Programming with C#

# Asynchronous Programming Practice

1. Start Visual Studio 2013.
2. Locate the GraphDemo solution from the companion content.
3. In Solution Explorer, expand the GraphDemo.xaml node and open the GraphDemo.xaml.cs file in the Code and Text Editor window.
4. In the *GraphWindow* class, locate the *plotButton\_Click* method.

The code in this method looks like this:

private void plotButton\_Click(object sender, RoutedEventArgs e)   
{   
 Random rand = new Random();   
 redValue = (byte)rand.Next(0xFF);   
 greenValue = (byte)rand.Next(0xFF);   
 blueValue = (byte)rand.Next(0xFF);   
   
 tokenSource = new CancellationTokenSource();   
 CancellationToken token = tokenSource.Token;   
 Stopwatch watch = Stopwatch.StartNew();   
  
 try   
 {   
 generateGraphData(data, 0, pixelWidth / 2, token);   
 duration.Text = string.Format("Duration (ms): {0}", watch.ElapsedMilliseconds);   
 }   
 catch (OperationCanceledException oce)  
 {

duration.Text = oce.Message;   
 }   
  
 Stream pixelStream = graphBitmap.PixelBuffer.AsStream();

pixelStream.Seek(0, SeekOrigin.Begin);   
 pixelStream.Write(data, 0, data.Length);   
 graphBitmap.Invalidate();   
 graphImage.Source = graphBitmap;   
}

This is a simplified version of the application from the previous chapter. It invokes the *generate GraphData* method directly from the user interface thread and does not use *Task* objects to generate the data for the graph in parallel.

**Note** If you reduced the size of the *pixelWidth* and *pixelHeight* fields in the exercises in Chapter 23 to save memory, do so again in this application before proceeding with the next step.

1. On the Debug menu, click Start Debugging.
2. In the GraphDemo window, click Plot Graph. While the data is being generated, try to click Cancel.

Notice that the user interface is completely unresponsive while the graph is being generated and displayed. This is because the *plotButton\_Click* method performs all of its work synchro­nously, including the generation of the data for the graph.

1. Return to Visual Studio and stop debugging.
2. In the Code and Text Editor window displaying the *GraphWindow* class, above the *generate GraphData* method, add a new private method called *generateGraphDataAsync*.

This method should take the same list of parameters as the *generateGraphData* method, but it should return a *Task* object rather than a *void*. The method should also be marked as *async*, and it should look like this:

private async Task generateGraphDataAsync(byte[] data, int partitionStart, int partitionEnd, CancellationToken token)   
{   
}

**Note** It is recommended practice to name asynchronous methods with the *Async* suffix.

1. In the *generateGraphDataAsync* method, add the statements shown here in bold.

private async Task generateGraphDataAsync(byte[] data, int partitionStart, int partitionEnd, CancellationToken token)   
{   
 **Task task = Task.Run(() => generateGraphData(data, partitionStart, partitionEnd, token));   
 await task;**   
}

This code creates a Task object that runs the generateGraphData method and uses the await operator to wait for the Task to complete. The task generated by the compiler as a result of the await operator is the value returned from the method.

1. Return to the plotButton\_Click method and change the definition of this method to include the async modifier, as shown in bold in the following code:

private async void plotButton\_Click(object sender, RoutedEventArgs e)   
{   
 ...   
}

1. In the try block in the plotButton\_Click method, modify the statement that generates the data for the graph to call the generateGraphDataAsync method asynchronously, as shown here in bold:

try   
 {   
 **await generateGraphDataAsync(data, 0, pixelWidth / 2, token);**   
 duration.Text = string.Format("Duration (ms): {0}", watch.ElapsedMilliseconds);   
 }  
 ...

1. On the Debug menu, click Exceptions. In the Exceptions dialog box, expand Common Language Runtime Exceptions, expand System, clear the User-unhandled check box for the System.OperationCanceledException exception, and then click OK.
2. This step prevents the debugger from intercepting the System.OperationCanceledException exception while continuing to report any other exceptions that might occur.
3. On the Debug menu, click Start Debugging.
4. In the GraphDemo window, click Plot Graph and verify that the application generates the graph correctly.
5. Click Plot Graph and, while the data is being generated, click Cancel.
6. This time, the user interface should be responsive. Only part of the graph should be gener¬ated, and the duration TextBlock should display the message “The operation was cancelled.”
7. Return to Visual Studio and stop debugging