CGT 353: Principles of Interactive and Dynamic Media
Events and Event Handling with AS 3.0

Introduction:

- Previous discussions of events and listeners were very basic…
- Did not place much importance in understanding the underlying functions…
- Texts discuss these functions in-depth.
- Important for greater manipulation and control of the event handling process….

Terminology:

- **Event –** an occurrence that can trigger a response. Each event is identified by an ***event name***, which is accessible via a constant
- **Event object –** An object representing a single occurrence of an event that determines what information about the event is available to listeners
- **Event Target** – The object to which the event pertains and that acts as the destination object of a dispatched event. Event targets can register listeners.
- **Event Listener** – A function or method registered to receive event notification from an event target or target ancestor
- **Event Dispatching** – the process of sending notification of the event to the event target, which triggers any listeners. Also known as **event propogation**

Two Types of Events:

1. **Built-in Events:** describe changes to the state of the runtime environment
2. **Custom Events:** describe changes to the state of the program

- In a pure AS program, everything after the main class constructor method is triggered by events.

How it all Works:

- Again, **event listeners** are methods or functions that are registered to be executed once a particular **event** takes place.
- The process of notifying event listeners when an event takes place is **event dispatch.**
- When an event is about to start, AS creates an **event object** that represents the event (always an instance of the *Event* class.)
- All events listeners executed during dispatch are passed a reference to the event object as an **argument.**
- Listeners can use these event object variables to access info relating to the event…very useful.
- All events have **string names**.
- During dispatch, this name can be retrieved with the **type** variable.
- Each event dispatch has an **event target –** which is basically the thing that was manipulated, clicked, typed on, whatever.
- To respond to an event, listeners must **register** with the event target.
- All event target objects are instances of a class that inherits from the ***EventDispatcher*** class.
- *EventDispatcher* provides methods like *addEventListener()* and *removeEventListener()*
- When the event target is a display object, listeners can also register with the event target's **display ancestors**

Registering an Event Listener:

1. Determine the name of the event's event type.
2. Determine the datatype of the event object representing the event.
3. Create an event listener to respond to the event.
4. Use *EventDispatcher* class's instance method *addEventListener( )* to register the event listener with the event target (or, any display ancestor of the event target).
5. Wait for the event to occur.

Step 1 - Determining the Event Type's Name:

- Each event type name is accessible through a constant of the Event class or one of its descendants.
- Ex. For "operation complete" – Event.COMPLETE (string value = "complete")
- Ex. For "mouse pressed" – Event.MOUSE_DOWN (string value = "mouseDown)
- **Note:** Look at list of events in ActionScript 3.0 Language and Components Reference (Event, EventDispatch, MouseEvent, KeyBoard Event)
- **These are the event types you'll use most often.**

For URLLoader Class:

- Look under "events" header in the AS3.0 Language reference

complete event
Event object type: flash.events.Event
Event.type property = flash.events.Event.COMPLETE
Dispatched after all the received data is decoded and placed in the data property of the URLLoader object. The received data may be accessed once this event has been dispatched.

So in the code we would see:

theURLLoader.addEventListener (**Event.COMPLETE**, *someListener*);

Step 2 – Determine the Event Object's DataType:

- For the example above, the datatype of Event.COMPLETE's event object is

flash.events.Event

Step 3 – Create the Event Listener:

```
private function completeListener (e:Event):void {
trace("Load complete");
}
```

Some Important Rules:

- Note the parameter "e"…this will receive the event object at runtime.
- All event listeners have a return type of *void.*
- Event listeners that are methods are declared *private* so they can't be invoked elsewhere.
- Naming: Use *eventname*Listener, where *eventName* is the string name of the event.

Step 4 - Register for the Event:

```
// create the event target class
var urlLoader:URLLoader = new URLLoader( );

// register the listener
urlLoader.addEventListener(Event.COMPLETE, completeListener);
```

Complete method signature:

addEventListener(*type*, *listener*, *useCapture*, *priority*, *useWeakReference*)

- ***useCapture*** not needed for now
- ***priority*** refers to the precedence of execution of the listener
- ***useWeakReference*** refers to memory condition

```
package {
        import flash.display.*;
        import flash.net.*;
        import flash.events.*;

        public class FileLoader extends Sprite {
        public function FileLoader ( ) {

        // Create the event target
        var urlLoader:URLLoader = new URLLoader( );

        // Register the event listener
        urlLoader.addEventListener(Event.COMPLETE, completeListener);

        // Start the operation that will trigger the event
        urlLoader.load(new URLRequest("someFile.txt"));
        }


        // Define the event listener

                private function completeListener (e:Event):void {
```

```
                trace("Load complete");
                }
        }
}
```

Accounting for the possibility of error:

```
package {
        import flash.display.*;
        import flash.net.*;
        import flash.events.*;

        public class FileLoader extends Sprite {
        public function FileLoader ( ) {

        // Create the event target
        var urlLoader:URLLoader = new URLLoader( );

        // Register the event listener
        urlLoader.addEventListener(Event.COMPLETE, completeListener);

        // Start the operation that will trigger the event
        urlLoader.load(new URLRequest("someFile.txt"));
        }


        // Define the event listener

                private function completeListener (e:Event):void {
                trace("Load complete");

                }

                private function ioErrorListener (e:Event):void {
                trace("Error loading file.");
        }
}
```

Un-Registering an Event Listener:

- Very important to clear from memory, as it will persist

    eventTargetOrTargetAncestor.removeEventListener(*type*, *listener*, *useCapture*)

- Again, only type and listener usually required.

Accessing the Target Object:

- Remember that during dispatch, the event object passed to a listener defines a target variable that provides a reference to the target object.

- Use the instance variable **target**

```
package {
        import flash.display.*;
        import flash.net.*;
        import flash.events.*;

public class FileLoader extends Sprite {
        public function FileLoader ( ) {
                var urlLoader:URLLoader = new URLLoader( );
                urlLoader.addEventListener(Event.COMPLETE, completeListener);
                urlLoader.load(new URLRequest("someFile.txt"));
                }

        private function completeListener (e:Event):void {
                trace("Load complete");
                }

        private function completeListener (e:Event):void {
        var loadedText:String = URLLoader(e.target).data;
                }

        }
}
```

Example 2:

```
package {
        import flash.display.*;
        import flash.events.*;
        import flash.text.*;

        // Changes a text field's background color to red when focused

        public class HighlightText extends Sprite {

        // Constructor

        public function HighlightText ( ) {

                // Create a Sprite object
                var s:Sprite = new Sprite( );
                s.x = 100;
                s.y = 100;

                // Create a TextField object

                var t:TextField = new TextField( );
                t.text = "Click here";
                t.background = true;
                t.border = true;
                t.autoSize = TextFieldAutoSize.LEFT;

                // Put the TextField in the Sprite
                s.addChild(t);

                // Add the Sprite to this object's display hierarchy
                addChild(s);

                // Register to be notified when the user focuses any of the Sprite
                // object's descendants (in this case, there's only one descendant:
                // the TextField, t)

                s.addEventListener(FocusEvent.FOCUS_IN, focusInListener);
                }

                // Listener executed when one of the Sprite object's descendants
                // is focused

                public function focusInListener (e:FocusEvent):void {
                        // Displays: Target of this event dispatch: [object TextField]
                        trace("Target of this event dispatch: " + e.target);

                        // Set the text field's background to red. Notice that, for added type
                        // safety, we cast Event.target to TextField-—the actual datatype of
                        // the target object.

                        TextField(e.target).backgroundColor = 0xFF0000;
                        }
                }
}
```