



beeblos®

Autore: **Gianluca Lovino**[embersun@gmail.com](mailto:embersun@gmail.com)

## SOMMARIO

Beeblos .....	1
Il file .xml.....	2
L'interfaccia .....	4
Livello dell'interfaccia .....	4
Livello del motore di ricerca .....	5
Livello del codice.....	5
Codice del motore di ricerca.....	7

---

## BEEBLOS

### Cos'è Beeblos?

Beeblos è un'applicazione flash che visualizza un insieme di dati estratti da un file .xml esterno.

### Quali e quanti tipi di dati possono essere visualizzati?

Non c'è un limite al tipo e al numero di dati che si possono visualizzare con Beeblos. Tutte le informazioni sono codificate nel linguaggio xml che, come è noto, non specifica la natura dei dati ma ne definisce solo la struttura. Pertanto, gli elementi visualizzabili possono essere indifferentemente testi, immagini, video e musica: è sufficiente istanziarli all'interno del file xml che si vuole utilizzare. In questo senso, Beeblos è un data displayer, ovvero un visualizzatore di dati di qualunque natura.

### Qualche esempio?

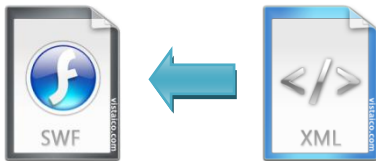
Beeblos può essere usato per realizzare un elenco bibliografico online, un catalogo di ricette illustrate e guidate, una galleria di immagini ... Nell'esempio pubblicato, ho realizzato un catalogo musicale dedicato ai Paradise Lost.

### Com'è stato scritto il codice di Beeblos?

Beeblos è stato sviluppato su Macromedia Flash MX 8 e supporta Actionscript 2.0. Ho rilasciato l'attuale versione iniziale 1.0 il 29/3/2009 anche se l'idea di base è molto più datata e risale allo sviluppo della Biblioteca Online creata per conto del Liceo Linguistico "Guido D'Arezzo" di Ruvo di Puglia (luglio 2006).

## Come funziona Beeblos?

Beeblos è un file .swf creato con Flash MX. Esso legge i dati strutturati contenuti in un file .xml esterno, li importa e quindi li visualizza:



[icone da [www.vistaicons.com](http://www.vistaicons.com)]

---

## IL FILE .XML

La struttura del file .xml è illustrata qui di seguito.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<dati>
```

```
  <elemento autore=" " lavoro=" " anno=" " casadiscografica=" " tracklist="" duratabrani=" " descrizione="
  " immagine=" " video=" " />
```

```
</dati>
```

Il nodo primario <dati> viene popolato di nodi – figlio <elemento> a ciascuno dei quali viene assegnato un gruppo di attributi. Sono proprio questi ultimi a contenere tutte le informazioni che si vogliono visualizzare. In particolare, gli attributi autore, lavoro, anno, casa discografica, tracklist, durata brani e descrizione contengono solo testo, invece immagine e video contengono il percorso in cui si trovano le immagini e i file swf.

Ad esempio, il primo record che ho preparato è fatto così:

