Developpez.com
Club des développeurs

Comprendre les bases du framework PixLib

par Xavier Martin (aka zeflasher or xxlm)

Date de publication : 04/07/2007

Dernière mise à jour :

Traduction de l'**article original en anglais** par **Grégory Dumas**
Cet article vous permettra de vous initier au framework PixLib, de comprendre son architecture, son fonctionnement, ...

I - Introduction
II - Pré-requis
III - Le fla
IV - Le code
    IV-A - Application
    IV-B - La ViewList, les vues et comment les utiliser
    IV-C - La ModelList et le ModelClock
    IV-D - L'EventList et le XEventBroadcaster
    IV-E - Le FrontController
    IV-F - Les commandes
V - Conclusion

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

I - Introduction

Ce tutoriel est fait pour vous donner les bases vous permettant d'utiliser l'excellent framework de Francis Bourre : PixLib. Pour cela, vous devez connaître au minimum ce que sont les motifs de conceptions **MVC** et Commande.

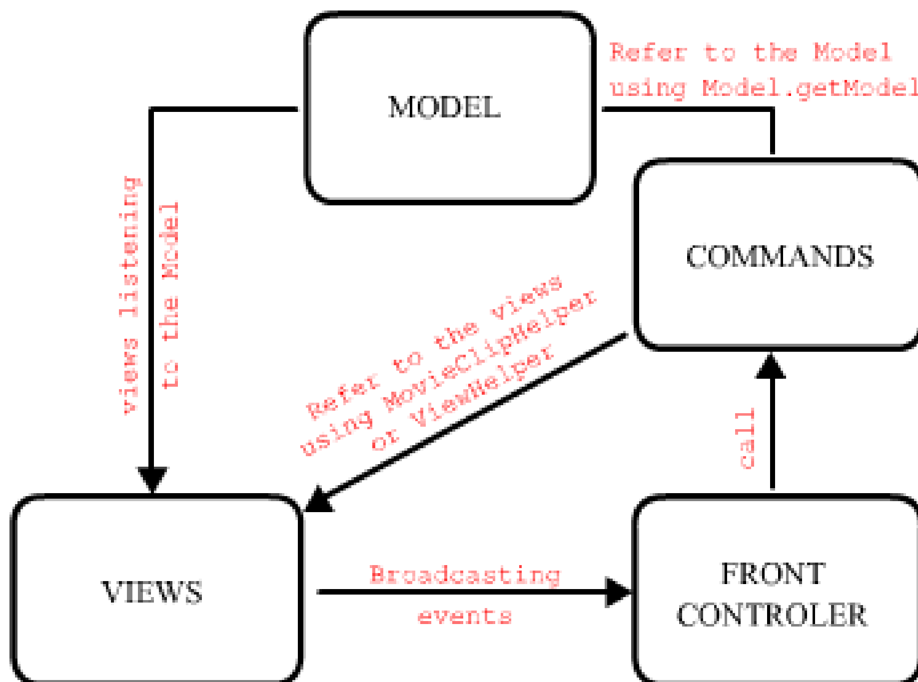
Généralement, en utilisant le motif MVC, chaque vue a un contrôleur qui met à jour le modèle (si nécessaire) et toutes les vues qui doivent être changées. Ceci peut être réalisé à partir de la bibliothèque PixLib en utilisant la classe dans com.bourre.mvc. Mais ce tutoriel ne traitera pas ce modèle MVC « habituel ».

Francis a ajouté à son framework quelque chose de réellement adaptable; appelé FrontController. Ce contrôleur reçoit les événements de chaque vue et appelle les Commandes associées.

A présent, votre code est vraiment propre et pratique : une classe = une commande, donc votre code destiné à une action peut en être réellement simplifié, de plus, vous pouvez voir toutes les actions qui peuvent être exécutées en regardant le code du "FrontController", tout est dedans!

De plus, vous pouvez voir toutes les actions qui peuvent être exécutées en regardant le code du ForntController, tout s'y trouve. Vous n'aurez pas besoin de regarder dans X classes pour essayer de trouver le code nécessaire pour faire une action particulière, vous irez droit au but.

