What's New in WPF 4.5

DEVELOPMENTOR

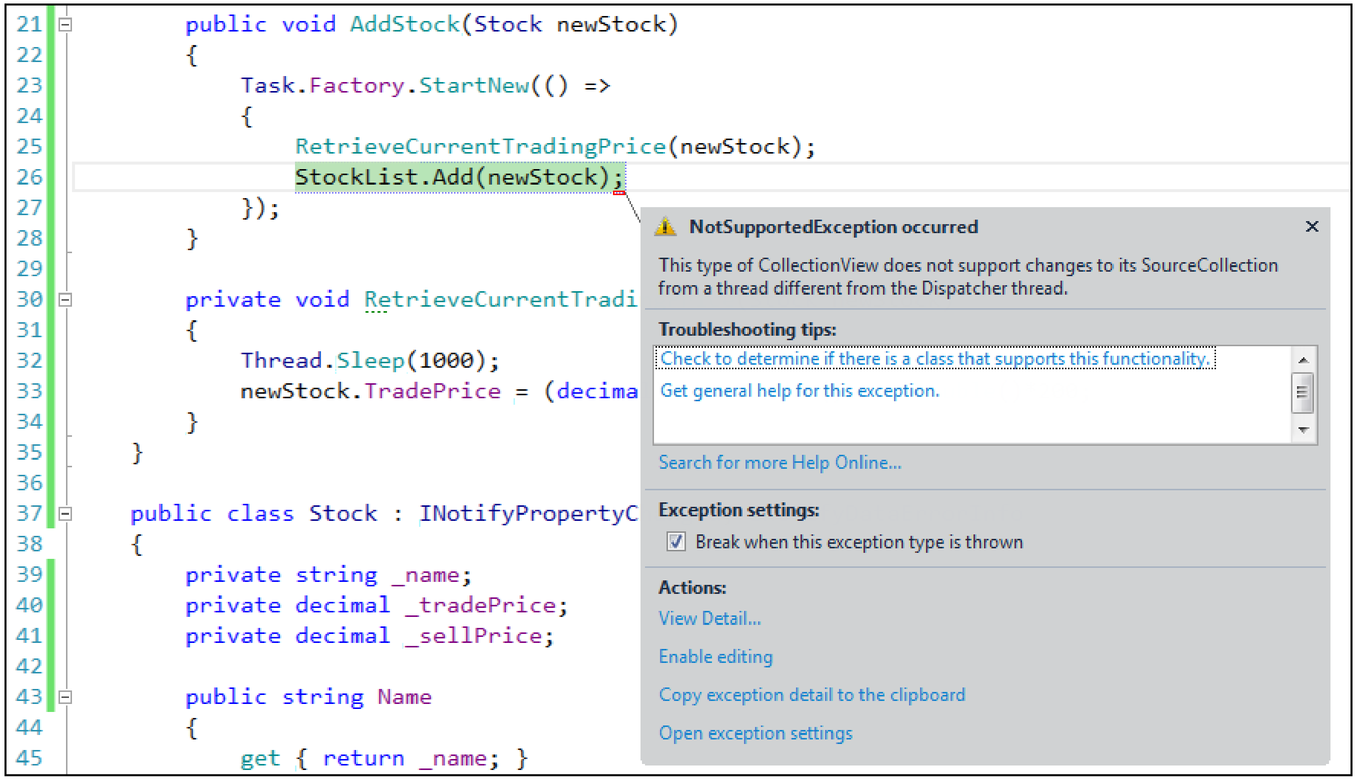
DEVELOPING PEOPLE WHO DEVELOP SOFTWARE

Agenda

- What's New?
  - several data binding improvements
  - UI Virtualization changes
  - new markup extension capabilities
  - weak events
  - language async support
  - new controls

Data Binding [performance]

- Data binding supports background thread INPC but not collection changes
  - must switch to UI thread before modifying collection
  - performance issue when collection is heavily modified



```
        public void AddStock(Stock newStock)
        {
            Task.Factory.StartNew(() =>
            {
                RetrieveCurrentTradingPrice(newStock);
                StockList.Add(newStock);
            });
        }

        private void RetrieveCurrentTradi
        {
            Thread.Sleep(1000);
            newStock.TradePrice = (decima
        }
    }

    public class Stock : INotifyPropertyC
    {
        private string _name;
        private decimal _tradePrice;
        private decimal _sellPrice;

        public string Name
        {
            get { return _name; }
```

NotSupportedException occurred

This type of CollectionView does not support changes to its SourceCollection from a thread different from the Dispatcher thread.

