application logic, WF can define compensators that are invoked if an activity fails.

Workflows

By using WF, rather than injecting the workflow logic into the source code, each step in the business

process is defined explicitly and executed by the workflow engine providing a cleaner, reusable, and maintainable implementation. By providing a common workflow technology, WF used to build workflow solutions across various Microsoft products and custom development is streamlined.

WF supports both system workflows and human workflows. To meet these requirements, WF supports two built-in workflow types: sequential workflows and state machine workflows. Both rely on the same runtime environment and use the same set of standard activities that come with WF. A workflow can also be a composite of both types, containing activities that rely on the system and on human intervention.

Sequential Workflows

Sequential workflows execute activities in a pre-defined pattern. This type is better suited for system workflows. With sequential workflows, execution of activities closely resembles a flowchart with branches, decision logic, loops, and other control structures.

State Machine Workflows

State machine workflows execute activities as external events occur. This type, based on the finite state machine pattern, is better suited for workflows involving human intervention. The next activity to be executed depends on the current state and the event it has received. State machine workflows are useful when the sequence of events is not known in advance or when the number of possibilities makes defining all possible paths impractical.

Workflows created in a designer are stored in a XAML file. XAML is a declarative XML-based language used to define objects, their properties, relationships, and interactions. On execution, the runtime engine takes the XAML workflow and creates workflow instances. While there is only one XAML-based workflow file, there can be multiple workflow instances running at any time.

WF supports passivation and activation of the instance state, allowing a workflow instance to be serialized to a data store such as SQL Server. The workflow instance can be restored to the original execution state at any time by de-serializing the data based on events or messages.

Activities

A workflow is a sequence of activities executed by the workflow engine. An activity should be modeled as a real-world step required for completing the business process. An activity is a class that encapsulates the logic required by the business process. Activities can potentially be reused across different workflows.

Out of the box, WF includes several activities known as the base activity library. Activities from the base activity library commonly used with sequential workflows are listed in Table 2.

Activity	Description
Looping and synchronization activities	
IfElse	This activity allows conditions to be specified in the workflow. The runtime engine evaluates a condition and acts upon it based on the result. The IfElse activity may contain other IfElse activities and a default IfElse activity if no other condition is met.
While	The While activity accepts a condition and evaluates it at the beginning of every iteration. If the condition is true, the child activity is run repeatedly until the condition becomes false.
Replicator	The Replicator activity executes a child activity a given number of times. This is similar to the foreach statement in C#.
Sequence	This activity is used to execute a group of activities one at a time in a predefined order.
Parallel	This activity executes two or more sequences of activities in parallel or in an interleaved manner. All sequence activities must be completed before

	the workflow moves to the next activity.
Activities that enable human intervention	
Listen	This activity is used to idle the workflow process and wait for a wake up call. The Listen activity is typically used when human interaction is required; it serializes the workflow and goes into a passive mode when it is waiting for human intervention. On receiving an event, it reactivates the workflow and continues with the processing logic.
EventDriven	The Listen activity must contain child activities that represent human events. This is implemented by using the <code>EventDriven</code> activity. A Listen activity must contain <code>EventDriven</code> activities.
HandleExternalEvent	The <code>HandleExternalEvent</code> is invoked when an event specified in an interface with an <code>EventName</code> is raised. The <code>HandleExternalEvent</code> activity is used by WF to communicate with an external service.
Delay	This activity is used to suspend the execution of the workflow for a specified amount of time.
Activities that invoke and execute code	
Code	The <code>Code</code> activity allows source code to be injected directly into the workflow. It fires the <code>ExecuteCode</code> event which executes the code. The <code>Code</code> can call an external assembly.
CallExternalMethod	This activity is used to call a method in a class available to the workflow. The interface and its implementation must be available in the same assembly.
InvokeWorkflow	This activity invokes another workflow to start executing.
InvokeWebService	This activity invokes a Web service external to the workflow application. This activity creates a Web reference to a Web service and allows operations on the service to be invoked.
WebServiceInput	The <code>WebServiceInput</code> activity enables a workflow to receive a Web service request.
WebServiceOutput	A <code>WebServiceOutput</code> activity pairs with a <code>WebServiceInput</code> activity to respond to a service request. In order to use this activity, the <code>WebServiceInput</code> activity must be configured first.
TransactionScope	This activity is used to represent <code>System.Transactions</code> in WF. It supports all the properties currently supported by <code>System.Transactions</code> .
Terminate	This activity is used to terminate the execution of the workflow.

