**Delegates & Events Tutorial**

Contents

[**Delegates Tutorial** 1](#_Toc426118439)

[**Sample Files** 1](#_Toc426118440)

[**Further Reading** 2](#_Toc426118441)

[**Tutorial** 2](#_Toc426118442)

[**Delegates and Events** 2](#_Toc426118443)

[**Delegates vs. Interfaces** 2](#_Toc426118444)

[**Events Tutorial** 2](#_Toc426118445)

[**Sample Files** 2](#_Toc426118446)

[**Further Reading** 2](#_Toc426118447)

[**Tutorial** 2](#_Toc426118448)

[**Events and Inheritance** 2](#_Toc426118449)

[**Events in Interfaces** 2](#_Toc426118450)

[**.NET Framework Guidelines** 2](#_Toc426118451)

[Handling and Raising Events 2](#_Toc426118452)

[**Events and Delegates** 2](#_Toc426118453)

[**Consuming Events** 2](#_Toc426118454)

[**Consuming an Event in a Web Forms Application** 2](#_Toc426118455)

[**Consuming an Event in a Windows Forms Application** 2](#_Toc426118456)

[**The Event Pattern** 2](#_Toc426118457)

[**Raising an Event** 2](#_Toc426118458)

[**Event Sample** 2](#_Toc426118459)

[Raising Multiple Events 2](#_Toc426118460)

This tutorial demonstrates the delegate types. It shows how to map delegates to static and instance methods, and how to combine them (multicast).

**Sample Files**

See [Delegates Sample](https://msdn.microsoft.com/en-us/library/3sw9sk6y(v=vs.71).aspx) to download and build the sample files discussed in this tutorial.

**Further Reading**

* [delegate](https://msdn.microsoft.com/en-us/library/900fyy8e(v=vs.71).aspx)
* [15. Delegates](https://msdn.microsoft.com/en-us/library/aa664602(v=vs.71).aspx)
* [Events Tutorial](https://msdn.microsoft.com/en-us/library/aa645739(v=vs.71).aspx)
* [Asynchronous Delegates](https://msdn.microsoft.com/en-us/library/aa735790(v=vs.71).aspx)

**Tutorial**

A delegate in C# is similar to a function pointer in C or C++. Using a delegate allows the programmer to encapsulate a reference to a method inside a delegate object. The delegate object can then be passed to code which can call the referenced method, without having to know at compile time which method will be invoked. Unlike function pointers in C or C++, delegates are object-oriented, type-safe, and secure.

A delegate declaration defines a type that encapsulates a method with a particular set of arguments and return type. For static methods, a delegate object encapsulates the method to be called. For instance methods, a delegate object encapsulates both an instance and a method on the instance. If you have a delegate object and an appropriate set of arguments, you can invoke the delegate with the arguments.

An interesting and useful property of a delegate is that it does not know or care about the class of the object that it references. Any object will do; all that matters is that the method's argument types and return type match the delegate's. This makes delegates perfectly suited for "anonymous" invocation.

**Note**Delegates run under the caller's security permissions, not the declarer's permissions.

This tutorial includes two examples:

* Example 1 shows how to declare, instantiate, and call a delegate.
* Example 2 shows how to combine two delegates.

In addition, it discusses the following topics:

* Delegates and Events
* Delegates vs. Interfaces

**Example 1**

The following example illustrates declaring, instantiating, and using a delegate. The BookDB class encapsulates a bookstore database that maintains a database of books. It exposes a method ProcessPaperbackBooks, which finds all paperback books in the database and calls a delegate for each one. The **delegate** type used is called ProcessBookDelegate. The Test class uses this class to print out the titles and average price of the paperback books.

The use of delegates promotes good separation of functionality between the bookstore database and the client code. The client code has no knowledge of how the books are stored or how the bookstore code finds paperback books. The bookstore code has no knowledge of what processing is done on the paperback books after it finds them.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_9a14b3a7-a02e-4c96-85f2-c0a320900f39');" \o "Copy to clipboard.)

// bookstore.cs

using System;

// A set of classes for handling a bookstore:

namespace Bookstore

{

using System.Collections;

// Describes a book in the book list:

public struct Book

{

public string Title; // Title of the book.

public string Author; // Author of the book.

public decimal Price; // Price of the book.

public bool Paperback; // Is it paperback?

public Book(string title, string author, decimal price, bool paperBack)

{

Title = title;

Author = author;

Price = price;

Paperback = paperBack;

}

}

// Declare a delegate type for processing a book:

public delegate void ProcessBookDelegate(Book book);

// Maintains a book database.

public class BookDB

{

// List of all books in the database:

ArrayList list = new ArrayList();

// Add a book to the database:

public void AddBook(string title, string author, decimal price, bool paperBack)

{

list.Add(new Book(title, author, price, paperBack));

}

// Call a passed-in delegate on each paperback book to process it:

public void ProcessPaperbackBooks(ProcessBookDelegate processBook)

{

foreach (Book b in list)

{

if (b.Paperback)

// Calling the delegate:

processBook(b);

}

}

}

}

// Using the Bookstore classes:

namespace BookTestClient

{

using Bookstore;

// Class to total and average prices of books:

class PriceTotaller

{

int countBooks = 0;

decimal priceBooks = 0.0m;

internal void AddBookToTotal(Book book)

{

countBooks += 1;

priceBooks += book.Price;

}

internal decimal AveragePrice()

{

return priceBooks / countBooks;

}

}

// Class to test the book database:

class Test

{

// Print the title of the book.

static void PrintTitle(Book b)

{

Console.WriteLine(" {0}", b.Title);

}

// Execution starts here.

static void Main()

{

BookDB bookDB = new BookDB();

// Initialize the database with some books:

AddBooks(bookDB);

// Print all the titles of paperbacks:

Console.WriteLine("Paperback Book Titles:");

// Create a new delegate object associated with the static

// method Test.PrintTitle:

bookDB.ProcessPaperbackBooks(new ProcessBookDelegate(PrintTitle));

// Get the average price of a paperback by using

// a PriceTotaller object:

PriceTotaller totaller = new PriceTotaller();

// Create a new delegate object associated with the nonstatic

// method AddBookToTotal on the object totaller:

bookDB.ProcessPaperbackBooks(new ProcessBookDelegate(totaller.AddBookToTotal));

Console.WriteLine("Average Paperback Book Price: ${0:#.##}",

totaller.AveragePrice());

}

// Initialize the book database with some test books:

static void AddBooks(BookDB bookDB)

{

bookDB.AddBook("The C Programming Language",

"Brian W. Kernighan and Dennis M. Ritchie", 19.95m, true);

bookDB.AddBook("The Unicode Standard 2.0",

"The Unicode Consortium", 39.95m, true);

bookDB.AddBook("The MS-DOS Encyclopedia",

"Ray Duncan", 129.95m, false);

bookDB.AddBook("Dogbert's Clues for the Clueless",

"Scott Adams", 12.00m, true);

}

}

}

**Output**

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_a780d6cb-192c-4f3c-8b42-3352cadee261');)

Paperback Book Titles:

The C Programming Language

The Unicode Standard 2.0

Dogbert's Clues for the Clueless

Average Paperback Book Price: $23.97

**Code Discussion**

* **Declaring a delegate**The following statement:

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_2c60984d-1fda-4981-88ee-4c8e114d1b68');)

public delegate void ProcessBookDelegate(Book book);

declares a new delegate type. Each delegate type describes the number and types of the arguments, and the type of the return value of methods that it can encapsulate. Whenever a new set of argument types or return value type is needed, a new delegate type must be declared.

* **Instantiating a delegate**Once a delegate type has been declared, a delegate object must be created and associated with a particular method. Like all other objects, a new delegate object is created with a **new** expression. When creating a delegate, however, the argument passed to the **new** expression is special — it is written like a method call, but without the arguments to the method.

