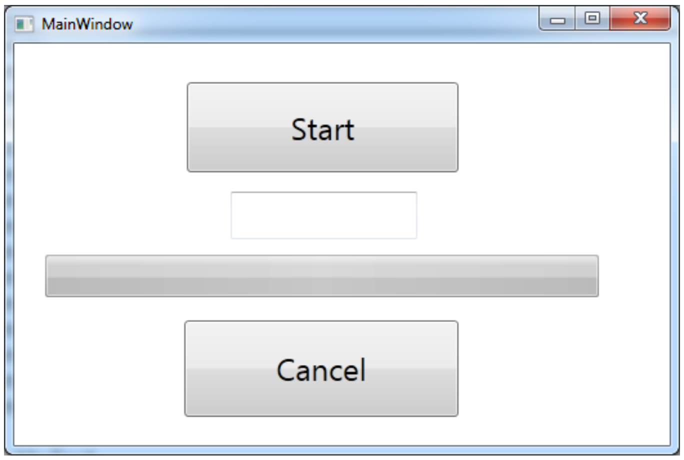
May 13, 2016

EE 356
Background Worker Example

Example 1

This example shows a single background worker. The work done is to sleep for a few milliseconds. The start button begins the work and the progress bar shows progress on the work. The percent of work done is also shown in the text box. The work may be cancelled using the cancel button. The main window is shown in Figure 1. The XAML code and the code behind is on the following pages.



Notes on the program

1. In WPF the progress bar and background worker have not been fully integrated into the system. For the background worker you need to create all of the events yourself.
2. The background worker runs some task (in this case sleep) on a separate thread from the thread pool. That means it cannot interact with the user interface since that would be crossing threads and this is not safe. The background worker provides a built in delegate to make this easier – this is the progress changed reporting.
3. The following line creates a background worker named bkw1.

```
static BackgroundWorker bkw1 = new BackgroundWorker();
```

You will need the using System.ComponentModel for this.

4. In the initialization you need to create the events like this using the += operator:

```
bkw1.DoWork += new DoWorkEventHandler(bkw1_DoWork);
bkw1.ProgressChanged += new
            ProgressChangedEventHandler(bkw1_ProgressChanged);
bkw1.RunWorkerCompleted += new
            RunWorkerCompletedEventHandler(bkw1_RunWorkerCompleted);
```

We also use two properties which we initialize to true.

```
bkw1.WorkerReportsProgress = true;
bkw1.WorkerSupportsCancellation = true;
```

5. The DoWork event gets implemented like this. It checks to see if the worker is being cancelled and if not it does work by sleeping for 100 milliseconds. The progress is reported as a percent of the total. In this case it's just the loop counter. The stub of this event and the others is not entered for you – you have to type them in yourself.

```
 //The DoWork module
 void bkw1_DoWork(object sender, DoWorkEventArgs e)
    {int i = 1;
     for(i=0;i<100;i++)                  //runs Sleep(100) 100 times
       {if(bkw1.CancellationPending)     //See if we should cancel
          {e.Cancel = true;
           return;
          }
        Thread.Sleep(100);
        bkw1.ReportProgress(i);         //Report progress as percent done
       }
     e.Result = "Done";                 //Send Done note on completion
    }
```

6. The main program, which in this case is the btnStart click event runs the worker with the statement

```
         bkw1.RunWorkerAsync("Message to Worker");
```

The message to the worker is optional.

