Developer Guide Document

Introduction

Aim of this document is constituting development standardization, teaching MVC logic and helping developers in their development process.
This document is written for actionscript developers and every developer whichever junior to senior can take advantage of this.

This document includes following issues below;

A-) Code Standards	v 0.1
B-) Common Terms	v 0.3
C-) MVC (Model-View-Controller) Pattern	v 0.5
D-) Multiplatform Development	v 0.7
E-) Starting a New Project	v 0.9
F-) Working with a Team	v 1.0

Current Version 0.5

For any contribution you can send email to s.sahin@triodor.nl

Author
Software Architect
Suat SAHIN

After this document following documents should be read;

1- Optimization Flash Platform
http://help.adobe.com/en_US/as3/mobile/flashplatform_optimizing_content.pdf

2- Debugging with Scout
http://www.adobe.com/devnet/scout/articles/adobe-scout-getting-started.html

3- Starling Framework
http://www.adobe.com/devnet/flashplayer/articles/introducing_Starling.html

Useful web sites you can follow for daily news;

1- http://www.bytearray.org/
2- http://flashdaily.net/
3- http://gaming.adobe.com/

Index

1- Analysing
2- Setup Working Enviroment
3- Designing Code Architecture
4- Developing Features
5- Optimization
6- Security
7- Test and Debug
8- Deployment
9- Maintenance

F-) Working with a Team

1- Paralel Working
2- Communication
3- Code Comments & Reviews
4- Sharing Knowledge

A-) Code Standards

1- Naming Convention

- Give specific and meaninful names, dont give generic or hard to understand short names.

wrong - not understandable
```
var rpname:String;
```

correct
```
var rivalName:String;
```

wrong - too generic
```
var root:Sprite
```

correct
```
var mainScreen:Sprite
```

- Variables must be includes their data types in their names except primitives.

Non - Primitives
```
var playBtn:Button;
var lightEffectMc:MovieClip;
var usersDic:Dictionary;
var usersAr:Array;
var kitchenFocusArea:FocusAreaDataVO;
```

```
Primitives
var userCount:int;
var warningMessage:String;
var isCaptured:Boolean;
```

- Name of the private variables must be started with '_'

```
private var _penCount:int;
private var _description:String;
private var _isCaptured:Boolean;
```

- Variable names must be lower camelcase except Constants, and variable names consist of only letters, none digits, non special charecters etc.

```
correct
var camelCaseVarName:String;

wrong
var wrongvariablename:String;

correct
const STAGE_HEIGHT:int;

wrong
var FA_01:FocusAreaDataVO;

correct
var kitchenFocusArea:FocusAreaDataVO;
```

- Class names must to be upper camel case

```
public class ResourceManagerSystem
```

- Look at the 'MVC (Model-View-Controller) Pattern' chapter to see naming convention for Commands, Mediators, Proxies etc. which are related with MVC framework.

2- Coding Convention

Target of coding convention is improving code quality, making code easy to read for every developer and preventing developing buggy code.

- Dont use ' * ' , use absoulute path

```
wrong
import mystery.game.asset.*

correct
import mystery.game.asset.EmbedAssets;
```

- Use equal signs as aligned

```
hard to read
var focusAreaData:FocusAreaDataVO = areaData;
var groupDic:Dictionary = new Dictionary();
var groupDataItemGroup:ItemGroupVO = new ItemGroupVO();
var nonGroupItemList:Vector.<String> = new Vector.<String>();
var backgroundList:Vector.<String> = new Vector.<String>();

easy to read
var focusAreaData:FocusAreaDataVO     = areaData;
var groupDic:Dictionary               = new Dictionary();
var groupDataItemGroup:ItemGroupVO    = new ItemGroupVO();
var nonGroupItemList:Vector.<String>  = new Vector.<String>();
var backgroundList:Vector.<String>    = new Vector.<String>();
```