The following statement:

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_7d07a629-616e-4766-b470-43339a3b7f34');)

bookDB.ProcessPaperbackBooks(new ProcessBookDelegate(PrintTitle));

creates a new delegate object associated with the static method Test.PrintTitle. The following statement:

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_df9abd82-0ad8-4f63-900c-edd68b1d6e23');)

bookDB.ProcessPaperbackBooks(new

ProcessBookDelegate(totaller.AddBookToTotal));

creates a new delegate object associated with the nonstatic method AddBookToTotal on the object totaller. In both cases, this new delegate object is immediately passed to the ProcessPaperbackBooks method.

Note that once a delegate is created, the method it is associated with never changes — delegate objects are immutable.

* **Calling a delegate**Once a delegate object is created, the delegate object is typically passed to other code that will call the delegate. A delegate object is called by using the name of the delegate object, followed by the parenthesized arguments to be passed to the delegate. An example of a delegate call is:

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_7aa6d619-3d3b-4fa5-a6da-3e8318767992');)

processBook(b);

A delegate can either be called synchronously, as in this example, or asynchronously by using [BeginInvoke](https://msdn.microsoft.com/en-us/library/system.windows.forms.control.begininvoke(v=vs.71).aspx) and [EndInvoke](https://msdn.microsoft.com/en-us/library/zbdewc0s(v=vs.71).aspx) methods.

**Example 2**

This example demonstrates composing delegates. A useful property of delegate objects is that they can be composed using the "**+**" operator. A composed delegate calls the two delegates it was composed from. Only delegates of the same type can be composed.

The "**-**" operator can be used to remove a component delegate from a composed delegate.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_b2f101cd-b6be-4120-98f5-997d835602f0');)

// compose.cs

using System;

delegate void MyDelegate(string s);

class MyClass

{

public static void Hello(string s)

{

Console.WriteLine(" Hello, {0}!", s);

}

public static void Goodbye(string s)

{

Console.WriteLine(" Goodbye, {0}!", s);

}

public static void Main()

{

MyDelegate a, b, c, d;

// Create the delegate object a that references

// the method Hello:

a = new MyDelegate(Hello);

// Create the delegate object b that references

// the method Goodbye:

b = new MyDelegate(Goodbye);

// The two delegates, a and b, are composed to form c:

c = a + b;

// Remove a from the composed delegate, leaving d,

// which calls only the method Goodbye:

d = c - a;

Console.WriteLine("Invoking delegate a:");

a("A");

Console.WriteLine("Invoking delegate b:");

b("B");

Console.WriteLine("Invoking delegate c:");

c("C");

Console.WriteLine("Invoking delegate d:");

d("D");

}

}

**Output**

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_d962cbc8-c4df-4e89-ab2c-b6a79b854320');)

Invoking delegate a:

Hello, A!

Invoking delegate b:

Goodbye, B!

Invoking delegate c:

Hello, C!

Goodbye, C!

Invoking delegate d:

Goodbye, D!

**Delegates and Events**

Delegates are ideally suited for use as events — notifications from one component to "listeners" about changes in that component. For more information on the use of delegates for events, see the [Events Tutorial](https://msdn.microsoft.com/en-us/library/aa645739(v=vs.71).aspx).

**Delegates vs. Interfaces**

Delegates and interfaces are similar in that they enable the separation of specification and implementation. Multiple independent authors can produce implementations that are compatible with an interface specification. Similarly, a delegate specifies the signature of a method, and authors can write methods that are compatible with the delegate specification. When should you use interfaces, and when should you use delegates?

Delegates are useful when:

* A single method is being called.
* A class may want to have multiple implementations of the method specification.
* It is desirable to allow using a static method to implement the specification.
* An event-like design pattern is desired (for more information, see the [Events Tutorial](https://msdn.microsoft.com/en-us/library/aa645739(v=vs.71).aspx)).
* The caller has no need to know or obtain the object that the method is defined on.
* The provider of the implementation wants to "hand out" the implementation of the specification to only a few select components.
* Easy composition is desired.

Interfaces are useful when:

* The specification defines a set of related methods that will be called.
* A class typically implements the specification only once.
* The caller of the interface wants to cast to or from the interface type to obtain other interfaces or classes.

**Events Tutorial**

**Visual Studio .NET 2003**

This tutorial shows how to declare, invoke, and hook up to events in C#.

**Sample Files**

See [Events Sample](https://msdn.microsoft.com/en-us/library/aa645506(v=vs.71).aspx) to download and build the sample files discussed in this tutorial.

**Further Reading**

* [event](https://msdn.microsoft.com/en-us/library/8627sbea(v=vs.71).aspx)
* [delegate](https://msdn.microsoft.com/en-us/library/900fyy8e(v=vs.71).aspx)
* [Handling and Raising Events](https://msdn.microsoft.com/en-us/library/aa720046(v=vs.71).aspx)
* [Delegates Tutorial](https://msdn.microsoft.com/en-us/library/aa288459(v=vs.71).aspx)
* [Introduction to Event Handlers in Windows Forms](https://msdn.microsoft.com/en-us/library/aa983763(v=vs.71).aspx)

**Tutorial**

An *event* in C# is a way for a class to provide notifications to clients of that class when some interesting thing happens to an object. The most familiar use for events is in graphical user interfaces; typically, the classes that represent controls in the interface have events that are notified when the user does something to the control (for example, click a button).

Events, however, need not be used only for graphical interfaces. Events provide a generally useful way for objects to signal state changes that may be useful to clients of that object. Events are an important building block for creating classes that can be reused in a large number of different programs.

Events are declared using delegates. If you have not yet studied the Delegates Tutorial, you should do so before continuing. Recall that a delegate object encapsulates a method so that it can be called anonymously. An event is a way for a class to allow clients to give it delegates to methods that should be called when the event occurs. When the event occurs, the delegate(s) given to it by its clients are invoked.

In addition to the examples on declaring, invoking, and hooking up to events, this tutorial also introduces the following topics:

* Events and Inheritance
* Events in Interfaces
* .NET Framework Guidelines

**Example 1**

The following simple example shows a class, ListWithChangedEvent, which is similar to the standard ArrayList class, but also invokes a Changed event whenever the contents of the list change. Such a general-purpose class could be used in numerous ways in a large program.

For example, a word processor might maintain a list of the open documents. Whenever this list changes, many different objects in the word processor might need to be notified so that the user interface could be updated. By using events, the code that maintains the list of documents doesn't need to know who needs to be notified — once the list of documents is changed, the event is automatically invoked and every object that needs to be notified is correctly notified. By using events, the modularity of the program is increased.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_e88db0cf-d7c1-48c4-beac-d973691c33f9');)

// events1.cs

using System;

namespace MyCollections

{

using System.Collections;

// A delegate type for hooking up change notifications.

public delegate void ChangedEventHandler(object sender, EventArgs e);

// A class that works just like ArrayList, but sends event

// notifications whenever the list changes.

public class ListWithChangedEvent: ArrayList

{

// An event that clients can use to be notified whenever the

// elements of the list change.

public event ChangedEventHandler Changed;

// Invoke the Changed event; called whenever list changes

protected virtual void OnChanged(EventArgs e)

{

if (Changed != null)

Changed(this, e);

}

// Override some of the methods that can change the list;

// invoke event after each

public override int Add(object value)

{

int i = base.Add(value);

OnChanged(EventArgs.Empty);

return i;

}

public override void Clear()

{

base.Clear();

OnChanged(EventArgs.Empty);

}

public override object this[int index]

{

set

{

base[index] = value;

OnChanged(EventArgs.Empty);

}

}

}

}

namespace TestEvents

{

using MyCollections;

class EventListener

{

private ListWithChangedEvent List;

public EventListener(ListWithChangedEvent list)

{

List = list;

// Add "ListChanged" to the Changed event on "List".

