# **Event Based Programming model**

**Event Based Programming model** depends on the following:

* Your method should be executed asynchronously.
* It should raise an event once the execution has come to an end.

 To implement this pattern, a class needs to:

* have a method with the name XXXAsync, where XXX is the name of the synchronous method;
* the XXXAsync method receives the same parameters that are expected by the synchronous version plus an optional object parameter (used to pass extra state that should be consumed later);
* expose an event with the name XXXCompleted, which should be fired when the asynchronous task completes;
* the EventArgs type of that event should be an AsyncCompletedEventArgs derived class;
* expose a CancelAsync method (which might optionally receive an object parameter) that is responsible for cancelling the async operation;
* optionally expose a ProgressChanged event (of type ProgressChangedEventHandler), that can be consumed for getting info  on the progress of the operation.

public class AsyncExample

{

// Synchronous methods.

public int Method1(string param);

public void Method2(double param);

// Asynchronous methods.

public void Method1Async(string param);

public void Method1Async(string param, object userState);

public event Method1CompletedEventHandler Method1Completed;

public void Method2Async(double param);

public void Method2Async(double param, object userState);

public event Method2CompletedEventHandler Method2Completed;

public void CancelAsync(object userState);

public bool IsBusy { get; }

// Class implementation not shown.

}

 You can use ***Thread, ThreadStart, Threadpool, BackgroundWorker,***or any other technique of your choice.

public void GetPrimeCountAsync(int min, int count)

{

int result = -1;

ManualResetEvent evt = new ManualResetEvent(false);

WaitCallback wait = new WaitCallback((x) =>

{

try

{

result = GetPrimeCount(min, count);

}

catch (Exception ex)

{

*//ToDo handle the exception*

}

finally

{

evt.Set();

}

});

ThreadPool.QueueUserWorkItem(wait);

evt.WaitOne();

evt.Close();

On\_GetPrimeCount\_Completed(new GetPrimeEventArg()

{ Minimum = min, Count = count, Result = result });

}

**When to use Event Based Asynchronous Model**

* GUI Components
* Cancel Async operation is needed
* Update progress on Async Operaration

[**WebClient**](http://msdn.microsoft.com/en-us/library/system.net.webclient.aspx) and **[BackgroundWorker](http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx)** are two typical components that use this pattern

**<http://blogs.msmvps.com/luisabreu/blog/2009/06/16/multithreading-introducing-the-event-based-asynchronous-pattern/>**

<http://www.codeproject.com/Articles/646239/NET-Asynchronous-Patterns>