

EVENTOS

Los eventos pueden ser generados por los dispositivos de usuario, como el ratón y el teclado, o cualquier otra entrada externa, como el retorno de una llamada de un web service.

Los eventos también se desencadenan cuando ocurren cambios en el ciclo de la apariencia o la vida de un componente, como la creación o destrucción de un componente o cuando el componente cambia de tamaño.

Cualquier interacción del usuario con la aplicación puede generar eventos. Los eventos también pueden ocurrir sin ninguna interacción directa del usuario, por ejemplo, cuando finaliza la carga de datos de un servidor o cuando una cámara conectada se activa. Se pueden "manejar" estos eventos en el código mediante un controlador de eventos. Los controladores de eventos son las funciones o métodos que se escriben para responder a eventos específicos. También se les conoce a veces como detectores de eventos.

Las aplicaciones Flash Player, tienen la habilidad de disparar eventos de forma asíncrona, en paralelo, siendo ninguna de estas dependientes entre sí, permitiendo a las actividades "transparentes" e ir por "detrás" mientras que la aplicación interactúa, y proporciona información al usuario en tiempo real (en lugar de tener código ejecutándose sincrónica o secuencialmente, mientras que el usuario espera a que se complete).

Por el contrario, al hacer una petición a un servidor web, una página HTML visualiza mediante un navegador web la página de forma vertical desde la parte superior de la página a la página del inferior, a menos que JavaScript se utilice para actualizar de forma asíncrona la página después de que la página se ha cargado.

La técnica para especificar determinadas acciones que deben realizarse como respuesta a eventos concretos se denomina gestión de eventos. Cuando se escribe código ActionScript para llevar a cabo la gestión de eventos, se deben identificar tres elementos importantes:

- El origen del evento: ¿en qué objeto va a repercutir el evento? Por ejemplo, ¿en qué botón se hará click o qué objeto Loader está cargando la imagen? El origen del evento también se denomina objetivo del evento, ya que es el objeto al que Flash Player (donde tiene lugar realmente el evento) destina el evento.
- El evento: ¿qué va a suceder, a qué se va a responder? Es importante identificar esto porque muchos objetos activan varios eventos.
- La respuesta: ¿qué pasos hay que llevar a cabo cuando ocurra el evento?

El **flujo del evento** describe el modo en el que un objeto de evento se desplaza por la lista de visualización. La lista de visualización se organiza en una jerarquía que puede describirse como un árbol. En la parte superior de la jerarquía de la lista de visualización se encuentra el objeto Stage (o Escenario) , que es un contenedor de objeto de visualización especial que actúa como raíz de la lista de visualización.



Cuando Flash Player distribuye un objeto de evento para un evento relacionado con la lista de visualización, éste realiza un viaje de ida y vuelta desde el objeto Stage (escenario) hasta el nodo de destino. El nodo de destino es el objeto de la lista de visualización en el que se ha producido el evento. Por ejemplo, si un usuario hace clic en un objeto de la lista de visualización denominado child1, Flash Player distribuirá un objeto de evento usando child1 como nodo de destino.

En ActionScript ,se pueden registrar detectores de eventos en un nodo de destino y en cualquier nodo a lo largo del flujo del evento. No todos los eventos, sin embargo, participan en las tres fases.

El flujo del evento se divide conceptualmente en tres partes:

FASE DE CAPTURA :

La primera parte del flujo del evento se llama la fase de captura. Esta fase comprende la totalidad de los nodos desde el nodo raíz hasta el elemento principal del nodo de destino.

FASE DE DESTINO :

La segunda parte del flujo del evento, la fase de destino, consiste únicamente en el nodo de destino.

FASE DE PROPAGACION:

La tercera parte del flujo del evento, la fase de propagación, se compone de todos los nodos del padre del nodo destino al nodo raíz. Comenzando con el padre del nodo de destino, Flash Player establece los valores correspondientes en el objeto de evento y luego llama a los detectores de eventos en cada uno de estos nodos.

La Clase Event.