List.Changed += new ChangedEventHandler(ListChanged);

}

// This will be called whenever the list changes.

private void ListChanged(object sender, EventArgs e)

{

Console.WriteLine("This is called when the event fires.");

}

public void Detach()

{

// Detach the event and delete the list

List.Changed -= new ChangedEventHandler(ListChanged);

List = null;

}

}

class Test

{

// Test the ListWithChangedEvent class.

public static void Main()

{

// Create a new list.

ListWithChangedEvent list = new ListWithChangedEvent();

// Create a class that listens to the list's change event.

EventListener listener = new EventListener(list);

// Add and remove items from the list.

list.Add("item 1");

list.Clear();

listener.Detach();

}

}

}

**Output**

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_4e8fbb16-3e6d-4967-a69b-510c32f696a6');)

This is called when the event fires.

This is called when the event fires.

**Code Discussion**

* **Declaring an event**To declare an event inside a class, first a delegate type for the event must be declared, if none is already declared.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_5fa3d270-b143-4103-955d-72f3ef693a90');)

public delegate void ChangedEventHandler(object sender, EventArgs e);

The delegate type defines the set of arguments that are passed to the method that handles the event. Multiple events can share the same delegate type, so this step is only necessary if no suitable delegate type has already been declared.

Next, the event itself is declared.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_eedc95cb-f96e-4fcc-b726-6d8c5f7af405');)

public event ChangedEventHandler Changed;

An event is declared like a field of delegate type, except that the keyword **event** precedes the event declaration, following the modifiers. Events usually are declared public, but any accessibility modifier is allowed.

* **Invoking an event**Once a class has declared an event, it can treat that event just like a field of the indicated delegate type. The field will either be null, if no client has hooked up a delegate to the event, or else it refers to a delegate that should be called when the event is invoked. Thus, invoking an event is generally done by first checking for null and then calling the event.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_4eeceacc-2d0b-4364-a5b1-8f0283237f18');)

if (Changed != null)

Changed(this, e);

Invoking an event can only be done from within the class that declared the event.

* **Hooking up to an event**From outside the class that declared it, an event looks like a field, but access to that field is very restricted. The only things that can be done are:
  + Compose a new delegate onto that field.
  + Remove a delegate from a (possibly composite) field.

This is done with the **+=** and **-=** operators. To begin receiving event invocations, client code first creates a delegate of the event type that refers to the method that should be invoked from the event. Then it composes that delegate onto any other delegates that the event might be connected to using **+=**.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_dac931ee-5c7c-466b-ad91-01e42604ce3e');)

// Add "ListChanged" to the Changed event on "List":

List.Changed += new ChangedEventHandler(ListChanged);

When the client code is done receiving event invocations, it removes its delegate from the event by using operator **-=**.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_55a5782f-c729-4a01-a418-779a6d70bc5f');)

// Detach the event and delete the list:

List.Changed -= new ChangedEventHandler(ListChanged);

**Events and Inheritance**

When creating a general component that can be derived from, what seems to be a problem sometimes arises with events. Since events can only be invoked from within the class that declared them, derived classes cannot directly invoke events declared within the base class. Although this is sometimes what is desired, often it is appropriate to give the derived class the freedom to invoke the event. This is typically done by creating a protected invoking method for the event. By calling this invoking method, derived classes can invoke the event. For even more flexibility, the invoking method is often declared as virtual, which allows the derived class to override it. This allows the derived class to intercept the events that the base class is invoking, possibly doing its own processing of them.

In the preceding example, this has been done with the OnChanged method. A derived class could call or override this method if it needed to.

**Events in Interfaces**

One other difference between events and fields is that an event can be placed in an interface while a field cannot. When implementing the interface, the implementing class must supply a corresponding event in the class that implements the interface.

**.NET Framework Guidelines**

Although the C# language allows events to use any delegate type, the .NET Framework has some stricter guidelines on the delegate types that should be used for events. If you intend for your component to be used with the .NET Framework, you probably will want to follow these guidelines.

The .NET Framework guidelines indicate that the delegate type used for an event should take two parameters, an "object source" parameter indicating the source of the event, and an "e" parameter that encapsulates any additional information about the event. The type of the "e" parameter should derive from the **EventArgs** class. For events that do not use any additional information, the .NET Framework has already defined an appropriate delegate type: **EventHandler**.

**Example 2**

The following example is a modified version of Example 1 that follows the .NET Framework guidelines. The example uses the **EventHandler** delegate type.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_4cdcf055-8540-4331-aa63-1baa8c4a2054');)

// events2.cs

using System;

namespace MyCollections

{

using System.Collections;

// A class that works just like ArrayList, but sends event

// notifications whenever the list changes:

public class ListWithChangedEvent: ArrayList

{

// An event that clients can use to be notified whenever the

// elements of the list change:

public event EventHandler Changed;

// Invoke the Changed event; called whenever list changes:

protected virtual void OnChanged(EventArgs e)

{

if (Changed != null)

Changed(this,e);

}

// Override some of the methods that can change the list;

// invoke event after each:

public override int Add(object value)

{

int i = base.Add(value);

OnChanged(EventArgs.Empty);

return i;

}

public override void Clear()

{

base.Clear();

OnChanged(EventArgs.Empty);

}

public override object this[int index]

{

set

{

base[index] = value;

OnChanged(EventArgs.Empty);

}

}

}

}

namespace TestEvents

{

using MyCollections;

class EventListener

{

private ListWithChangedEvent List;

public EventListener(ListWithChangedEvent list)

{

List = list;

// Add "ListChanged" to the Changed event on "List":

List.Changed += new EventHandler(ListChanged);

}

// This will be called whenever the list changes:

private void ListChanged(object sender, EventArgs e)

{

Console.WriteLine("This is called when the event fires.");

}

public void Detach()

{

// Detach the event and delete the list:

List.Changed -= new EventHandler(ListChanged);

List = null;

}

}

class Test

{

// Test the ListWithChangedEvent class:

public static void Main()

{

// Create a new list:

ListWithChangedEvent list = new ListWithChangedEvent();

// Create a class that listens to the list's change event:

EventListener listener = new EventListener(list);

// Add and remove items from the list:

list.Add("item 1");

list.Clear();

listener.Detach();

}

}

}

**Output**

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_be7d0d94-0bf1-46d6-9a8a-7c35cf2e4c57');)

This is called when the event fires.

This is called when the event fires.

**See Also**

[C# Tutorials](https://msdn.microsoft.com/en-us/library/aa288436(v=vs.71).aspx)

# 

# Handling and Raising Events

Events in the .NET Framework are based on the delegate model. (Those familiar with design patterns in object-oriented programming will notice the observer design pattern.) This section contains topics that describe the delegate model, show how to consume events in applications, and describe how to raise events from your class.

For details about the event syntax in a specific programming language, see the documentation for that language.

#### In This Section

[Events and Delegates](https://msdn.microsoft.com/en-us/library/aa720047(v=vs.71).aspx)

Provides an overview of the event model and describes delegates in the .NET Framework.

[Consuming Events](https://msdn.microsoft.com/en-us/library/aa720052(v=vs.71).aspx)

Provides an overview of how .NET Framework applications consume events.

[Raising an Event](https://msdn.microsoft.com/en-us/library/aa720442(v=vs.71).aspx)

Shows how to provide event functionality in your class.

[Event Sample](https://msdn.microsoft.com/en-us/library/aa720051(v=vs.71).aspx)

Contains a detailed sample that walks through the steps for raising an event from a class.

[Raising Multiple Events](https://msdn.microsoft.com/en-us/library/aa720454(v=vs.71).aspx)

Shows a technique for optimizing storage for multiple events.

#### Related Sections

Web Server Control Event Model

Describes the details of the event model for ASP.NET Web Forms.

[Events in ASP.NET Server Controls](https://msdn.microsoft.com/en-us/library/cc425426(v=vs.71).aspx)

Describes how to raise events from custom ASP.NET server controls.

