

Composite Application Guidance for WPF and Silverlight (AKA “Prism 2”)

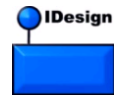
Brian Noyes
www.idesign.net

©2008 Brian Noyes, IDesign Inc. All rights reserved



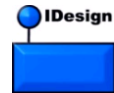
About Brian

- Chief Architect, IDesign Inc. (www.idesign.net)
- Microsoft Regional Director / MVP
- Publishing
 - Developing Applications with Windows Workflow Foundation, LiveLessons training DVD, June 2007.
 - Smart Client Deployment with ClickOnce, Addison Wesley, January 2007
 - Data Binding in Windows Forms 2.0, Addison Wesley, January 2006
 - MSDN Magazine, MSDN Online, CoDe Magazine, The Server Side .NET, asp.netPRO, Visual Studio Magazine
- Speaking
 - Microsoft TechEd US, Europe, Malaysia, Visual Studio Connections, DevTeach, INETA Speakers Bureau, MSDN Webcasts
- Participates in Microsoft Design Reviews
- E-mail: brian.noyes@idesign.net
- Blog: <http://briannoyes.net>



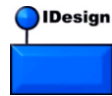
Agenda

- Composite Application Guidance Overview
- Application Architecture
- CAL Features
 - Regions
 - Modules
 - Commands
 - Events
- Patterns



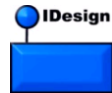
Composite Application Guidance

- Developed by Microsoft patterns and practices
- Designed as a successor to CAB/SCSF for WPF
- Ground up redesign
 - Leverage the things that are unique about WPF
 - Same motivations / fundamental requirements as CAB
 - Use lessons learned from CAB
 - ▲ Lighter weight / less intrusive
 - ▲ Stronger typing
 - ▲ Don't lock into a single container model
 - ▲ Many others



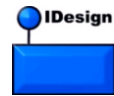
Composite Application Guidance

- Consists of:
 - Composite Application Libraries (CAL)
 - Stock Trader Reference Implementation (RI)
 - Quickstarts
 - Documentation
 - How-Tos
 - Published Spikes
- Current Version: 2.0



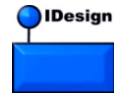
Why Composites?

- Loose coupling
- Separation of concerns
- Agility
- Maintainability
- Extensibility



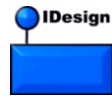
Agenda

- Composite Application Guidance Overview
- [Application Architecture](#)
- CAL Features
 - Regions
 - Modules
 - Commands
 - Events
- Patterns



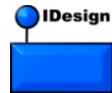
Composite WPF App Parts

- Container
- Bootstrapper
- Shell
- Modules
- Views
- Services



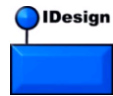
Dependency Injection

- Also known as Inversion of Control (IoC) or DI for short
- Pattern for indirect construction of objects
- Delegate responsibility for construction of objects to a Container
- Container constructs objects and injects their dependencies



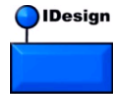
Unity Application Block

- Dependency Injection Container
 - Default for Prism (can use others though)
- Main functions:
 - Register – Tell Unity it is responsible for a type or object
 - Resolve – Obtain an object reference from the container
 - BuildUp – Inject dependencies in an existing object
 - Configure – Register based on configuration info
- Many overloads / variants
- Generic and non-generic versions



Bootstrapper

- Startup code for your application
- Alternative to putting everything in the Main() method
- Well defined initialization point in your application
- Not required, but recommended



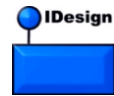
Bootstrapping Process

Configure the Container

Configure the Region
Mappings

Create the Shell

Initialize the Module



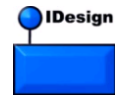
Shell

- Main window of the application
- Could be more than one
- Presented at startup (typically, could be minimized to System Tray)
- Root element for the whole UI
 - Even modular views that know nothing about the shell
- Typically knows nothing about the views that it is composed of
 - Just provides a “shell” to contain them, thus the name
- May define regions for dynamic plug in of views
- May explicitly load views into containers in the UI



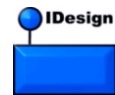
Shell Implementation

- Just a normal WPF Window
- Defines the overall layout for the application
- May contribute styles and templates that flow down to child view through resources
- Take Window1.xaml created by VS2008 and rename it to Shell.xaml
- Do WPF stuff from there
- Add regions for views to be added by modules