La clase base para todos los objetos de eventos es la clase `flash.events.Event`. Todos los objetos de eventos son instancias de esta clase, o instancias de una subclase de la clase `Event`. En la siguiente tabla se describen las propiedades públicas de la clase `Event`.

Property	Type	Description
<code>type</code>	String	Es el nombre del evento, por ejemplo, "click". El constructor de eventos establece esta propiedad.
<code>target</code>	EventDispatcher	Es una referencia a la instancia del componente que dispara el evento. Esta propiedad se establece mediante el método <code>dispatchEvent()</code> , no se puede cambiar a un objeto diferente.
<code>currentTarget</code>	EventDispatcher	Es una referencia a la instancia del componente que dispara el Evento, esta propiedad se establece mediante el método <code>dispatchEvent()</code> .
<code>eventPhase</code>	uint	Devuelve la fase actual en el flujo del evento. La propiedad puede contener los siguientes valores: <ul style="list-style-type: none"><code>EventPhase.CAPTURING_PHASE</code>: The capture phase<code>EventPhase.AT_TARGET</code>: The target phase<code>EventPhase.BUBBLING_PHASE</code>: The bubbling phase
<code>bubbles</code>	Boolean	Si el evento puede propagarse, el valor de esta propiedad es true, de lo contrario, es falsa. Se puede opcionalmente pasar esta propiedad como argumento del constructor de la clase <code>Event</code> .
<code>cancelable</code>	Boolean	Si el evento se puede cancelar, el valor para este valor es true, de lo contrario, es false. Se puede opcionalmente pasar esta propiedad como argumento del constructor de la clase <code>Event</code> .

UTILIZAR UN EVENTO

El uso de eventos en Flex es un proceso de dos pasos.

En primer lugar, se escribe una función o un método de clase, conocido como un detector de eventos o controlador de eventos, que responde a los eventos.

El siguiente ejemplo muestra una simple función de detector de eventos que informa cuando un control activa el evento que se está escuchando:

```
<?xml version="1.0"?>

<!-- events/SimpleEventHandler.mxml -->

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp();">

    <mx:Script><![CDATA[

        import mx.controls.Alert;

        private function initApp():void {
            b1.addEventListener(MouseEvent.CLICK, myEventHandler);
        }

        private function myEventHandler(event:Event):void {
            Alert.show("An event occurred.");
        }

    ]]></mx:Script>

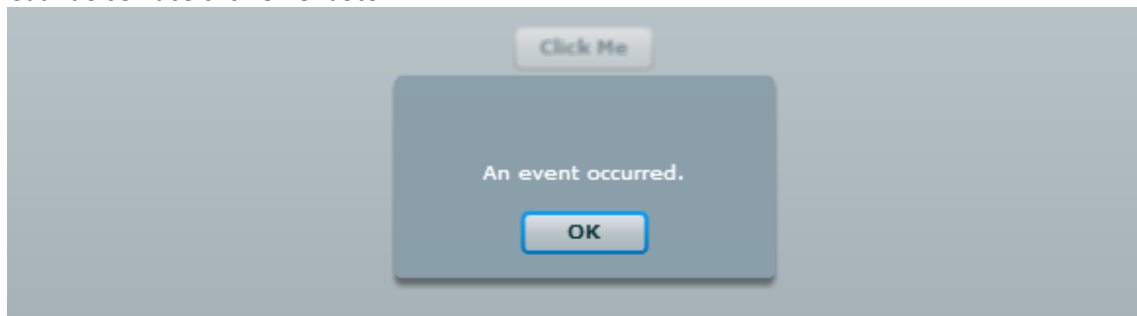
    <mx:Button id="b1" label="Click Me"/>

</mx:Application>
```

El ejemplo de este código sería :



Cuando se hace click en el botón :



