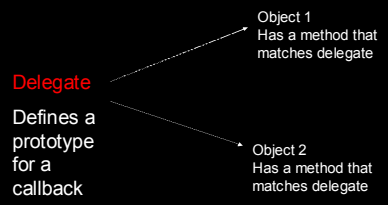
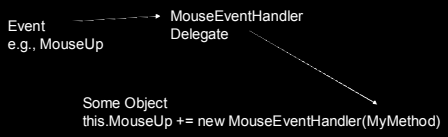
Custom Events

Delegates



Predefined Events



Custom Events

- If a class wants to define a custom event (one that is not defined in the FCL), it must use syntax that follows this model:
- public event <DelegateName><EventName>
- DelegateName will be a new delegate, probably declared in the same class as the event.

Custom Events

- Often, events will be accompanied by additional information that is passed in an object of type System.EventArgs
- e.g., MouseEventArgs carries with it the X and Y locations of where the mouse was clicked on the form

Custom Events

- If your custom events need to send additional information, you should put the information in a class that descends from System.EventArgs.
- **Extended Example:** We are going to make a class named EmailMgr that will expose an event named EmailEvent

Custom Events

- Other classes can then register their interest in this event so that when the EmailEvent is fired, they will be notified.
- The EmailEvent will be fired such that additional information (sender, subject and message) will accompany it.
- What follows is what the developer of the EmailMgr class must do.

Example

- **1.) Define a type that will hold any additional information that should be sent to receivers of the event.**

```
public class EmailEventArgs : EventArgs
{
        string sender, subject, message;
        public EmailEventArgs(string sender, string subject, string msg): base()
        {
           this.sender = sender; this.subject = subject; message = msg;
        }

        public string GetMail
        {
           get { return sender + " " + subject + " " + message; }
        }
}
```

EventArgs

- EventArgs is defined in the FCL and looks like this:

```
[Serializable]
public class EventArgs
{
  public static readonly EventArgs Empty = new
    EventArgs();
  public EventArgs();
}
```

- As you can see, does not offer much. Just serves as a base type.

EventArgs

- Many events have no additional info to pass along (e.g., Button Click just notifies listeners it has been clicked.) No additional info in EventArgs.
- If your custom classes have no additional info to pass along, just use EventArgs.Empty rather than construct a new EventArgs object.

Example

2.) Define a delegate type that specifies the prototype of the method that will be called when the event fires.

```
public delegate void
   EmailEventHandler(object sender,
   EmailEventArgs eea);
```

Event Handlers

- By convention, this delegate ends with *EventHandler*.
- Also by convention, this method should be *void* and should take 2 parameters.
- 1st arg is the object sending the notification, and 2nd is EventArgs (derived) type with additional info that receivers of the notification require.

Event Handlers

- If you are defining an event that has no additional info, you don't have to define a new delegate.
- You can use FCL's System.EventHandler delegate and pass EventArgs.Empty
- its prototype is

public delegate void EventHandler(object sender, EventArgs e)

- For example, when you register a Win Form as a listener for a Button click, VS.NET generates a line of code that reads:

MyButton.Click += new System.EventHandler(MethodName);

Example

3.) Define an event of your delegate type

public event EmailEventHandler EmailEvent;

Example

4.) Define a protected, virtual method responsible for notifying registered objects of the event.

```
protected virtual void
   OnEmailEvent(EmailEventArgs eea)
        {
           if (EmailEvent != null)
              EmailEvent(this, eea);
        }
```

Example

- **5.) Define a method that will execute OnEmailEvent to get the ball rolling**

```
public void SimulateEmailEvent()
{
// Create instance of special EventArgs object, prime the property,
// call OnEmailEvent
    EmailEventArgs eea = new EmailEventArgs( "Brenda",
    "Meeting",  "Let's make an appointment." );

    OnEmailEvent(eea);
}
```

Compiler Actions

- When the C# compiler sees the line of code

public event EmailEventHandler EmailEvent;

- it generates 3 things:

Compiler Actions

1.) A private delegate field that is initialized to null

private EmailEventHandler EmailEvent = null;

Compiler Actions

- Then it generates 2 other public methods

```
public void add_EmailEvent(EmailEventHandler handler)
{
EmailEvent = (EmailEventHandler) Delegate.Combine(EmailEvent,
    handler);
}
And

public void remove_EmailEvent(EmailEventHandler handler)
{
  EmailEvent = (EmailEventHandler) Delegate.Remove(EmailEvent,
    handler);
}
```

Example

- Design a type that listens for the event

```
public class BlackBerry
{
      public BlackBerry() { }

      // Method with required signature that gets called back
      public void YouHaveMail(object sender, EmailEventArgs eea)
      {
           string email = eea.GetMail;
           Console.WriteLine("Email has arrived.");
            Console.WriteLine("The email is " + email);
            Console.WriteLine("This method was called by " +
    sender.ToString());

       }

}
```

Example

In Main(), get everything going:

- Create an instance of the EmailMgr class

EmailMgr em= new EmailMgr();

- Create an instance of the class to be notified about the event

BlackBerry bb = new BlackBerry();

Example

- Wire up the BlackBerry's event handler for the EmailEvent

em.EmailEvent += new EmailMgr.EmailEventHandler(bb.YouHaveMail);

- Compiler turns this into

```
em.add_EmailEvent(new EmailEventHandler(bb.YouHaveMail));
```

Example

- Finally, call the method in EmailMgr that gets the ball rolling

em.SimulateEmailEvent();