7. Run the program and note that the progress bar and other user interface elements are not frozen as the worker runs.

May 13, 2016

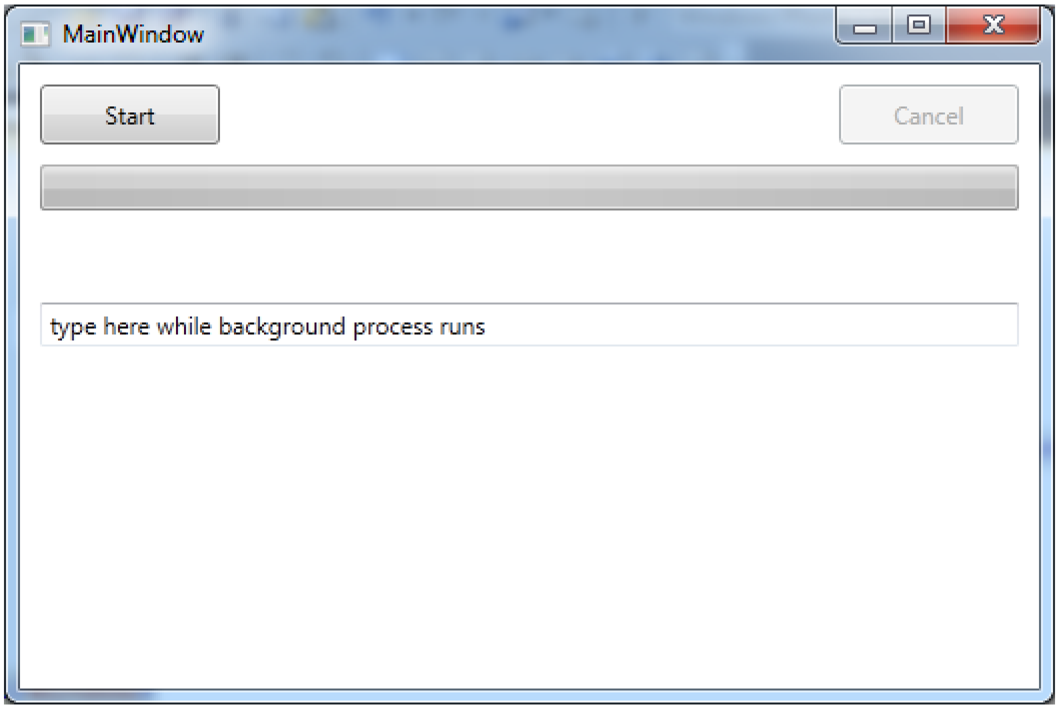
```xml
<Window x:Class="bkwTest.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
    <Grid>
      <Button Content="Start"
          Height="70"
          HorizontalAlignment="Left"
          Margin="134,30,0,0"
          Name="btnStart"
          VerticalAlignment="Top"
          Width="211"
          Click="btnStart_Click"
          FontSize="24" />
      <ProgressBar
          Height="33"
          HorizontalAlignment="Left"
          Margin="24,164,0,0"
          Name="pgb1"
          VerticalAlignment="Top"
          Width="430" />
      <Button Content="Cancel"
          Height="75"
          HorizontalAlignment="Left"
          Margin="132,215,0,0"
          Name="btnCancel"
          VerticalAlignment="Top"
          Width="213"
          FontSize="24"
          Click="btnCancel_Click" />
      <TextBox
         Height="37"
         HorizontalAlignment="Left"
         Margin="168,115,0,0"
         Name="txtPercent"
         VerticalAlignment="Top"
         Width="145"
         FontSize="24" />
   </Grid>
</Window>
```

May 13, 2016

```
using System;
using System.Collections.Generic;
using System.Linq;
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.Navigation;
using System.Windows.Shapes;
using System.ComponentModel;
using System.Threading;

namespace bkwTest
   {/// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
      {//Create a background worker.  Need using System.ComponentModel
       static BackgroundWorker bkw1 = new BackgroundWorker();

       public MainWindow()
         {InitializeComponent();
          //Add DoWork, ProgressChanged, and RunWorkerCompleted events
          bkw1.DoWork += new DoWorkEventHandler(bkw1_DoWork);
          bkw1.ProgressChanged += new ProgressChangedEventHandler(bkw1_ProgressChanged);
          bkw1.RunWorkerCompleted += new
                   RunWorkerCompletedEventHandler(bkw1_RunWorkerCompleted);
          //Set Progress and Cancellation properties to true
          bkw1.WorkerReportsProgress = true;
          bkw1.WorkerSupportsCancellation = true;
         }

      private void btnStart_Click(object sender, RoutedEventArgs e)
         {btnStart.IsEnabled = false;
          btnCancel.IsEnabled = true;
          pgb1.Value = 0;
          //Run the worker asynchronously
          bkw1.RunWorkerAsync("Message to Worker");
         }
      //The DoWork module
      void bkw1_DoWork(object sender, DoWorkEventArgs e)
         {int i = 1;
          for(i=0;i<100;i++)                  //runs Sleep(100) 100 times
            {if(bkw1.CancellationPending)     //See if we should cancel
               {e.Cancel = true;
                return;
               }
             Thread.Sleep(100);
             bkw1.ReportProgress(i);         //Report progress as percent done
            }
          e.Result = "Done";                 //Send Done note on completion
         }
      //The ProgressChanged event from ReportProgress
      void bkw1_ProgressChanged(object sender, ProgressChangedEventArgs e)
        {pgb1.Value = e.ProgressPercentage;    //Write to progress bar
         txtPercent.Text = (e.ProgressPercentage).ToString();  //and text box
        }
      //The completion method
      void bkw1_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
        {btnStart.IsEnabled = true;
         btnCancel.IsEnabled = false;
         pgb1.Value = 0;
         txtPercent.Text = (e.Result).ToString();
        }
      //The cancellation buttioni
      private void btnCancel_Click(object sender, RoutedEventArgs e)
         {bkw1.CancelAsync();
          btnCancel.IsEnabled = false;
          btnStart.IsEnabled = true;
         }
      }
   }
```