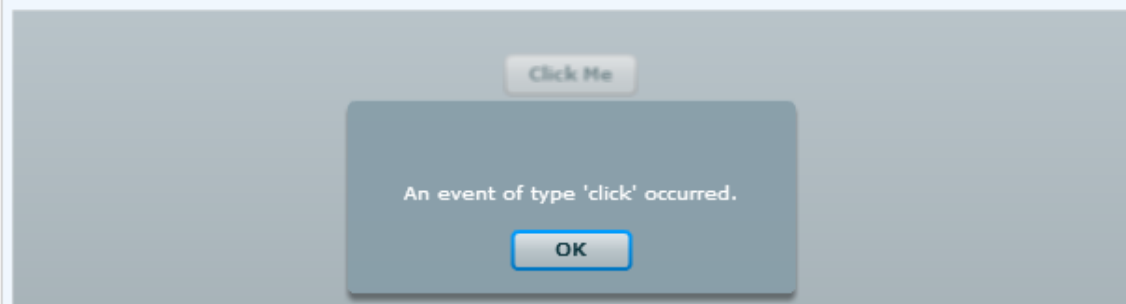
La mayoría de los controladores FLEX simplifican el registro de detectores de eventos, ya que permite especificar el controlador dentro de la etiqueta MXLM, por ejemplo en lugar de utilizar el método `addEventListener()` para especificar una función de detección de evento click del control Button, se especifica en el atributo `click` del tag `<mx:Button>`, y obtenemos el mismo resultado que con el código anterior. Ej:

```
<?xml version="1.0"?>
<!-- events/EventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import mx.controls.Alert;

        private function myEventHandler(e:Event):void {
            Alert.show("An event of type '" + e.type + "' occurred.");
        }
    ]]></mx:Script>

    <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>
</mx:Application>
```

The executing SWF file for the previous example is shown below:



Esto es equivalente al método `addEventListener()` en el ejemplo de código anterior. Sin embargo, se recomienda utilizar el método `addEventListener()`. Este método le da un mayor control sobre el evento y configuración del mismo.

Además, si utiliza `addEventListener()` para agregar un controlador de eventos, puede utilizar `removeEventListener()` para eliminar el controlador cuando ya no lo necesite.

Se puede acceder a las propiedades del evento.

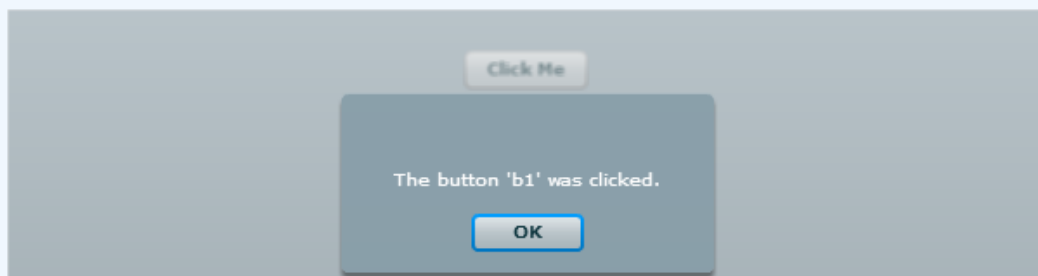
Los objetos Evento incluyen una referencia a la instancia del componente de envío (destino), lo que significa que se puede acceder a todas las propiedades y métodos de la instancia en un detector de eventos. El siguiente ejemplo accede al id del control Button que desencadenó el evento:

```
<?xml version="1.0"?>
<!-- events/AccessingCurrentTarget.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import mx.controls.Alert;

        private function myEventHandler(e:Event):void {
            Alert.show("The button '" + e.currentTarget.id + "' was clicked.");
        }
    ]]></mx:Script>

    <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>
</mx:Application>
```

The executing SWF file for the previous example is shown below:



Removiendo un Evento Listener.

Se puede utilizar el método `removeEventListener()` para eliminar un detector de eventos que ya no se necesita. Los parámetros requeridos son los parámetros `eventName` y `listener`, que son los mismos que los parámetros necesarios para el método `addEventListener()`.

