ASYNCHRONOUS PROGRAMING WITH async AND await

Prior to the introduction of Task Parallel Library (TPL) in .net framework 4.0, writing an asynchronous program was quite a tedious task. One had to write too many lines of code in order to accomplish a simple multi-threaded operation; for example creating wait handles, writing callbacks, complex error handling, etc.

Visual C# 2012 introduces the async modifier and await operator to greatly simplify the asynchronous programming.

async and await

The **async** keyword can be used as a modifier to a method or anonymous method to tell the C# compiler that the respective method holds an asynchronous operation in it.

The **await** keyword is used while invoking the async method, to tell the compiler that the remaining code in that particular method should be executed after the asynchronous operation is completed.

Use async

1. Use async modifier to a method, anonymous method or lambda expression, which will have the await operation in it.

```
public async void GetStringAsync()
```

2. All method that have the await operations should be marked with the async keyword. This applies to control events as well.

```
public async void button1_OnClick(object sender, ClickEventArgs e)
```

3. The await operator can be used only on a method marked with async modifier.

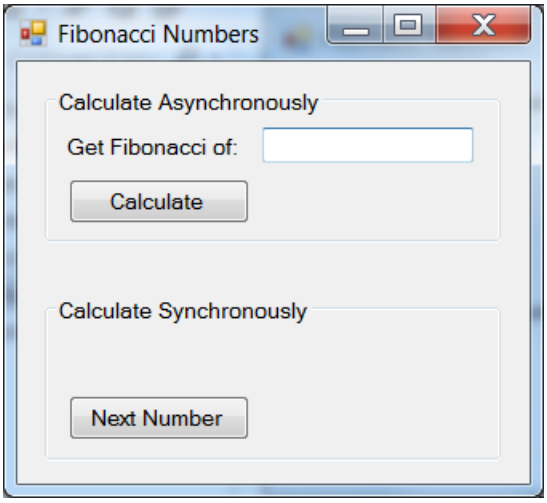
4. An async method can return only void, Task or Task<T>. Basically it should be an awaitable type.

5. Microsoft recommends suffixing the asynchronous method name with ‘Async’ like GetStringAsync instead of GetString.

6. Multiple await statements can be added into a single async method.

Sample Code

Fibonacci test




```
// Performing a compute-intensive calculation from a GUI app
using System;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FibonacciTest
{
   public partial class FibonacciForm : Form
   {
      private long n1 = 0; // initialize with first Fibonacci number
      private long n2 = 1; // initialize with second Fibonacci number
      private int count = 1; // current Fibonacci number to display

      public FibonacciForm()
      {
         InitializeComponent();
      } // end constructor

      // start an async Task to calculate specified Fibonacci number
      private async void calculateButton_Click(
         object sender, EventArgs e )
      {
         // retrieve user's input as an integer
         int number = Convert.ToInt32( inputTextBox.Text );

         asyncResultLabel.Text = "Calculating...";

         // Task to perform Fibonacci calculation in separate thread
         Task< long > fibonacciTask =
            Task.Run( () => Fibonacci( number ) );

         // wait for Task in separate thread to complete
         await fibonacciTask;

         // display result after Task in separate thread completes
```

