**C# Delegates**

A delegate is a type that safely encapsulates a method, like a function pointer in C and C++.

* A picture containing font, text, screenshot, graphics

  Description automatically generatedThe type of a delegate is defined by the name of the delegate.
* A screenshot of a computer program

  Description automatically generated with medium confidenceFor example, we declare a delegate named **Del** that can encapsulate a method that takes a **string** as an argument and return **void**
* Any parameters passed **to** delegate by caller are passed to the method.
* Any return value, if any, from wrapped method, will be returned to the caller.

Because the instantiated delegate is an object, it can be **passed as an argument**, or **assigned to a property**. This allows a method to accept a delegate as a parameter, and call the delegate at some later time. This is known as an asynchronous callback, and is a common method of notifying a caller when a long process has completed.

A screen shot of a computer

Description automatically generated with low confidence

A picture containing font, text, graphics, screenshot

Description automatically generated

A delegate can call more than one method when invoked. This is referred to as multicasting.

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer code

Description automatically generated with low confidence

When allMethodsDelegate is invoked, it passes all parameters to all methods within it (d1, d2, d3)

**Event with Delegates (EventHandler)**

* The recipient creates a method to handle an event.
* Creates a delegate for that method. Multicast delegate to the event source.
* Event source calls delegate when event occurs (invokes delegate), delivering data to recipient.

Events enable a class or object to notify other classes or objects when something of interest occurs. The class that sends (or raises) the event is called the publisher and the classes that receive (or handle) the event are called subscribers.

* How does the publisher “raise” the event? It invokes a method!
* So how does the publisher know **which** method to invoke? A **contract** is needed between publisher and subscriber
* EventHandler in Subscribers: A screen shot of a computer

  Description automatically generated with low confidence
* Multicasted delegates of (EventHandlers of Subscribers) in Publisher!

**\*\*\* Publisher\*\*\***

**A picture containing text, font, graphics, graphic design

Description automatically generated**

**A screen shot of a computer code

Description automatically generated with low confidence**

1. **Define a delegate (determines the signature of the method)** – also EventHandler of subscribers.



*source: source of the event*

*args: any additional data we send with the event*

1. **Define an event based on that delegate.**



*event: a* ***collection*** *of delegates of the subscribers, to be delegate-invoked.*

1. **Publish (raise) an event.**

 (when we want to raise the event)



A screen shot of a computer

Description automatically generated with low confidence

* Calls all delegates registered in “VideoEncoded”, pass in “this” as source, with data in “EventArgs”

**\*\*\*Subscriber\*\*\***

1. **Define EventHandler for subscriber**

A picture containing text, screenshot, font

Description automatically generated

1. **Multicast to “subscribe list”**

A picture containing text, screenshot, font

Description automatically generated

**NOTE:**

* We can decide to pass some data by creating a class that inherits from EventArgs

A picture containing text, font, screenshot, handwriting

Description automatically generated

* We can simplify part 1 and 2 of Publisher to:



*(omit the <VideoEventArg> if no data is passed)*

* We can use syntactic sugar for part 3 of Publisher:

if (VideoEncoded != null) VideoEncoded(<param>);

* VideoEncoded?.Invoke(<param>);