

pixlib

ActionScript 2.0 framework

pixlib (short) story

At the beginning, it was designed to develop games.

Private beta launched at the end of 2004.

It becomes a generic framework.

Pixlib is NOT (anymore)

Classes library to develop games.

Components framework.

Complete framework to fulfil every need.

pixlib IS designed to

Be light, easy and powerful.

Encourage polymorphism, strong-typing and debugging.

Be Flash player 6 compatible.

What we will see together.

log

events

data.libs

transitions

commands

log

Package designed to handle logging.

Logger.LOG outputs any data to your debugging console.

```
Logger.LOG( mc._alpha );
```

LogLevel acts as a filter.

Logger.getInstance().addLogListener

It will add any class (implementing **LogListener** interface) for receiving log events.

use Luminic Flash inspector

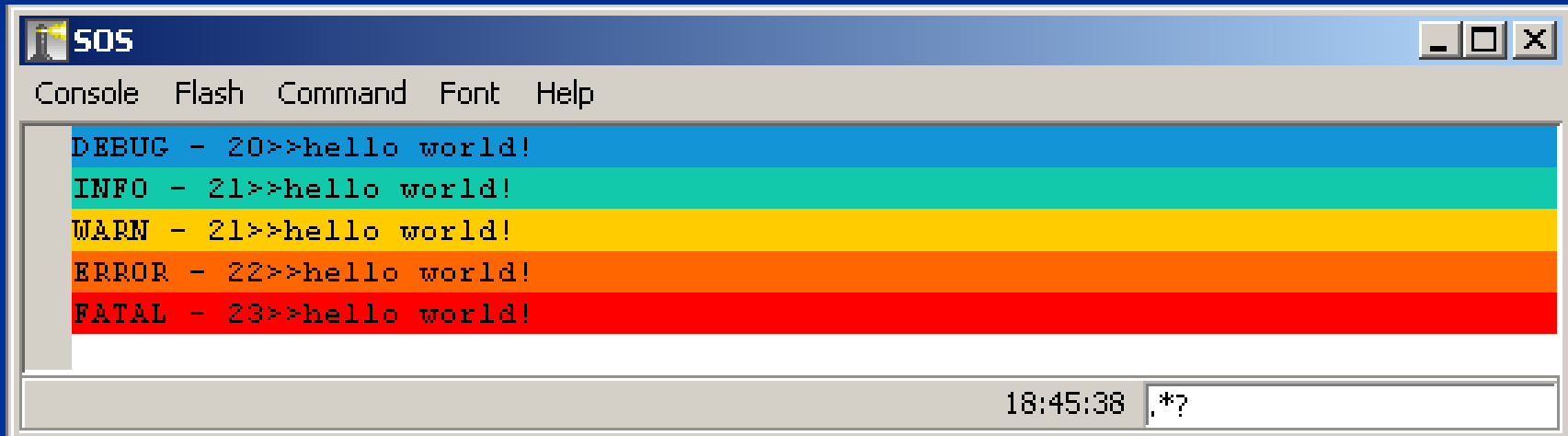
```
com.bourre.utils.LuminicTracer.getInstance();
```



<http://www.luminicbox.com/blog/?page=post&id=2>

use Powerflasher SOS

```
com.bourre.utils.SOSTracer.getInstance();
```



<http://sos.powerflasher.com/english.html>

**or use your own debugging
console**

Just implement LogListener interface.

```
public function onLog( e : LogEvent ) : Void;
```

**Use LogEvent properties to retrieve
data.**

LogEvent.content

LogEvent.level

Log basical example

```
import com.bourre.log.*;
import com.bourre.utils.*;

// Adds Flash Inspector as listener
Logger.getInstance().addLogListener(
    LuminiTracer.getInstance() );
// Adds Powerflasher SOS as Listener
Logger.getInstance().addLogListener(
    SosTracer.getInstance() );

// Basical message
Logger.LOG( "Hello world" );
// Message with filter level
Logger.LOG( "Error message", LogLevel.ERROR );
```

events

EventBroadcaster class is the cornerstone of events package.

For a quick start, you can use Macromedia's **EventDispatcher** polymorphism.

```
addEventListener(t:String, oL) : Void;  
removeEventListener(t:String, oL) : Void;  
dispatchEvent(o:Object) : Void;
```

event type

Each event class implements IEvent interface.

```
public function getType() : EventType;  
public function getTarget();  
public function setType( e : EventType ) : Void;  
public function setTarget( target ) : Void;
```

BasicEvent is the most basic implementation of this interface.