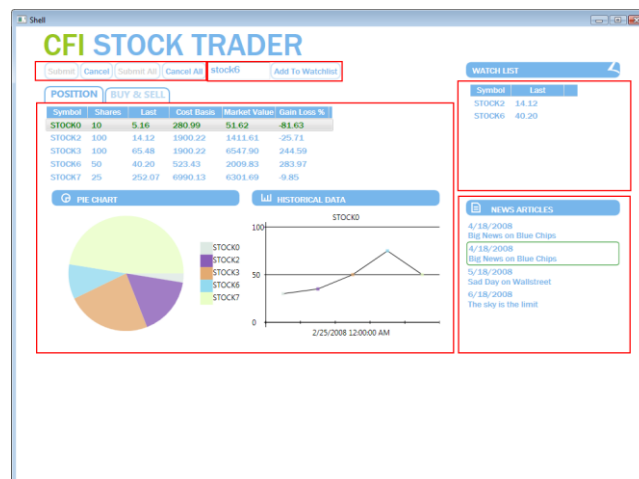


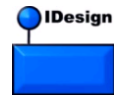
Views

- Composite parts (Legos) of your UI
- Used to decompose your window from one big monolithic blob into a bunch of semi-autonomous parts
- Can be defined as:
 - User control
 - Custom control
 - WPF Data Template
 - XAML Resource
 - Dynamically constructed UI Element tree
- Ultimately in WPF:
 - A UIElement with some backing logic



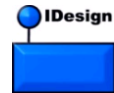
Views





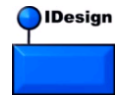
Composite View

- View that contains other views
- Child views may be added
 - By composite view statically
 - ▲ XAML declaration
 - ▲ Programmatic addition
 - Through regions within the composite view
- Composite view generally responsible for composing itself out of child views
- May have some content of its own as well



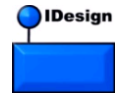
Services

- Common abstraction in composites
- Does not (necessarily) mean Web or WCF services
 - Application (in-proc) service might be point of encapsulation for the proxy that calls an external service
- Provide shared services across the application
 - Modules and shell
- Typically singleton instance model
- Container provides the glue



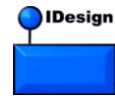
Services

- CAL Provided:
 - Logging
 - Region Manager
 - Event Aggregator
 - Module Catalog
- Custom
 - Customer service



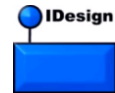
Agenda

- Composite Application Guidance Overview
- Application Architecture
- CAL Features
 - Modules
 - Regions
 - Commands
 - Events
- Patterns



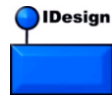
Modules

- Unit of composition for the application
 - As opposed to views as the unit of composition for the UI
- Modules contain the artifacts for a portion of your application
 - Self-contained
 - Decoupled
 - Reusable
- Typically defined as a Visual Studio project / assembly
- Can have more than one module per assembly
 - Avoid



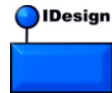
Modules

- Main purpose:
 - Initialize the objects in the module
- Like a Main() method for a library
- Known entry point in the module
- Initializes
 - Container types
 - Views
 - Regions
 - Services
 - Controllers
 - Etc



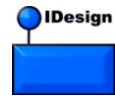
Module Loading

- CAL supports:
 - Statically
 - Dynamically
 - ^ Directory scan
 - ^ Config
 - On-demand
- Modules can be dependent on other modules
 - Need to resolve load order in that case



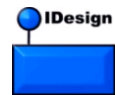
Agenda

- Composite Application Guidance Overview
- Application Architecture
- CAL Features
 - Modules
 - Regions
 - Commands
 - Events
- Patterns

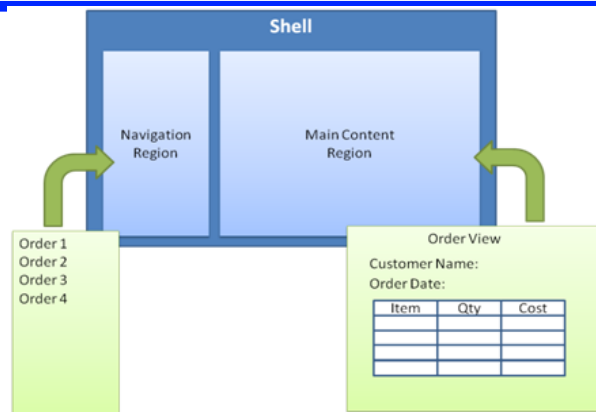


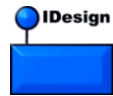
Regions

- Placeholder (named location) for view containment
 - Views dynamically added by app/module code
- Prism 2 supports two approaches:
 - View Discovery
 - View Injection
- Can be defined by the shell or a composite view
 - Shell – almost always
 - Composite View – less often



