Formatting Bound Data

Agenda

- StringFormat
- Data Converters
- Data Templates
  - DataTriggers

FORMATTING BOUND DATA WITH STRINGFORMAT

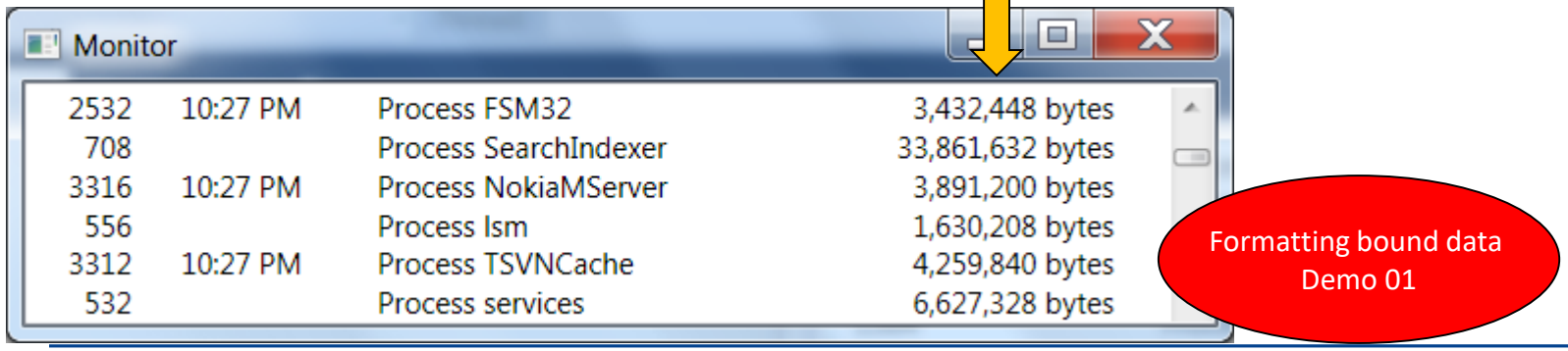
Use of StringFormat

- When you have some data, you can make it look much better by adding some simple formatting

```
<TextBlock Text="{Binding Path=WorkingSet,
                          StringFormat=N0}"
    />
```

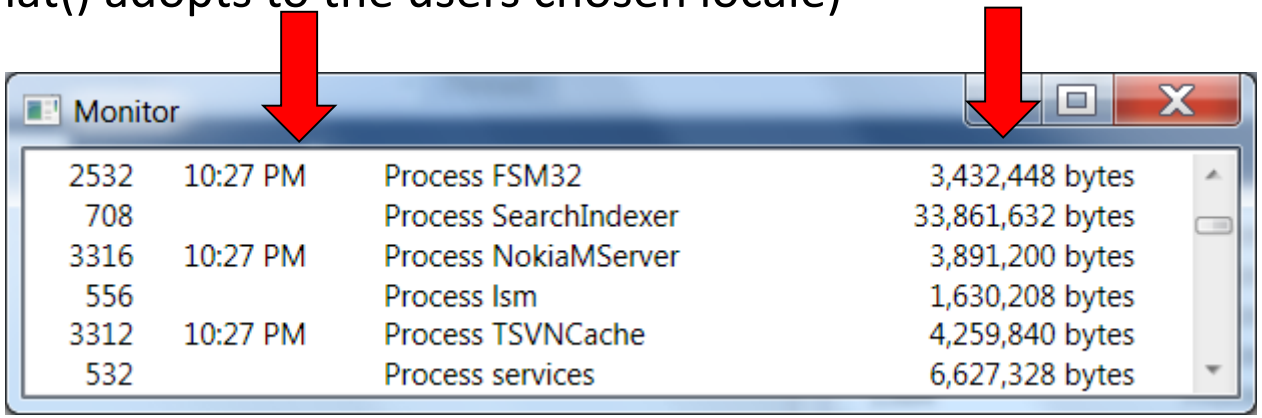
- Or if you want to add some text:

```
<TextBlock Text="{Binding Path=WorkingSet,
                          StringFormat=\{0:N0\} bytes}"
    />
```