- Dont use shorter if statements

```
Wrong however you can only use when parsing xml
item.@base ?          itemData.name = item.@base : itemData.name =     itemData.name;

Wrong one line if
if(itemAmount <= 0) break;

Wrong using without { }
if (listHolder.getChildAt(k) is Image)
     listHolder.getChildAt(k).removeFromParent(true);


Correct easy to read and debug
if(item.isCaptured == true){

}else{
}
```

- About using For loops

```
Wrong - dont use typeless iterator
```

```
for each(var item:* in configXmlList){
      Wrong - dont define variables in loops
      var itemData:HiddenItemDataVO = new HiddenItemDataVO();

Correct
var itemData:HiddenItemDataVO
for each(var item:XML in configXmlList){
      itemData                        = new HiddenItemDataVO();
```

Correct - When you use nested for loops iterator names must be 'i','k','m' . It makes that readable otherwise 'j' always seems like 'i'.

```
var i:int;
var k:int;
var m:int;
for(i=0
      for(k=0
            for(m=0
```

```
Wrong -no need to redefine iterator with strange names
for (var i2:int = 0; i2 < 6; i2++){
```

```
Correct - you can use same iterator again
for(i=0; i<6; i++){
```

- Dont use nested functions and typeless objects never. Function names also must be lower camel case.

```
Wrong
private function doSomething():void{
      inventoryList.addCallback( { onshow:function():void {
            trace('nooo');
      }
}
```

```
Correct
private function doSomething():void{
      inventoryList.addCallback(callbackFunc);
}

private function callbackFunc():void{
      trace('yeah');
}
```

- Dont make control with Strings

String objects must to be only used for representation data to user. Dont make any control with String such as name. Its a very common developer habit but its not a good way to handle a state.

Wrong

```
private function onComplete(e:Event):void {

      if (Object(e.target).name == "open"){

            _call(close);

      }else if (Object(e.target).name == "close") {

      }

}
```

Wrong - to get information its not a good way to get it from String name.

```
var qualifiedName:String = getQualifiedClassName(Lang_EN );

qualifiedName = qualifiedName.substr(0, qualifiedName.length - 2);
```

Correct - There must be a defined data object in a Dictionary

```
var languageData:LanguageDataVO = _languageDict[_languageName];
languageData.name
languageData.content
```

```
public function setMode(mode:String):void {

      Wrong

      if (mode == 'focus') {

      Correct

      if (mode == PlayGameMenuModeTypes.FOCUS) {
```

- Comments

Works are which planned to do later, you should write a comment like this

```
//TODO users data will be fetched here!
```

When you want to warm the other developers, you can write a comment

```
//ATTENTION item ids must be unique!
```

When you think something is wrong you can write a question comment

```
///WHY? item ids must be unique, we can use their names, cant we?
```

- Dont set undefined as a default value to any variable. Without setting default value its already null so you can use null control when you need.

Wrong

```
var wrongDefaultValue:Custom = undefined;
```

Wrong

```
if(wrongDefaultValue == undefined){
}
```

Correct

```
var correctDefaultValue:Custom = new Custom();
```

or

```
var correctDefaultValue:Custom; // its already null
```

Correct

```
if(correctDefaultValue == null){
}
```

3- About Strong Typing

Strong typing always better for code architecture, performance, maintenance and refactoring. We see it as standard, it can not be a design choice.

- All variables must be always strong typed.

wrong

```
var someValue:*;
```

```
var someOtherValue:Object
var anyOtherValue;

correct
var someValue:String;
var someOtherValue:Number;
var anyOther:Hero;
```

- Always define type for parameters and return values. Always make everything strong typed :) never use typeless.

```
Wrong
public function setPlayMenuItemList(areaName):* {

Correct
public function setPlayMenuItemList(areaName:String):void {
```

B-) Common Terms

1- Flash Platform

MVC : Model View Controller Code Pattern
SWC : An Adobe SWC file is a package of precompiled Flash symbols and Actionscript code that allows a Flash or Flex developers to distribute classes and assets
Texture : Image which stores multiple images and data. ex: Starling textures or Away3D textures
Stage 3D : is the native GPU rendering pipeline developed by Adobe, Starling Framework and all 3D engines use this technology.
RMFTP : Real-Time Media Flow Protocol developed by Adobe provides P2P connections between swf files which are in different endpoints.
AMF : Action Message Format is used for object passing between flash client and server. ex: AmfPHP
Scout : Profiller for flash
FLV, F4V : Flash video file

will be improved in next version!

