For developers, the computing world of today is becoming much more concurrent than just a few short years ago. Computer users expect an ever-increasing computational power from their electronic gadgets, including their mobile devices. Unfortunately, it seems that the only way manufacturers will be able to increase computational speed in the near future is through adding additional processors (instead of making a single processor faster, as has been the case over the last few decades). In the case of processors on personal computers, the industry has already hit a proverbial “brick wall” and has pretty much reached the maximum computational capacity available on a single processing unit. An average personal computer today comes with two or more processing units, and the number is certain to increase.

Mobile devices still have some processing speed to grow into before they max out the processing power of a single CPU. However, the average phone will shortly have several processing units as well. In addition to that, uninterrupted Internet access on the phone is assumed—resources needed for proper functioning of an application may be spread around the world (in the “cloud”), but the user is rarely aware of that. A phone application should have the ability to access those resources seamlessly as needed—i.e., it should not stop accepting all input from the user while these resources are accessed. Rather, an application should retrieve the resources without interrupting other functionality of an application—in other words, it should obtain these resources asynchronously. The future for both personal computers and mobile devices is both concurrent and asynchronous.

How do we approach concurrent and asynchronous programming on Windows Phone 7? The answer is, certainly, with great caution, since it is not easy. To help tame that complexity, a powerful framework emerged on the .NET scene at the end of 2009. That framework, called the Reactive Extensions for .NET (also known as Rx.NET), is now available for Windows Phone 7 and provides sophisticated mechanisms to make event processing and asynchronous programming more intuitive. In this chapter, you will learn the concepts behind Reactive Extensions for .NET and build two simple applications using the Rx.NET framework. The first will search and retrieve images on Flickr asynchronously. The second will display the current weather after accepting a zip code as input from the user. As with several other chapters of this volume, a whole book could be written on the subject of Reactive Extensions alone, especially since there is a version of Reactive Extensions for JavaScript available as well.

In this chapter, you will learn the basics of Rx.NET and, we hope, leave with a basic understanding and appreciation of this technology. However, to leverage the power of Rx.NET, you need a good understanding of LINQ. Although the examples in this chapter should be relatively easy for the novice C# programmer even without an in-depth knowledge of LINQ, for expert coverage of the topic we recommend the excellent book Pro LINQ: Language Integrated Query in C#2008, by Joseph C. Rattz. Another good resource is “101 LINQ Samples,” available for free online at http://msdn.microsoft.com/en-us/vcsharp/aa336746.aspx. Rx.NET also relies heavily on general object-oriented principles; if the concept of interfaces is not familiar to you, it may be a good idea to understand those before reading this chapter. Finally, Rx.NET constructs make extensive use of the newer concepts of the .NET framework, such as lambda expressions and extension methods. While it will be possible to follow examples in this book without in-depth understanding of either of those
concepts, to leverage the power of Rx.NET on your own, you will have to understand these features of the .NET framework.

The power of Reactive Framework can be applied to deal with a wide range of computational issues. In this chapter, we will focus on the way that the framework deals with a problem that is probably as old as the personal computer: how do you provide a responsive user interface while utilizing the full computational resources available? And how can you do so in a manner that makes code readable and easy to manage/maintain?

**Introducing Reactive Programming**

Rx.NET aims to revolutionize reactive programming in the .NET framework. In reactive programming, you register an interest in something and have items of interest handed over, or pushed to the attention of the application, asynchronously, as they become available. A classic example an application that relies heavily on the reactive programming model is the spreadsheet, where an update to a single cell triggers cascading updates to every cell that references it. This concept of having things pushed down as they become available is particularly well suited to applications that use constantly changing data sources, such as the weather application that you will be building later on in this chapter.

Reactive programming is often contrasted with interactive programming, where the user asks for something and then waits for it until it is delivered. To help further differentiate these concepts, let’s take a look at a car shopping analogy. Usually, when shopping for a car, you go to a car dealership (or look online) and look at different car makes and models. You pick the ones you like and test-drive them. This arrangement is an example of interactive programming, where you asked for a car and got it in return. In a reactive approach, you would send a note to a dealership expressing interest in a certain make and model and then continue going on with your daily routine. The dealer locates items of interest and notifies you when they become available.

Let’s see if we can carry over this analogy to event processing on Windows Phone 7. For the sample application that you will be building later in this chapter, you will want to read the contents of a text box once it can be determined that no keystroke has occurred a half a second since the previous one. In the sample, this will be taken to mean that the user has finished typing and is ready for the application to do something. If you were to use an interactive approach, you would implement this by wiring up the KeyDown event for the text box, and then checking some sort of timer to see whether enough time had elapsed between keystrokes. In a reactive approach, as you will see shortly, things are much simpler: you express interest in being notified of KeyDown events only after a half-second has elapsed between a user’s keystrokes. Once notified of such an event, you take action—searching for photos online, in this case.

Before you learn how to search for photos in a reactive manner, however, you will walk through several short examples to get a feeling for how Reactive Extensions implement the core Observer pattern, which forms the basis of the Reactive Framework and is described in detail in the following sidebar.