# Appendix A: Glossary



Microsoft® patterns & practices
proven practices for predictable results

This glossary includes definitions of important terms that appear in the Prism documentation.

**bootstrapper**. The class responsible for the initialization of an application built using the Prism Library.

**command**. A loosely coupled way for you to handle user interface (UI) actions. Commands bind a UI gesture to the logic that performs the action.

**composite application**. A composite application is composed of a number of discrete and independent modules. These components are integrated together in a host environment to form a single, seamless application.

**composite command**. A command that has multiple child commands.

**container**. Provides a layer of abstraction for the creation of objects. Dependency injection containers can reduce the dependency coupling between objects by providing the facility to instantiate instances of classes and manage their lifetime based on the configuration of the container.

**DelegateCommand**. Allows delegating the commanding handling logic to selected methods instead of requiring a handler in the code-behind. It uses .NET Framework delegates as the method of invoking a target handling method.

**EventAggregator**. A service that is primarily a container for events that allows publishers and subscribers to be decoupled so they can evolve independently. This decoupling is useful in modularized applications because new modules can be added that respond to events defined by the shell or other modules.

**modularity**. The ability to create complex applications from discrete functional units named *modules*. When you develop in a modularized fashion, you structure the application into separate modules that can be individually developed, tested, and deployed by different teams. It also helps you address separation of concerns by keeping a clean separation between the UI and business functionality.

**model**. Encapsulates the application's business logic and data.

**module**. A logical unit of separation in the application.

**ModuleCatalog**. Defines the modules that the end user needs to run the application. The module catalog knows where the modules are located and the module's dependencies.

Silverlight Class Library

**ModuleManager**. The main class that manages the process of validating the module catalog, retrieving modules if they are remote, loading the modules into the application domain, and invoking the module's **Initialize** method.

**module management phases**. The phases that lead to a module being initialized. These phases are module discovery, module loading, and module initialization.

**multi-targeted code**. Code that targets two different platforms with largely the same code-base. This allows binaries targeting two different technologies to be produced while keeping the code as much the same as possible. The technologies that Prism helps you multi-target are Windows Presentation Foundation (WPF) and Silverlight.

**navigation**. The process by which the application coordinates changes to its UI as a result of the user's interaction with the application, or as a result of internal application state changes.

**presenter-first composition**. The composition approach where the presenter is logically created first, followed by the view.

**on-demand module**. A module that is retrieved and initialized only when it is explicitly requested by the application.

**region**. A named location that you can use to define where a view will appear. Modules can locate and add content to a region in the layout without exact knowledge of how and where the region is visually displayed. This allows the appearance and layout to change without affecting the modules that add the content to the layout.

**RegionContext**. A technique that can be used to share context between a parent view and child views that are hosted in a region. The **RegionContext** can be set through code or by using data binding XAML.

**RegionManager**. The class responsible for maintaining a collection of regions and creating new regions for controls. The **RegionManager** finds an adapter mapped to a WPF or Silverlight control and associates a new region to that control. The **RegionManager** also supplies the attached property that can be used for simple region creation from XAML.

**Separated Presentation pattern**. Pattern used to implement views, which separates presentation and business logic from the UI. Using a separated presentation allows presentation and business logic to be tested independently of the UI, makes it easier to maintain code, and increases re-use opportunities.

**shell**. The main window of a WPF application or the top-level **UserControl** of a Silverlight application where the primary UI content is contained.

**scoped region**. Regions that belong to a particular region scope. The region scope is delimited by a parent view and includes all the child views of the parent view.

**service**. A service provides functionality to other modules in a loosely coupled way through an interface and is often a singleton.

**state-based navigation**. Navigation accomplished via state changes to existing controls in the visual tree.

**UI composition**. The act of building an interface by composing it from discrete views at run time, likely from separate modules.

**view**. The main unit of UI construction within a composite UI application. The view encapsulates the UI and UI logic that you would like to keep as decoupled as

possible from other parts of the application. You can define a view as a user control, data template, or even a custom control.

**view-based navigation**. Navigation accomplished via the addition or removal of elements from the visual tree.

**view-first composition**. The composition approach where the view is logically created first, followed by the view model or presenter on which it depends.

**view discovery**. A way to add, show, or remove views in a region by associating the type of a view with a region name. Whenever a region with that name displays, the registered views will be automatically created and added to the region.

**view injection**. A way to add, show, or remove views in a region by adding or removing instances of a view to a region. The code interacting with the region does not have direct knowledge of how the region will handle displaying the view.

**view model**. Encapsulates the presentation logic and state for the view. It is responsible for coordinating the view's interaction with any model classes that are required.

Next Topic | Previous Topic | Home | Community
Last built: August 28, 2012