Troubleshooting tips:

Check to determine if there is a class that supports this functionality.

Get general help for this exception.

Search for more Help Online...

Exception settings:

☑ Break when this exception type is thrown

Actions:

View Detail...

Enable editing

Copy exception detail to the clipboard

Open exception settings

Data Binding [performance]

- **Collections may now be altered in background threads**
  - supports any collection, uses Monitor for synchronization

```
public class StockTrader {
   private object _lock = new object();
   public IList<Stock> StockList { get; private set; }

   public Company() {
      StockList = new ObservableCollection<Stock>();
      BindingOperations.EnableCollectionSynchronization(StockList, _lock);
   }

   void AddStock(Stock newStock) {
      Task.Run(() => {
         RetrieveCurrentTradingPrice(newStock);
         lock(_lock) { StockList.Add(newStock) };
       });
   }
}
```

Data Binding [property change notifications]

- **.NET 4.5 has new attribute to supply caller name to method**
  - can be used for INPC with no magic strings
  - better performance than LINQ-based expression tree

```
public class Stock : INotifyPropertyChanged
{
   private string _name;
   public string Name
   {
      get { return _name; }
      set { _name = value; RaisePropertyChanged(); }
   }

   public PropertyChangedEventHandler PropertyChanged = delegate {};
   private void RaisePropertyChanged([CallerMemberName] string prop = "")
   {
      PropertyChanged(this, new PropertyChangedEventArgs(prop));
   }
}
```

Data Binding [static property change notification]

- Binding to static properties
  - **Binding engine now recognizes static property changes**

```
public class StockManager
{
   public static event EventHandler<PropertyChangedEventArgs>
                                StaticPropertyChanged = delegate {};

  public static void RaiseStaticPropertyChanged(string property )
  {
     StaticPropertyChanged(null, new PropertyChangedEventArgs(propName));
  }

  private static int _traderCount = 10;
  public static int TraderCount
  {
     get { return _traderCount; }
     set { _count = value; RaiseStaticPropertyChanged("TraderCount"); }
  }
}
```

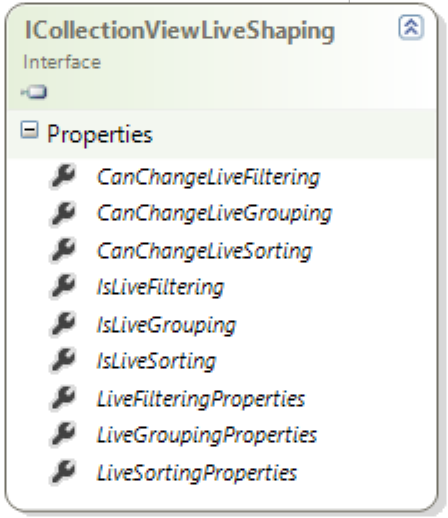
```
<TextBlock Text="{Binding Path=(me:StockManager.TraderCount)}" />
```

Data Binding [live data shaping]

- Live shaping
  - **Collection views support sorting, filtering and grouping**
    - ... but did not react to property changes
  - WPF 4.5 adds "live shaping" via **`ICollectionViewLiveShaping`**

```
var cv = CollectionViewSource.GetDefaultView(StockList);
var icvs = cv as ICollectionViewLiveShaping;
if (icvs != null)
{
  // Properties to perform live grouping on
  icvs.IsLiveGrouping = true;
  icvs.LiveGroupingProperties.Add("IsFalling");
  icvs.LiveGroupingProperties.Add("IsMajorMover");

  // Properties to perform live sorting on
  icvs.IsLiveSorting = true;
  icvs.LiveSortingProperties.Add("Delta");
  icvs.LiveSortingProperties.Add("AbsDelta");
  icvs.LiveSortingProperties.Add("Price");
}
```