[Events in Windows Forms Controls](https://msdn.microsoft.com/en-us/library/aa720050(v=vs.71).aspx)

Describes how to raise events from custom Windows Forms controls.

[Managed and Unmanaged Events](https://msdn.microsoft.com/en-us/library/aa720259(v=vs.71).aspx)

Describes how managed events in the .NET Framework interoperate with unmanaged COM events.

**Events and Delegates**

An event is a message sent by an object to signal the occurrence of an action. The action could be caused by user interaction, such as a mouse click, or it could be triggered by some other program logic. The object that raises (triggers) the event is called the event sender. The object that captures the event and responds to it is called the event receiver.

In event communication, the event sender class does not know which object or method will receive (handle) the events it raises. What is needed is an intermediary (or pointer-like mechanism) between the source and the receiver. The .NET Framework defines a special type ([Delegate](https://msdn.microsoft.com/en-us/library/system.delegate(v=vs.71).aspx)) that provides the functionality of a function pointer.

A delegate is a class that can hold a reference to a method. Unlike other classes, a delegate class has a signature, and it can hold references only to methods that match its signature. A delegate is thus equivalent to a type-safe function pointer or a callback. While delegates have other uses, the discussion here focuses on the event handling functionality of delegates. The following example shows an event delegate declaration.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_73143a2f-9401-443b-9545-c863ad19a7a9');)

// AlarmEventHandler is the delegate for the Alarm event.

// AlarmEventArgs is the class that holds event data for the alarm event.

// It derives from the base class for event data, EventArgs.

public delegate void AlarmEventHandler(object sender, AlarmEventArgs e);

[Visual Basic]' AlarmEventHandler is the delegate for the Alarm event.

' AlarmEventArgs is the class that holds event data for the alarm event.

' It derives from the base class for event data, EventArgs.

Public Delegate Sub AlarmEventHandler(sender As Object, e As AlarmEventArgs)

The syntax is similar to that of a method declaration; however, the **delegate** keyword informs the compiler that AlarmEventHandler is a delegate type.

By convention, event delegates in the .NET Framework have two parameters, the source that raised the event and the data for the event.

**Note**   A delegate declaration is sufficient to define a delegate class. The declaration supplies the signature of the delegate, and the common language runtime provides the implementation.

An instance of the AlarmEventHandler delegate can bind to any method that matches its signature, such as the AlarmRang method of the WakeMeUp class shown in the following example.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_6a4d9c8c-36e7-42a5-b439-016094263da6');)

public class WakeMeUp

{

// AlarmRang has the same signature as AlarmEventHandler.

public void AlarmRang(object sender, AlarmEventArgs e){...};

...

}

[Visual Basic]Public Class WakeMeUp

' AlarmRang has the same signature as AlarmEventHandler.

Public Sub AlarmRang(sender As Object, e As AlarmEventArgs)

...

End Sub

...

End Class

**To connect (wire) AlarmRang to an Alarm event:**

1. Create an instance of the AlarmEventHandler delegate that takes a reference to the AlarmRang method of the WakeMeUp instance in its constructor, as shown in the following example.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_696faec7-cee9-4eda-a605-b0316c5e2d7a');)

// Create an instance of WakeMeUp.

//

WakeMeUp w = new WakeMeUp();

// Instantiate the event delegate.

// The C# compiler provides a constructor for event handlers that takes

// one parameter, the reference to the method that performs the

// event handling logic. The two-parameter constructor for EventHandler

// provided in the class library is intended for developers of

// compilers and other tools.

//

AlarmEventHandler alhandler = new AlarmEventHandler(w.AlarmRang);

[Visual Basic]' Create an instance of WakeMeUp.

'

Dim w As New WakeMeUp()

' Instantiate the event delegate.

Dim alhandler As AlarmEventHandler = AddressOf w.AlarmRang

Now, whenever alhandler is called, it in turn calls the AlarmRang method of the WakeMeUp instance.

1. Register the alhandler delegate with the Alarm event. For details and a complete sample, see [Event Sample](https://msdn.microsoft.com/en-us/library/aa720051(v=vs.71).aspx).

Custom event delegates are needed only when an event generates event data. Many events, including some user-interface events such as mouse clicks, do not generate event data. In such situations, the event delegate provided in the class library for the no-data event, [System.EventHandler](https://msdn.microsoft.com/en-us/library/system.eventhandler(v=vs.71).aspx), is adequate. Its declaration follows.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_cb290680-5f65-491f-87b2-83ab62ae4776');)

// The base class for event data, EventArgs, does not have

// any data and hence can be used as the event data type for events

// that do not generate data.

//

delegate void EventHandler(object sender, EventArgs e);

[Visual Basic]' The base class for event data, EventArgs, does not have

' any data and hence can be used as the event data type for events

' that do not generate data.

'

Public Delegate Sub AlarmEventHandler(sender As Object, e As AlarmEventArgs)

Event delegates are multicast, which means that they can hold references to more than one event handling method. For details, see [Delegate](https://msdn.microsoft.com/en-us/library/system.delegate(v=vs.71).aspx). Delegates allow for flexibility and fine-grain control in event handling. A delegate acts as an event dispatcher for the class that raises the event by maintaining a list of registered event handlers for the event.

For details on using delegates to provide event functionality in your component or control, see [Raising an Event](https://msdn.microsoft.com/en-us/library/aa720442(v=vs.71).aspx).

For an overview of consuming events in your applications, see [Consuming Events](https://msdn.microsoft.com/en-us/library/aa720052(v=vs.71).aspx).

**See Also**

[Consuming Events](https://msdn.microsoft.com/en-us/library/aa720052(v=vs.71).aspx) | [Raising an Event](https://msdn.microsoft.com/en-us/library/aa720442(v=vs.71).aspx) | [Event Sample](https://msdn.microsoft.com/en-us/library/aa720051(v=vs.71).aspx)

**Consuming Events**

To consume an event in an application, you must provide an event handler (an event-handling method) that executes program logic in response to the event and register the event handler with the event source. This process is referred to as event wiring. For an overview of the event model in the .NET Framework, see [Events and Delegates](https://msdn.microsoft.com/en-us/library/aa720047(v=vs.71).aspx). The visual designers for Windows Forms and for Web Forms provide rapid application development (RAD) tools that simplify or hide the details of event wiring.

This topic walks you through a simple example that illustrates the details of event wiring. Suppose you want to create a user interface that has a button and a text box, and you want the background color of the text box to change when the user clicks the button. The following examples show how to program such an event-based user interface in Web Forms and in Windows Forms. For background about the Web Forms programming model, see Programming Web Forms. For background about Windows Forms, see Introduction to Windows Forms.

**Consuming an Event in a Web Forms Application**

Suppose you have created a simple Web Forms page (ASP.NET page) that has a **Button** control and a **TextBox** control, as shown in the following example.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_4e097fcf-d77f-4bbe-a548-b5ee4a73bd3f');)

<asp:TextBox id = "Box" Text = "Hello" BackColor = "Cyan" runat=server/>

<asp:Button id = "Button" Text = "Click Me" runat = server/>

If you want your page to handle a button click, you must know the type of the **Click** event. The **Click** event of [System.Web.UI.WebControls.Button](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button(v=vs.71).aspx)control is declared as follows.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_c7b8013c-3d1c-4fda-a15b-509643a83817');)

public event EventHandler Click;

[Visual Basic]Public Event Click As EventHandler

The declaration shows that the type for the **Click** event is [EventHandler](https://msdn.microsoft.com/en-us/library/system.eventhandler(v=vs.71).aspx), which is a delegate type. To handle the **Click** event, you must supply an event handler (event-handling method) that has the signature of **EventHandler**.The following code shows the signature of **EventHandler**.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_e7bd9e97-7d3d-448f-a1c2-f0ee7a235808');)

public delegate void EventHandler(object sender, EventArgs e);

[Visual Basic]Public Delegate Sub EventHandler(ByVal sender As Object, ByVal e As EventArgs)

By convention, event delegates in the .NET Framework have two parameters, the source that raised the event and the data for the event. The event data class derives from **System.EventArgs**. If the event does not generate data, it uses **EventArgs** as the event data type.