Voici un schéma représentant cette structure :



Trêve de blabla, rentrons dans le vif du sujet. Je vais vous expliquer qui est qui et qui fait quoi à partir d'exemples. Pour cela, nous allons essayer de réaliser un chronomètre avec affichage analogique et numérique.

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

II - Pré-requis

Téléchargez la dernière version du framework PixLib à cette adresse : **http://osflash.org/projects/pixlib**

Téléchargez les sources du tutoriel : **ici** (**miroir**)

Enfin, téléchargez la maquette du projet : **ici** (**miroir**)

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

III - Le fla

Pour commencer, examinons le fla. J'ai créé 3 MovieClips. Un pour l'affichage digital, un autre pour l'affichage analogique et un dernier pour la barre d'outils où se trouvent les boutons "Stop" et "Start". Ces MovieClips seront utilisés comme vues (interfaces).

Dans la première (et unique) image de la timeline du fla, vous avez ce code :

```
import net.webbymx.projects.tutorial01.Application;
var apz : Application = new Application( this );
```

Ici, nous importons Application.as qui est le point d'entrée du code, et en créons une instance en utilisant l'objet _root (ou _level0). Nous verrons plus tard dans quel but.

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

IV - Le code

Tout d'abord, regardons comment un projet utilisant le framework PixLib est structuré.

Pour vous aider, j'ai réalisé une maquette pour vous. Vous trouverez les fichiers nécessaires pour développer une application utilisant le modèle MVCC (MVC + Commande) avec la librairie PixLib.

Voici la structure :

et une rapide explication :

- **commands** : contient toutes les classes implémentant l'interface com.bourre.commands.Command. Contient également le FrontController.
- **events** : contient toutes les classes définissant un nouveau type d'événement (héritant de BasicEvent). Contient également votre propre classe EventBroadcaster et la classe EventList qui est seulement une énumération de tous les événements existants.
- **models** : contient le modèle et la ModelList
- **services** : contient chacune de classes qui communiquent avec des données externes (comme server, xml, ...)
- **uis** : contient toutes les classes qui définissent vos Interfaces Utilisateur
- **views** : contient toutes les classes qui héritent des classes MovieClipHelper ou de ViewHelper et qui seront liées à un MovieClip. Contient également la classe ViewList
- **vos** : Contient toutes les classes définissant vos propres objets
- **Application.as** : C'est la classe principale de votre application qui crée tout !

IV-A - Application

Désormais, regardons la classe Application.as :

```
//  Debug
import com.bourre.log.Logger;
import com.bourre.log.LogLevel;
import com.bourre.utils.LuminicTracer;

//  Views
import net.webbymx.projects.tutorial01.views.ViewAnalog;
import net.webbymx.projects.tutorial01.views.ViewDigital;
import net.webbymx.projects.tutorial01.views.ViewTools;
```


Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

```
// Controller
import net.webbymx.projects.tutorial01.commands.Controller;

// Models
import net.webbymx.projects.tutorial01.models.ModelClock;

// Personnal EventBroadcaster
import net.webbymx.projects.tutorial01.events.XEventBroadcaster;


class net.webbymx.projects.tutorial01.Application extends MovieClip {
/* ****************************************************************************
 * PRIVATE VARIABLES
 **************************************************************************** */

/* ****************************************************************************
 * CONSTRUCTOR
 **************************************************************************** */
 function Application(container) {
 // the movieclip is transtyped as a application object
  container.__proto__ = this.__proto__;
  container.__constructor__ = Application;
  this = container;

 // init the object
  _init();
 }


/* ****************************************************************************
 * PRIVATE FUNCTIONS
 **************************************************************************** */
/**
 * Init the Application
 * @param    Void
 */
 private function _init( Void ) : Void {
 // init the debugger
  Logger.getInstance().addLogListener( LuminicTracer.getInstance() );

 // Instanciating the views
  var digital : MovieClip = this.attachMovie( "mc_digital", "mc_digital", 0 );
  var vDigital  : ViewDigital = new ViewDigital ( digital );

  var analog : MovieClip = this.attachMovie( "mc_analog", "mc_analog", 1 );
  var vAnalog  : ViewAnalog = new ViewAnalog ( analog );

  var tools : MovieClip = this.attachMovie( "mc_tools", "mc_tools", 2 );
  var vTools  : ViewTools = new ViewTools ( tools );

 // create the model
  var mClock : ModelClock = new ModelClock();

 // the views are listening to the model
  mClock.addListener( vDigital );
  mClock.addListener( vAnalog );
  mClock.addListener( vTools );

 // init the controller with our custom EventBroadcaster class
  Controller.getInstance( XEventBroadcaster.getInstance() );
 }


/* ****************************************************************************
 * PUBLIC FUNCTIONS
 **************************************************************************** */
```


Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

```
/*************************************************************************************
 * GETTER & SETTER
 ************************************************************************************* */
}
```

Revenons sur quelques points de ce code

```
container.__proto__ = this.__proto__;
container.__constructor__ = Application;
this = container;
```

Ici, nous transtypons le "container" (passé en argument) qui est (et c'est le cas dans la majeure partie des cas) le _root (ou _level0) de votre fla, en un objet Application. Comme vous pouvez le voir, votre classe Application.as hérite de la classe MovieClip et donc conserve les propriétés et méthodes de celle-ci.

Nous appelons alors la fonction _init() chargée de tout initialiser.

Premièrement, nous définissons le Debugger

```
Logger.getInstance().addLogListener( LuminicTracer.getInstance() );
```

Deuxièmement, nous créons nos vues liées aux Movieclips que nous avons "attachés" sur la scène.

```
var digital : MovieClip = this.attachMovie( "mc_digital", "mc_digital", 0 );
var vDigital  : ViewDigital = new ViewDigital ( digital );

var analog : MovieClip = this.attachMovie( "mc_analog", "mc_analog", 1 );
var vAnalog  : ViewAnalog = new ViewAnalog ( analog );

var tools : MovieClip = this.attachMovie( "mc_tools", "mc_tools", 2 );
var vTools  : ViewTools = new ViewTools ( tools );
```

Troisièmement, nous créons le modèle

```
var mClock : ModelClock = new ModelClock();
```

et affectons les vues comme écouteurs de ce modèle. Aussi, lorsque le modèle diffusera un événement, les vues réagiront à celui-ci.

```
mClock.addListener( vDigital );
mClock.addListener( vAnalog );
mClock.addListener( vTools );
```

Enfin, nous créons le FrontController, qui écoutera le "XEventBroadcaster" et appellera la commande associée à l'événement.

```
Controller.getInstance( XEventBroadcaster.getInstance());
```

Maintenant, nous allons regarder chaque classe pour voir ce qu'elles font :

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

IV-B - La ViewList, les vues et comment les utiliser

Tout d'abord, la ViewList. Cette classe est seulement une énumération de toutes les vues (classes) que vous avez dans votre application. Comme Flash ne fournit pas de type Enumération, c'est une manière pour y arriver ...

Voici la classe :

```
/**
 * listing of the views
 */
class net.webbymx.projects.tutorial01.views.ViewList {
/* ****************************************************************************
 * PUBLIC STATIC VARIABLES
 **************************************************************************** */
 public static var VIEW_ANALOG    : String = "view_analog";
 public static var VIEW_DIGITAL   : String = "view_digital";
 public static var VIEW_TOOLS     : String = "view_tools";


/* ****************************************************************************
 * CONSTRUCTOR
 **************************************************************************** */
 private function ViewList() {}
}
```