ICollectionViewLiveShaping
Interface

Properties
- *CanChangeLiveFiltering*
- *CanChangeLiveGrouping*
- *CanChangeLiveSorting*
- *IsLiveFiltering*
- *IsLiveGrouping*
- *IsLiveSorting*
- *LiveFilteringProperties*
- *LiveGroupingProperties*
- *LiveSortingProperties*

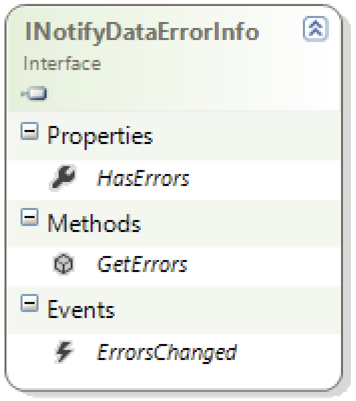
Data Binding [delay target to source transfer]

- **Update source transfer can now be delayed**
  - improves UI performance when source value generate intermediate values (sliders, selectors, etc.)
  - new `Delay` property specifies millisecond wait before source is transferred to target

```
<Slider Value="{Binding ZoomFactor, Delay=500}" />
   ...
<ListBox SelectedItem="{Binding SelectedStock, Delay=1000}" />
   ...
<TextBox Text="{Binding SellPrice, UpdateSourceTrigger=PropertyChanged,
                                   Delay=250}" />
```

Data Binding [asynchronous validations]

- **Binding validations now include asynchronous support**
  - built around new `INotifyDataErrorInfo`
  - identical to Silverlight support



implementation raises the `ErrorsChanged` event when new errors are identified, bindings associated to this source will then call `GetErrors` in response

```
<TextBox Text="{Binding SellPrice, ValidatesOnNotifyDataErrors=True}" />
```

Binding indicates participation through new flag

Data Binding [asynchronous validations]

- **Validations can now be done in background**

```
public partial class Stock
{
   private decimal _sellPrice;
   public decimal SellPrice
   {
      get { return _sellPrice; }
      set
      {
         _sellPrice = value;
         RaisePropertyChanged("SellPrice");
         Task.Run( () => ValidateSellPrice );
      }
   }
}
```

Data Binding [asynchronous validations]

```
public partial class Stock
{
   private readonly Dictionary<string,List<string>> _errors = ...;

   private void ValidateSellPrice() {
       ...
       if (invalidPrice)
          AddError("SellPrice", "Price not within approved range");
       else
          ClearErrors("SellPrice");
   }

   private void AddError(string property, string errMessage) {
      List<string> errLst = new List<string>();
      if (!_errors.TryGetValue(property, out errLst)
         _errors.Add(errLst);
      errLst.Add(errMessage);
      ErrorsChanged(this, new DataErrorsChangedEventArgs(property));
   }
}
```

Data Binding [asynchronous validations]

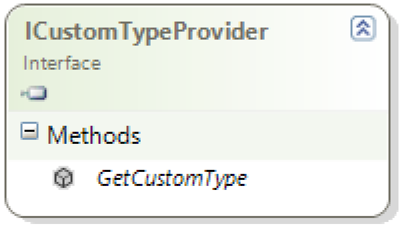
```
public partial class Stock : INotifyDataErrorInfo
{
   public event EventHandler<DataErrorsChangedEventArgs>
                                 ErrorsChanged = delegate { };

   public IEnumerable GetErrors(string propertyName)
   {
      List<string> errors = new List<string>();
      _errors.TryGetValue(propertyName, out errors);
      return errors;
   }

   public bool HasErrors { get { return _errors.Count > 0; } s}
}
```