Your event handler for the **Click** event must have the following signature.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_4b95216a-3b7f-4ea6-878f-bbd8fa74db05');)

void Button\_Clicked((object sender, EventArgs e){}

[Visual Basic]Sub Button\_Clicked(sender As Object, e As EventArgs)

**Note**   An event delegate in the .NET Framework is named *EventName***EventHandler**, while the term event handler in the documentation refers to an event-handling method. The logic behind the naming scheme is that an *EventName***EventHandler** delegate points to the event handler (the method) that actually handles the event.

To wire the Button\_Clicked method to the **Click** event of Button, add the syntax shown in bold in the following code.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_b7c7b210-34c0-48cd-92c1-121b7f3d8ccf');)

<asp:Button

id = "Button" OnClick = "Button\_Clicked" Text = "Click Me"

runat = server/>

The following Web Forms page handles the **Click** event of **Button** to change the background color of **TextBox**. The elements in bold in this example show the event handler code and how the event handler is wired to the **Click** event of **Button**.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_ff383dcd-b4c8-47f6-a092-f67aa03f1aba');)

<html>

<script language="C#" runat=server>

private void Button\_Clicked(object sender, EventArgs e){

Box.BackColor = System.Drawing.Color.LightGreen;

}

</script>

<body>

<form method="POST" action="Events.aspx" runat=server>

Click the button, and notice the color of the text box.<br><br>

<asp:TextBox

id = "Box" Text = "Hello" BackColor = "Cyan" runat=server/>

<br><br>

<asp:Button

id = "Button" OnClick = "Button\_Clicked" Text = "Click Me"

runat = server/>

</form>

</body>

</html>

[Visual Basic]<html>

<script language="VB" runat=server>

Private Sub Button\_Clicked(sender As Object, e As EventArgs)

Box.BackColor = System.Drawing.Color.LightGreen

End Sub

</script>

<body>

<form method="POST" action="Events.aspx" runat=server>

Click the button, and notice the color of the text box.<br><br>

<asp:TextBox

id = "Box" Text = "Hello" BackColor = "Cyan" runat=server/>

<br><br>

<asp:Button

id = "Button" OnClick = "Button\_Clicked" Text = "Click Me"

runat = server/>

</form>

</body>

</html>

To see how event handling works in Web Forms, save the page above to a file with an .aspx extension (which indicates that the file is an ASP.NET page) and deploy it anywhere in your IIS virtual root directory.

The following list summarizes the essential steps in the sample.

* The source of the event is an instance of the **System.Web.UI.WebControls.Button** server control.
* The button raises a **Click** event.
* The delegate for the **Click** event is **EventHandler**.
* The page has an event handler called Button\_Clicked.
* Button\_Clicked is wired to the **Click** event using the following page syntax: OnClick = "Button\_Clicked".

**Note**A Web Forms application developer can wire the event declaratively as shown in the last bullet item without directly working with the delegate. The ASP.NET page framework generates code that creates an instance of **EventHandler** that references Button\_Clicked and adds this delegate instance to the **Click** event of the **Button** instance.

**Consuming an Event in a Windows Forms Application**

Suppose you have created a simple form that contains a button and a text box control, as shown in the following code fragment.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_16f71ee5-4a86-40a8-b57a-1dc236dd33b9');)

private TextBox box;

private Button button;

[Visual Basic]Private box As TextBox

Private myButton As Button

[System.Windows.Forms.Button](https://msdn.microsoft.com/en-us/library/system.windows.forms.button(v=vs.71).aspx)has a **Click** event whose event delegate is [EventHandler](https://msdn.microsoft.com/en-us/library/system.eventhandler(v=vs.71).aspx). To handle the **Click** event, your event handler must have the signature of **EventHandler**. The following example shows an event handler, Button\_Clicked, that has the same signature as **EventHandler**.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_21e707e6-31af-41ea-805a-d63f226962cd');)

void Button\_Clicked(object sender, EventArgs e){}

[Visual Basic]Sub Button\_Clicked(sender As Object, e As EventArgs)

**Note**   An event delegate in the .NET Framework is named *EventName***EventHandler**, while the term event handler in the documentation refers to an event-handling method. The logic behind the naming scheme is that an *EventName***EventHandler** delegate points to the event handler (the method) that actually handles the event.

To wire your event handler to the **Button**, you must create an instance of **EventHandler** that takes a reference to Button\_Clicked in its argument and add this delegate instance to the **Click** event, as shown in the following example.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_258262e0-a849-4bfa-868e-59bce4764af3');)

button.Click += new EventHandler(this.Button\_Clicked);

[Visual Basic]AddHandler button.Click, AddressOf Me.Button\_Clicked

**Note**   A designer such as Visual Studio .NET will do this event wiring for you by generating code that is similar to that in the example.

The following sample shows a simple Windows Forms application that handles the **Click** event of **Button** to change the background color of **TextBox**. The elements in bold show the event handler and how it is wired to the **Click** event of the **Button**.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_ab4acb04-f639-4e36-9760-ea7f0ab9a7c0');)

using System;

using System.ComponentModel;

using System.Windows.Forms;

using System.Drawing;

public class MyForm : Form

{

private TextBox box;

private Button button;

public MyForm() : base()

{

box = new TextBox();

box.BackColor = System.Drawing.Color.Cyan;

box.Size = new Size(100,100);

box.Location = new Point(50,50);

box.Text = "Hello";

button = new Button();

button.Location = new Point(50,100);

button.Text = "Click Me";

// To wire the event, create

// a delegate instance and add it to the Click event.

button.Click += new EventHandler(this.Button\_Clicked);

Controls.Add(box);

Controls.Add(button);

}

// The event handler.

private void Button\_Clicked(object sender, EventArgs e)

{

box.BackColor = System.Drawing.Color.Green;

}

// STAThreadAttribute indicates that Windows Forms uses the

// single-threaded apartment model.

[STAThreadAttribute]

public static void Main(string[] args)

{

Application.Run(new MyForm());

}

}

[Visual Basic]Option Explicit

Option Strict

Imports System

Imports System.ComponentModel

Imports System.Windows.Forms

Imports System.Drawing

Public Class MyForm

Inherits Form

Private box As TextBox

Private WithEvents button As Button

Public Sub New()

box = New TextBox()

box.BackColor = System.Drawing.Color.Cyan

box.Size = New Size(100, 100)

box.Location = New Point(50, 50)

box.Text = "Hello"

button = New Button()

button.Location = New Point(50, 100)

button.Text = "Click Me"

Controls.Add(box)

Controls.Add(button)

End Sub

' The event handler.

Private Sub Button\_Clicked(sender As Object, e As EventArgs) Handles button.Click

box.BackColor = System.Drawing.Color.Green

End Sub

' STAThreadAttribute indicates that Windows Forms uses the

' single-threaded apartment model.

<STAThreadAttribute()> \_

Public Shared Sub Main(args() As String)

Application.Run(New MyForm())

End Sub

End Class

To see how event handling works in Windows Forms, save the preceding code to a file (with a .cs extension for a C# file and .vb for Visual Basic .NET), compile, and execute. For example, if the source file is named WinEvents.cs (or WinEvents.vb), execute the following command.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_35b77c21-28f4-4eb7-88ff-6d8612ce41e0');)

csc /r:System.DLL /r:System.Windows.Forms.DLL /r:System.Drawing.DLL WinEvents.cs

[Visual Basic]csc /r:System.DLL /r:System.Windows.Forms.DLL /r:System.Drawing.DLL WinEvents.vb

Your executable file will be named WinEvents.exe.

The following list summarizes the essential steps in the sample.

