

Events }

Advanced Multimedia Technologies

Events

- { Events
- { Types of Events
- { Registering Events
- { Event Listeners
- { Adding and Removing (Garbage Collection)
- { Custom Events



Events

Events

A condition for notification. Such as a mouse over, complete load or a change in value.

Event Usage

Events are used to trigger responses with an application. When something is clicked, do this.



Types of Events

Types of Events

- { Built in Events
- { Custom Events

Built in Events

Events that are apart of the general
ActionScript library.

Built in Events

- { Mouse
- { Keyboard
- { EnterFrame
- { Complete
- { Progress
- { Focus
- { Timer

Custom Events

Defined by the developer.

Custom Events

- { GAME_OVER
- { END_LOADING
- { CLOSE_PANEL
- { IN
- { OUT

Built in Event Examples

`MouseEvent.CLICK`

Built in Event Examples

`MouseEvent.CLICK`

**The Name of the
Event Class.**

Built in Event Examples

`MouseEvent.CLICK`

Remember the ActionScript is structured around Classes.

Built in Event Examples

MouseEvent.**CLICK**

This refers to the event taking place. (This is stored as a static constant)

Registering Events



Registering Events

Events must be registered in order for the Flash runtime to handle them.

Registering Events

```
addEventListener(MouseEvent.CLICK, go)
```

Registering Events

```
addEventListener(MouseEvent.CLICK, go)
```

**This method has
two arguments.**

Registering Events

```
addEventListener(MouseEvent.CLICK, go)
```

**The Name of the
Event Class.**

Registering Events

```
addEventListener(MouseEvent.CLICK, go)
```

**The event you
are listening for...**

Registering Events

```
addEventListener(MouseEvent.CLICK, go)
```

**...and the responding
function to be executed
once the event happens.**

Registering Events

```
addEventListener(MouseEvent.CLICK, go)
```

When the mouse is clicked a function named go will execute.

Event Listeners



Event Listeners

Commands that execute when registered event conditions are true.

Event Listener

```
addEventListener(MouseEvent.CLICK, go)
```

go is the Event Listener

Event Listener

```
addEventListener(MouseEvent.CLICK, go)
```

And must be defined.

Event Handlers

```
addEventListener(MouseEvent.CLICK, go)
function go(e:MouseEvent):void{
    trace("Mouse Click Event")
}
```

Event Handlers

```
addEventListener(MouseEvent.CLICK, go)
function go(e:MouseEvent):void{
    trace("Mouse Click Event")
}
```

**The Event Listener
declaration.**

Event Handlers

```
addEventListener(MouseEvent.CLICK, go)
function go(e:MouseEvent):void{
    trace("Mouse Click Event")
}
```

When the Mouse is clicked...

Event Handlers

```
addEventListener(MouseEvent.CLICK, go)
function go(e:MouseEvent):void{
    trace("Mouse Click Event")
}
```

...run this function.

Event Handlers

```
addEventListener(MouseEvent.CLICK, go)
function go(e:MouseEvent):void{
    trace("Mouse Click Event")
}
```

Resulting in a trace.



Removing Listeners

Removing Listeners

Every listener you register is taxing your applications resources. Continuously listening for that event until runtime ends or it's removed.

Removing Listeners

```
removeEventListener(MouseEvent.CLICK, go)
```

This removes the listener and the mouse click will no longer execute the go function.

Catch 22

Removing *ALL* of the listeners is impractical. You need some of them for basic operation. However nonessential event listeners should be removed when no longer needed.

Case Study

Primary navigation is needed to get around the navigation, but sub navigation is only needed at certain times. Don't tax your resources by making them available all the time. Use only what you need.

Exit Strategy

Keep this thought in the back of your head. If you create something, how will you destroy it? Do you have methods in place to handle creation as well as deletion of objects and event listeners?

Garbage Collection

By removing unused objects and event listeners they become available for garbage collection and they will be removed from the application freeing up resources.

Custom Events



Custom Events

Events, or notifications defined by you.

Custom events are the power behind state development.