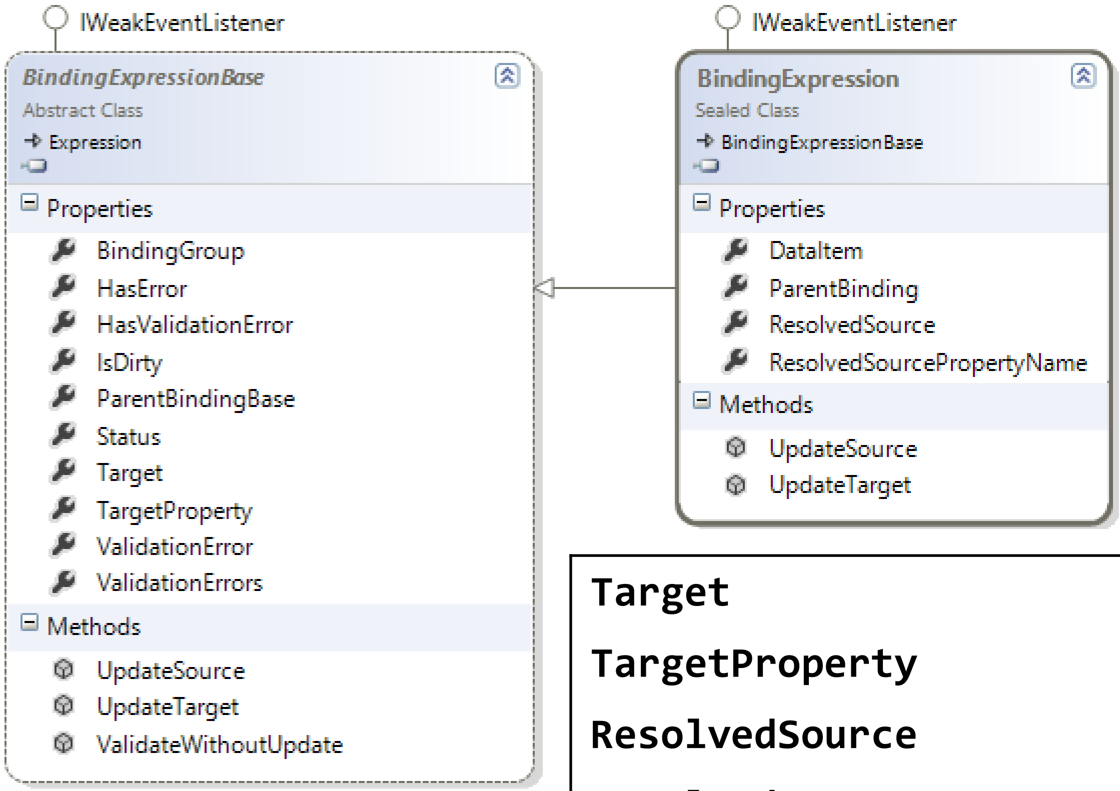
Data Binding [dynamic properties]

- New `ICustomTypeProvider` support enables dynamic binding
  - where object structure is not known until runtime (ex: JSON)
  - compatible with Silverlight 5 implementation
- Several steps necessary
  - must provide overridden **`Type`** definition
  - must provide storage for dynamic properties (**`PropertyInfo`**)
  - should implement **`INotifyPropertyChanged`**
- See blog post about using this approach with WPF
  - http://julmar.com/blog/mark/?p=201



Data Binding [extended binding information]

- `BindingExpression` extended to provide useful information



`Target`	target object of the binding
`TargetProperty`	target property of the binding
`ResolvedSource`	source object of the binding
`ResolvedSourceProperty`	source property of the binding
`BindingGroup`	group associated to the binding
`Owner`	owner of the binding group

Data Binding [extended binding information]

- New properties useful when evaluating specific cases
  - **BindingGroups**
  - behaviors
  - complex controls which need to control bindings (**DataGrid**)

```
void ValidateBindings(BindingGroup bindingGroup)
{
   foreach (BindingExpression be bindingGroup.BindingExpressions)
   {
      DependencyObject target = be.Target;
      DependencyProperty targetProperty = be.TargetProperty;
      object source = be.ResolvedSource;
      string sourceProperty = be.ResolvedSourcePropertyName;
      ...
   }
```