Table 2: Base library activities commonly used with sequential workflows.

State machine workflows provide a way of defining workflows that match an organizations business process. It is based on the state machine pattern and uses states, events, and transitions to model a workflow. A state represents a snapshot of the business process. An event is an outside stimulus. The workflow is always in one state and will transition to a new state when it receives an event. Typically some action will take place in the outside world for the state in the workflow to be transitioned to a new state. On reaching the final state, the workflow is completed.

The base activity library includes several activities designed to enable state machine workflows in WF, as listed in Table 3.

Activity	Description
----------	-------------

State	This activity represents a state in a state machine workflow. When an event arrives, the workflow will transition from one State activity to a new State activity.
EventDriven	An EventDriven activity represents an event handler in a state machine and is placed inside a State activity. They represent legal events for a state.
SetState	This activity is used to model transitions in a state machine workflow. It includes TargetStateName property that points to the destination state.
StateInitialization and StateFinalization	StateInitialization activity runs when the state machine transitions into the state containing the initialization activity. The StateFinalization activity runs when the state machine transitions out of a state. These activities are used to perform pre- and post-processing in a state.

Table 3: Base library activities commonly used with state machine workflows.

A state workflow is typically consumed by one or more user interface (UI) components. The UI components must reflect the current state of the workflow and allow users to only perform legal events. WF includes the StateMachineWorkflowInstance class which provides an API to manage and query a state machine workflow. The class includes properties used to fetch the current state name and find legal transitions for the state. It also includes properties that provide history of all the states the workflow has been through.

Workflow Runtime Environment

A workflow instance is created and executed by the workflow runtime engine. The runtime engine relies on several runtime services for persisting the workflow's state, managing transactions, tracking workflow's execution, and other features. Each instance of the runtime engine can support multiple instances of workflow concurrently. The workflow instance runs in a host process or in an application domain and can be hosted in ASP.NET Web sites, Windows forms, Windows services, Web services, or Sharepoint.

The runtime engine is powered by runtime services which provide a rich execution environment for transactions, persistence, tracking changes, timer, and threading. Runtime services can be augmented by plugging in custom services that allow changing the behavior of the runtime engine to meet the specific needs of the execution environment. For most workflow implementations, the default implementation of runtime services meets the needs. However, in some cases the behavior of the runtime engine may need to be altered. For example, the workflow may require the host application and the runtime engine to communicate differently and this would require building custom services.

The workflow execution starts by creating an instance of the workflow. It proceeds with executing activities until it is required to idle the execution, at which point the instance state is persisted to disk, as shown in Figure 3.

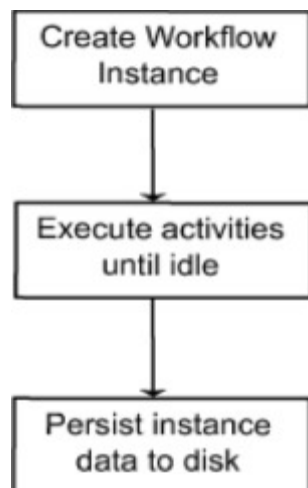


Figure 3: A workflow executed by the workflow runtime engine.

The WF Programming Model

Windows Workflow Foundation (WF) classes are encapsulated in three namespaces, as shown in Figure 4. The `System.Workflow.Runtime` assembly contains the `WorkflowRuntime` class which is used to create an instance of a workflow.