2- Other

GPU : Graphics Processing Unit
CPU : Central Processing Unit
SVN : Subversion (SVN) is a version control system ex: Tortoise svn
GIT : Distributed version control system ex: GitHub
IDE : Integrated Development Environment ex: Flash Builder, Eclipse
DB : Database ex: Mysql, Mongo

API : "Application Program Interface," An API is a set of commands, functions, and protocols
AI : Artificial Interface
P2P : Peer to peer data trasmission protocol
TCP/IP : Transmission Control Protocol/Internet Protocol
RMTP : Real Time Messaging Protocol
SEO : Search Engine Optimization
Backup : Archiving of files so it may be used to restore in future
Bandwidth : Refers to how much data that goes through a network.
Binary : Binary is a two-digit (Base-2) numerical system, 1 or 0
RGBA : Red Green Blue Alpha color model for web
Memory Leak: Extra memory usage because of cumulative unremoved objects
GUI : Graphical User Interface
HD: 'High definition' relevant to image and video quality
SSL: "Secure Sockets Layer."secure protocol
Stand Alone: Able to work independently, no depency on any platform
Streaming : Data streaming, commonly seen in the forms of audio and video streaming
Sync : Synchronizing between seperated data models
Syntax : Specified set of rules for code format, ex: c++ syntax diffrent than actionscript syntax
IP : "Internet Protocol." It provides a standard set of rules for sending and receiving data through the internet.
IT: "Information Technology," and is pronounced "I.T." It refers to anything related to computing technology
Compiler : A software which compile the specified code ex: Flex compiler
Runtime : When a program is running, or executing, it is said to be in runtime.
SDK : Software Development Kit, ex: Flex SDK, Air SDK etc.

C-) MVC (Model-View-Controller) Pattern

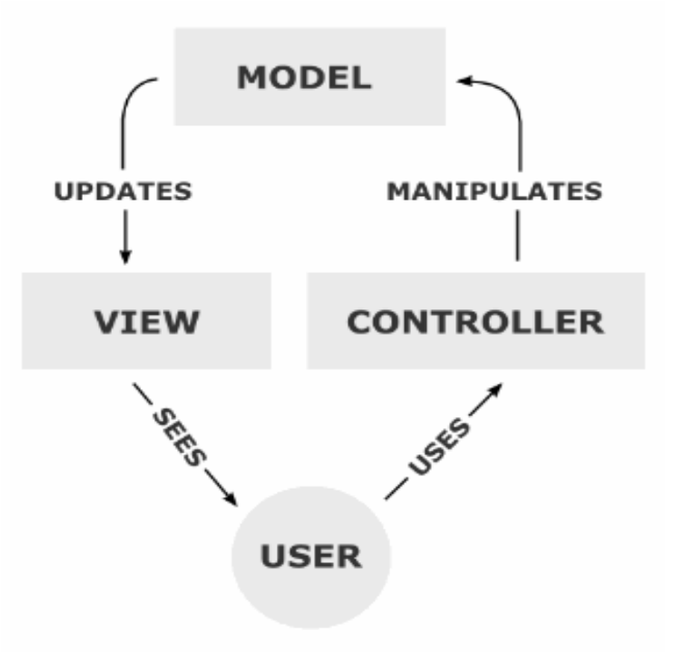
1- What is MVC?