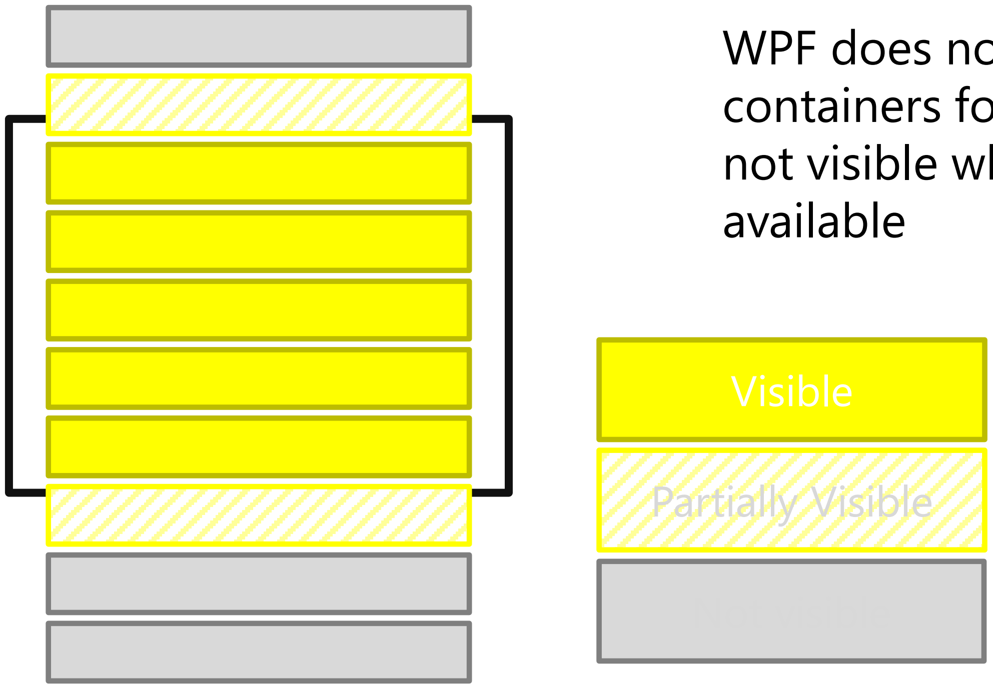
Data Binding [disconnected visuals]

- Accessing visual container DataContext sometimes failed
  - happened when container was disconnected from source
    - collection being changed while user interacts with UI
    - or in virtualization scenarios when container is thrown away
- WPF sets DataContext to a sentinel object in these cases
  - used to have to check for magic string "{DisconnectedItem}"
  - now exposed as **`BindingOperations.DisconnectedSource`**

```
private void OnTreeViewItemSelected(object sender, RoutedEventArgs e)
{
   // Get the prior selected item
   TreeViewItem item = (TreeViewItem) e.OriginalSource;
   if (item.DataContext == null ||
       item.DataContext == BindingOperations.DisconnectedSource)
      return;
    ...
}
```