If you need to build an event class, **extend it.**

EventBroadcaster complete API

```
dispatchEvent( o ) : Void  
addEventListener( event : String, o ) : Void  
removeEventListener( event : String, o ) : Void  
removeAllEventListeners( event : String ) : Void
```

```
addListener( o ) : Void  
removeListener( o ) : Void  
removeAllListeners() : Void
```

```
getListenerArray( event : String ) : ListenerArray  
listenerArrayExists( event : String ) : Boolean  
isEmpty() : Boolean
```

```
broadcastEvent( e : IEvent ) : Void
```

automatical event proxy

```
// register with proxy method
myScroller.addEventListener (
    ScrollerModel.onScrolleVENT,
    this,
    refreshView
);

// unregister the proxy method
myScroller.removeEventListener (
    ScrollerModel.onScrolleVENT, this );

// or unregister like that
myScroller.removeListener( this );
```

deploy event system with inheritance

```
class ScrollerModel extends EventBroadcaster
{
    public static var onScrolleVENT = new EventType("onScroll");

    public function ScrollerModel()
    {
        super( this );
    }
}
```

deploy event system with composition

```
class ScrollerModel
{
    private var _oEB : EventBroadcaster;
    public static var onScrollEVENT = new EventType("onScroll");

    public function ScrollerModel()
    {
        _oEB = new EventBroadcaster( this );
    }

    public function addEventListener( t : EventType, o ) : Void
    {
        _oEB.addEventListener.apply( _oEB, arguments );
    }

    public function removeEventListener( e : EventType, oL ) : Void
    {
        _oEB.removeEventListener( t, o );
    }
}
```

data.libs

Package designed to handle file loading.

- **GraphicLib**. Handles SWF loading.
- **XMLToObject**. Handles XML files loading and automatical object deserialization.
- **VideoDisplay**. Handles FLV loading and playing.
- **Libstack**. Polymorphic stack implementation (FIFO). You can enqueue

AbstractLib

Abstract class for handling file loading.

Features available in this class :

Name identifier and Url accessors.

Event system built-in with timeout handling.

Errors logging

Content getter.

LibEvent with getPerCent() and getName() accessors.

Extend it to add new file formats.

Graphiclib example

```
import data.libs.GraphicLib;
import data.libs.LibEvent;

// Callback method
function showIntro( e : LibEvent ) : Void
{
    // e.getView().play();
}

gl = new GraphicLib( mcTarget, nDepth );

gl.addEventListener( GraphicLib.onLoadInitEVENT,
    this, showIntro );

gl.load( "intro.swf" );
```

VideoDisplay basical example

```
import com.bourre.data.libs.LibEvent;
import com.bourre.medias.video.VideoDisplay;

// Callback method
function onVideo( e : LibEvent )
{
    // do something
}

var p:VideoDisplay = new VideoDisplay( holder,
    holder.videoObject );

p.addEventListener(VideoDisplay.onLoadInitEVENT,
    this, onVideo);

p.load( "file.flv" );
```

about XMLToObject

XMLToObject loads XML files and decorates a target object with typed data.

Each file's node becomes a (targeted object) property with specified type (with type attribute).

List of implemented types:

String

Number

Boolean

Array

Point

Class

XML file example

```
<config>
```

```
  <cost type= "number"> 12 </cost>
```

```
  <url type= "string" name= "blog"> www.tweenpix.net </url>
```

```
  <list type= "array"> "item0', "item1", true, 44 </list>
```

```
  <visible type = "boolean"> true </visible>
```

```
  <dvds>
```

```
    <film type= "string"> film0</film>
```

```
    <film type= "string"> film1</film>
```

```
  </dvds>
```

```
  <position type= "point"> 3, 7 </position>
```

```
  <event type= "class"> "com.bourre.events.EventType",  
  "onTest" </event>
```

```
</config>
```

XMLToObject basic example

```
import com.bourre.data.libs.*;

// callback method
function run( e : XMLToObjectEvent ) : Void
{
    doSomethingWith( e.getObject() );
}

_xto = new XMLToObject( new Object() );
_xto.addListener( XMLToObject.onLoadInitEVENT,
    this, run );
_xto.load( "config.xml" );
```

Add new types to XMLToObject

```
import com.bourre.data.libs.*;

function getSquare( data ) : Square
{
    // return parsed and casted data
}

_xto = new XMLToObject ( new Object() );
_deserializer = _xto.getDeserializer();

_deserializer.addType("square", this, getSquare );
```

You can build your own deserializer.
Just implement **IXMLToObjectDeserializer**

LibStack example

```
import com.bourre.data.libs.*;

_g1 = new GraphicLib( mcTarget, 10);
_vd = new VideoDisplay( mcTarget );

_lib = new LibStack();

_lib.enqueue( _g1, "uis", "uis.swf" );
_lib.enqueue( _vd, "video", "file.flv" );

_lib.addEventListener( LibStack.onLoadCompleteEVENT,
    this, launchApplication );

_lib.execute();
```

Gimme the tempo!

IFrameBeacon is a metronome interface.