Regions



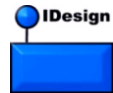


RI Regions

The screenshot shows the CFI Stock Trader application window. The interface is divided into several regions:

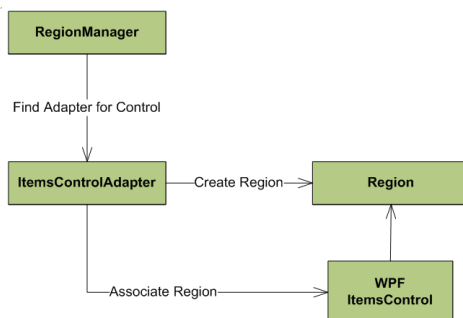
- Toolbar Region:** Located at the top of the application, containing buttons like 'Submit', 'Cancel', 'Submit All', 'Cancel All', and 'Add To Watchlist'.
- Watch Region:** A 'WATCH LIST' table on the right side of the application.
- Main Content Region:** The central area containing a 'POSITION' table, a 'PIE CHART', and 'HISTORICAL DATA' chart.
- News Region:** A 'NEWS ARTICLES' section at the bottom right of the main content area.

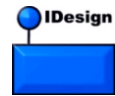
Symbol	Shares	Lot	Cost Basis	Market Value	Gain/Loss %
STOCK0	10	56.68	280.99	566.89	101.72
STOCK2	100	46.01	1900.22	4601.29	142.15
STOCK3	100	54.54	1900.22	5454.02	187.02
STOCK6	50	44.11	523.43	2205.35	321.33
STOCK7	25	241.15	6990.43	6028.69	-13.75



Region Manager

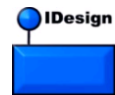
- CAL Service
- Registration point for named regions
- Modules get a region references from the region manager
- Use the region to add their views



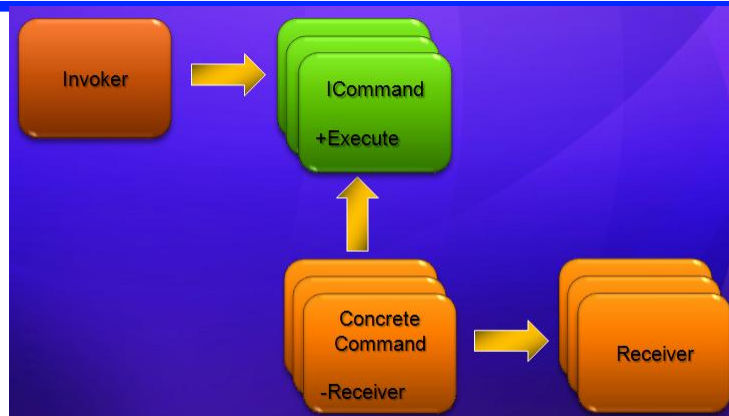


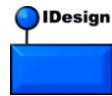
Agenda

- Composite Application Guidance Overview
- Application Architecture
- CAL Features
 - Modules
 - Regions
 - Commands
 - Events
- Patterns



Command Pattern

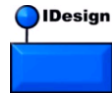




WPF Routed Command Limitations

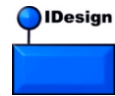
- Tied to the visual tree
 - Use routed events as the command dispatch mechanism
 - ▲ Bubbling and tunneling
 - Means no way to put the command logic outside the view without a handoff from the view
- Tied to the focused element
 - Can get in focus hell if handler is not up the element tree from the invoker
- Single handler invoked
 - First-in wins during tunneling and bubbling process
 - No multi-dispatch