Pour commencer, regardons la vue ViewTool, car dans la petite application que nous allons réaliser. C'est celle qui va utiliser toutes les fonctionnalités de base.

```
//  Debug
import com.bourre.log.Logger;
import com.bourre.log.LogLevel;

//  Delegate
import com.bourre.commands.Delegate;

//  Event Broadcasting
import com.bourre.events.IEvent;
import com.bourre.events.BasicEvent;
import com.bourre.events.EventType;
import net.webbymx.projects.tutorial01.events.EventList;
import net.webbymx.projects.tutorial01.events.XEventBroadcaster;

//  MovieClipHelper
import com.bourre.visual.MovieClipHelper;

//  list of Views
import net.webbymx.projects.tutorial01.views.ViewList;

class net.webbymx.projects.tutorial01.views.ViewTools extends MovieClipHelper {
/* ****************************************************************************
 * PRIVATE VARIABLES
 **************************************************************************** */
//  Assets
 private var _txtColor    : TextField;
 private var _btnStart    : MovieClip;
 private var _btnStop     : MovieClip;
 private var _btnColor    : MovieClip;

/* ****************************************************************************
 * CONSTRUCTOR
 **************************************************************************** */
```


Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

```
	function ViewTools( mc : MovieClip ) {
	 super( ViewList.VIEW_TOOLS, mc );
	 _init();
	}

	/* *****************************************************************************
	 * PRIVATE FUNCTIONS
	 ***************************************************************************** */
	/**
	 * Init the view
	 * @param    Void
	 */
	 private function _init( Void ) : Void {
	 //  Position the MovieClip
	 view._x = 33;
	 view._y = 180;

	 //  init var, assets, events, ...
	 _txtColor = view.txt_color;
	 _btnStart = view.btn_start;
	 _btnStop = view.btn_stop;
	  _btnColor = view.btn_color;

	 //  Mouse events
	 _btnStart.onRelease = Delegate.create( this, _fireEvent, new BasicEvent( EventList.START_CLOCK )
	);
	 _btnStop.onRelease = Delegate.create( this, _fireEvent, new BasicEvent( EventList.STOP_CLOCK ) );
	 _btnColor.onRelease = Delegate.create( this, _fireEvent, new BasicEvent(
	EventList.CHANGE_CLOCK_COLOR) );

	 disableStart();
	}

	/**
	 * Broadcast the event
	 * @usage    _fireEvent( new BasicEvent( EventList.MYTYPE, Object ) );
	 * @param    e
	 */
	 private function _fireEvent( e : IEvent ) : Void {
	 XEventBroadcaster.getInstance().broadcastEvent( e );
	}

	/* *****************************************************************************
	 * PUBLIC FUNCTIONS
	 ***************************************************************************** */
	 public function disableStart( Void ) : Void {
	  _btnStop.enabled = true;
	 _btnStop._alpha = 100;
	 _btnStart.enabled = false;
	 _btnStart._alpha = 60;
	}

	public function disableStop( Void ) : Void {
	 _btnStop.enabled = false;
	 _btnStop._alpha = 60;
	 _btnStart.enabled = true;
	 _btnStart._alpha = 100;
	}


	/* *****************************************************************************
	 * GETTER & SETTER
	 ***************************************************************************** */
}
```

Maintenant, regardons comment cela marche.

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

Une vue est une interface. Seul le code dans cette classe agit sur l'aspect visuel. Votre vue (classe) est également associée à un objet visuel (un MovieClip). Ici, nous n'utilisons pas l'héritage comme nous l'avons fait pour l'application, mais la composition. Notre classe ViewTool a une propriété appelée view où la référence au MovieClip sera stockée.

Regardons le constructeur

```
function ViewTools( mc : MovieClip ) {
 super( ViewList.VIEW_TOOLS, mc );
 _init();
}
```

Nous pouvons voir que nous appelons le constructeur de la super classe (MovieClipHelper). Celui-ci stocke le nom de votre vue et sa référence dans une Map (Tableau associatif d'objets) et la référence de votre MovieClip dans la propriété view.

Pourquoi cela ?

La classe MovieClipHelper a une fonction statique et publique nommée getMovieClipHelper. Cette fonction renvoie la vue (classe) voulue. Vous l'utilisez de cette manière :

```
ViewTool( MovieClipHelper.getMovieClipHelper( ViewList.ViewTool ) )
```

Cela signifie que vous pouvez accéder à votre vue de n'importe où et que vous avez le contrôle de votre MovieClip (avec les propriétés de la vue ou en utilisant des fonctions publiques)

D'accord pour cette partie. Maintenant, regardons le code dans la fonction _init.