Metronome : Clicking pendulum indicates the exact tempo of a piece of music.

start() : Void

stop() : Void

isPlaying() : Boolean

addFrameListener(o : IFrameListener) :
Void

removeFrameListener(o : IFrameListener) :
Void

Listen to the beat now!

Implementing **IFrameListener** means :
« *I'm ready to move on the beat, to act on the tempo.* »

```
public function onEnterFrame() : Void
{
    // Do something on each click.
}
```

FPSBeacon

Broadcasts each click on each player's frame.

Use **FPSBeacon.getInstance()**
Singleton implementation.

```
import com.bourre.transitions.FPSBeacon;  
FPSBeacon.getInstance().addFrameListener(  
    animation );
```

Global play and pause

```
FPSBeacon.getInstance().stop();
```

```
FPSBeacon.getInstance().play();
```

Will stop and start/resume the process of all the classes listed below:

CommandFPS

CommandManagerFPS

BasicTweenFPS

TweenFPS

MSBeacon

Broadcasts each click on specified framerate.

```
import com.bourre.transitions.MSBeacon;  
  
var msb:MSBeacon = new MSBeacon();  
msb.setFPS( 25 );  
msb.addFrameListener( animation );
```

Global access with
MSBeacon.getInstance().

Basic TweenMS and TweenMS use

IBasicTween

IBasicTween is the most basic tween interface of the framework.

start() : Void

stop() : Void

setEasing(f : Function) : Void

BasicTweenMS and **BasicTweenFPS** are concrete implementations of **IBasicTween**.

BasicTweenFPS uses **FPSBeacon**.

BasicTweenMS uses **MSBeacon**.

ITween

ITween have event system built-in.

TweenEventType list:

onStartEVENT

onStopEVENT

onMotionFinishedEVENT

onMotionChangedEVENT

```
ITween.addEventListener( TweenEventType, listener);
```

TweenMS and **TweenFPS** classes are concrete implementations of **ITween**.

Tween object's property

```
import com.bourre.transitions.*;

var t:TweenFPS = new TweenFPS( this.mc, "_x", 550, 90 );

t.addEventListener( TweenMS.onMotionFinishedEVENT, this,
    restartTween);

t.start();

// Callback method
function restartTween( e : TweenEvent ) : Void
{
    e.getTween().start();
}
```

Tween setter

```
import com.bourre.transitions.*;

// Setter
function setX( n : Number ) : Void
{
    this.mc._x = n;
}

t = new TweenMS( this, setX, 550, 3000, 0 );

// Change framerate
MSBeacon.getInstance().setFPS( 40 );

t.start();
```

commands

Libs, transitions and commands package use command pattern.

```
public function execute( e: IEvent ) : Void
{
    // Do something when called.
}
```

AbstractLib. All inherited classes have **execute()** method.

ITween. All Tween classes have **execute()** method.

Delegate. You can transform any object's method call to command.

Batch. Macro-command implementation.

Batch

Handles several commands as one command.

```
import com.bourre.commands.*;

var b:Batch = new Batch();

b.addCommand( myAlphaTween );
b.addCommand( myPositionTween );

b.execute();
```

Delegate

It works exactly as `mx.utils.Delegate` does.

So, what are the new additions ?

You can use constructor to encapsulate function call.

You can pass parameters and add some more further with `setArguments` and `addArguments` methods.

It Implements `Command` interface.

It Implements `IEventListener` interface.

CommandFPS

Store Command objects and execute each one on each frame.

```
import com.bourre.commands.*;

var t:CommandFPS = new CommandFPS();
var d:Delegate = new Delegate(this, test);

function test(s:String) : Void
{
    trace("hello world");
}

t.push( d );
```

CommandMS

Store Command objects and loop their execution at specified speed.

```
import com.bourre.commands.*;

var t:CommandMS = new CommandMS();
var d:Delegate = new Delegate(this, test);

function test(s:String) : Void
{
    trace("hello world");
}

t.push( d, 1000 );
```

more features

Collections package

Iterator package

Sound package

```
mySoundFactory.init( soundLib );  
mySoundFactory.addSounds( [ "loop", "scroll", "quick" ] );
```

Structures package

some advanced features

HashCodeFactory

AbstractFactory

GraphicLibLocator

ViewHelper

FrontController

MVC package

What's next ?

Brush up the lack of documentation !!!

(Top Priority)

CommandSequencer time-based and event-based.

Finish some refactoring and tests on **FLVPlayer**, **FLVLoop**, **FLVLoopEvent** and **FLVLoopManager** classes.

Polish the Event bubbling API part.

Extend **AbstractLib** with more file formats.

Useful links

Official url:

<http://www.osflash.org/pixlib>

Mailing-list:

http://osflash.org/mailman/listinfo/pixlib_osfl

SVN repository

<http://svn.sourcesecure.co.uk/osflash/pixlib/trunk>

My blog:

<http://www.tweenpix.net/blog>