UI Virtualization

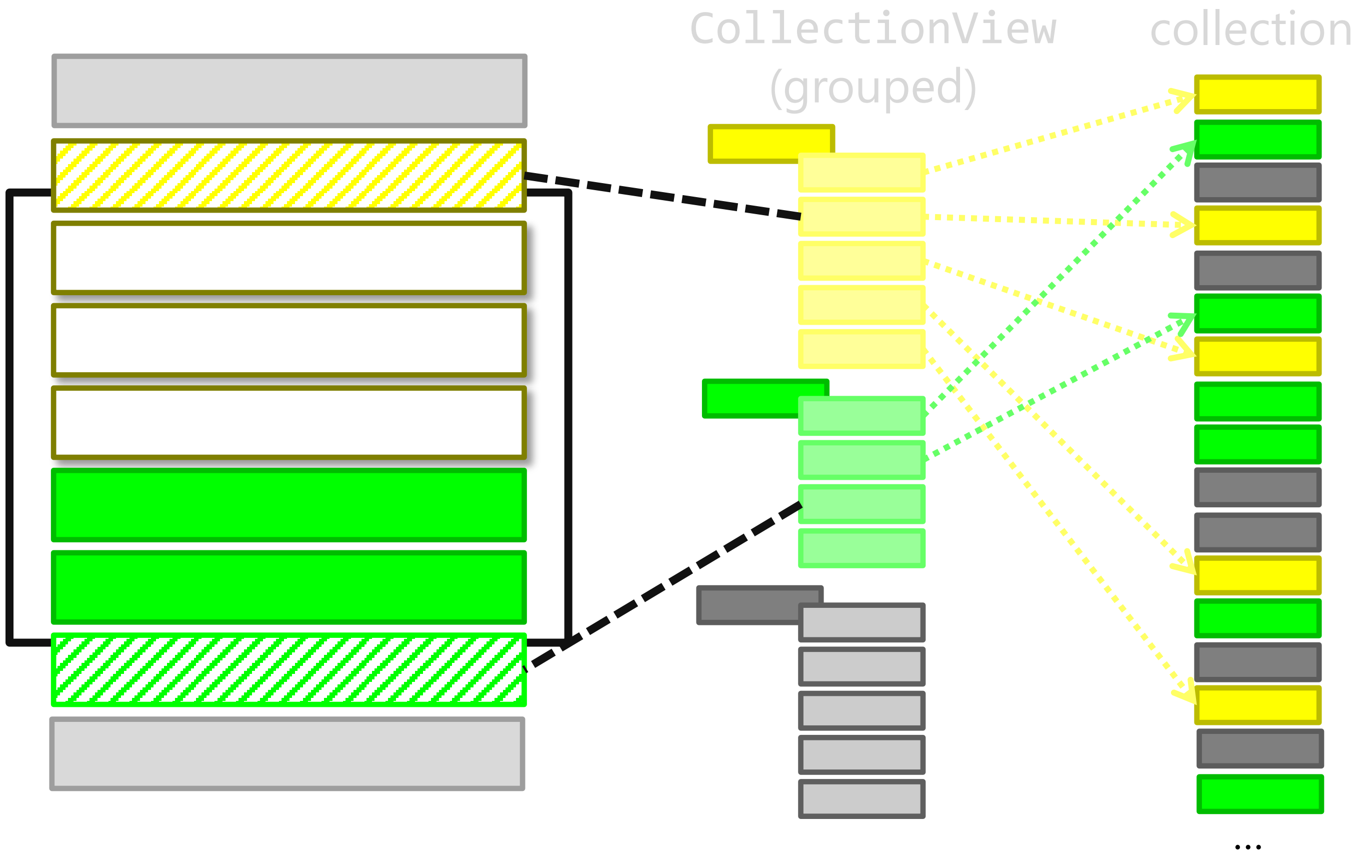
- **Key to large list performance in WPF is UI virtualization**
  - affects load time, scrolling and memory footprint
  - common scenarios often disabled virtualization[1]



WPF does not have to generate UI containers for elements that are not visible when UI virtualization is available