* The source of the event is an instance of the **System.Windows.Forms.Button** control.
* The button raises a **Click** event.
* The delegate for the **Click** event is **EventHandler**.
* The form has an event handler called Button\_Clicked.
* Button\_Clicked is wired to the **Click** event.

The code in this sample was written without using a visual designer such as Visual Studio .NET and contains only essential programming elements. If you use a designer, it will generate additional code.

**The Event Pattern**

While the details of event wiring differ in Windows Forms and Web Forms because of the different levels of support provided by different RAD tools, both scenarios follow the same event pattern, which has the following characteristics.

* A class that raises an event named *EventName* has the following member.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_27846e99-57e9-4282-b527-8d40eadfe374');)

public event EventNameEventHandler EventName;

[Visual Basic]Public Event EventName As EventNameEventHandler

* The event delegate for the *EventName* event is *EventName*EventHandler, with the following signature.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_1b733b07-7afe-4730-94f3-100a346557f1');)

public delegate void EventNameEventHandler(object sender, EventNameEventArgs e);

[Visual Basic]Public Delegate Sub EventNameEventHandler(sender As Object, e As EventNameEventArgs)

To consume the *EventName* event, your event handler must have the same signature as the event delegate.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_49dd9c0c-02dc-4b15-abc3-9c4c09a7e6cf');)

void EventHandler (object sender, EventNameEventArgs e) {}

[Visual Basic]Sub EventHandler(sender As Object, e As EventNameEventArgs)

The **Click** event in the examples does not have any associated data. It uses the **EventArgs** class for event data and **EventHandler** as the delegate. Events that do have associated data use classes that derive from **EventArgs** from the event data type, and the corresponding event delegate type. For example, if you want to handle a **MouseUp** eventin a Windows Forms application, the event data class is **MouseEventArgs** and the event delegate is **MouseEventHandler**. Note that several mouse events use a common class for event data and a common event delegate, so the naming scheme does not exactly match the convention described above. Your event handler must have the following signature.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_4f2fff3a-daca-4218-8d42-7856aae9a6be');)

void Mouse\_Moved(object sender, MouseEventArgs e){}

[Visual Basic]Sub Mouse\_Moved(sender As Object, e As MouseEventArgs)

In the simple examples in this topic, the logic provided by the event handlers does not depend on the sender or on the event data. In general, the event handler provides logic that uses information provided by event source object and by the event data object. For an example, see [Event Sample](https://msdn.microsoft.com/en-us/library/aa720051(v=vs.71).aspx).

**Note**   Events also arise outside the context of user interfaces (UIs), and, in fact, the .NET Framework includes many non-UI classes that raise events. However, all events have the pattern described here.

For information on raising events from a class, see [Raising an Event](https://msdn.microsoft.com/en-us/library/aa720442(v=vs.71).aspx).

**See Also**

[Events and Delegates](https://msdn.microsoft.com/en-us/library/aa720047(v=vs.71).aspx) | [Raising an Event](https://msdn.microsoft.com/en-us/library/aa720442(v=vs.71).aspx)

**Raising an Event**

Event functionality is provided by three interrelated elements: a class that provides event data, an event delegate, and the class that raises the event. The .NET Framework has a convention for naming classes and methods related to events. If you want your class to raise an event named *EventName,* you need the following elements.

* A class that holds event data, named *EventName*EventArgs. This class must derive from [System.EventArgs](https://msdn.microsoft.com/en-us/library/system.eventargs(v=vs.71).aspx).
* A delegate for the event, named *EventName*EventHandler.
* A class that raises the event. This class must provide:
  1. An event declaration.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_780a979b-e16a-4dae-b092-9d4d79ebb4e6');)

public event EventNameEventHandler EventName;

[Visual Basic]Public Event EventName As EventNameEventHandler

* 1. A method named On*EventName* that raises the event.

The event data class and the event delegate class might already be defined in the .NET Framework class library or in a third-party class library. In that case, you do not have to define those classes.

If you are not familiar with the delegate model for events in the .NET Framework, see [Events and Delegates](https://msdn.microsoft.com/en-us/library/aa720047(v=vs.71).aspx).

**To provide event functionality**

1. Define a class that provides data for the event. This class must derive from [System.EventArgs](https://msdn.microsoft.com/en-us/library/system.eventargs(v=vs.71).aspx), which is the base class for event data. An example follows.

**Note**This step is not needed if an event data class already exists for the event or if there is no data associated with your event. If there is no event data, use the base class **System.EventArgs**.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_c3220ca8-7cec-4e01-b596-680996e2363f');)

public class AlarmEventArgs : EventArgs {

private readonly int nrings = 0;

private readonly bool snoozePressed = false;

//Properties.

public string AlarmText {

...

}

public int NumRings {

...

}

public bool SnoozePressed{

...

}

...

}

[Visual Basic]Public Class AlarmEventArgs

Inherits EventArgs

Private nrings As Integer = 0

Private \_snoozePressed As Boolean = False

'Properties.

Public ReadOnly Property AlarmText() As String

...

End Property

Public ReadOnly Property NumRings() As Integer

...

End Property

Public ReadOnly Property SnoozePressed() As Boolean

...

End Property

...

End Class

1. Declare a delegate for the event, as in the following example.

**Note**You do not have to declare a custom delegate if the event does not generate data. In that case, use the base event handler [System.ComponentModel.EventHandler](https://msdn.microsoft.com/en-us/library/system.eventhandler(v=vs.71).aspx).

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_c98e2295-605e-49d4-a696-987589ff8b57');)

public delegate void AlarmEventHandler(object sender, AlarmEventArgs e);

[Visual Basic]Public Delegate Sub AlarmEventHandler(sender As Object, e As AlarmEventArgs)

1. Define a public event member in your class using the **event** keyword whose type is an event delegate, as in the following example.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_00f26499-54b2-4dcd-985c-146135de0408');)

public class AlarmClock

{

...

public event AlarmEventHandler Alarm;

}

[Visual Basic]Public Class AlarmClock

...

Public Event Alarm As AlarmEventHandler

End Class

In the AlarmClock class the Alarm event is a delegate of type AlarmEventHandler. When the compiler encounters an **event** keyword, it creates a private member such as

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_ad3d5969-7d83-4948-a884-d38f03448b1f');)

private AlarmEventHandler al = null;

and the two public methods add\_Alarm and remove\_Alarm. These methods are event hooks that allow delegates to be combined or removed from the event delegate al. The details are hidden from the programmer.

**Note**In languages other than C# and Visual Basic .NET, the compiler might not automatically generate the code corresponding to an event member, and you might have to explicitly define the event hooks and the private delegate field.

1. Provide a protected method in your class that raises the event. This method must be named **On***EventName.* The**On***EventName* methodraises the event by invoking the delegates. The code example at the end of this topic shows an implementation of **On***EventName*.

**Note**   The protected **On***EventName* method also allows derived classes to override the event without attaching a delegate to it. A derived class must always call the **On***EventName* method of the base class to ensure that registered delegates receive the event.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_a839540e-0399-4b85-bc87-74f3b6e789ed');)

public class AlarmClock

{

...

public event AlarmHandler Alarm;

protected virtual void OnAlarm(AlarmEvent e){...}

}

[Visual Basic]Public Class AlarmClock

...

Public Event Alarm As AlarmEventHandler

Protected Overridable Sub OnAlarm(e As AlarmEventArgs)

...

End Sub

End Class

The following code fragment puts together all of the elements discussed in this section. For a complete sample that implements and uses events, see [Event Sample](https://msdn.microsoft.com/en-us/library/aa720051(v=vs.71).aspx).

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_d976e494-ba12-458b-af18-d608a09f5729');)

//Step 1. Class that defines data for the event

//

public class AlarmEventArgs : EventArgs

{

private readonly bool snoozePressed = false;

private readonly int nrings = 0;

// Constructor.

public AlarmEventArgs(bool snoozePressed, int nrings) {...}

// Properties.

public int NumRings{ get { return nrings;}}

public bool SnoozePressed { get { return snoozePressed;}}

public string AlarmText { get {...}}

}

//Step 2. Delegate declaration.

