Using Delegates

.NET Framework 1.1

With asynchronous programming, the caller must define the delegate when calling a method, if the caller uses a delegate. In the following code sample, the delegate is first defined, then an instance of it created, and then it is called. The sample below shows the caller defining a pattern for the invoking the **Factorize** method asynchronously:

```
using System;
using System.Runtime.Remoting;

public delegate bool FactorizingAsyncDelegate(
        int factorizableNum,
        ref int primefactor1,
        ref int primefactor2);

// This is a class that receives a callback when the results are available.
public class ProcessFactorizedNumber
{
  private int _ulNumber;

  public ProcessFactorizedNumber(int number)
  {
    _ulNumber = number;
  }

  // Note the qualifier one-way.
  [OneWayAttribute()]
  public void FactorizedResults(IAsyncResult ar)
  {
    int factor1=0, factor2=0;

    // Extract the delegate from the AsyncResult.
    FactorizingAsyncDelegate fd =
```

```
        (FactorizingAsyncDelegate) ((AsyncResult)ar).AsyncDelegate;
      // Obtain the result.
      fd.EndInvoke(ref factor1, ref factor2, ar);

      // Output the results.
      Console.WriteLine("On CallBack: Factors of {0} : {1} {2}",
                  _ulNumber, factor1, factor2);
    }
}
Asynchronous Variation 1 – call
// The Asynchronous Variation 1 call, calls
// the ProcessFactorizedNumber.FactorizedResults callback
// when the call completes.
public void FactorizeNumber1()
{
   // The following is the Client code.
   PrimeFactorizer pf = new PrimeFactorizer();
   FactorizingAsyncDelegate fd = new FactorizingAsyncDelegate (pf.Factorize);

   // Asynchronous Variation 1
   int factorizableNum = 1000589023, temp=0;

   // Create an instance of the class that is going
   // to be called when the call completes.
   ProcessFactorizedNumber fc = new ProcessFactorizedNumber(factorizableNum);

   // Define the AsyncCallback delegate.
   AsyncCallback cb = new AsyncCallback(fc.FactorizedResults);

   // You can use any object as the state object.
   Object state = new Object();

   // Asynchronously invoke the Factorize method on pf.
   IAsyncResult ar = fd.BeginInvoke(
                     factorizableNum,
```

```
                ref temp,
                ref temp,
                cb,
                state);

    //
    // Do some other useful work.
    //. . .
}
Asynchronous Variation 2
// Asynchronous Variation 2
// Waits for the result.
public void FactorizeNumber2()
{
    // The following is the Client code.
    PrimeFactorizer pf = new PrimeFactorizer();
    FactorizingAsyncDelegate fd = new FactorizingAsyncDelegate (pf.Factorize);

    // Asynchronous Variation 1
    int factorizableNum = 1000589023, temp=0;

    // Create an instance of the class that is going
    // to called when the call completes.
    ProcessFactorizedNumber fc = new ProcessFactorizedNumber(factorizableNum);

    // Define the AsyncCallback delegate.
    AsyncCallback cb =
    new AsyncCallback(fc.FactorizedResults);

    // You can use any object as the state object.
    Object state = new Object();

    // Asynchronously invoke the Factorize method on pf.
    IAsyncResult ar = fd.BeginInvoke(
                    factorizableNum,
```

```
                ref temp,
                ref temp,
                null,
                null);

    ar.AsyncWaitHandle.WaitOne(10000, false);

    if (ar.IsCompleted)
    {
      int factor1=0, factor2=0;

      // Obtain the result.
      fd.EndInvoke(ref factor1, ref factor2, ar);

      // Output the results.

      Console.WriteLine("Sequential : Factors of {0} : {1} {2}",
              factorizableNum, factor1, factor2);

    }
}
```

Note Calling **EndInvoke** before the asynchronous operation is complete will block the caller. Calling it the second time with the same **IAsyncResult** is undefined.