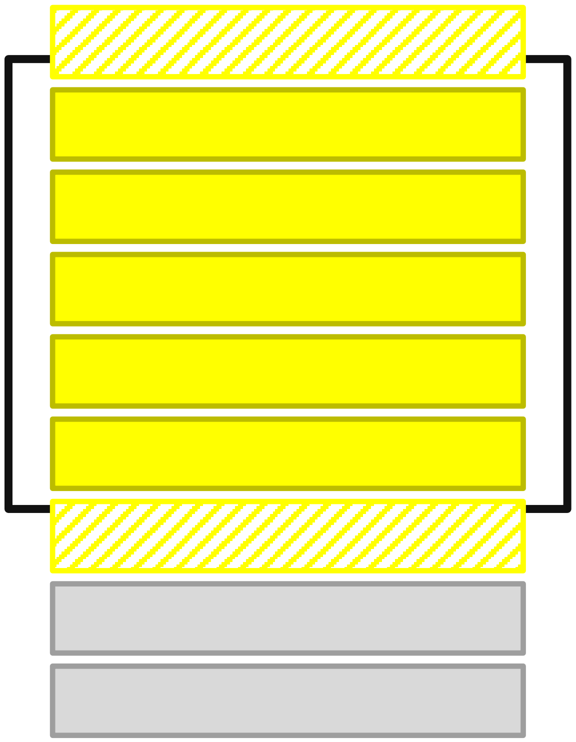
UI Virtualization [grouping]

```
<ListBox ItemsSource="{Binding GroupedStocks}"
         VirtualizingPanel.IsVirtualizingWhenGrouping="true" ... />
```



UI Virtualization [visual caching]

- **Scrolling performance suffers as WPF creates virtualized UI**
  - 4.5 caching feature solves that by pre-creating cached visuals in the background



```
<ListBox
   VirtualizingPanel.CacheUnit="Item"
   VirtualizingPanel.CacheLength="3" />
```

`CacheUnit` can be `Pixel`, `Item` or `Page`

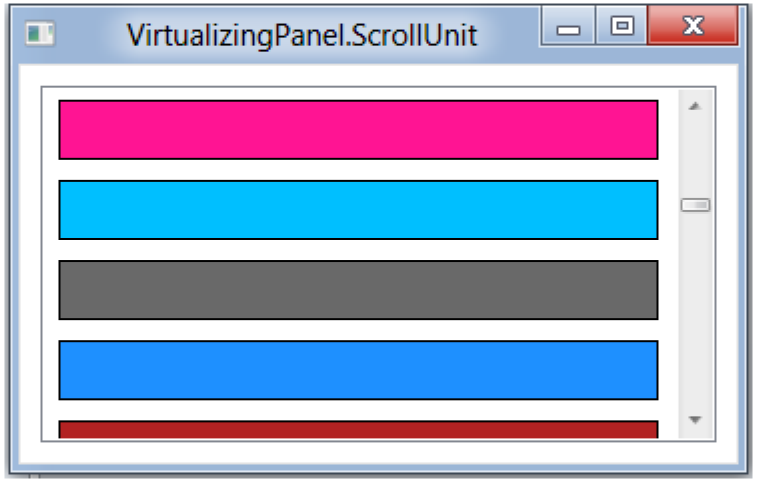
`CacheLength` is then defined in terms of the unit

UI Virtualization [scrolling]

- **Scroll units specified with `VirtualizingPanel.ScrollUnit`**
  - can scroll by unit or pixel

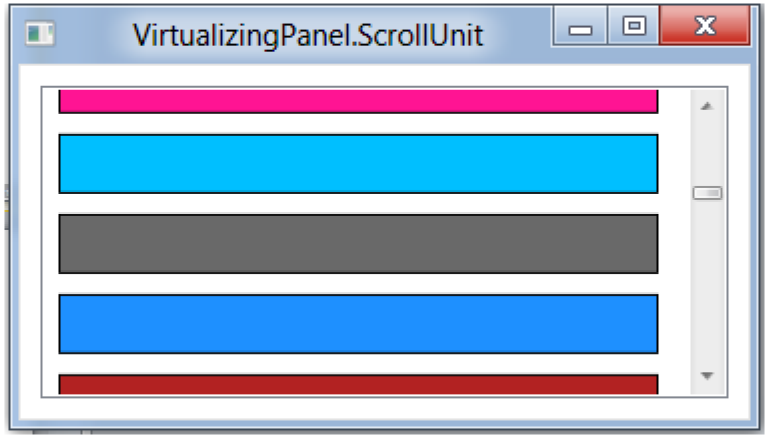
```
<ListBox ItemsSource="{Binding Colors}"
         VirtualizingPanel.ScrollUnit="Item" ... />
```

