




Build Loosely Coupled Silverlight Business Applications

Brian Noyes
www.idesign.net

©2010 Brian Noyes, IDesign Inc. All rights reserved



About Brian

<p>Chief Architect IDesign Inc. (www.idesign.net)</p>	<p>Publishing</p> <p><i>Developing Applications with Windows Workflow Foundation, LiveLessons training DVD, June 2007</i></p> <p><i>Smart Client Deployment with ClickOnce, Addison Wesley, January 2007</i></p> <p><i>Data Binding in Windows Forms 2.0, Addison Wesley, January 2006</i></p> <p><i>MSDN Magazine, MSDN Online, CoDe Magazine, The Server Side .NET, asp.netPRO, Visual Studio Magazine</i></p>
<p>Microsoft Regional Director (www.theregion.com)</p>	<p>Speaking</p> <p><i>Microsoft TechEd US, Europe, Malaysia, Visual Studio Connections, DevTeach, INETA Speakers Bureau, MSDN Webcasts</i></p>
<p>Microsoft MVP Connected Systems</p>	

E-mail: brian.noyes@idesign.net
Blog: <http://briannoyes.net>
Twitter: @briannoyes

Agenda

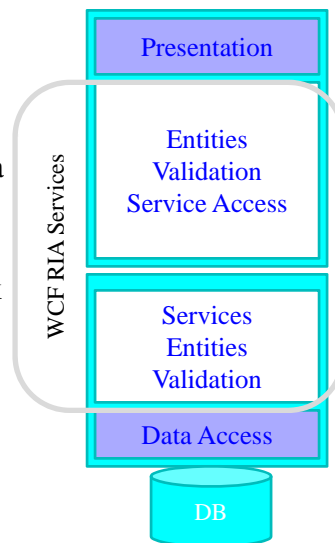


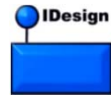
- Quick Intro to WCF RIA Services
- Quick Intro to MVVM
- Quick Intro to MEF
- Tying them all together

WCF RIA Services Overview



- Simplifies building N-tier applications
- Focused on Line of Business (LOB) applications
 - Highly dependent on pull-push of data from back end
- Architecture and tools for building the glue code between the client and the back end
 - Silverlight, ASP.NET; future: WPF, other clients
 - Entity Framework, LINQ to SQL; future: WCF Data Services (Astoria), SQL Azure





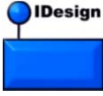
What do you need?

- Silverlight Tools for Visual Studio 2010
 - Core bits
- RIA Services Toolkit
 - Other endpoint types (SOAP, JSON)
 - ASP.NET support
 - Client tools for proxies



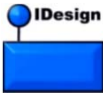
WCF RIA Services Overview

- Link client project to server project
- Define server side domain services
 - Base classes: DomainService, LinqToEntitiesDomainService, LinqToSqlDomainService
- Compile
- WCF RIA Services code generates client side “proxy” code
 - Domain context
 - Entities
 - Validation
 - Shared code



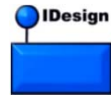
Application Architecture

- Web project
 - Silverlight host
 - Domain service host
 - Domain Services
 - ▲ EnableClientAccess attribute
- Client project (i.e. Silverlight)
 - Generated code folder
 - DomainContext classes
 - ▲ One per domain service
 - DomainDataSources
 - ▲ Facilitates simple data binding
- Supporting class libraries as needed



Query Operations

- Read operations
 - Return object, IEnumerable<T>, IQueryable<T>
 - Query attribute on methods
 - ▲ IsComposable=false when returning single entity
 - Naming convention without Query attribute
 - Use LoadOperation<T> on client side to load data
 - Load related types with a metadata class with [Include] on related members
- CUD Operations (CRUD – R)
 - Naming convention in domain service to identify methods
 - ▲ InsertXXX, UpdateXXX, DeleteXXX where XXX is entity type
 - DomainContext contains an EntityContainer
 - EntityContainer tracks changes on entities with EntityChangeSet



DomainDataSource

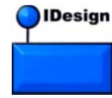
- Silverlight only, eventually WPF as well
- Similar to ObjectDataSource and XmlDataSource in WPF
- Allow you to execute a DomainContext method to query data
 - Including updates
- Drag/drop data binding is VS 2010
- Supports parameterized queries
- Supports sorting, grouping, filtering, paging
- Not a good fit with MVVM
 - Puts query “logic” into the view



Agenda

- Quick Intro to WCF RIA Services
- [Quick Intro to MVVM](#)
- Quick Intro to MEF
- Tying them all together

Why use separated UI patterns?