Examples

Dispatching events for GAME_OVER,
SHIP_HIT, APP_START,
PANEL_CLOSED

Custom Events

- { Define the Event
- { Dispatch the Event
- { Register and listen for Event
- { Execute listener

Custom Event - Definition

```
package{
import flash.events.*
    public static const IN:String = "in"
    class CustomEvent extends Event{

        public function CustomEvent(type){
            super(type,true,true)
        }
        public override function clone():Event{
            return new CustomEvent(type)
        }
    }
}
```

Custom Event - Definition

```
package{  
import flash.events.*  
    public static const IN:String = "in"  
class CustomEvent extends Event{  
    public function CustomEvent(type) {  
        super(type)  
    }  
    public override function clone():Event {  
        return new CustomEvent(type)  
    }  
}  
}
```

**Basic Class
Definition**

Custom Event - Definition

```
package{  
import flash.events.*  
public static const IN:String = "in"  
class CustomEvent extends Event{  
  
public function CustomEvent(type){  
super(type)  
}  
public override function toString():Event{  
return  
}  
}  
}
```

**Although we are
extending the
Event Class.**

Custom Event - Definition

```
package{
import flash.events.*
    public static const IN:String = "in"
class CustomEvent extends Event{

    public function CustomEvent(type){
        super(type)
    }
    public override function toString():Event{
        return
    }
}
}
```

Define the static constant used to reference the event.

Custom Event - Definition

```
package{
import flash.events.*
    public static const IN:String = "in"
class CustomEvent extends Event{

    public function CustomEvent(type) {
        super(type,true,true)
    }
    public override function clone():Event{
        return new CustomEvent(type)
    }
}
}
```

**The constructor
sets up the
custom event.**

Custom Event - Definition

```
package{
import flash.events.*
    public static const IN:String = "in"
    class CustomEvent extends Event{

        public function CustomEvent(type){
            super(type,true,true)
        }
        public override function clone():Event{
            return new CustomEvent(type)
        }
    }
}
```

**This allows the
new custom event
to dispatch.**

Custom Events - Dispatch

```
var ce:CustomEvent = new CustomEvent(CustomEvent.IN)  
dispatchEvent(ce)
```

Custom Events - Dispatch

```
var ce:CustomEvent = new CustomEvent(CustomEvent.IN)  
dispatchEvent(ce)
```

**Create a custom
event instance.**

Custom Events - Dispatch

```
var ce:CustomEvent = new CustomEvent(CustomEvent.IN)  
dispatchEvent(ce)
```

**Note this is the
static constant we
just created.**

Custom Events - Dispatch

```
var ce:CustomEvent = new CustomEvent(CustomEvent.IN)  
dispatchEvent(ce)
```

**This will dispatch
the newly created
custom event.**

Custom Events - Dispatch

```
var ce:CustomEvent = new CustomEvent(CustomEvent.IN)  
dispatchEvent(ce)
```

**It's a behind the scenes look
at forcing at how an event fires.**

Custom Events - Register

```
var ce:CustomEvent = new CustomEvent(CustomEvent.IN)  
dispatchEvent(ce)
```

```
addEventListener(CustomEvent.IN, go)  
function go():void{  
    trace("custom event executed")  
}
```

Custom Events - Register

```
var ce:CustomEvent = new CustomEvent(CustomEvent.IN)  
dispatchEvent(ce)
```

```
addEventListener(CustomEvent.IN, go)  
function go():void{  
    trace("custom event executed")  
}
```

**Register the listener
for the custom event.**

Custom Events - Register

```
var ce:CustomEvent = new CustomEvent(CustomEvent.IN)  
dispatchEvent(ce)
```

```
addEventListener(CustomEvent.IN, go)  
function go():void{  
    trace("custom event executed")  
}
```

**Handle it as soon as
the event happens.**

Custom Events

A great way to organize features to happen at specific times.

Events

- { Events
- { Types of Events
- { Registering Events
- { Event Listeners
- { Adding and Removing (Garbage Collection)
- { Custom Events