

OpenLaszlo: Come definire la LOGICA

Scripting

- ❑ LZX permette di programmare la logica dell'interfaccia per mezzo di un **linguaggio procedurale** basato su *JavaScript*
- ❑ Il codice viene eseguito sul **client**, *quindi attenzione ad un carico eccessivo*
- ❑ L'**interprete** del linguaggio è interno al Flash Player (*no JavaScript del browser*), quindi verrà eseguito su ogni client

❑ Sintassi essenziale (simile a C e Java)

- **Ogni “cosa” è un oggetto**

(String, Number, Array, Boolean, Date, Function, Math, Object...)

- **Strutture di controllo e condizionali**

if (...condizione...) { ...istruzioni... } **else** { ...istruzioni... }

while (...condizione...) { ...istruzioni... }

for (..... ; ;) { ...istruzioni... }

switch (...espressione...) {

case CASE_1 : ...istruzioni...; **break**;

default : ...istruzioni...;

}

- **Funzioni e procedure**

function (arg1, arg2, arg3) {

 ...istruzioni...; **return** ...espressione...;

}

❑ **Attenzione ai caratteri illegali** in XML!!!

```
<script>  
  if ( this.x < 100 ) Debug.info( this.x );  
</script>
```

Questo carattere va in contrasto con
l'eventuale apertura di un nuovo tag!!!

Soluzioni:

- sostituire i caratteri incriminati con **entità HTML**

```
<script>  
  if ( this.x &lt; 100 ) Debug.info( this.x );  
</script>
```

- inglobare il codice in una **sezione CDATA**

```
<script>  
  <![CDATA[  
    if ( this.x < 100 ) Debug.info( this.x );  
  ]]>  
</script>
```

- ❑ Per dettagli e differenze da *JavaScript standard* vedere nella guida:

<http://www.openlaszlo.org/lps4.2/docs/developers/ecmascript-and-lzx.html>

- ❑ Le applicazioni OpenLaszlo possono include **codice procedurale**:

- nei metodi di un oggetto (tag `<method>`)
- negli eventi di un oggetto (tag `<handler>`)
- nei tag `<script>`
- nelle espressioni all'interno dei *constraints* per il calcolo del valore di un attributo

I tag *<script>*

- ❑ Il codice inserito all'interno dei tag *<script>* viene **eseguito immediatamente**
- ❑ Possono comparire solo nel *<canvas>*
- ❑ Utili per definizioni/impostazioni iniziali
- ❑ Esempio:

<script>

```
Debug.write( myView.xoffset )  
myText.setAttribute("text", "hello world");
```

</script>

I tag *<method>*

- ❑ Il codice inserito dentro un tag *<method>* viene **eseguito quando viene richiamato** con la chiamata *nomeOggetto.nomeMetodo()*

- ❑ Sintassi:

<method name="nomeMetodo" args="arg_1, ..., arg_n">

...codice del metodo...

</method>

- ★ Un metodo può avere **zero o più argomenti** in ingresso e
- ★ **può restituire un valore** in uscita con il comando *return* espressione;

❑ E' possibile l'**overloading dei metodi**:

- il metodo originale (*o della superclasse*) non viene richiamato una volta che viene ridefinito
- per richiamarlo esplicitamente si usa la sintassi **super.nomeMetodo(...)**

❑ Esempi di metodi:

- **<view>**
getAttribute(), setAttribute()
play(), stop(), seek(), setVolume(), unload(), destroy()
- **<animator>**
doStart(), pause(), stop()
- **<window>**
open(), close(), bringToFront(), sendToBack()

I tag *<handler>*

❑ Paradigma di **programmazione ad eventi**:

- normalmente l'esecuzione delle istruzioni segue percorsi fissi, che si ramificano soltanto in ben determinati punti predefiniti dal programmatore (*if, switch ecc*)
- nel paradigma ad eventi, invece, **il flusso del programma è largamente determinato dal verificarsi di eventi esterni** (*quando viene premuto il mouse, al termine del caricamento di una immagine ecc*)

❑ Il codice inserito dentro un tag *<handler>* viene **eseguito al verificarsi delle condizioni** che determinano quel particolare evento

❑ Sintassi:

<handler name="nomeEvento">

...codice per questo evento...

</handler>

❑ Esempi di eventi:

- **<view>**

onclick, ondblclick, onmouseover, onmouseout
onfocus, onkeydown, onkeyup
oninit, onload, onerror, ontimeout

- **<animator>**

onstart, onstop, onpaused

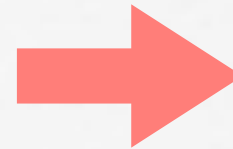
- **<text>**

ontext

❑ **Eventi per la modifica degli attributi:**

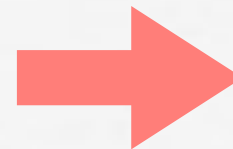
ogniqualevolta si modifica un attributo con il metodo `setAttribute()` viene generato un conseguente evento:

`myView.setAttribute('x',this.x+10)`



Evento **onx**
per l'oggetto *myView*

`myWin.setAttribute('width',400)`



Evento **onwidth**
per l'oggetto *myWin*

- Questi eventi stanno alla base del meccanismo interno dei *constraints*
- Per non scatenare i conseguenti eventi, basta modificare gli attributi direttamente:

`myView.x = this.x + 10;` oppure `myWin.width = 400;`

Note su *error handling*

Quando si effettua una richiesta HTTP per un dato (*immagini, suoni o dati XML*) ci si deve sempre ricordare di **controllare l'esito della richiesta**:

- **onload/ondata** viene generato al termine della ricezione corretta del dato richiesto;
- **onerror** è lanciato al verificarsi di errori durante la trasmissione o se il formato del dato è errato;
- **ontimeout** viene generato dopo che non si ricevono dati per un periodo di tempo prestabilito (default 30s).

3° esercitazione

1. Creare una classe che implementi un **pulsante** che:

- si ingrandisca al passaggio del mouse;
- visualizzi icone differenti a seconda dello stato;
- emetta dei suoni alle eventuali azioni del mouse.

2. Creare una **finestra** che:

- non possa essere spostata oltre un box posizionato in alto;
- sia pilotata da un pulsante;
- si apra/chiuda con una animazione articolata;
- il cui stato (aperto/chiuso) sia correttamente rappresentato dal pulsante.

3. Dentro la finestra, caricare una **immagine** a scelta tra quelle presenti in una **combobox**:

- visualizzare correttamente l'immagine (*dimensioni, clipping, stretching*);
- gestire eventuali errori nel caricamento;
- visualizzare le informazioni dell'immagine (dimensioni, filename ecc).

4. Dare la possibilità di effettuare lo **zoom** ed il **pan** tramite 6 pulsanti grafici (es.      )

5. Provare a caricare **più immagini** e gestirle in un layout, mantenendo l'aspect-ratio ed applicando i parametri di zoom/pan a tutte