

Building Extensible Desktop Applications

Brian Noyes

Chief Architect, IDesign Inc (www.idesign.net)

brian.noyes@idesign.net, [@briannoyes](https://twitter.com/briannoyes)



About Brian

Chief Architect

IDesign Inc. (www.idesign.net)

Microsoft Regional Director

(www.theregion.com)

Microsoft MVP

Silverlight

E-mail: brian.noyes@idesign.net

Twitter: [@briannoyes](https://twitter.com/briannoyes)

Blog: <http://briannoyes.net>

Publishing

Developers Guide to Microsoft Prism 4, O'Reilly & Assoc., March 2011

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, DevConnections, DevTeach, others



Agenda

- Extensibility overview
- Quick Intro to MEF
- Quick Intro to MVVM
- Quick Intro to Prism
- Defining the Application “Core”
- Extending the Application



What Does It Mean To Be Extensible?

- Incrementally add new features to your application over time with minimal effort
- Minimize changes to existing code to add features
- Develop new features independently of one another and then main application changes



What Does It Mean To Be Extensible?

- Everything in the code behind – surgical changes for every addition
- Well factored code – everything in its place – hard coded dependencies and knowledge of concrete types
- Pluggable architecture dependent only on base types and interfaces – no changes to core code to add new functionality



Not Extensible

```

public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        NorthwindEntities context = new NorthwindEntities();
        var customers = (from c in context.Customers select c).ToList();
        DataContext = customers;
    }

    private void button1_Click(object sender, RoutedEventArgs e)
    {
        var selectedCustomer = customersDataGrid.SelectedItem as Customer;
        NorthwindEntities context = new NorthwindEntities();
        var orders = (from o in context.Orders.Include("Order_Details")
                     where o.CustomerID == selectedCustomer.CustomerID select o);
        decimal sum = 0;
        foreach (Order order in orders)
        {
            foreach (Order_Detail detail in order.Order_Details)
            {
                sum += detail.UnitPrice * detail.Quantity;
            }
        }
        MessageBox.Show("Sales for customer " + selectedCustomer.CompanyName + " = " + sum.ToString());
    }
}

```

Business logic

Data Access

UI manipulation

Is This Code Extensible?

```

public class MyDevice
{
    public DeviceOperation1 Operation1 { get; set; }
    public DeviceOperation2 Operation2 { get; set; }
    public DeviceOperation3 Operation3 { get; set; }
}

public class DeviceOperation1 { public void DoSomething() { } }
public class DeviceOperation2 { public void SomeOperation() { } }
public class DeviceOperation3 { public void OtherStuff() { } }

public class Consumer
{
    void ProcessOperations(string op)
    {
        MyDevice device = GetMyDevice();
        switch (op)
        {
            case "op1":
                device.Operation1.DoSomething();
                break;
            case "op2":
                device.Operation2.SomeOperation();
                break;
            case "op3":
                device.Operation3.OtherStuff();
                break;
        }
    }
    private MyDevice GetMyDevice() { return null; }
}

```

Better Extensibility

```

public class Device { }
public class Operation { public virtual void Execute() { } }

public class MyDevice : Device
{
    public List<Operation> Operations { get; set; }
}

public class DeviceOperation1 : Operation { }
public class DeviceOperation2 : Operation { }
public class DeviceOperation3 : Operation { }

public class Consumer
{
    void ProcessOperations(Type op)
    {
        MyDevice device = GetMyDevice();
        var opToExecute = device.Operations.Where(o => o.GetType() == op).FirstOrDefault();
        if (opToExecute != null) opToExecute.Execute();
    }
    private MyDevice GetMyDevice() { return null; }
}

```

To Be Extensible – You Need

- A set of abstractions that represent the domain and the views
 - Base classes and interface
 - Domain model objects, view models
- A base set of functionality (the “core”) that ties the app together
 - Model retrieval and update
 - Container views and view models
- A plug-in mechanism to add new functionality without disturbing the core
 - MEF, Prism, MVVM
- Decoupling between the parts
 - Physical factoring, minimizing shared dependencies



Extensibility Overview

- Key technologies:
 - MEF – Plug in pattern + IOC container
 - Can use any IOC container
 - MVVM – Separation of concerns and decoupling in the presentation layer
 - Also supports dynamic composition of child views
 - Prism – Toolkit for composing application from loosely coupled parts



Managed Extensibility Framework

- Part of .NET 4 / SL 4/5
- IOC container under the covers
- Define classes in your app as “parts”
- Declare Exports to say what parts are creatable
- Declare Imports to create a part
- Create the container that manages it all



MVVM

- Model-View-ViewModel
- Keeps different concerns of the presentation layer separated and decoupled from each other
- Well adopted in the industry
- Works well with composing parent and child views dynamically
- Heavily reliant on data binding



Prism

- Toolkit for composite apps
- Features
 - Modularity
 - UI Composition
 - Loosely coupled communications
 - Commands and events
 - Navigation
 - App structuring



Defining the App Core

- Base classes for model objects
- Parent views that children plug into
- Base view model classes
- Top level navigation logic to switch views
- Imports for the derived model/view model / view classes to bring them in dynamically



Extending Future Versions

- With each new release, you need to be able to add:
 - View + ViewModel + Model
 - With minimal effort



Extending the Model

- Derive types from base model objects
- Tie them in to the appropriate view model types
- Source objects for data binding in the views



Extending the View Models / Views

- Where the real UI functionality lives for the app
- Base view model types allow the core to use them without being coupled to them
- Views bind to the properties of the view model or the model objects exposed by them



Plugging in Whole New Functionality

- **Prism modules**
 - Allow you to dynamically add sets of functionality to the app with no code mods to the main app
 - If designed with this in mind originally
- **Prism Regions**
 - UI plug-in points to add new chunks of UI
 - Commands to access new functionality
 - Views to present new functionality



Resources

- MEF: <http://msdn.microsoft.com/en-us/magazine/ee291628.aspx>
- MVVM: <http://msdn.microsoft.com/en-us/magazine/dd419663.aspx>
- Prism: <http://compositewpf.codeplex.com/releases/view/55580>

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



Your Feedback is Important

Please fill out a session evaluation form
drop it off at the conference registration
desk.

Thank you!