Bottom Line: Insufficient for composites

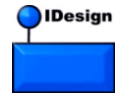
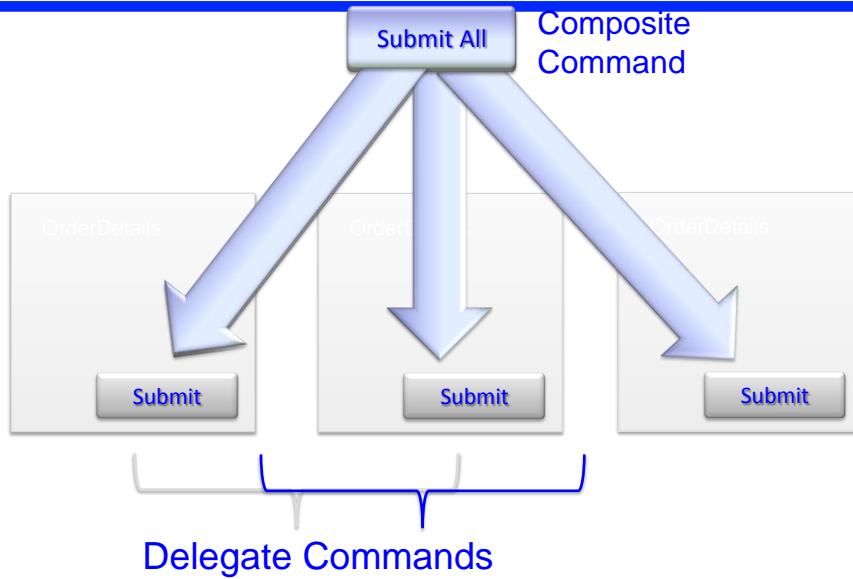


CAL Commands

- Based on WPF ICommand interface
- Gets the handling out of the visual tree
 - Handlers can be presenters or controllers
- Breaks the dependency on focus
- Allows multiple targets

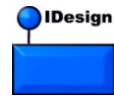


Composite Commands



Agenda

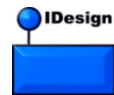
- Composite Application Guidance Overview
- Application Architecture
- CAL Features
 - Modules
 - Regions
 - Commands
 - Events
- Patterns



WPF Routed Events

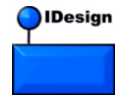
- New feature of WPF
- More than just .NET events
- Tied to the visual tree
 - Only allows elements within the visual tree to handle an event raised by another element in the visual tree
- Supported routing schemes:
 - Bubbling
 - Tunneling
 - Direct

Bottom Line: Insufficient for composites

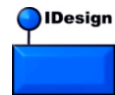
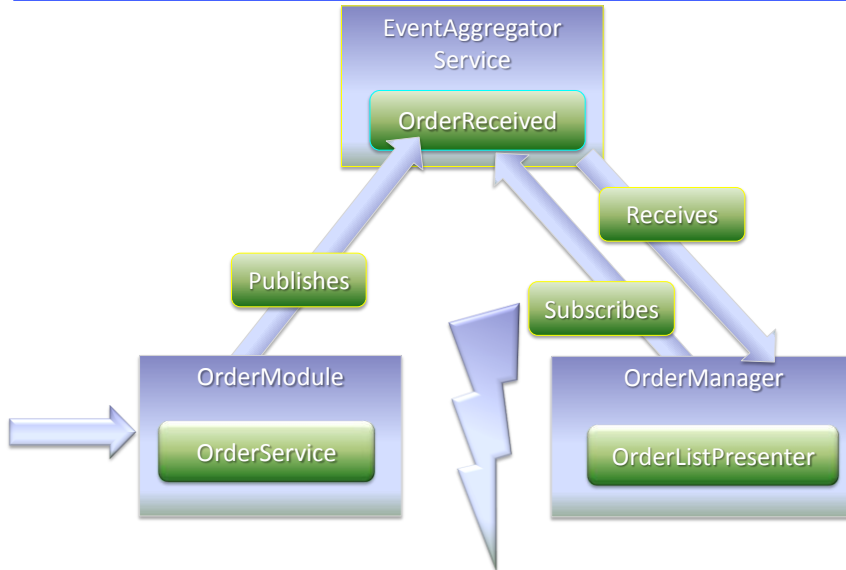


CAL Events

- Based on pub/sub pattern
- Uses an event aggregator service
- Get event from aggregator
- Subscribe or publish
- Decouples publishers and subscribers
 - Type
 - Lifetime
- Handles threading issues
- Provides filtering capability
- Allows communications between loosely coupled, non-visual parts
 - Presenters and controllers