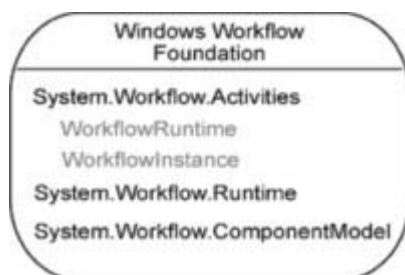


Figure 4: Workflow Foundation classes are encapsulated in three namespaces.

The workflow application presented here is a simple calculator function shown in Figure 5. The sequential workflow contains a calculator activity. The host instantiates the workflow and provides it with parameters including the input values and the operation to perform. In this case, we use a simplistic scenario where we have one activity to which we provide a couple of values. The business activity adds the values and outputs the result to the host application.

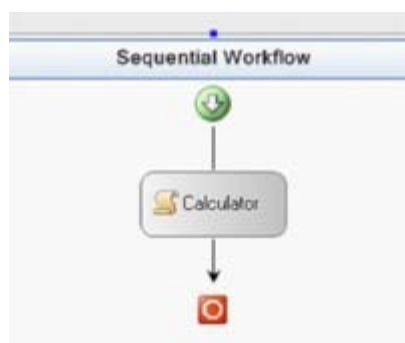


Figure 5: A simple sequential workflow which contains one activity that implements a calculator.

The workflow application is hosted in a console application as shown in the code fragment below. Workflow activities are typically coarser business activities and not as fine-grained or task-specific as shown in the example. In this case, a simplistic scenario is used to explain the programming model.

```
#C#
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Workflow.Runtime;
using System.Workflow.Runtime.Hosting;
#endregion
namespace WFWorkflow
{
    class Program
    {
        static void Main(string[] args)
        {
            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(WFWorkflow.CalcWorkflow));
            }
        }
    }
}
```

```

        instance.Start();
        waitHandle.WaitOne();
    }
}
}
}

```

```

#VB.NET
#Region "Using directives
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.Threading
Imports System.Workflow.Runtime
Imports System.Workflow.Runtime.Hosting
#End Region
Namespace WFWorkflow
Class Program
    Shared Sub Main(ByVal args() As String)
        WorkflowRuntime workflowRuntime = Functio
        WorkflowRuntime() As Shadows
        waitHandle.Set()
    End Function
    workflowRuntime.WorkflowTerminated Function
    delegate(ByVal sender As Object, ByVal e As
WorkflowTerminatedEventArgs) As +=
        Console.WriteLine(e.Exception.Message)
        waitHandle.Set()
    End Function
    Dim instance As WorkflowInstance =
        workflowRuntime.CreateWorkflow(Type.GetType(WFWorkflow.CalcWorkflow))

        instance.Start()
        waitHandle.WaitOne()
    End Sub
End Class
End Namespace

```

As listed above, the host process starts the workflow runtime and then starts the workflow method itself. The workflow runtime is started by instantiating the `workflowruntime` class. A workflow instance class is then created by passing in the workflow type to the `CreateWorkflow` method. The start method on the workflow instance kicks off the workflow business process.

There are several events raised by the workflow runtime environment. These include `WorkflowCompleted` and `WorkflowTerminated`. `WorkflowTerminated` is called when there is an error. The code fragment above uses anonymous delegates; `WorkflowTerminatedEventArgs` provides information on the exception that was generated. When the `WorkflowCompleted` event is called, we just set the `waitHandle`.

Workflow-Enabled Web Services

WF makes it possible to expose a workflow as a Web service. Only a workflow that uses the `WebServiceReceive` activity can be published as a Web service. A simple scenario would be to create a workflow project and add `WebServiceReceive` and `WebServiceResponse` activities to it. In this case, the workflow is activated by calling a method on the Web service and returns a value when the processing is complete.

In Visual Studio, a workflow project can be published as a Web service by right-clicking a workflow project and selecting "Publish as Web service". This action will create an ASP.NET project, with ASMX and Web.Config files to host the workflow as a Web service.

Business Rules in Windows Workflow

