Introducing Windows Workflow Foundation 4.0

A few years back, Microsoft shipped an API named Windows Workflow Foundation (WF) with the release of NET 3.0. This API allowed you to model, configure, monitor, and execute the workflows (which are used to model a business process) used internally by a given .NET program. The out-of-the-box functionality provided by the first release of WF was very a welcomed step in the right direction, as we were no longer required to manually develop complex infrastructure to support workflow-enabled applications.

As intriguing a technology as the first release of WF was, it was not without its warts. A number of developers felt the designer experience provided by Visual Studio 2008 was lackluster and that it was too cumbersome to navigate complex workflow during development. As well, the initial release of WF required a good deal of boilerplate code to get a workflow up and running, and even the act of building the workflow itself was a bit clunky, given the fact that your C# code base and the related workflow designer representation did not mesh together as seamlessly as hoped.

.NET 4.0 gives us a complete reboot of the entire WF API. Going forward, workflows are modeled (by default) using a declarative XML-based grammar named XAML where data used by the workflow is treated as a first class citizen. As well, the Visual Studio 2010 WF designers have been completely overhauled and rewritten using Windows Presentation Foundation (WPF) technologies. So, if you used the previous version of the WF API and were a tad dissatisfied, I encourage you to read on with a fresh set of eyes.

If you are new to the topic of WF, this chapter begins by defining the role of business processes and describes how they relate to the WF API. As well, you will be exposed to the concept of a WF activity, the two major flavors of workflows under 4.0 (flowchart and sequential), and various project templates and programming tools. Once we’ve covered the basics, we’ll build several example programs that illustrate how to leverage the WF programming model to establish business processes that execute under the watchful eye of the WF runtime engine.

---

**Note** The entirety of WF 4.0 cannot be covered in a single introductory chapter. If you require a deeper treatment of the topic than presented here, check out *Pro WF: Windows Workflow in .NET 4.0* by Bruce Bukovics (Apress, 2010).
Defining a Business Process

Any real-world application must be able to model various business processes. Simply put, a business process is a conceptual grouping of tasks that logically work as a collective whole. For example, assume you are building an application that allows a user to purchase an automobile online. Once the user submits the order, a large number of activities are set in motion. You might begin by performing a credit check. If the user passes the credit verification, you might start a database transaction in order to remove the entry from an Inventory table, add a new entry to an Orders table, and update the customer account information. After the database transaction has completed, you still might need to send a confirmation e-mail to the buyer, and then invoke a remote service to place the order at the car dealership. Collectively, all of these tasks could represent a single business process.

Historically speaking, modeling a business process was yet another detail that programmers had to account for, often by authoring custom code to ensure that a business process was not only modeled correctly but also executed correctly within the application itself. For example, you may need to author code to account for points of failure, tracing, and logging support (to see what a given business process is up to); persistence support (to save the state of long-running processes); and whatnot. As you may know firsthand, building this sort of infrastructure from scratch entails a great deal of time and manual labor.

Assuming that a development team did, in fact, build a custom business process framework for their applications, their work was not yet complete. Simply put, a raw C# code base cannot be easily explained to nonprogrammers on the team who also need to understand the business process. The truth of the matter is that subject matter experts (SMEs), managers, salespeople, and members of a graphical design team often do not speak the language of code. Given this, as programmers, we were required to make use of other modeling tools (such as Microsoft Visio, the office whiteboard, etc) to graphically represent our processes using skill set–neutral terms. The obvious problem here is we now have two entities to keep in sync: If we change the code, we need to update the diagrams. If we change the diagrams, we need to update the code.

Furthermore, when building a sophisticated software application using the 100% code approach, the code base has very little trace of the internal “flow” of the application. For example, a typical .NET program might be composed of hundreds of custom types (not to mention the numerous types used within the base class libraries). While programmers may have a feel for which objects are making calls on other objects, the code itself is a far cry from a living document that explains the overall sequence of activity. While the development team may build external documentation and workflow charts, again there is the problem of multiple representations of the same process.

The Role of WF 4.0

In essence, the Windows Workflow Foundation 4.0 API allows programmers to declaratively design business processes using a prefabricated set of activities. Thus, rather than only using a set custom of assemblies to represent a given business activity and the necessary infrastructure, we can make use of the WF designers of Visual Studio 2010 to create our business process at design time. In this respect, WF allows us to build the skeleton of a business process, which can be fleshed out through code where required.

When programming with the WF API, a single entity can then be used to represent the overall business process as well as the code that defines it. In addition to being a friendly visual representation of the process, since a single WF document is used to represent the code driving the process, we no longer need to worry about multiple documents falling out of sync. Better yet, this WF document will clearly illustrate the process itself. With a little bit of guidance, even the most non-technical of staff members should be able to get a grip on what your WF designer is modeling.