Building a great user interface for a professional application is challenging, given all of the competing priorities, such as presenting data, interaction models, network connectivity, security, managing background tasks, and localization. It’s important to build a rich application on a solid architecture that helps the different pieces work together.

This chapter delves more deeply into user interface development with Silverlight, covering a wide range of topics. It builds on previous chapters, but especially on Chapter 2, which covered Silverlight user interface development. This chapter extends the topic of user interface development to include how to architect an application with separation of concerns without sacrificing support for tooling or ease of development.

This chapter starts with an introduction of the most prevalent application architecture model in Silverlight development, Model-View-ViewModel (MVVM), which provides excellent support for well-designed user interfaces with separation of concerns. With Windows Phone OS 7.1 (Mango), Windows Phone adds compatibility with the Silverlight 4 programming model including much improved commanding support.

After MVVM, the Silverlight for Windows Phone Toolkit is next, covering the additional controls and capabilities including the new controls available in the update for Windows Phone OS 7.1 (Mango). The section following is on creating transitions and interactivity in Expression Blend. The final section is on the Microsoft Advertising SDK, which provides an excellent vehicle to monetize applications.

The Model-View-ViewModel Architecture

The Model-View-ViewModel (MVVM) architecture originated when the Microsoft Windows Presentation Foundation (WPF) team were building the first version of Expression Blend. WPF is Microsoft’s desktop XAML development model, and Expression Blend is written in WPF. MVVM is similar to other separation-of-concerns architectures, like the tried-and-true Model-View-Controller (MVC) model; however, MVVM is optimized to take advantage of XAML’s rich data binding, data templates, commands, and event routing capabilities. The next section covers the architecture in more detail.
**MVVM Overview**

In this section, the MVVM pattern is defined to help you grasp how it works with XAML. If you are familiar with MVC, MVVM will look somewhat familiar to you—but it is much more than just MVC. MVVM relies heavily on XAML data-binding capabilities to allow the UI to bind to both data and commands. Figure 6-1 depicts the MVVM architecture.

![The MVVM architecture](image)

*Figure 6-1. The MVVM architecture*

In Chapter 4, there is a simple example that displays a list of fake Vendor data made available via JSON. The REST+JSON service project, named WcfRemoteServicesSimpleRestJSON, is added to the Chapter 6 solution. The BasicMVVM sample re-architects the AdventureWorksRestJSONPage.xaml page from the Chapter 4 CallingRemoteServices project to use MVVM in the Chapter 6 project named BasicMVVM.

The BasicMVVM and the WcfRemoteServicesSimpleRestJSON projects are configured as the startup project. Three folders are added to the project, named Model, View, and ViewModel. The sections that follow cover the major components of MVVM in the BasicMVVM example.

**BasicMVVM - Model**

The Model contains the building blocks of the application. It consists of the underlying data objects that are populated via a data access layer. Examples of Model classes are Customer, Store, Product, and so on. When you create a class to represent an object in an application, it most likely belongs as part of the Model. The Model sits behind the ViewModel. The View will data bind to lists or individual objects based on classes in the Model.

To get started, copy over the Vendor class from the WcfRemoteServicesSimpleRestJSON services project to the BasicMVVM Models folder. The class implements the INotifyPropertyChanged interface to