StringFormat Limitations

- XAML's StringFormat has a very serious limitation:
  - It assumes use of US culture!
  - (C#'s string.Format() adopts to the users chosen locale)



- To avoid this you must either
  - implement your own converter in C#, or
  - **you can add some code to your App class that fixes the problem:**

```
protected override void OnStartup(StartupEventArgs e)  {
  Thread.CurrentThread.CurrentUICulture = Thread.CurrentThread.CurrentCulture;
  FrameworkElement.LanguageProperty.OverrideMetadata(typeof(FrameworkElement),
                   new FrameworkPropertyMetadata(XmlLanguage.GetLanguage(CultureInfo.CurrentCulture.Name)));
  base.OnStartup(e);
}
```

DATA CONVERTERS

Data Converters

- A data converter is a chunk of code that converts one value into another
  - E.g. we can take a number like 3723264 and convert it to 3 MiB,
  - or we could take the same number and, via some algorithm, convert it to a color
- Data converters add a huge amount of power to what you can do with XAML
- To create a data converter, you create a class that implements the `IValueConverter` interface

- Advice:
  - Don’t reinvent the wheel …
  - You can find many converters on the Internet
  - E.g. https://github.com/kentcb/WPFConverters

Data Converter Example

```
class NumberToFormattedTextValueConverter : IValueConverter
{
  public object Convert(object value, Type targetType,
                        object parameter,
                        System.Globalization.CultureInfo culture)
  {
    Int64 size = System.Convert.ToInt64(value);
    size = size / 1024;
    if (size < 1024)
      return size.ToString() + " KiB";
    else
      return (size / 1024).ToString() + " MiB";
  }

  public object ConvertBack(object value, Type targetType,
                            object parameter,
                            System.Globalization.CultureInfo culture)
  {
    throw new NotImplementedException();
  }
}
```

```
class NumberToFormattedTextValueConverter : IValueConverter
{
  public object Convert(object value, Type targetType,
                        object parameter,
                        System.Globalization.CultureInfo culture)
  {
    Int64 size = System.Convert.ToInt64(value);
    string units = (parameter != null) ? parameter.ToString() : "IEC";

    switch (units)
    {
      case "IEC":
           size = size / 1024;
           if (size < 1024)
             return size.ToString() + " KiB";
           else
             return (size / 1024).ToString() + " MiB";
      case "BINARYSI":
             size = size / 1024;

                   if (size < 1024)
                       return size.ToString() + " KB";
                   else
                       return (size / 1024).ToString() + " MB";
      case "SI":
           "
```

Formatting bound data
Demo 02

DATA TEMPLATES

Control ContentPresenter Algorithm

1. If Content is of type UIElement, then add it to the display tree
2. **If ContentTemplate/ItemTemplate is set, use that to create a UIElement instance and add it to the display tree**
3. If ContentTemplateSelector is set, use that to find a template, use the template to create a UIElement instance, and add it to the display tree
4. If the data type of Content has a data template associated with it, use that to create a UIElement instance
5. If the data type of Content has a TypeConverter instance associated with it that can convert to type UIElement, convert Content and add it to the display tree
6. If the data type of Content has a TypeConverter instance associated with it that can convert to a string, wrap Content in TextBlock and add it to the display tree
7. Finally, call ToString on Content, wrap it in TextBlock, and add it to the display tree

Data Templates

- A DataTemplate is a class in the WPF framework that we use to specify the visualization of some data objects
  - a data template is a tree of elements to expand in a particular context
- **We use data templates to provide an application with the capability to render nonvisual objects**
- DataTemplate objects are particularly useful when you are binding an ItemsControl such as a ListBox to an entire collection

```
<ListBox.ItemTemplate>
  <DataTemplate>
    <TextBlock Text="{Binding Path=ProcessName}"/>
  </DataTemplate>
</ListBox.ItemTemplate>
```