Model–view–controller (**MVC**) is a software architecture pattern that separates the representation of information from the user's interaction with it. The *model* consists of application data and business rules, and the *controller* mediates input, converting it to commands for the model or view.A *view* can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible, such as a pie chart for management and a tabular view for accountants. The central ideas behind MVC are code reusability and separation of concerns.
- Quote from Wikipedia

There are two popular MVC Framework for actionscript;

1- PureMVC http://puremvc.org/
2- Robotlegs http://www.robotlegs.org/

In this document we will go through PureMVC. Note that Robotlegs framework was built on same MVC logic and flow.



As you can see on image above, flow starts with a user action then related controller gets dispatched and the controller changes the model data, as final step view refreshs itself.

Lets look at a basic scenario below;

1- After filling the form, user clicks the saveProfile button which exists in the Profile mediator
2- SaveProfile button dispatches a local event only for Profile mediator
3- Profile mediator catches the event and sends SaveProfile notification with defined parameter
4- SaveProfile notification automaticly executes SaveProfile command with the parameter
5- SaveProfile command updates the UserProfile proxy
6- At final all mediators which listen the SaveProfile notification refresh their viewcomponents.

So the user sees the different data on the screen. Does it seem complicated? yes it is if you are going to use MVC first time.

You can find more detailed information about PureMVC on web, just google it :)

2- Best Practices

PureMVC

- Extended Facade structure can be like this;

```
public class GameFacade extends Facade implements IFacade{


      public static function get instance():GameFacade{

            if (Facade.instance == null){
                  Facade.instance = new GameFacade();
            }

            return GameFacade(Facade.instance);
      }


      /**
       * All Proxies will be registered in initializeModel
      */
      override protected function initializeModel():void {

            super.initializeModel();

            registerProxy(new DBConnectorProxy());
            registerProxy(new AccountProxy());
            registerProxy(new LanguageProxy());
            registerProxy(new AssetProxy());
            registerProxy(new PaymentProxy());
            registerProxy(new AreaProxy());
      }

      /**
       * All Commands will be registered in initializeController
      */
      override protected function initializeController():void    {

            super.initializeController();

            registerCommand(NotificationList.START_UP, StartUpCMD);
            registerCommand(NotificationList.SHOW_AREA, ShowAreaCMD);
            registerCommand(NotificationList.PLAY_GAME, PlayGameCMD);
      }
```

```
	/**
	 * Singleton Mediators will be registered in initializeView
	*/
	override protected function initializeView():void {

		super.initializeView();

		registerMediator(new GameScreenMed());
	}
```

- Command structure;

```
public class ShowAreaCMD extends SimpleCommand implements ICommand{

	private var _areaProxy:AreaProxy= AreaProxy(GameFacade.instance.retrieveProxy(AreaProxy.NAME));

		override public function execute(notification:INotification):void{

			var param:ShowAreaParamVO  = ShowAreaParamVO(notification.getBody());

			_areaProxy.setCurrentAreaName(param.areaName);

		}
	}
```

```
	Every command must to be executed with a specified ParamVO
	public class ShowAreaParamVO{
		public var areaName:String;
	}
```

```
	You can call command like this;

	var showAreaParam:ShowAreaParamVO	= new ShowAreaParamVO();
	showAreaParam.areaName			= _areaName;
	sendNotification(NotificationList.SHOW_AREA, showAreaParam);
```

1- As you see above ShowAreaCMD has suffix 'CMD', every command must to have that suffix.
2- Every command has a parameter which is strongly typed object with same name but different suffix 'ParamVO' ShowAreaParamVO.
3- ParamVO only can be used for passing parameter to command. Its not a model data. Its just a temporary parameter object.

4- Commands can hold proxy referances.
5- You can call commands from mediators and other commands, dont call them from proxies!
6- There is a NotificationList class which holds the string names of the notifications through that we know which notifications we can use.

