Create a Background Worker Thread in XAML

Start

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

using System;
using System.ComponentModel;
using System.Threading;
using System.Windows;
using System.Windows.Input;

namespace WpfApplication1
{
public partial class Window1 : Window
{
private readonly BackgroundWorker worker;

public Window1()
{
InitializeComponent();
worker = this.FindResource("backgroundWorker") as
BackgroundWorker;
}

private void button_Click(object sender, RoutedEventArgs e)
{
```

```
if(!worker.IsBusy)
{
this.Cursor = Cursors.Wait;
worker.RunWorkerAsync();
button.Content = “Cancel”;
}else{
worker.CancelAsync();
}
}
```

```
private void BackgroundWorker_DoWork(object sender, System.ComponentModel.DoWorkEventArgs e)
{
for(int i = 1; i <= 100; i++) { if(worker.CancellationPending) break; Thread.Sleep(100); worker.ReportProgress(i); } } private void BackgroundWorker_RunWorkerCompleted(object sender, System.ComponentModel.RunWorkerCompletedEventArgs e) { this.Cursor = Cursors.Arrow; Console.WriteLine(e.Error.Message); button.Content = "Start"; } private void BackgroundWorker_ProgressChanged(object sender, System.ComponentModel.ProgressChangedEventArgs e) { progressBar.Value = e.ProgressPercentage; } } } [/csharp]
```

Show a ProgressBar While Processing on a Background Thread

Start

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

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

public Window1()
{
InitializeComponent();
worker.WorkerReportsProgress = true;
worker.DoWork += new DoWorkEventHandler(worker_DoWork);
worker.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(worker_RunWorkerCompleted);
worker.ProgressChanged += worker_ProgressChanged;
}

private void button_Click(object sender, RoutedEventArgs e)
{
worker.RunWorkerAsync();
this.Cursor = Cursors.Wait;
button.IsEnabled = false;
}

private void worker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
{
this.Cursor = Cursors.Arrow;
if(e.Error != null)
MessageBox.Show(e.Error.Message);
```

```
button.IsEnabled = true;
}
private void worker_DoWork(object sender, DoWorkEventArgs e)
{
for(int i = 1; i <= 100; i++) { Thread.Sleep(100);
worker.ReportProgress(i);    }    }    private    void
worker_ProgressChanged(object sender, ProgressChangedEventArgs
e) { progressBar.Value = e.ProgressPercentage; } } } [/csharp]
```

Use BackgroundWorker to run task at background

Start

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

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

public Window1()
```

```
{
InitializeComponent();
worker.WorkerReportsProgress = true;
worker.DoWork += new DoWorkEventHandler(worker_DoWork);
worker.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(worker_RunWorkerCompleted);
worker.ProgressChanged += worker_ProgressChanged;
}

private void button_Click(object sender, RoutedEventArgs e)
{
worker.RunWorkerAsync();
this.Cursor = Cursors.Wait;
button.IsEnabled = false;
}

private void worker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
{
this.Cursor = Cursors.Arrow;
if(e.Error != null)
MessageBox.Show(e.Error.Message);
button.IsEnabled = true;
}
private void worker_DoWork(object sender, DoWorkEventArgs e)
{
for(int i = 1; i <= 100; i++) { Thread.Sleep(100);
worker.ReportProgress(i); } } private void
worker_ProgressChanged(object sender, ProgressChangedEventArgs
e) { progressBar.Value = e.ProgressPercentage; } } } [/csharp]
```

A Cancellable ProgressBar While Processing on a Background Thread

Start

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

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

public Window1()
{
InitializeComponent();
worker.WorkerReportsProgress = true;
worker.WorkerSupportsCancellation = true;
worker.DoWork += new DoWorkEventHandler(worker_DoWork);
worker.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(worker_RunWorkerCompleted);
worker.ProgressChanged += worker_ProgressChanged;
```

```
}

private void button_Click(object sender, RoutedEventArgs e){
if(!worker.IsBusy)
{
this.Cursor = Cursors.Wait;
worker.RunWorkerAsync();
button.Content = “Cancel”;
}else{
worker.CancelAsync();
}
}

private void worker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
{
this.Cursor = Cursors.Arrow;
if(e.Cancelled)
{
MessageBox.Show(“Operation was canceled”);
}
else if(e.Error != null)
{
MessageBox.Show(e.Error.Message);
}

button.Content = “Start”;
}

private void worker_DoWork(object sender, DoWorkEventArgs e)
{
for(int i = 1; i <= 100; i++) { if(worker.CancellationPending)
{ e.Cancel = true; return; } Thread.Sleep(100);
worker.ReportProgress(i); } } private void
worker_ProgressChanged(object sender, ProgressChangedEventArgs
e) { progressBar.Value = e.ProgressPercentage; } } } [/csharp]
```

Show a Continuous Progress Bar While Processing on a Background Thread

Start

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

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

public Window1()
{
InitializeComponent();
worker.WorkerSupportsCancellation = true;
worker.DoWork += new DoWorkEventHandler(worker_DoWork);
worker.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(worker_RunWorkerCompleted);
}
```

```
private void button_Click(object sender, RoutedEventArgs e)
{
if (!worker.IsBusy)
{
this.Cursor = Cursors.Wait;
progressBar.IsIndeterminate = true;
button.Content = “Cancel”;
worker.RunWorkerAsync();
}
else
{
worker.CancelAsync();
}
}

private void worker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
this.Cursor = Cursors.Arrow;

if (e.Error != null)
{
MessageBox.Show(e.Error.Message);
}

button.Content = “Start”;
progressBar.IsIndeterminate = false;
}

private void worker_DoWork(object sender, DoWorkEventArgs e){
for (int i = 1; i <= 100; i++) { if (worker.CancellationPending) break; Thread.Sleep(50); } } } }
[/csharp]
```

Using a BackgroundWorker: progress changed and completed

```
//File:Window.xaml.cs
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.ComponentModel;
using System.Threading;

namespace BackgroundWorkerExample
{
partial class MyWindow : Window
{
BackgroundWorker bw = new BackgroundWorker();

public MyWindow()
{
bw.DoWork += new DoWorkEventHandler(bw_DoWork);
bw.ProgressChanged += bw_ProgressChanged;
bw.RunWorkerCompleted += bw_RunWorkerCompleted;
```

```
bw.WorkerReportsProgress = true;
bw.RunWorkerAsync();
}

void bw_DoWork(object sender, DoWorkEventArgs e)
{
for (int i = 0; i < 10; ++i) { int percent = i * 10; bw.ReportProgress(percent); Thread.Sleep(1000); } } void bw_ProgressChanged(object sender, ProgressChangedEventArgs e) { this.Title = "Working: " + e.ProgressPercentage + "%"; } void bw_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e) { this.Title = "Finished"; } } } [/csharp]
```

Unblock Thread with BackgroundWorker