The binding mechanism assumes that we want to bind to whatever object we have available - in this case, the Process object in the current row of the ListBox

Data Template Example

- You will typically use a Panel of some kind as the top level element on a data template

```
<ListView.ItemTemplate>
  <DataTemplate>
    <WrapPanel>
      <TextBlock Text="{Binding Path=Id}" MinWidth="80" />
      <TextBlock Text="{Binding Path=ProcessName}"
MinWidth="180" />
      <TextBlock>
        <TextBlock.Text>
          <Binding Path="WorkingSet" />
        </TextBlock.Text>
      </TextBlock>
    </WrapPanel>
  </DataTemplate>
</ListView.ItemTemplate>
```

Control ContentPresenter Algorithm

1. If Content is of type UIElement, then add it to the display tree
2. If ContentTemplate is set, use that to create a UIElement instance and add it to the display tree
3. If ContentTemplateSelector is set, use that to find a template, use the template to create a UIElement instance, and add it to the display tree
4. **If the data type of Content has a data template associated with it, use that to create a UIElement instance**
5. If the data type of Content has a TypeConverter instance associated with it that can convert to type UIElement, convert Content and add it to the display tree
6. If the data type of Content has a TypeConverter instance associated with it that can convert to a string, wrap Content in TextBlock and add it to the display tree
7. Finally, call ToString on Content, wrap it in TextBlock, and add it to the display tree

TEMPLATES BASED ON TYPE

- If you have a list of different types of objects you can specify a DataType for your DataTemplates

  <DataTemplate DataType="{x:Type io:Directory}">

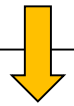
- When a DataTemplate is needed for the specified type, the template targeted at that type will automatically be picked up

```
<Window xmlns:local="clr-namespace:TicTacToe">
  <Window.Resources>
    <Style TargetType="{x:Type Button}">
      <Setter Property="HorizontalContentAlignment" Value="Stretch" />
      <Setter Property="VerticalContentAlignment" Value="Stretch" />
    </Style>
    <DataTemplate DataType="{x:Type io:Directory}">
       ...
    </DataTemplate>
    <DataTemplate DataType="{x:Type io:File}">
       ...
    </DataTemplate>
```

DATATRIGGERS

DataTriggers

- DateTemplates have a DataTriggers property that can be used to set one ore more data triggers
- A data trigger is based on a data value of some kind
  - E.g. if a particular threshold is passed, then your text turns red



```
<DataTemplate.Triggers>
    <DataTrigger Binding="{Binding Path=PriorityClass}" Value="High">
        <Setter TargetName="wrapPanel1" Property="Background">
            <Setter.Value>
                <LinearGradientBrush>
                    <GradientStop Color="Salmon" Offset="0" />
                    <GradientStop Color="Salmon" Offset="0.4" />
                    <GradientStop Color="White" Offset="1" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </DataTrigger>
</DataTemplate.Triggers>
```

Smart use of DataTrigger

- The fact that you can only do a single comparison for a trigger may seem like a serious limitation
  - but it can be over come by use of binding and a converter
- E.g. if we want to highlight rows that have a memory size greater than a certain size, we can create an IsLargeValueConverter that checks for a particular value and returns true if the size is larger

```
<DataTrigger Binding="{Binding Path=WorkingSet64,
             Converter={StaticResource isLarge},
             ConverterParameter=40000000}"
             Value="true" >
   <Setter TargetName="wrapPanel1" Property="Background">
      <Setter.Value>
         <SolidColorBrush Color="RosyBrown" />
      </Setter.Value>
   </Setter>
</DataTrigger>
```

Demo 03

```
public class IsLargeValueConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
                          System.Globalization.CultureInfo culture)
    {
            Int64 convertedValue = System.Convert.ToInt64(value);
            Int64 threshold = System.Convert.ToInt64(parameter);
            if (convertedValue > threshold)
                return true;
            return false;
        }
```