```
<elemento
```

```
  autore="Paradise Lost"
```

```
  lavoro="In Requiem"
```

```
  anno="2007"
```

```
  casadiscografica="Century Media"
```

```
  tracklist="01. Never for the Damned
```

```
02. Ash & Debris
```

```
03. The Enemy
```

```
04. Praise Lamented Shade
```

```
05. Requiem
```

```
06. Unreachable
```

```
07. Prelude to Descent
```



08. Fallen Children

09. Beneath Black Skies

10. Sedative God

11. Your own Reality

12. Missing

13. Silent in Heart

14. Sons of Perdition"

duratabrani=""

descrizione=""

immagine="inrequiem.jpg"

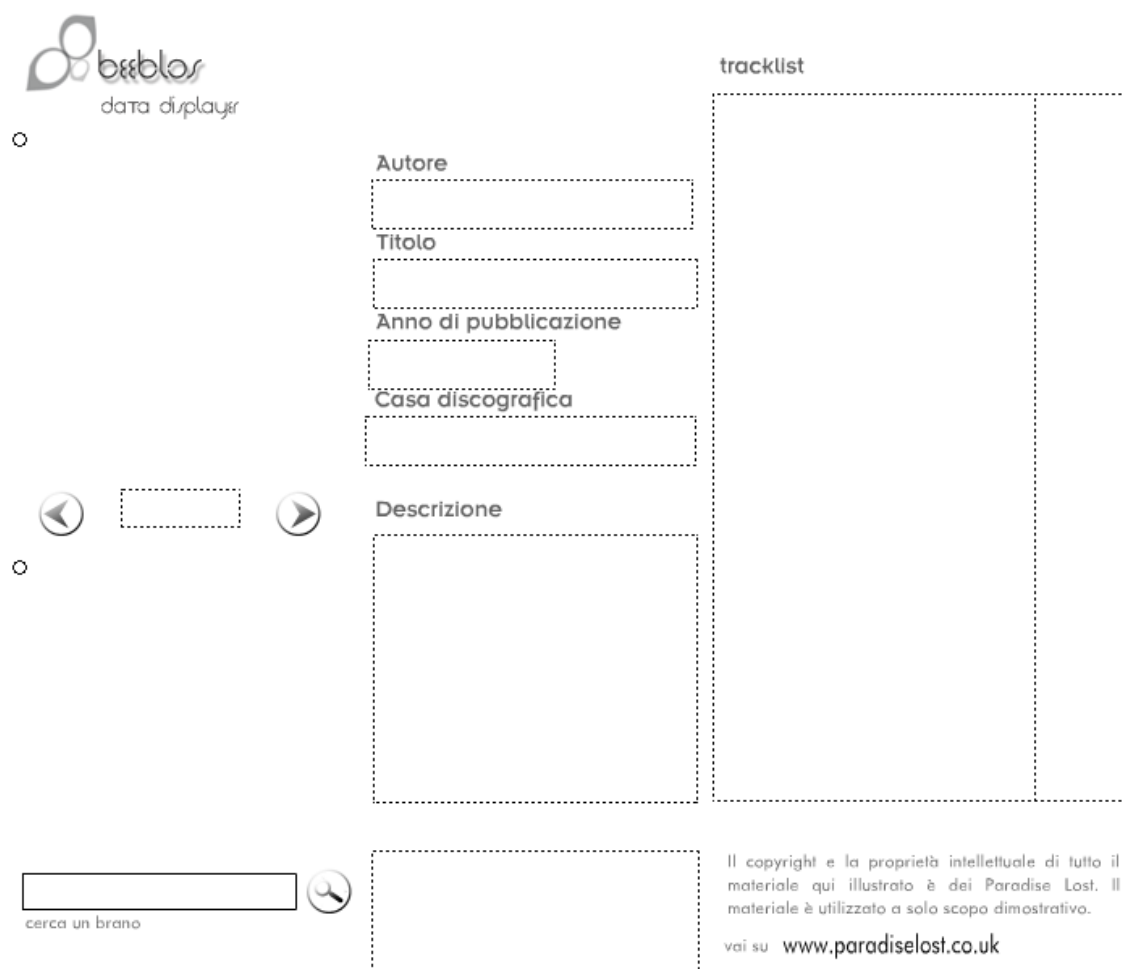
video="the enemy.swf"

/>

Gli altri sono stati riempiti in modo analogo.

## L'INTERFACCIA

Nella figura in basso è illustrata la struttura dell'applicazione come mostrata nell'IDE di Flash MX



L'applicazione è costituita da 3 livelli, uno dedicato all'interfaccia dei dati visualizzabili, uno che accoglie il motore di ricerca interno e uno in cui è riportato il codice di funzionamento.

### LIVELLO DELL'INTERFACCIA

Contiene i campi etichettati con Autore, Titolo, Anno di pubblicazione, Casa discografica, descrizione, tracklist e la colonna a destra di quest'ultima che conterrà la durata dei brani. Sono tutti testi dinamici cui sono assegnati i nomi di variabile `autore_txt`, `titolo_txt`, `anno_txt`, `casadiscografica_txt`, `descrizione_txt`, `tracklist_txt`, `duratabrani_txt` rispettivamente.

Sulla sinistra (negli spazi vuoti che hanno un pallino vuoto in alto a sinistra) ci sono i due istanze degli holder di immagini e di video cui sono assegnati i nomi di istanza `imageLoader` e `videoLoader`.