En el ejemplo de abajo tenemos 2 líneas de botones. El `button1` se le agrega en el atributo `Click` como dentro del tag de `MXLM`, mientras que el `button2` se agrega con el evento `Click` mediante la función `addEventListener`.

Si se presiona tanto el `Button1` como `Button2` se mostrar las alertas, mientras que si hacemos click en los botones de `Remove Listener`, el `Button2` ya no tendrá efecto y no se mostrará la alerta por lo que se ha eliminado satisfactoriamente el Evento, mientras que en el `Button1` seguirá apareciendo la alerta, ya que estos eventos embebidos en `MXML` no se pueden eliminar.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
  backgroundColor="#FFFFFF" backgroundGradientColors="#FFFFFF #FFFFFF"
  width="400" height="400" creationComplete="add_listener()">
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      private function add_listener():void{
        button2.addEventListener(MouseEvent.CLICK, show_alert2);
      }
      private function show_alert1():void{
        Alert.show('Button1 Clicked')
      }
      private function show_alert2(event:MouseEvent):void{
        Alert.show('Button2 Clicked');
      }
    ]]>
  </mx:Script>
  <mx:Text text="Try to remove click event for the first and second button"/>
  <mx:HBox verticalGap="10">
    <mx:Button click="show_alert1()" id="button1" label="Button1"/>
    <mx:Button click="button1.removeEventListener(MouseEvent.CLICK, show_alert1)"
      label="Remove listener 1"/>
  </mx:HBox>
  <mx:HRule width="350"/>
  <mx:HBox verticalGap="10">
    <mx:Button id="button2" label="Button2"/>
    <mx:Button click="button2.removeEventListener(MouseEvent.CLICK, show_alert2)"
      label="Remove listener 2"/>
  </mx:HBox>
</mx:Application>
```

EVENTOS PERSONALIZADOS.

Se pueden crear eventos personalizados como parte de la definición MXML y componentes ActionScript. Los eventos personalizados permiten agregar funcionalidad a medida a sus componentes para responder a las interacciones del usuario.

Creando Eventos Personalizados.

Todos los componentes MXML puede disparar eventos, tanto los heredados por los componentes de sus superclases.

Cuando se desarrollan componentes MXML, se pueden agregar sus propios tipos de eventos.

Un ejemplo:

Para cada evento personalizado enviado por el componente, debe hacer lo siguiente:

- 1) Crear un objeto Evento que describa el evento.
- 2) (opcional) usar la etiqueta [Event] para que el evento sea publico y accesible desde otro componente.
- 3) Disparar el evento mediante el método [dispatchEvent\(\)](#).

Ejemplo :

En este ejemplo, se define un nuevo componente llamado TextAreaEnabled.mxml que utiliza una etiqueta <mx:TextArea> como su etiqueta raíz. Este componente también define una nueva propiedad llamada enableTA, que si está en true permitir la introducción de texto y en false para deshabilitarlo. La etiqueta [Evento] identifica el evento en el compilador MXML para que se pueda hacer referencia a él desde otro componente.

La sintaxis para [Evento] es :

```
<mx:Metadata>
    [Event (name="eventName", type="eventType")]
</mx:Metadata>
```

Con el evento dispatchEvent() se dispara el nuevo evento, como muestra el código abajo.

```
<?xml version="1.0"?>
<!-- mxmmlAdvanced/myComponents/TextAreaEnabled.mxml -->

<mx:TextArea xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Metadata>
        [Event (name="enableChanged", type="flash.events.Event")]
    </mx:Metadata>

    <mx:Script>
        <![CDATA[
```

```

import flash.events.Event;

// Define private variable to hold the enabled state.
private var __enableTA:Boolean;

// Define a setter method for the private variable.
public function set enableTA(val:Boolean):void {
    __enableTA = val;
    enabled = val;

    // Define event object, initialize it, then dispatch
    it.    dispatchEvent(new Event("enableChanged"));
}

// Define a getter method for the private variable.
public function get enableTA():Boolean {
    return __enableTA;
}