`ScrollUnit = "Item"`



top of the panel always shows a whole item

`ScrollUnit = "Pixel"`



allows display of a partial item at the top of the panel

Custom Markup Extensions [events]

- **Custom markup extensions can now be used as event targets**
  - allows event handler to be decided at runtime in markup
- **`IProvideValueTarget`** has been extended to support **`EventInfo`**
  - `TargetObject` is the source (sender) of the event
  - `TargetProperty` is the relevant `MethodInfo`
    - `MethodInfo` for "attached" events (e.g. Mouse Events)
    - `EventInfo` for "normal" events (e.g. Button Click)
    - Useful for yielding the type of the required delegate

```
<Button Content="OK" Padding="20,5" Margin="5"
         Click="{me:CallMethod Submit}" />
```

events can now use {extension} syntax to locate proper event handler – for example on the target object's `DataContext` (ViewModel)

Language async support

- **C# and VB.NET now support async / await**
  - makes it much easier to support asynchronous code
- **`Dispatcher`** also gains some new features
  - awaitable `[Begin]InvokeAsync` methods

```
async void UpdateUI() {
  var result = await Dispatcher.InvokeAsync<string>( () => ... );
  // do something with result
}
```

  - `CancelationToken` support

```
void UpdateUI(CancellationToken token, string message) {
  Dispatcher.Invoke(() => txtResult.Text = "Hello",
                    DispatcherPriority.Background, token);
}
```

  - Invoke methods to support Func<T>

```
var text = Dispatcher.Invoke<string>( () =>
                            DateTime.Now.ToLongTimeString() );
```

Weak events

- WPF has always has support for "weak" events
  - `WeakEventManager` and `IWeakEventListener`
  - not easy to work with
- New **`WeakEventManager<TSource,TEventArgs>`** makes it easier
  - subscriber does not need to implement `IWeakEventListener`

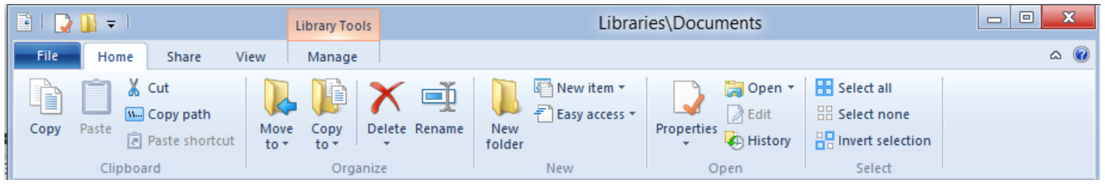
```
public void SubscribeToLongLivedEvent(MainWindow someWindow)
{
   // might leak unless you un-subscribed
   // someWindow.Activated += ProcessOnWindowActivated;

   WeakEventManager<MainWindow,RoutedEventArgs>
      .AddHandler(someWindow, "Activated", this.ProcessWindowActivated);
   // use .RemoveHandler() to unsubscribe
}
```

Warning: subscription will not keep delegate target alive!

New Controls [Ribbon]

- Office Ribbon control is becoming standard business UI
  - now even appearing in Windows 8 apps [explorer]
- WPF now includes **Ribbon** and **RibbonWindow** controls
  - see http://bit.ly/xoc4JK for details
  - also available for WPF 4.0 as add-on
  - keep in mind it has specific terms and conditions of use



```
<RibbonWindow x:Class="Office2013.MainWindow" ...>
  <DockPanel>
     <Ribbon DockPanel.Dock="Top"> ... </Ribbon>
  </DockPanel>
</RibbonWindow>
```

Summary

- WPF 4.5 was primarily about performance and polish
  - now fully supports async for full data manipulation / validation
  - much better UI virtualization capabilities
  - **`CollectionView`** support for live data group/sort/filtering
- Several features were added to increase compatibility with SL
  - **`INotifyDataErrorInfo`**
  - **`ICustomTypeProvider`**
- Some features to improve tooling capabilities
  - custom markup extension support for events
  - [**`CallerMemberName`**]
- Not a revolutionary release, but important for business apps