Infine, tra i due holder ci sono i pulsanti `avanti_btn` e `indietro_btn` che racchiudono il testo dinamico `elementoNum_txt` che visualizzerà il record attuale rispetto al numero totale di record disponibili.

## LIVELLO DEL MOTORE DI RICERCA

Fanno parte di questo livello il testo di input `query`, il pulsante `cerca_btn` e il testo dinamico `risultato`. Nell'esempio che ho pubblicato, il motore serve per ricercare la posizione di un brano all'interno di un lavoro discografico. L'idea è che, digitando anche una sola parola di una canzone, il risultato della ricerca fornisca l'elenco degli album in cui questa parola è contenuta nel titolo di una canzone. Il codice per il funzionamento è associato al pulsante `cerca_btn`.

Si noti che in questa versione, il motore è limitato dal fatto di richiamare la funzione `inedxOf()` che è case-sensitive. Dunque è possibile che la ricerca manchi il risultato solo per aver scritto un brano con la lettera minuscola piuttosto che con la maiuscola. Questo difetto verrà sistemato nelle prossime versioni.

Completano il livello il pulsante link che rimanda al sito dei Paradise Lost e un disclaimer (doveroso). Infatti tutto il materiale che Beeblos illustra è stato preso dal sito ufficiale del gruppo [www.paradiselost.co.uk](http://www.paradiselost.co.uk) ed è appena il caso di osservare che tutti i diritti e la proprietà intellettuale dei lavori sono di esclusiva proprietà di tali autori: in Beeblos utilizzo alcuni di questi lavori solo a scopo illustrativo dell'applicazione.

## LIVELLO DEL CODICE

Il codice che fa funzionare Beeblos è commentato qui di seguito.

Creiamo un'istanza della classe XML tramite la parola chiave `new` e chiamiamola `catalogo`:

