Implement INotifyPropertyChanged to notify the binding targets when the values of properties change.

Bind to an Existing Object Instance

Digital Clock

Add ApplicationCommands.Cut to TextBox with TextBox.CommandBindings

---

Execute a Method Asynchronously Using a Background Worker Thread

Start

```
//File:Window.xaml.cs
using System;
using System.Windows;
using System.ComponentModel;

namespace WpfApplication1
{
public partial class Window1 : Window
{
private BackgroundWorker worker = new BackgroundWorker();

private long from= 1;
private long to = 200;
private long biggestPrime;

public Window1(): base()
{
InitializeComponent();
worker.DoWork += new DoWorkEventHandler(worker_DoWork);
worker.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(worker_RunWorkerCompleted);
}

private void Start_Click(object sender, RoutedEventArgs e)
{
worker.RunWorkerAsync();
btnStart.IsEnabled = false;
txtBiggestPrime.Text = string.Empty;
}

private void worker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
{
btnStart.IsEnabled = true;
txtBiggestPrime.Text = biggestPrime.ToString();
}

private void worker_DoWork(object sender, DoWorkEventArgs e)
{
for(long current = from; current <= to; current++) {
biggestPrime = current; } } } } [/csharp]
```

Track the Progress of a

Background Worker Thread

Start

```
//File:Window.xaml.cs
using System;
using System.ComponentModel;
using System.Windows;

namespace WpfApplication1
{
public partial class Window1 : Window
{
private BackgroundWorker worker = new BackgroundWorker();

private long from = 2;
private long to = 20000;
private long biggestPrime;

public Window1() : base()
{
InitializeComponent();
worker.WorkerReportsProgress = true;

worker.DoWork += new DoWorkEventHandler(worker_DoWork);
worker.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(worker_RunWorkerCompleted);
worker.ProgressChanged += worker_ProgressChanged;
}

private void StartStop_Click(object sender, RoutedEventArgs e)
{
worker.RunWorkerAsync();
btnStartStop.IsEnabled = false;
txtBiggestPrime.Text = string.Empty;
}

private void worker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
{
btnStartStop.IsEnabled = true;
txtBiggestPrime.Text = biggestPrime.ToString();
}

private void worker_DoWork(object sender, DoWorkEventArgs e)
{
for(long current = from; current <= to; current++){
biggestPrime = current; int percentComplete =
Convert.ToInt32(((double) current / to) * 100d);
worker.ReportProgress(percentComplete);
System.Threading.Thread.Sleep(10); } } private void
worker_ProgressChanged(object sender, ProgressChangedEventArgs
e) { txtPercent.Text = e.ProgressPercentage.ToString() + "%";
} } } [/csharp]
```

Support the Cancellation of a Background Worker Thread

Start

```
//File:Window.xaml.cs

using System;
using System.ComponentModel;
using System.Windows;

namespace WpfApplication1
{
public partial class Window1 : Window
{
private BackgroundWorker worker = new BackgroundWorker();

private long from = 2;
private long to = 2000;
private long biggestPrime;

public Window1(): base()
{
InitializeComponent();
worker.WorkerSupportsCancellation = true;
worker.DoWork += new DoWorkEventHandler(worker_DoWork);
worker.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(worker_RunWorkerCompleted);
}
private void StartStop_Click(object sender, RoutedEventArgs e)
{
if(!worker.IsBusy)
{
worker.RunWorkerAsync();
btnStartStop.Content = “Cancel”;
txtBiggestPrime.Text = string.Empty;
}
else
{
worker.CancelAsync();
}
}
private void worker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
{
if(e.Cancelled)
{
MessageBox.Show("Operation was canceled");
}

btnStartStop.Content = "Start";
txtBiggestPrime.Text = biggestPrime.ToString();
}

private void worker_DoWork(object sender, DoWorkEventArgs e)
{
for(long current = from; current <= to; current++) {
if(worker.CancellationPending) { e.Cancel = true; return; }
biggestPrime = current; } } } } [/csharp]
```