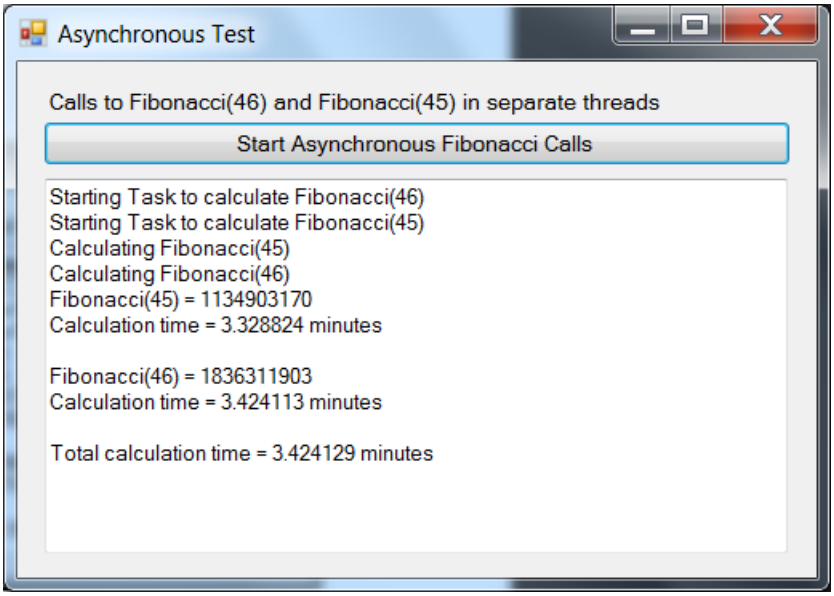
```
            asyncResultLabel.Text = fibonacciTask.Result.ToString();
         } // end method calculateButton_Click

         // calculate next Fibonacci number iteratively
         private void nextNumberButton_Click( object sender, EventArgs e )
         {
            // calculate the next Fibonacci number
            long temp = n1 + n2; // calculate next Fibonacci number
            n1 = n2; // store prior Fibonacci number in n1
            n2 = temp; // store new Fibonacci
            ++count;

            // display the next Fibonacci number
            displayLabel.Text = string.Format( "Fibonacci of {0}:", count );
            syncResultLabel.Text = n2.ToString();
         } // end method nextNumberButton_Click

         // recursive method Fibonacci; calculates nth Fibonacci number
         public long Fibonacci( long n )
         {
            if ( n == 0 || n == 1 )
               return n;
            else
               return Fibonacci( n - 1 ) + Fibonacci( n - 2 );
         } // end method Fibonacci
      } // end class FibonacciForm
} // end namespace FibonacciTest
```

Asynchronous Execution of two computed intensive tasks



```
// Fibonacci calculations performed in separate threads
using System;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FibonacciAsynchronous
{
   public partial class AsynchronousTestForm : Form
   {
      public AsynchronousTestForm()
      {
         InitializeComponent();
      } // end constructor

      // start asynchronous calls to Fibonacci
      private async void startButton_Click( object sender, EventArgs e )
      {
         outputTextBox.Text =
            "Starting Task to calculate Fibonacci(46)\r\n";

         // create Task to perform Fibonacci(46) calculation in a thread
         Task< TimeData > task1 =
            Task.Run( () => StartFibonacci( 46 ) );

         outputTextBox.AppendText(
            "Starting Task to calculate Fibonacci(45)\r\n" );

         // create Task to perform Fibonacci(45) calculation in a thread
         Task< TimeData > task2 =
            Task.Run( () => StartFibonacci( 45 ) );

         await Task.WhenAll( task1, task2 ); // wait for both to complete

         // determine time that first thread started
         DateTime startTime =
            ( task1.Result.StartTime < task2.Result.StartTime ) ?
            task1.Result.StartTime : task2.Result.StartTime;

         // determine time that last thread ended
         DateTime endTime =
            ( task1.Result.EndTime > task2.Result.EndTime ) ?
            task1.Result.EndTime : task2.Result.EndTime;

         // display total time for calculations
         outputTextBox.AppendText( String.Format(
            "Total calculation time = {0:F6} minutes\r\n",
            endTime.Subtract( startTime ).TotalMilliseconds /
            60000.0 ) );
      } // end method startButton_Click

      // starts a call to fibonacci and captures start/end times
      TimeData StartFibonacci( int n )
      {
         // create a ThreadData object to store start/end times
         TimeData result = new TimeData();

         AppendText( String.Format( "Calculating Fibonacci({0})", n ) );
         result.StartTime = DateTime.Now; // time before calculation
```

```
            long fibonacciValue = Fibonacci( n );
            result.EndTime = DateTime.Now; // time after calculation

            AppendText( String.Format( "Fibonacci({0}) = {1}",
               n, fibonacciValue ) );
            AppendText( String.Format(
               "Calculation time = {0:F6} minutes\r\n",
               result.EndTime.Subtract(
                  result.StartTime ).TotalMilliseconds / 60000.0 ) );

            return result;
         } // end method StartFibonacci

         // Recursively calculates Fibonacci numbers
         public long Fibonacci( long n )
         {
            if ( n == 0 || n == 1 )
               return n;
            else
               return Fibonacci( n - 1 ) + Fibonacci( n - 2 );
         } // end method Fibonacci

         // append text to outputTextBox in UI thread
         public void AppendText( String text )
         {
            if ( InvokeRequired ) // not GUI thread, so add to GUI thread
               Invoke( new MethodInvoker( () => AppendText( text ) ) );
            else // GUI thread so append text
               outputTextBox.AppendText( text + "\r\n" );
         } // end method AppendText
      } // end class AsynchronousTestForm

} // end namespace FibonacciAsynchronous
```