Au lieu d'accéder aux propriétés du MovieClip en faisant

```
ViewTool.view.txt_color
```

Je préfère initialiser des variables privées de ma classe qui font référence à ces propriétés.

```
_txtColor = view.txt_color;
_btnStart = view.btn_start;
_btnStop = view.btn_stop;
_btnColor = view.btn_color;
```

Finalement, pour gérer les événements boutons, nous avons ceci :

```
_btnStart.onRelease = Delegate.create( this, _fireEvent, new BasicEvent( EventList.START_CLOCK ) );
_btnStop.onRelease = Delegate.create( this, _fireEvent, new BasicEvent( EventList.STOP_CLOCK ) );
_btnColor.onRelease = Delegate.create( this, _fireEvent, new BasicEvent(
 EventList.CHANGE_CLOCK_COLOR) );
```

Chaque bouton déclenchera un événement (depuis l'énumération de la classe EventList)

Ici, j'ai choisi d'utiliser les Delegate (plus propre) mais vous pouvez utiliser :

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

```
_btnStart.onRelease = function () {
XEventBroadcaster.getInstance().broadcastEvent( new BasicEvent( EventList.START_CLOCK ) );
}
```

Nous avons donc deux fonctions publiques qu'une commande appellera pour activer/désactiver les boutons stop/start.

IV-C - La ModelList et le ModelClock

La ModelList ressemble la ViewList. Il s'agit juste d'une classe listant les variables, pour émuler le type Enumération qui n'existe pas dans Flash. Aussi, dans cette classe, nous listons tous les noms des modèles utilisés dans notre application (ici seulement). Et oui, nous pouvons créer plus d'un modèle si on en a besoin (par exemple pour séparer les différentes logiques). Cependant, je ne l'ai jamais fait encore ...

Le ModelClock est une classe qui contient le code de la logique, tout ce qui est "derrière la scène" pour faire fonctionner votre application. Dans notre exemple, nous y mettrons le code qui déclenchera un événement chaque seconde.

```
//  Debug
import com.bourre.log.Logger;
import com.bourre.log.LogLevel;

//  Model
import com.bourre.core.Model;

//  Import the model list
import net.webbymx.projects.tutorial01.models.ModelList;

//  Broadcasting event
import com.bourre.events.EventType;
import com.bourre.events.BasicEvent;

class net.webbymx.projects.tutorial01.models.ModelClock extends Model {
/* ****************************************************************************
 * PRIVATE STATIC VAR
 **************************************************************************** */
 private static var CLOCK_TICK : EventType =  new EventType( "onClockTicking" );
/* ****************************************************************************
 * PRIVATE VAR
 **************************************************************************** */
 private var _siTick : Number;
 private var _dDate  : Date;


/* ****************************************************************************
 * CONSTRUCTOR
 **************************************************************************** */
  function ModelClock() {
   super( ModelList.MODEL_CLOCK);
  _startTicking();
 }



/* ****************************************************************************
 * PRIVATE FUNCTIONS
 **************************************************************************** */
  private function _startTicking( Void ) : Void {
 //  set up the date actual date and time
  _dDate = new Date();
```


Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.
http://gdumas.developpez.com/traductions/articles/flash/pixlib/bases/

```
    _tick();
    _siTick = setInterval( this, "_tick", 1000 );
   }

   private function _stopTicking( Void ) : Void {
    clearInterval( _siTick );
   }

   private function _tick( Void ) : Void {
   //  add one second to the date object
    _dDate.setSeconds( _dDate.getSeconds() + 1 );

   //  create the object time which will be sent during the broadcasting
    var oTime = new Object();
    oTime.seconds = _dDate.getSeconds().toString();
    oTime.minutes = _dDate.getMinutes().toString();
    oTime.hours = _dDate.getHours().toString();

    if ( oTime.hours.length == 1 ) oTime.minutes = "0"+oTime.hours;
    if ( oTime.minutes.length == 1 ) oTime.minutes = "0"+oTime.minutes;
    if ( oTime.seconds.length == 1 ) oTime.seconds = "0"+oTime.seconds;

   //  Broadcasting the event to the view listening to the model ( init in the Application.as class )
    _oEB.broadcastEvent( new BasicEvent( CLOCK_TICK, oTime ) );
   }
  /* ***************************************************************************
   * PUBLIC FUNCTIONS
   *************************************************************************** */
  /**
   * called by the startClock command
   * @param    Void
   */
    public function startClock( Void ) : Void {
    _startTicking();
   }

  /** * called by the stopClock command
   * @param    Void
   */
    public function stopClock( Void ) : Void {
     _stopTicking();
   }

  /* ***************************************************************************
   * GET & SET
   *************************************************************************** */
  }
```