//

public delegate void AlarmEventHandler(object sender, AlarmEventArgs e);

// Class definition.

//

public class AlarmClock

{

//Step 3. The Alarm event is defined using the event keyword.

//The type of Alarm is AlarmEventHandler.

public event AlarmEventHandler Alarm;

//

//Step 4. The protected OnAlarm method raises the event by invoking

//the delegates. The sender is always this, the current instance of

//the class.

//

protected virtual void OnAlarm(AlarmEventArgs e)

{

if (Alarm != null)

{

//Invokes the delegates.

Alarm(this, e);

}

}

}

[Visual Basic]'Step 1. Class that defines data for the event

'

Public Class AlarmEventArgs

Inherits EventArgs

Private \_snoozePressed As Boolean = False

Private nrings As Integer = 0

' Constructor.

Public Sub New(snoozePressed As Boolean, nrings As Integer)

...

End Sub

' Properties.

Public ReadOnly Property NumRings() As Integer

Get

Return nrings

End Get

End Property

Public ReadOnly Property SnoozePressed() As Boolean

Get

Return \_snoozePressed

End Get

End Property

Public ReadOnly Property AlarmText() As String

Get

...

End Get

End Property

End Class

'Step 2. Delegate declaration.

'

Public Delegate Sub AlarmEventHandler(sender As Object, e As AlarmEventArgs)

' Class definition.

'

Public Class AlarmClock

'Step 3. The Alarm event is defined using the event keyword.

'The type of Alarm is AlarmEventHandler.

Public Event Alarm As AlarmEventHandler

'

'Step 4. The protected OnAlarm method raises the event by invoking

'the delegates. The sender is always this, the current instance of

'the class.

'

Protected Overridable Sub OnAlarm(e As AlarmEventArgs)

'Invokes the delegates.

RaiseEvent Alarm(Me, e)

End Sub

End Class

**See Also**

[Events and Delegates](https://msdn.microsoft.com/en-us/library/aa720047(v=vs.71).aspx) | [Event Sample](https://msdn.microsoft.com/en-us/library/aa720051(v=vs.71).aspx)

**Event Sample**

The sample presented here builds on the various pieces discussed in [Events and Delegates](https://msdn.microsoft.com/en-us/library/aa720047(v=vs.71).aspx) and [Raising an Event](https://msdn.microsoft.com/en-us/library/aa720442(v=vs.71).aspx).

This sample shows how to raise an event from your class and also how to handle the event. It defines the following classes.

* AlarmClock is the class that raises the Alarm event.
* AlarmEventArgs defines the data for the Alarm event.
* AlarmEventHandler is the delegate for the alarm event.
* WakeMeUp is a class that has a method, AlarmRang, that handles the alarm event.
* AlarmDriver is a class that demonstrates how events are wired. It instantiates AlarmClock and WakeMeUp. It then instantiates the AlarmEventHandler delegate with a reference to the AlarmRang method of the WakeMeUp instance. AlarmDriver completes the event wiring by registering the delegate with the instance of AlarmClock, using the += syntax for adding delegates to an event.

**To compile and run this sample**

1. Copy the code below to a source file (such as EventSample.cs.or EventSample.vb).
2. From the directory that contains source file, execute the following command:

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_f1b5878c-2e0b-4847-a826-a80acfb1c9e2');)

csc /r:System.dll EventSample.cs[Visual Basic]vbc /r:System.dll EventSample.vb

(The /r option references the System assembly.)

1. Run the executable that is created, EventSample.exe.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_501f9d8b-2623-421c-bd09-a23efcd64bb0');)

// EventSample.cs.

//

namespace EventSample

{

using System;

using System.ComponentModel;

// Class that contains the data for

// the alarm event. Derives from System.EventArgs.

//

public class AlarmEventArgs : EventArgs

{

private readonly bool snoozePressed ;

private readonly int nrings;

//Constructor.

//

public AlarmEventArgs(bool snoozePressed, int nrings)

{

this.snoozePressed = snoozePressed;

this.nrings = nrings;

}

// The NumRings property returns the number of rings

// that the alarm clock has sounded when the alarm event

// is generated.

//

public int NumRings

{

get { return nrings;}

}

// The SnoozePressed property indicates whether the snooze

// button is pressed on the alarm when the alarm event is generated.

//

public bool SnoozePressed

{

get {return snoozePressed;}

}

// The AlarmText property that contains the wake-up message.

//

public string AlarmText

{

get

{

if (snoozePressed)

{

return ("Wake Up!!! Snooze time is over.");

}

else

{

return ("Wake Up!");

}

}

}

}

// Delegate declaration.

//

public delegate void AlarmEventHandler(object sender, AlarmEventArgs e);

// The Alarm class that raises the alarm event.

//

public class AlarmClock

{

private bool snoozePressed = false;

private int nrings = 0;

private bool stop = false;

// The Stop property indicates whether the

// alarm should be turned off.

//

public bool Stop

{

get {return stop;}

set {stop = value;}

}

// The SnoozePressed property indicates whether the snooze

// button is pressed on the alarm when the alarm event is generated.

//

public bool SnoozePressed

{

get {return snoozePressed;}

set {snoozePressed = value;}

}

// The event member that is of type AlarmEventHandler.

//

public event AlarmEventHandler Alarm;

// The protected OnAlarm method raises the event by invoking

// the delegates. The sender is always this, the current instance

// of the class.

//

protected virtual void OnAlarm(AlarmEventArgs e)

{

if (Alarm != null)

{

// Invokes the delegates.

Alarm(this, e);

}

}

// This alarm clock does not have

// a user interface.

// To simulate the alarm mechanism it has a loop

// that raises the alarm event at every iteration

// with a time delay of 300 milliseconds,

// if snooze is not pressed. If snooze is pressed,

// the time delay is 1000 milliseconds.

//

public void Start()

{

for (;;)

{

nrings++;

if (stop)

{

break;

}

else if (snoozePressed)

{

System.Threading.Thread.Sleep(1000);

{

AlarmEventArgs e = new AlarmEventArgs(snoozePressed,

nrings);

OnAlarm(e);

}

}

else

{

System.Threading.Thread.Sleep(300);

AlarmEventArgs e = new AlarmEventArgs(snoozePressed,

nrings);

OnAlarm(e);

}

}

}

}

// The WakeMeUp class that has a method AlarmRang that handles the

// alarm event.

//

public class WakeMeUp

{

public void AlarmRang(object sender, AlarmEventArgs e)

{

Console.WriteLine(e.AlarmText +"\n");

if (!(e.SnoozePressed))

{

if (e.NumRings % 10 == 0)

{

Console.WriteLine(" Let alarm ring? Enter Y");

Console.WriteLine(" Press Snooze? Enter N");

Console.WriteLine(" Stop Alarm? Enter Q");

String input = Console.ReadLine();

if (input.Equals("Y") ||input.Equals("y")) return;

else if (input.Equals("N") || input.Equals("n"))

{

((AlarmClock)sender).SnoozePressed = true;

return;

}

else

{

((AlarmClock)sender).Stop = true;

return;

}

}

}

else

{

Console.WriteLine(" Let alarm ring? Enter Y");

Console.WriteLine(" Stop Alarm? Enter Q");

String input = Console.ReadLine();

if (input.Equals("Y") || input.Equals("y")) return;

else

{

((AlarmClock)sender).Stop = true;

return;

}

}

}

}

// The driver class that hooks up the event handling method of

// WakeMeUp to the alarm event of an Alarm object using a delegate.

// In a forms-based application, the driver class is the

// form.

//

public class AlarmDriver

{

public static void Main (string[] args)

{

// Instantiates the event receiver.

WakeMeUp w= new WakeMeUp();

// Instantiates the event source.