```
var catalogo:XML = new XML();
```

Tutti gli spazi bianchi del codice XML devono essere ignorati:

```
catalogo.ignoreWhite = true;
```

La sorgente dei dati XML che funzionerà da factory per l'applicazione sono contenuti nel file `dati.xml`, da cui l'istanza `catalogo` deve caricarli:

```
catalogo.load("dati.xml");
```

Inizializziamo tre variabili numeriche per le routine dei pulsanti:

```
var record:Number = 0;
```

```
var totale:Number = 0;
```

```
var corrente:Number = 0;
```

L'evento `onLoad` deve fare da trigger per la funzione di ricerca e visualizzazione dei dati:

```
catalogo.onLoad = function(success) {
    if (success) {
```

Le caselle di testo `autore_txt`, `lavoro_txt`, `anno_txt`, `casa discografica_txt`, `tracklist_txt`, `durata brani_txt` e `descrizione_txt` vengono riempite con dati contenuti negli attributi appropriati del file `.xml` sorgente.

Tutti i campi sono indicizzati dalla variabile `[_root.record]` che, inizialmente, è settata a 0.

```
    autore_txt = this.firstChild.childNodes[_root.record].attributes.autore;
```

```
    lavoro_txt = this.firstChild.childNodes[_root.record].attributes.lavoro;
```

```

anno_txt = this.firstChild.childNodes[_root.record].attributes.anno;

casadiscografica_txt = this.firstChild.childNodes[_root.record].attributes.casadiscografica;

tracklist_txt = this.firstChild.childNodes[_root.record].attributes.tracklist;

duratabrani_txt = this.firstChild.childNodes[_root.record].attributes.duratabrani;

descrizione_txt = this.firstChild.childNodes[_root.record].attributes.descrizione;

```

Qui invece vengono riempiti gli holder delle immagini e degli eventuali filmati i cui indirizzi sono specificati negli attributi appropriati del file .xml sorgente:

```

loadMovie(this.firstChild.childNodes[_root.record].attributes.immagine, imageLoader);

loadMovie(this.firstChild.childNodes[_root.record].attributes.video, videoLoader);

```

In questa sezione, inizia la codifica del comportamento dei pulsanti indietro\_btn, avanti\_btn e link\_btn.

La variabile totale definisce il numero di nodi figlio presenti nel documento: in pratica, conta il numero di dati (lavori discografici, in questo caso) complessivamente disponibili. La variabile corrente, invece, si riferisce al nodo figlio attualmente visualizzato.

```

totale = this.firstChild.childNodes.length;

corrente = record+1;

```

Il campo di testo elementoNum\_txt viene riempito con le due variabili precedenti in modo da orientare l'utente sul quale lavoro sta consultando e quanti ve ne sono complessivamente.

```

    elementoNum_txt = corrente+" di "+totale;
}
};

stop();

```

Il pulsante indietro\_btn riduce l'indice record di una unità alla volta; se il lavoro attualmente visualizzato è quello iniziale, il cui record è 0 [if (record == 0)], al clic sul pulsante viene visualizzato l'ultimo record disponibile [record = totale-1]. Successivamente carica i record dal file dati.xml [catalogo.load("dati.xml")].

```

indietro_btn.onPress = function() {

    if (record == 0) {

        record = totale-1;

    } else {

        record -= 1;

    }

    catalogo.load("dati.xml");
}

```

Analogamente il pulsante `avanti_btn` aumenta l'indice `record` di una unità alla volta; se il lavoro attualmente visualizzato è quello finale, il cui `record` è 0 [if (`record +1== totale`)], al clic sul pulsante viene visualizzato il primo record disponibile [`record = 0`]. Successivamente carica i record dal file `dati.xml` [`catalogo.load("dati.xml")`].

```
avanti_btn.onPress = function() {
    if (record+1 == totale) {
        record = 0;
    } else {
        record += 1;    }
}
```

```
catalogo.load("dati.xml");
```

Questo è un semplice pulsante contenente un link,

```
link_btn.onPress = function ()
{
    getURL("http://www.paradiselost.co.uk", "_blank");
}
};
```

## CODICE DEL MOTORE DI RICERCA

Al pulsante `cerca_btn` è associata una routine per la ricerca del titolo di un brano (o di una sua parte) all'interno di un lavoro discografico.

```
on (release)
```

```
{
```

Creiamo una nuova classe XML denominata `sorgente` e agganciamo il file `dati.xml` per la lettura dei record:

```
var sorgente:XML = new XML();
sorgente.ignoreWhite = true;
sorgente.load("dati.xml");
sorgente.onLoad = function(success)
{
    if(success)
    {
        var nodo = this.firstChild.childNodes;
        for(var i = 0; i < nodo.length ; i++)
```



```
{
```



Ad ogni ciclo, l'applicazione esamina l'attributo `tracklist` e controlla se il testo immesso nel campo di ricerca `query` è in esso contenuto `[brano.indexOf(word.text)!=-1]` dove `[brano= nodo[i].attributes.tracklist]`. Questa condizione è posta in AND con la richiesta che il campo di `query` non sia vuoto `[query.text!=""]`.

Se l'AND logico delle due richieste restituisce 1 (vero), allora il campo di testo risultato viene riempito con il nome del lavoro, dove `[lavoro = nodo[i].attributes.lavoro]`, altrimenti non accade nulla.

```
var lavoro = nodo[i].attributes.lavoro;  
  
var brano = nodo[i].attributes.tracklist;  
  
if(brano.indexOf(query.text)!=-1 && query.text!="")  
    {risultato.text+=lavoro+"\n";}  
}
```

```
}
```

```
};
```

```
}
```