Dans le constructeur, nous allons faire la même chose que pour la vue. Je veux dire que nous allons envoyer au constructeur le nom du modèle (récupéré à partir de la ModelList)

Le constructeur enregistrera le nom de ce modèle et sa référence.

Vous serez alors capable de récupérer votre modèle de cette manière :

```
ModelClock( Model.getModel( ModelList.MODEL_CLOCK ) )
```

Comme vous pouvez le voir à la ligne 21, nous avons créé (comme dans l'EventList) une liste d'événements que le ModelClock déclenchera. Ici, c'est légèrement différent. Le modèle déclenchera l'événement et la vue écoutant le modèle exécutera la fonction fournie par l'argument EventType, "onClockTicking" dans notre exemple.

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

Dans notre modèle, chaque seconde la fonction "onClockTicking" sera appelée. La vue recevra l'événement et exécutera la fonction, qui fera avancer la trotteuse ou modifiera l'affichage numérique.

Vous pouvez donc voir que le modèle a 2 fonctions publiques pour démarrer et stopper le chronomètre. Ces fonctions seront appelées depuis une commande. N'oubliez pas qu'une commande peut appeler une fonction depuis le modèle ET les vues. Regardez la "StartCommand" par exemple.

IV-D - L'EventList et le XEventBroadcaster

L'EventList ressemble, elle aussi, à la ViewList. Dans cette classe, nous listerons tous les noms des événements que nous utiliserons dans notre application.

Le **XEventBroadcaster** est EventBroadcaster particulier. Par défaut, dans le framework PixLib, vous avez un EventBroadcaster par défaut créé implicitement. Je préfère utiliser le mien, comme cela, si vous créez une application complexe qui utilise des modules (et donc plus d'un EventBroadcaster), ce ne sera pas le désordre. Chaque module aura le sien et les événements ne seront pas envoyés aux FrontControllers qui ne sont pas concernés. Souvenez-vous que le FrontController écoute l'EventBroadcaster. C'est pour cela que, dans Application.as, nous avons créé un nouveau contrôleur qui écoute le XEventBroadcaster...

IV-E - Le FrontController

Le FrontController est assez simple.

```
//  Debug
import com.bourre.log.Logger;
import com.bourre.log.LogLevel;

//  Type
import com.bourre.core.HashCodeFactory;
import com.bourre.events.FrontController;

//  Commands and Event list
import net.webbymx.projects.tutorial01.commands.*;
import net.webbymx.projects.tutorial01.events.EventList;

class net.webbymx.projects.tutorial01.commands.Controller extends FrontController {
/* ****************************************************************************
 * PRIVATE STATIC VAR
 **************************************************************************** */
 private static var _oI : Controller;


/* ****************************************************************************
 * PUBLIC STATIC FUNCTIONS
 **************************************************************************** */
 public static function getInstance( myDispatcher ) : Controller  {
  if (!_oI) _oI = new Controller( myDispatcher );
  return _oI;
 }

/* ****************************************************************************
 * CONSTRUCTOR
 **************************************************************************** */
 private function Controller( myDispatcher ) {
  super( myDispatcher );
  _oI = this;
  _init();
 }
```


Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

```
/* ***************************************************************************
 * PUBLIC FUNCTIONS
 *************************************************************************** */
/**
 * Push the event in the controller and link them to a command
 * @param    Void
 */
  private function _init( Void ) : Void {
   push ( EventList.START_CLOCK , new StartClock() );
  push ( EventList.STOP_CLOCK, new StopClock() );
 }

/**
 * Return the instance id
 * @return
 */
  public function toString() : String {
   return 'net.webbymx.projects.echo.commands.Controller' + HashCodeFactory.getKey( this );
 }
}
```

C'est un singleton. Sa structure est toujours la même. Il suffit d'éditer la fonction _init() pour ajouter les paires événement/commande.