Example 2
This example is from:
`//http://www.tanguay.info/web/index.php?pg=codeExamples&id=232`
This example also uses the background worker but it uses a slightly different implementation making use of Lambda notation. The example runs two background workers with a progress bar, cancel button, and text blocks. The two background workers use the sleep function to do their work. The progress bar and text blocks are active while the workers are running. Both workers support cancellation and both are terminated with the same cancel button.

The XAML code produces the main window shown in Figure 2.



type here while background process runs

Figure 2
The user interface for the background worker example.

Notes on the program:
1. Unlike a forms project there is no background worker user interface element that you can drag onto the screen. All of the properties and events have to be user created.
2. If you attempt to use a progress bar from a running program you will find that in WPF it does not update until the running program ends and gives control of execution back to the operating system. The progress bar can be updated however, if you run the program using a background worker that reports its own progress as an event. The same is true for text boxes and other user interface elements that run on the UI thread.
3. For this example the first background worker get created and initialized with the following four statements:

```
  private BackgroundWorker bkw1;
  bkw1 = new BackgroundWorker();          //Initialize background worker 1 to
  bkw1.WorkerReportsProgress = true;      //  report progress and support
  bkw1.WorkerSupportsCancellation = true; //  cancellation
```

3. The following code shows how to create the DoWork event using Lambda notation. The Lambda expression allows us to create the worker function and register it as an event all in one (somewhat long) statement.

```
bkw1.DoWork += (s, args) =>
  {BackgroundWorker worker = s as BackgroundWorker;
   int numberOfTasks = 300;
   for (int i = 1; i <= numberOfTasks; i++)
      {if (worker.CancellationPending)
          {args.Cancel = true;
           return;
          }
       Thread.Sleep(10);                 //Work is to sleep for 10 msec
       float percentageDone = (i / (float)numberOfTasks) * 100f;
       worker.ReportProgress((int)percentageDone);
      }
 };
```

May 13, 2016

```
<Window x:Class="BackGroundWorkerTest.MainWindow"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="MainWindow" Height="350" Width="525">
   <StackPanel Margin="10">
      <DockPanel LastChildFill="True">
         <Button x:Name="btnStart"
                     DockPanel.Dock="Left"
                 Content="Start"
                 Width="90"
                 Height="30"
                 HorizontalAlignment="Left"
                     Click="btnStart_Click"
                 Margin="0 0 0 0"
                 />
         <Button x:Name="btnCancel"
                 DockPanel.Dock="Right"
                 Content="Cancel"
                 Width="90"
                 Height="30"
                 HorizontalAlignment="Right"
                     Click="btnCancel_Click"
                 Margin="10 0 0 0"
                 />
      </DockPanel>
      <ProgressBar x:Name="pgb1"
                     Margin="0 10 0 0"
                     Height="23"
                      Minimum="0"
                      Maximum="100"
                      />

      <TextBlock x:Name="txbMsg1"
                      Margin="0 10 0 0"
                      />

      <TextBox x:Name="txbInput"
                  Margin="0 20 0 0"
                  Text="type here while background process runs"
                  />
      <TextBlock x:Name="txbMsg2"
                  Margin="0 20 0 0"
                  />
   </StackPanel>
</Window>
```