AlarmClock clock = new AlarmClock();

// Wires the AlarmRang method to the Alarm event.

clock.Alarm += new AlarmEventHandler(w.AlarmRang);

clock.Start();

}

}

}

[Visual Basic]' EventSample.vb.

'

Option Explicit

Option Strict

Imports System

Imports System.ComponentModel

Imports Microsoft.VisualBasic

Namespace EventSample

' Class that contains the data for

' the alarm event. Derives from System.EventArgs.

'

Public Class AlarmEventArgs

Inherits EventArgs

Private \_snoozePressed As Boolean

Private nrings As Integer

'Constructor.

'

Public Sub New(snoozePressed As Boolean, nrings As Integer)

Me.\_snoozePressed = snoozePressed

Me.nrings = nrings

End Sub

' The NumRings property returns the number of rings

' that the alarm clock has sounded when the alarm event

' is generated.

'

Public ReadOnly Property NumRings() As Integer

Get

Return nrings

End Get

End Property

' The SnoozePressed property indicates whether the snooze

' button is pressed on the alarm when the alarm event is generated.

'

Public ReadOnly Property SnoozePressed() As Boolean

Get

Return \_snoozePressed

End Get

End Property

' The AlarmText property that contains the wake-up message.

'

Public ReadOnly Property AlarmText() As String

Get

If \_snoozePressed Then

Return "Wake Up!!! Snooze time is over."

Else

Return "Wake Up!"

End If

End Get

End Property

End Class

' Delegate declaration.

'

Public Delegate Sub AlarmEventHandler(sender As Object, e As AlarmEventArgs)

' The Alarm class that raises the alarm event.

'

Public Class AlarmClock

Private \_snoozePressed As Boolean = False

Private nrings As Integer = 0

Private stopFlag As Boolean = False

' The Stop property indicates whether the

' alarm should be turned off.

'

Public Property [Stop]() As Boolean

Get

Return stopFlag

End Get

Set

stopFlag = value

End Set

End Property

' The SnoozePressed property indicates whether the snooze

' button is pressed on the alarm when the alarm event is generated.

'

Public Property SnoozePressed() As Boolean

Get

Return \_snoozePressed

End Get

Set

\_snoozePressed = value

End Set

End Property

' The event member that is of type AlarmEventHandler.

'

Public Event Alarm As AlarmEventHandler

' The protected OnAlarm method raises the event by invoking

' the delegates. The sender is always this, the current instance

' of the class.

'

Protected Overridable Sub OnAlarm(e As AlarmEventArgs)

RaiseEvent Alarm(Me, e)

End Sub

' This alarm clock does not have

' a user interface.

' To simulate the alarm mechanism it has a loop

' that raises the alarm event at every iteration

' with a time delay of 300 milliseconds,

' if snooze is not pressed. If snooze is pressed,

' the time delay is 1000 milliseconds.

'

Public Sub Start()

Do

nrings += 1

If stopFlag Then

Exit Do

Else

If \_snoozePressed Then

System.Threading.Thread.Sleep(1000)

If (True) Then

Dim e As New AlarmEventArgs(\_snoozePressed, nrings)

OnAlarm(e)

End If

Else

System.Threading.Thread.Sleep(300)

Dim e As New AlarmEventArgs(\_snoozePressed, nrings)

OnAlarm(e)

End If

End If

Loop

End Sub

End Class

' The WakeMeUp class that has a method AlarmRang that handles the

' alarm event.

'

Public Class WakeMeUp

Public Sub AlarmRang(sender As Object, e As AlarmEventArgs)

Console.WriteLine((e.AlarmText + ControlChars.Cr))

If Not e.SnoozePressed Then

If e.NumRings Mod 10 = 0 Then

Console.WriteLine(" Let alarm ring? Enter Y")

Console.WriteLine(" Press Snooze? Enter N")

Console.WriteLine(" Stop Alarm? Enter Q")

Dim input As String = Console.ReadLine()

If input.Equals("Y") Or input.Equals("y") Then

Return

Else

If input.Equals("N") Or input.Equals("n") Then

CType(sender, AlarmClock).SnoozePressed = True

Return

Else

CType(sender, AlarmClock).Stop = True

Return

End If

End If

End If

Else

Console.WriteLine(" Let alarm ring? Enter Y")

Console.WriteLine(" Stop Alarm? Enter Q")

Dim input As String = Console.ReadLine()

If input.Equals("Y") Or input.Equals("y") Then

Return

Else

CType(sender, AlarmClock).Stop = True

Return

End If

End If

End Sub

End Class

' The driver class that hooks up the event handling method of

' WakeMeUp to the alarm event of an Alarm object using a delegate.

' In a forms-based application, the driver class is the

' form.

'

Public Class AlarmDriver

Public Shared Sub Main()

' Instantiates the event receiver.

Dim w As New WakeMeUp()

' Instantiates the event source.

Dim clock As New AlarmClock()

' Wires the AlarmRang method to the Alarm event.

AddHandler clock.Alarm, AddressOf w.AlarmRang

clock.Start()

End Sub

End Class

End Namespace

**See Also**

[Handling and Raising Events](https://msdn.microsoft.com/en-us/library/aa720046(v=vs.71).aspx)

# Raising Multiple Events

If your class raises multiple events and you program these as described in [Raising an Event](https://msdn.microsoft.com/en-us/library/aa720442(v=vs.71).aspx), the compiler generates one field per event delegate instance. If the number of events is large, the storage cost of one field per delegate may not be acceptable. For those situations, the .NET Framework provides a construct called event properties that you can use together with another data structure (of your choice) to store event delegates.

Event properties consist of event declarations accompanied by event accessors, as shown in the following example. Event accessors are methods you define to allow event delegate instances to be added or removed from the storage data structure. Note that event properties are slower than event fields, as each event delegate has to be retrieved before it can be invoked. The trade-off is between memory and speed. If your class defines many events that are infrequently raised, you will want to implement event properties. Windows Forms controls and ASP.NET server controls use event properties instead of event fields.

The following C# example shows how to use event properties to implement events. The keywords of the event property construct are in bold type.

**Note**   Event properties are not supported in Visual Basic .NET in this release.

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_348c0c19-e04a-467f-9955-de67cc8242a3');)

// The class Control defines two events-MouseUp and MouseDown-that

// use event properties.

class Control: Component {

// omit start

Delegate \_handler = null;

// omit end

// Defines a unique key for each event.

static readonly object mouseDownEventKey = new object();

static readonly object mouseUpEventKey = new object();

// Defines a method to retrieve the event delegate associated with key.

protected Delegate GetEventHandler(object key) {...}

// Defines the helper method for the add accessor.

protected void AddEventHandler(object key, Delegate handler) {...}

// Defines the helper method for the remove accessor.

protected void RemoveEventHandler(object key, Delegate handler) {...}

// Defines the MouseDown event property.

public event MouseEventHandler MouseDown {

// The add event accessor method.

add { AddEventHandler(mouseDownEventKey, value); }

// The remove event accessor method.

remove { RemoveEventHandler(mouseDownEventKey, value); }

}

// Defines the MouseUp event property.

public event MouseEventHandler MouseUp {

// The add event accessor method.

add { AddEventHandler(mouseUpEventKey, value); }

// The remove event accessor method.

remove { RemoveEventHandler(mouseUpEventKey, value); }

}

// Define a private data structure to store the

// event delegates.

// To do - Define your data stucture;

}

The .NET Framework provides a data structure for storing event delegates, the [System.ComponentModel.EventHandlerList](https://msdn.microsoft.com/en-us/library/system.componentmodel.eventhandlerlist(v=vs.71).aspx)class, which is used by classes in the Framework that raise multiple events. You can use this class or define your own data structure for storage.

#### See Also

[Handling and Raising Events](https://msdn.microsoft.com/en-us/library/aa720046(v=vs.71).aspx)