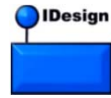
- Design with:
 - Loose coupling
 - Separation of concerns
- Benefits
 - Presentation logic code (ViewModel, Controllers, etc) becomes more testable
 - Designers and developers can work more independently in view/ViewModel respectively
 - More maintainable – easier to make changes to view or ViewModel without affecting each other

Presentation Model / ViewModel



- Model-View-ViewModel
- MVVM or ViewModel for short
- Specialized implementation of Presentation Model pattern for WPF and Silverlight





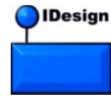
ViewModel

- Offers up state to the view through simple properties
 - Facilitates simple binding expressions in the view
- Notifies view of property changes
 - INotifyPropertyChanged / INotifyCollectionChanged
 - On ViewModel or its exposed property types
- Encapsulates interaction logic to support the view
 - Initial load of data
 - Command handlers
 - Service invocation
 - Validation logic
 - Value conversion



ViewModel in WPF/Silverlight

- View.DataContext = ViewModel
- POCO (Plain old CLR object)
 - No required base classes
 - However, often factor out common code into base class
- Public properties for view controls to bind to
 - Properties may expose model objects directly
 - Properties may be special data just for display purposes
 - Properties for commands
- Initialization / interaction logic
- “Married” to the view through construction code
 - Set as DataContext for the view
 - Several approaches



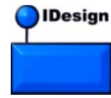
Agenda

- Quick Intro to WCF RIA Services
- Quick Intro to MVVM
- Quick Intro to MEF
- Tying them all together



MEF Overview

- Managed Extensibility Framework
- Composition mechanism for loosely coupled applications
- Mechanism for extensibility (plug ins)
- Dynamically discover and compose application out of parts
- No configuration required
- Available in .NET 4 Fx and Silverlight 4
 - `System.ComponentModel.Composition` library



MEF Imports and Exports

- In MEF, you build your application out of “Parts”
- Parts can be connected to other parts automatically
 - Wiring up of dependencies
- Primarily attribute based programming model
 - Import – indicate a dependency
 - Export – indicate an ability to satisfy a dependency
- CompositionContainer
 - Contains the parts (via contained Catalogs)
 - Performs composition (matching imports with exports)



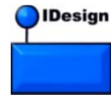
MEF Exports

- Put Export attribute on class


```
[Export]
public partial class MainWindow : Window
{ ... }
```
- Can also export an interface type


```
[Export(typeof(IMyView))]
public partial class MainWindow : Window, IMyView
{ ... }
```
- Can export a contract name when multiple matching types may exist
 - With or without a corresponding type
 - Example: multiple views that implement an IView marker interface

```
[Export("View2",typeof(IView))]
public partial class View2 : UserControl, IView
{ ... }
```



MEF Imports

- Add Import attribute on a property

```
[Import("View2",typeof(IView))]
public IView View2 { get; set; }
```

- Can indicate:
 - No argument – uses type of property
 - Just a type
 - Just a contract name
 - Both
- Import evaluated when:
 - Type is constructed by MEF
 - ComposeParts called on this



Setting Up Container

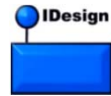
- Construct CompositionContainer
- Add catalogs
- ComposeParts on root object

```
public partial class App : Application
{
    private CompositionContainer _Container;

    private void Application_Startup(object sender, StartupEventArgs e)
    {
        var catalog = new AssemblyCatalog(typeof(App).Assembly);
        _Container = new CompositionContainer(catalog);
        _Container.ComposeParts(this);

        RootVisual = uiService.MainWindow as UserControl;

        ...
    }
    ...
}
```



Resources

- WCF RIA Services
 - WCF RIA Services (10 parts – in work):
<http://www.silverlightshow.net/items/WCF-RIA-Services-Part-1-Getting-Started.aspx>
- MVVM
 - WPF Apps with the Model-View-ViewModel Pattern, Josh Smith, MSDN Magazine, Feb 2009, <http://msdn.microsoft.com/en-us/magazine/dd419663.aspx>
 - Model-View-ViewModel in Silverlight 2 Apps, Shawn Wildermuth, MSDN Magazine, March 2009, <http://msdn.microsoft.com/en-us/magazine/dd458800.aspx>
- MEF
 - Building Composable Apps in .NET with WEF, Glenn Block, MSDN Magazine, Feb 2010, <http://msdn.microsoft.com/en-us/magazine/ee291628.aspx>