- Proxy structure;

```
public class AssetProxy extends Proxy implements IProxy{

	public static const NAME:String = 'assetproxy';

	public function AssetProxy()	{
		super(NAME, new AssetProxyDataVO());
	}

	public function get proxyData():AssetProxyDataVO{
		return AssetProxyDataVO(getData());
	}

	public function init():void{

	}
```

1- We set the name of Proxy as static const. When we need to retrieve Proxy from Facade makes it easy.
ex: _assetProxy = AssetProxy(GameFacade.instance.retrieveProxy(AssetProxy.NAME));
2- As you see above AssetProxy has suffix 'Proxy', every proxy must to have that suffix.
3- Main model objects are singleton, we use only one instance per Proxy object in application.
4- Proxy consists of functions which manipulate the contained data (DataVO-Data Value Object).
5- AssetProxyDataVO has very similar name with AssetProxy only it has suffix as 'DataVO'.
6- Its alwasy better to create init function to initialize objects, constructor is not good way to do that.
7- proxyData function is for getting DataVO as casted object.

- Mediator structure;

- Singleton mediator; there must be only one instance

```
public class AreaScreenMed extends Mediator implements IMediator{

	public static const NAME:String = "areascenemed";

	public function AreaScreenMed(){
```

```
            super(NAME, new AreaScreen());
        }

        public function get visual():AreaScreen{
            return AreaScreen(getViewComponent());
        }

        public function init():void{
            visual.addEventListener(Event.TRIGGERED, onVisualTriggered);
        }
```

- Standard Mediator; there can be multiple instances

```
    public class HiddenItemMed extends Mediator implements IMediator{

        public static const NAME_PREFIX:String = "hiddenitem";

        public var _data:HiddenItemDataVO;

        public function HiddenItemMed(mediatorName:String, visual:PixelButtonTouch) {

            super(NAME_PREFIX + mediatorName, visual);
        }

        public function get visual():PixelButtonTouch{
            return PixelButtonTouch(getViewComponent());
        }

        public function init():void{
            visual.addEventListener(Event.TRIGGERED, onVisualTriggered);
        }
```

1- There are 2 mediator structures above with little diffrences.
2- As you see above AssetProxy has suffix 'Proxy', every proxy must to have that suffix.
3- There is NAME constant which is defined for mediator constructor. When we need to retrieve Mediator we call the mediator through its Name.
_areaSceneMed = AreaScreenMed(GameFacade.instance.retrieveMediator(AreaScreenMed.NAME));
4- There are 2 kind of mediators which we use, standard mediator and singleton mediator, usally we use them together.
5- Standard mediators can be useful for dynamicly created and removed in game objects such as EnemySoldierMed, BuildingMed, CarMed etc.
6- Singleton mediators can be useful for screens, static menus etc.
7- Standard mediators' name are dynamicly created with a NAME_PREFIX to prevent possible conflicts with other mediators. ViewComponent also dynamic and its defined from out of scope.
8- Mediator contains viewComponent referance which is display object on the screen. Mediators responsible for their viewComponents, they listens the events which dispatched from viewComponents.

Also they updates and sets data into viewComponent.
9- Note that every mediator must to have only one viewComponent
10- visual function is for getting viewComponent as casted.

3- About Feature Packaging Method

We use feature packaging method with seperated view. Every feature must to have their own packet which contains commands, proxy, dataVO, paramVO objects must be in.. but mediators and viewComponents must be in view packet!. Through the feature packaging method its easy to maintain application and easy to move features into other projects, so there is no need to reinvent the wheel for every application :)

It looks like this;

- mystery
    - game
        - account
            - AccountProxy
            - AccountProxyDataVO
            - CreateAccountCMD
        - minigame
        - map
        - book
    - view
        - core
        - display
        - med
            - HiddenItemMed
            - OverviewItemMed

- If you look at the account package it includes AccountProxy and CreateAccountCMD on the same level. As you can see there is no Mediator or Viewcomponent in that package because they must to exist in view package.
- View always should be separated, its very common mistake that developers are always inclined to use view elements as controller. View only responsible to show data and listen user actions. View is not concerned to make decision about bussiness logic. For example you can not put a function which finishes the game into a class exists in view package. You must to create a command like FinishGameCMD.

D-) Multiplatform Development

v 0.7

E-) Starting a New Project

v 0.9

F-) Working with a Team
v 1.0