]]>
</mx:Script>
</mx:TextArea>

```

Con el siguiente código Main asociado,

```

<?xml version="1.0"?>
<!-- mxmmlAdvanced/MainTextAreaEnable.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:MyComp="myComponents.*">

    <mx:Script>
        <![CDATA[

            import flash.events.Event;
            import myComponents.TextAreaEnabled;

            public function handleEnableChangeEvent(eventObj:Event):void {

```

```

        var tempTA:TextAreaEnabled =
            eventObj.currentTarget as TextAreaEnabled;
        if (tempTA.enableTA) {
            myButton.label="Click to disable";
        }
        else {
            myButton.label="Click to enable";
        }
    }
]]>
</mx:Script>

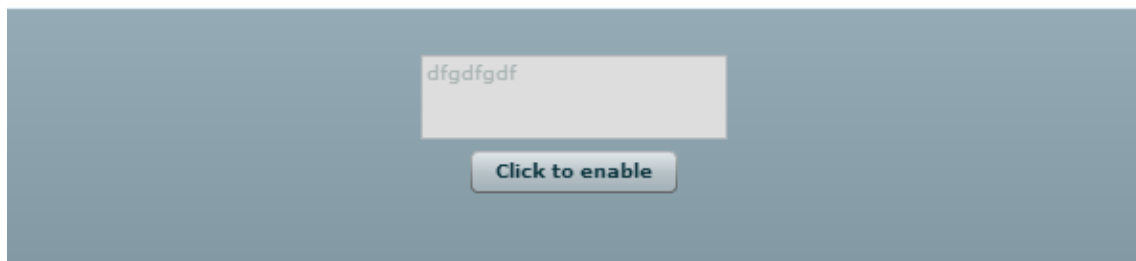
<MyComp:TextAreaEnabled id="myTA" enableTA="false"
    enableChanged="handleEnableChangeEvent(event);" />

<mx:Button id="myButton" label="Click to enable"
    click="myTA.enableTA=!myTA.enableTA;" />

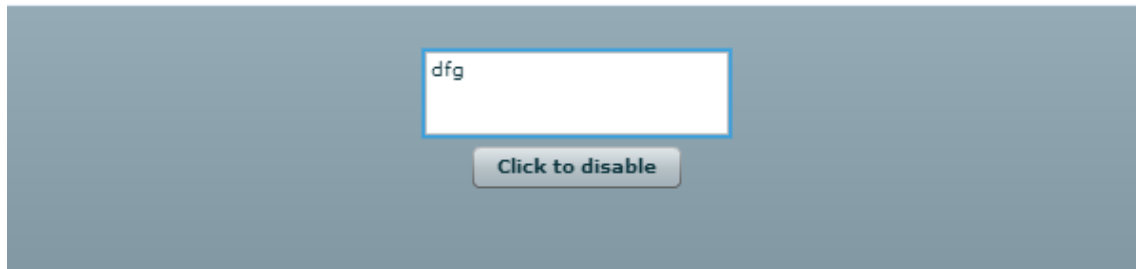
</mx:Application>

```

Visualizándose en ejecución queda :



Y al habilitarlo:



Si no se utiliza la etiqueta [Evento] en el archivo para definir el evento `enableChanged`, el compilador MXML genera un mensaje de error cuando se hace referencia al nombre del evento en un archivo MXML.

Cualquier componente puede registrar un detector de eventos en ActionScript mediante el método `addEventListener()`.

Stopping event propagation.

En cualquier de las fases del evento (capture, target, or bubbling) se pueden detener los eventos, llamando a uno de los siguientes metodos :

- [`stopPropagation\(\)`](#)
- [`stopImmediatePropagation\(\)`](#)

Se puede llamar a cualquiera de los dos métodos para evitar que el mismo siga su camino a través del flujo. Los dos métodos son casi idénticos, el método `stopPropagation()` impide que el evento avance hasta el siguiente nodo, pero sólo después de que se que todos los demás detectores de eventos del nodo se hayan ejecutado.

El método `stopImmediatePropagation()` termina todos los disparadores de eventos del objeto.