Your Short Cut to Knowledge

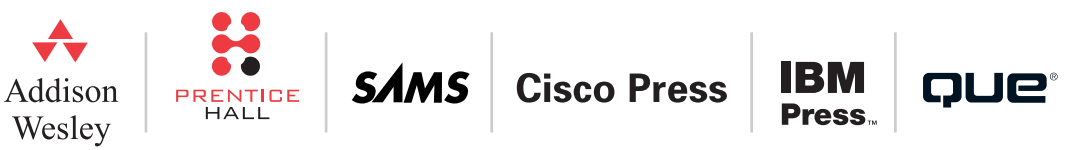
The following is an excerpt from a Short Cut published by one of the Pearson Education imprints.

Short Cuts are short, concise, PDF documents designed specifically for busy technical professionals like you.

We've provided this excerpt to help you review the product before you purchase. Please note, the hyperlinks contained within this excerpt have been deactivated.

Tap into learning—NOW!

Visit **www.informit.com/shortcuts** for a complete list of Short Cuts.



Requirements for Workflow Infrastructure

Given the structure of modern business applications, what kinds of problems does a workflow foundation have to solve?

First, business processes often take days, weeks or months to complete. Since human actions and decisions are part of the process, much of this time is spent waiting for something to occur rather than in constant calculation.

This has several significant implications:

- Workflows often have to wait days, weeks, or months for events to occur, or for other activities to finish.
- Workflows should not consume processor cycles while waiting.
- Workflows must allow for scalability with many multiple instances.
- Workflows and their associated state must survive machine resets.
- Workflows often require asynchronous control flow because they must wait for human decisions, or the results of other business processes
- Workflows often require the use of a compensation model, instead of traditional database transactions.

Second, business analysts need to design their business processes using concepts they understand. In other words they must use domain specific knowledge. The workflow design tool they use might be visual, but it does not have to be. In any case, the design tool must allow the business

© 2007 Pearson Education. All rights reserved.
This publication is protected by copyright. Please see page 2 for more details.

analyst to change the workflow in response to rapid changes in the business environment. To use the current jargon *du jour*, the business must be *agile*.

The implication here is that this designer must be capable of turning the analyst's design into executable code. You can see the attempt to do this in the application integration space with the BizTalk designer generating BPEL orchestrations.

These requirements for workflow translate into the following requirements for the Windows Workflow infrastructure:

- Workflow programs are ***reactive***; they must be capable of responding to actions and data that are generated by humans or machines. These actions and data may or may not be generated within the workflow.
- To avoid consuming processor cycles while waiting,[4] and to survive the inevitable machine resets that will occur over the long time workflows take, the state of the workflow must be capable of being saved to a storage medium, and then brought back into memory, and ***resumed***, possibly even on a different machine. This means that the workflow itself must be call stack, thread, process, and machine agnostic. You have to treat a workflow as pure data.
- While a workflow can be created independently of a design tool, any design tool must be capable of creating a workflow that can be translated into an executable.

[4] Imagine many threads waiting for input and being either blocked or busy waiting. This is not the way to build a scalable system.

© 2007 Pearson Education. All rights reserved.
This publication is protected by copyright. Please see page 2 for more details.

Windows Workflow Foundation

How Does Workflow Foundation Relate to the Requirements for the Workflow Infrastructure?

Windows Workflow Foundation (WF) is the workflow infrastructure that satisfies the requirements just outlined. WF supplies a runtime engine and a framework that you use to implement a workflow. You focus on the business logic, rules, and policies of the business process rather than building workflow infrastructure plumbing. Of course, you can customize parts of this infrastructure if you need to.

We will use five examples to illustrate how Windows Workflow Foundation satisfies the requirements we just listed. These examples only use the command line C# compiler. While most of our other examples will use Visual Studio.NET, it is important to understand the fundamental principles independent of the programming tools and design tools used. By doing this you will understand the principles underlying more sophisticated applications instead of just feeling like you are following a recipe.

None of these five examples use the activities that ship with Windows Workflow Foundation. There is nothing special or privileged about these included activities. They use the same technology that you would use to build your own activities.

The first two examples introduce the fundamental concepts of WF. The last three show how these concepts are the foundation for building modern business applications.

© 2007 Pearson Education. All rights reserved.
This publication is protected by copyright. Please see page 2 for more details.

Don't feel overwhelmed if you don't understand everything all at once. We will revisit in more detail later, the principles and classes we introduce here. You could skip over these examples and go directly to the section entitled "Using Visual Studio.NET" and come back here later. I hope you don't do that because a little patience here will translate into real understanding.

Information on setting up all the examples is found in the installation instructions that accompany the sample programs.

Host, Runtime, and Activity

The three must fundamental concepts in WF are host, runtime, and activity. These are illustrated in the first example which is just about the simplest workflow you can write. Run compile.bat to build the "Workflow and Host" example. Run the program HelloWorkflow.exe. Enter a carriage return to terminate the program. Following tradition, the sample just prints out "Hello world."

Let us examine the two files that make up this simple workflow.

The HelloActivity.cs file contains the simplest activity you can create. An ***activity*** is the means by which a workflow accomplishes some task or action. Later we will examine more complicated activities, but every activity has an *Execute* method. At the minimum, an activity goes from the Executing state to the Closed state. As we see in the code below, in the *Executing* state this activity just writes a simple string to the console before entering the *Closed* state.

```
using System;
using System.Workflow.ComponentModel;

namespace HelloWorld
```


© 2007 Pearson Education. All rights reserved.
This publication is protected by copyright. Please see page 2 for more details.

```
{
    public class HelloWorld : Activity
    {
    protected override ActivityExecutionStatus
                                Execute(ActivityExecutionContext context)
    {
        Console.WriteLine("Hello, world.");
        return ActivityExecutionStatus.Closed;
    }
    }
}
```

But who schedules and runs the activity? The workflow ***runtime*** is responsible for scheduling activities and executing them by invoking their *Execute* method. But the workflow runtime has to be created within an application. This could be a console application, a Windows Form application, a web service, or sharepoint to name a few possibilities. The application that creates and runs the workflow is known as the workflow ***host***. In these first five examples we shall use a console application as the host. The host.cs file contains the simplest host that one can write. Here is the code:

```
using System;
using System.Workflow.Runtime;

class Program
{
    static void Main(string[] args)
    {
        WorkflowRuntime workflowRuntime = new WorkflowRuntime();
```


© 2007 Pearson Education. All rights reserved.
This publication is protected by copyright. Please see page 2 for more details.

```
            workflowRuntime.StartRuntime();

            Type type = typeof(HelloWorld.HelloWorld);
            WorkflowInstance instance =
                                        workflowRuntime.CreateWorkflow(type);
            instance.Start();

            Console.ReadLine();
            workflowRuntime.StopRuntime();
        }
}
```

The host starts the workflow runtime by creating an instance of the **WorkflowRuntime** class and then invoking the *StartRuntime* method. After that it creates and starts the workflow instance. The workflow is represented as the type that implements the activity. Note that this type is identical to the type of the activity defined in the HelloActivity.cs file.

We use a *Console.ReadLine* to keep the console application alive while the workflow runs. After you enter a carriage return, the host stops the workflow runtime.

Activities and Workflow

While, in theory, you can write an entire workflow in one activity this would not be very practical. First, the activity would be enormously complex. Second, you would lose the ability to have reusable activities. In other words, if parts of different business processes shared similar functionality you would not be able to reuse that functionality.

© 2007 Pearson Education. All rights reserved.
This publication is protected by copyright. Please see page 2 for more details.