Le FrontController écoutera le XEventBroadcaster. Quand un événement est déclenché, il vérifie recherche dans un tableau l'événement et appelle la commande associée.

IV-F - Les commandes

Nous avons donc vu que le FrontController pour un événement créera une nouvelle instance d'une commande. Il appellera la fonction *execute* de la commande indiquée.

Examinons la commande StartClock

```
//  Debug
import com.bourre.log.Logger;
import com.bourre.log.LogLevel;

//  Implement
import com.bourre.commands.Command;

//  Hash
import com.bourre.core.HashCodeFactory;

//  Type
import com.bourre.events.IEvent;

//  Model
import com.bourre.core.Model;
import net.webbymx.projects.tutorial01.models.ModelClock;
import net.webbymx.projects.tutorial01.models.ModelList;

//  Views
import com.bourre.visual.MovieClipHelper;
import net.webbymx.projects.tutorial01.views.ViewList;
import net.webbymx.projects.tutorial01.views.ViewTools;

class net.webbymx.projects.tutorial01.commands.StartClock implements Command {
/* ***************************************************************************
```


Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

```
 * CONSTRUCTOR
 ************************************************************************* */
 function StartClock() {}

/* *************************************************************************
 * PUBLIC FUNCTIONS
 ************************************************************************* */
/**
 * This called by the FrontController when a event associated to it is triggered
 * @param    e
 */
  public function execute( e : IEvent ) : Void {
  //  execute the stopClock method of the ModelClock object
  //  here the command name is the same as the function name, BUT it could be different
   ModelClock( Model.getModel( ModelList.MODEL_CLOCK ) ).startClock();
 //  disable the start button on the ViewTools and enable the stop button
  ViewTools( MovieClipHelper.getMovieClipHelper( ViewList.VIEW_TOOLS ) ).disableStart();
 }

 public function toString() : String {
  return 'net.webbymx.projects.tutorial01.commands.StartClock' + HashCodeFactory.getKey( this );
 }
}
```

Nous pouvons la fonction execute. Cette fonction a un argument qui un objet implémentant l'interface IEvent. Cela peut être un BasicEvent ou un type d'événement que vous avez créé.

Vous pouvez alors appeler une fonction publique d'un vue en utilisant :

```
YourViewType( MovieClipHelper.getMovieClipHelper( YourViewList.YourView ) ).yourFunction ( args )
```

ou bien d'un modèle, de cette manière :

```
YourModelType( Model.getModel( ModelList.MyModel ) ).yourFunction( args )
```

Vous pouvez voir ici que nous utilisons le typage fort (YourViewType(...), YourModelType(...)). Ainsi, par exemple, dans FlashDevelop, vous obtiendrez toutes les fonctions publiques de cette class dans l'IDE.

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

V - Conclusion

Nous avons donc vu comment utiliser les classes basiques pour créer une application utilisant le motif MVC + Commande.

Cela peut sembler difficile au début mais cela devient vraiment simple lorsque vous comprenez comment les données sont transmises.

Regardez attentivement le schéma. Les vues diffusent (vie XEventBroadcaster) les événements au FrontController. Celui-ci appelle la commande associée à cet événement. Une commande peut alors appeler une fonction publique d'une vue ou d'un modèle.

Enfin, un modèle diffuse l'événement aux vues qui l'écoutent...

Voilà, c'est la fin de ce tutoriel, J'espère qu'il vous aidera à mieux comprendre l'architecture de ce framework.

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

Copyright © 2006-2007 Xavier Martin/Grégory Dumas. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur (Xavier Martin) et du traducteur (Grégory Dumas). Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.