**Creating Events and Delegates**  
  
When you create an event in a struct or a class, you need a way of enabling other code to subscribe to your event. In Visual C#, you accomplish this by creating a delegate.  Once again, a delegate is a special type that defines a method signature; in other words, the return type and the parameters of a method. As the name suggests, a delegate behaves like a representative for methods with matching signatures.

When you define an event, you associate a delegate with your event. To subscribe to the event from client code, you need to:

* Create a method with a signature that matches the event delegate. This method is known as the event handler.
* Subscribe to the event by giving the name of your event handler method to the event publisher, in other words, the object that will raise the event.

When the event is raised, the delegate invokes all the event handler methods that have subscribed to the event.  
  
Suppose you create a struct named Coffee. One of the responsibilities of this struct is to keep track of the stock level for each Coffee instance. When the stock level drops below a certain point, you might want to raise an event to warn an ordering system that you are running out of beans.   
  
The first thing you need to do is to define a delegate. To define a delegate, you use the delegate keyword. A delegate includes two parameters:

* The first parameter is the object that raised the event—in this case, a Coffee instance.
* The second parameter is the event arguments—in other words, any other information that you want to provide to consumers. This must be an instance of the EventArgs class, or an instance of a class that derives from EventArgs.

Next, you need to define the event. To define an event, you use the event keyword. You precede the name of your event with the name of the delegate you want to associate with your event.  
  
The following example shows how to define delegates and events:  
  
//Defining a Delegate and an Event  
public struct Coffee  
{  
   public EventArgs e;  
   public delegate void OutOfBeansHandler(Coffee coffee, EventArgs args);  
   public event OutOfBeansHandler OutOfBeans;  
}  
  
In this example, you define an event named OutOfBeans. You associate a delegate named OutOfBeansHandler with your event. The OutOfBeansHandler delegate takes two parameters, an instance of Coffee that will represent the object that raised the event and an instance of EventArgs that could be used to provide more information about the event.

**Raising Events**  
  
After you have defined an event and a delegate, you can write code that raises the event when certain conditions are met. When you raise the event, the delegate associated with your event will invoke any event handler methods that have subscribed to your event.  
  
To raise an event, you need to do two things:

1. Check whether the event is null. The event will be null if no code is currently subscribing to it.
2. Invoke the event and provide arguments to the delegate.

For example, suppose a Coffee struct includes a method named MakeCoffee. Every time you call the MakeCoffee method, the method reduces the stock level of the Coffee instance. If the stock level drops below a certain point, the MakeCoffee method will raise an OutOfBeans event.  
  
The following example shows how to raise an event:  
  
// Raising an Event  
public struct Coffee  
{  
   // Declare the event and the delegate.  
   public EventArgs e = null;  
   public delegate void OutOfBeansHandler(Coffee coffee, EventArgs args);  
   public event OutOfBeansHandler OutOfBeans;  
   int currentStockLevel;  
   int minimumStockLevel;  
   public void MakeCoffee()  
   {  
      // Decrement the stock level.  
      currentStockLevel--;  
      // If the stock level drops below the minimum, raise the event.  
      if (currentStockLevel < minimumStockLevel)  
      {  
         // Check whether the event is null.  
         if (OutOfBeans != null)  
         {  
            // Raise the event.  
            OutOfBeans(this, e);   
         }  
      }  
   }  
}

To raise the event, you use a similar syntax to calling a method. You provide arguments to match the parameters required by the delegate. The first argument is the object that raised the event. Note how the this keyword is used to indicate the current Coffee instance. The second parameter is the EventArgs instance, which can be null if you do not need to provide any other information to subscribers.

**Subscribing to Events**

If you want to handle an event in client code, you need to do two things:

* Create a method with a signature that matches the delegate for the event.
* Use the addition assignment operator (+=) to attach your event handler method to the event.

For example, suppose you have created an instance of the Coffee struct named coffee1. In your Inventory class, you want to subscribe to the OutOfBeans that may be raised by coffee1.  
  
    *Note: The previous topic shows how the Coffee struct, the OutOfBeans event, and the OutOfBeansHandler delegate are defined.*  
  
The following example shows how to subscribe to an event:  
  
// Subscribing to an Event  
public class Inventory  
{  
  public void HandleOutOfBeans(Coffee sender, EventArgs args)  
   {  
      string coffeeBean = sender.Bean;  
      // Reorder the coffee bean.  
   }  
   public void SubscribeToEvent()  
   {  
      coffee1.OutOfBeans += HandleOutOfBeans;  
   }  
}

In this example, the signature of the HandleOutOfBeans method matches the delegate for the OutOfBeans event. When you call the SubscribeToEvent method, the HandleOutOfBeans method is added to the list of subscribers for the OutOfBeans event on the coffee1 object.  
  
To unsubscribe from an event, you use the subtraction assignment operator (-=) to remove your event handler method from the event.  
  
The following example shows how to unsubscribe from an event:  
  
// Unsubscribing from an Event  
public void UnsubscribeFromEvent()  
{  
   coffee1.OutOfBeans -= HandleOutOfBeans;  
}