May 13, 2016

```
using System;
using System.Collections.Generic;
using System.Linq;
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.Navigation;
using System.Windows.Shapes;
using System.ComponentModel;
using System.Threading;
namespace BackGroundWorkerTest
//This from
//http://www.tanguay.info/web/index.php?pg=codeExamples&id=232
// <summary>
// Interaction logic for MainWindow.xaml
// </summary>
public partial class MainWindow : Window
   {private BackgroundWorker bkw1;         //Create two background workers
    private BackgroundWorker bkw2;
    int thread1percentageFinished = 0;     //  and two thread variables
    int thread2percentageFinished = 0;

    public MainWindow()
      {InitializeComponent();
       btnCancel.IsEnabled = false;        //Disable cancel button on start up
      }
    //The start button
    private void btnStart_Click(object sender, RoutedEventArgs e)
      {bkw1 = new BackgroundWorker();          //Initialize background worker 1 to
       bkw1.WorkerReportsProgress = true;      //  report progress and support
       bkw1.WorkerSupportsCancellation = true; //  cancellation
       //Create the DoWork event using a Lambda expression
       bkw1.DoWork += (s, args) =>
          {BackgroundWorker worker = s as BackgroundWorker;
           int numberOfTasks = 300;
           for (int i = 1; i <= numberOfTasks; i++)
             {if (worker.CancellationPending)
                {args.Cancel = true;
                 return;
                }
              Thread.Sleep(10);                //Work is to sleep for 10 msec
              float percentageDone = (i / (float)numberOfTasks) * 100f;
              worker.ReportProgress((int)percentageDone);
             }
          };
        //Create the ProgressChanged event using Lambda expression
        bkw1.ProgressChanged += (s, args) =>
           {thread1percentageFinished = args.ProgressPercentage;
            //Report percent done to progress bar
            pgb1.Value = thread1percentageFinished;
            txbMsg1.Text = thread1percentageFinished + "% finished";
            //Make text change colors depending on % done
            if (thread1percentageFinished >= 70)
              {txbInput.Foreground = new SolidColorBrush(Colors.Red);
              }
            else if (thread1percentageFinished >= 40)
              {txbInput.Foreground = new SolidColorBrush(Colors.Orange);
              }
            else if (thread1percentageFinished >= 10)
              {txbInput.Foreground = new SolidColorBrush(Colors.Brown);
              }
            else
              {txbInput.Foreground = new SolidColorBrush(Colors.Black);
              }
          };
        //Create Completion event using a Lambda expression
        bkw1.RunWorkerCompleted += (s, args) =>
           {btnStart.IsEnabled = true;
            btnCancel.IsEnabled = false;
            pgb1.Value = 0;
            //Report status to text box
            if (thread1percentageFinished < 100)
              {txbMsg1.Text = "stopped at " + thread1percentageFinished + "%";
              }
            else
              {txbMsg1.Text = "first thread finished";
              }
          };

       //Create a second background worker.  This is identical to bkw1
       //  except it sleeps for a different amount of time and does not
       //  report progress if it is cancelled.
       bkw2 = new BackgroundWorker();
       bkw2.WorkerReportsProgress = true;
       bkw2.WorkerSupportsCancellation = true;

       bkw2.DoWork += (s, args) =>
            {BackgroundWorker worker2 = s as BackgroundWorker;
             int numberOfTasks = 1000;
             for (int i = 1; i <= numberOfTasks; i++)
                {if (worker2.CancellationPending)
                    {args.Cancel = true;
                        return;
                    }

                 Thread.Sleep(5);
                 float percentageDone = (i / (float)numberOfTasks) * 100f;
                 worker2.ReportProgress((int)percentageDone);
                }
            };

        bkw2.ProgressChanged += (s, args) =>
          {thread2percentageFinished = args.ProgressPercentage;
           txbMsg2.Text = String.Format("Second thread: {0}, ({1}% finished)",
             DateTime.Now.ToString("yyyy-MM-dd hh:mm:ss"), thread2percentageFinished);
              };
            //This is the main program that runs the two background workers
            bkw2.RunWorkerAsync();
            bkw1.RunWorkerAsync();
            btnStart.IsEnabled = false;
            btnCancel.IsEnabled = true;

        }

        //The cancel button.
        private void btnCancel_Click(object sender, RoutedEventArgs e)
           {bkw1.CancelAsync();
            bkw2.CancelAsync();
           }
```

May 13, 2016

```
        }
  }
```