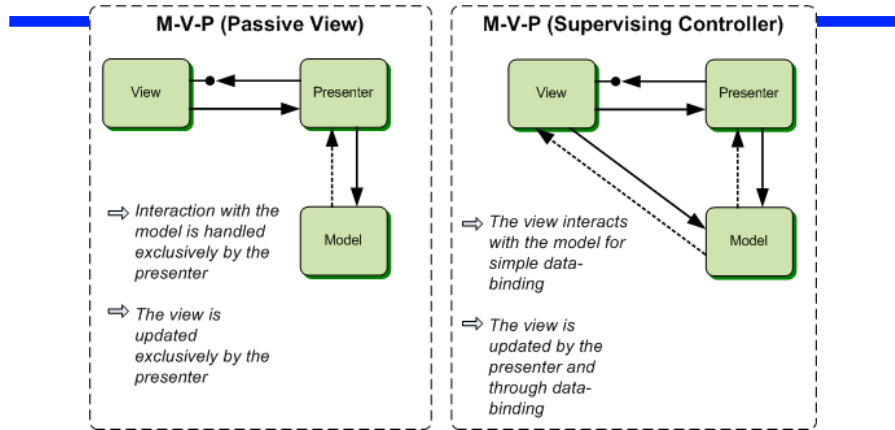
Event Aggregation



Agenda

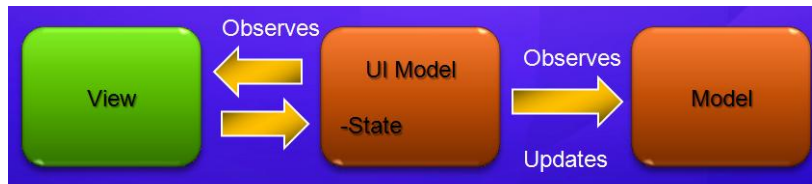
- Composite Application Guidance Overview
- Application Architecture
- CAL Features
 - Modules
 - Regions
 - Commands
 - Events
- **Patterns**

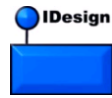
Model View Presenter Variants



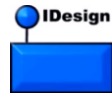
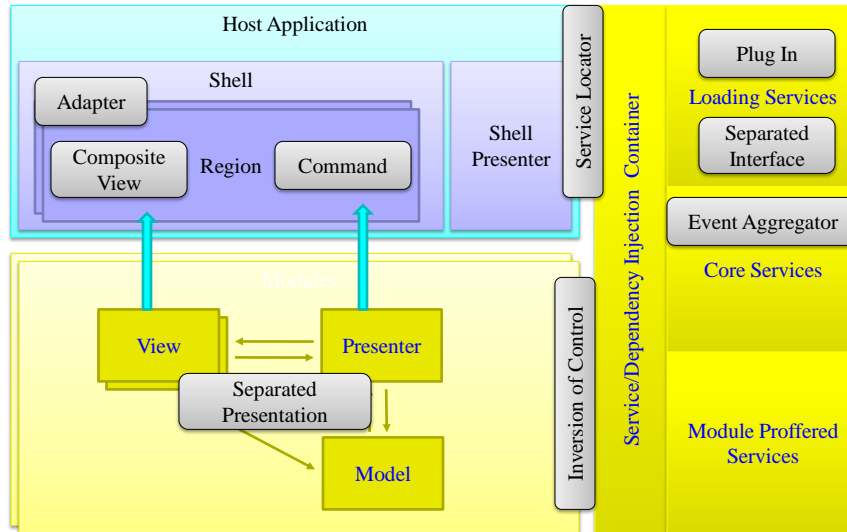
Presentation Model

- Also known as Model-View-ViewModel





Patterns in CAL



Resources

- Build Composite WPF Applications, Brian Noyes, CoDe Magazine, Nov/Dec 2008, <http://www.code-magazine.com>.
- Patterns for Building Composite Applications with WPF, Glenn Block, MSDN Magazine, Sep 2008, <http://msdn.microsoft.com/en-us/magazine/cc785479.aspx>.
- Understanding Routed Events and Commands, Brian Noyes, MSDN Magazine, September 2008, <http://msdn.microsoft.com/en-us/magazine/cc785480.aspx>.
- .NET Rocks! Interview: <http://www.dotnetrocks.com/default.aspx?showNum=374>
- Prism podcast, Brian Noyes: <http://pixel8.infragistics.com/shows/prism.aspx>.
- DNRTV: <http://www.dnrtv.com/default.aspx?showNum=124>, <http://www.dnrtv.com/default.aspx?showNum=132>