

Flex 3 development

Best practices

Document History

Version	Date	Description of change
1.0	2009-10-28	Initial version, created by Andriy Tabachyn. Presented on Lviv Flex conference 2009-10-24

Table of Contents

Table of Contents	3
Read this before you start	4
Project folder structure	4
Code conventions	4
Component development	7
Component life cycle	7
Write item renderers correctly	7
Several patterns what we use	9
Services Pattern	9
Code generation & Generation gap pattern	10
Code Behind	10
Several useful components / ideas	12
Alert Dialog and faults	12
Default Formatters	12
Debug Panel	12
Advanced Data Grid extensions	13
Misc	14
Frameworks	14
Child properties access.....	14
Dynamic classes.....	14
Support selection on component level	14
Use binding carefully	14
callLater usage.....	15
Create components on demand.....	15
Common components library.....	15
Memory leaking and weakly referenced listeners	15
Compiler Warnings.....	16
Use meta tags	16
Use namespaces	16
Use Transient meta tag	16
For each and array vs Array collection	16
Contact Us	18

Read this before you start

<http://seanthelexguy.com/blog/2009/01/08/flex-best-practices-presentations/>

Project folder structure

We propose to start from the following structure, and extend if in case of need:

```
com
---company
-----project
-----collections
-----components
-----componenClasses
-----controls
-----controlClasses
-----events
-----formatters
-----models
-----renderers
-----services
-----skins
-----views
-----util
```

Code conventions

<http://blog.dclick.com.br/wp-content/uploads/adobe-flex-coding-guidelines-v12-english.pdf>

We added several changes into adobe conventions:

1. All private variables should have `_` prefix and be declared in the end of the file.

```
private var _selectedItems : Array = [];
private var _selectedItemsChanged : Boolean = false;
```

2. MXML namespaces declaration order.

```
<Application
  xmlns="com.myproject.app.*"
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:controls="com.myproject.app.controls.*"
  xmlns:model="com.myproject.app.model.*"
  width="200"
  height="200">
```

3. MXML declare each attribute in separate line, id should be declared in first line

```
<mx:List id="list"
  width="100%"
  height="100%"
  wordWrap="true"
  variableRowHeight="true">
```

4. CSS style declaration

```
.genericPadding0
{
  paddingLeft: 0;
  paddingRight: 0;
  paddingTop: 0;
```

Flex 3 development - best practices

```
paddingBottom: 0;
}

ToggleButtonBar
{
    horizontalGap: 2;
    buttonStyleName: "internatToggleButtonBarButtonStyle";
}

.internatToggleButtonBarButtonStyle
{
    paddingLeft : 0;
    paddingLight : 0;
}
```

5. Do not use anonymous functions, use internal functions.

```
Alert.show("Are you sure ?", "Question", Alert.YES | Alert.NO, null, onClose);

function onClose(event:CloseEvent):void // <--- CORRECT
{
    if (event.detail == Alert.YES)
    {
        Alert.show("You rock !");
    }
}

Alert.show("Are you sure ?", "Question", Alert.YES | Alert.NO, null,
function (event:CloseEvent):void // <--- WRONG
{
    if (event.detail == Alert.YES)
    {
        Alert.show("You rock !");
    }
}
)
```

6. For event handlers use "on" convention:
- onClick
 - onChange
 - onChildCheckBoxChange

7. Declare MXML imports in the end of the file:

```
<?xml version="1.0" encoding="utf-8"?>
<MyComponentBase
    xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Label
        text="Total: {DefaultFormatters.currencyFormatter.format(presentationModel.usageData.totalUsage)}"
        fontSize="20"
        fontWeight="bold" />

    <mx:Script>
        <![CDATA[
            import com.myproject.core.formatters.DefaultFormatters;
        ]]>
    </mx:Script>

</ MyComponentBase >
```

Flex 3 development - best practices

Also we used on several modules this eclipse plug-in: <http://flexformatter.sourceforge.net/>. Fast growing and well supported project.

Component development

Component life cycle

http://livedocs.adobe.com/flex/3/html/help.html?content=ascomponents_advanced_3.html

<http://flexcomps.wordpress.com/2008/05/09/flex-component-life-cycle/>

Before you start development you should read mentioned docs. You should know main lifecycle methods before you start development. You should use them as it was designed by adobe otherwise you'll have big problems☺:

1. constructor
2. createChildren()
3. commitProperties()
4. measure()
5. layoutChrome()
6. updateDisplayList()

Tips and tricks:

1. Do not call createChildren, commitProperties, measure, updateDisplayList directly. Just call invalidate methods.
2. Do not use x, y, width, height setters in updateDisplayList, use move and setActualSize instead.
3. Try to avoid using of creationComplete event. You can always override needed lifecycle method.
4. Try to avoid using of callLater method. When you need to use it, it should be signal for you that you might did something wrong.

Write item renderers correctly

http://www.adobe.com/devnet/flex/articles/itemrenderers_pt1.html

http://www.adobe.com/devnet/flex/articles/itemrenderers_pt2.html

http://livedocs.adobe.com/flex/3/html/help.html?content=cellrenderer_2.html

Sample of inline item renderer:

```
<mx:TileList x="29" y="542" width="694" dataProvider="{testData.book}" height="232" columnWidth="275" rowHeight="135" >
  <mx:itemRenderer>
    <mx:Component>
      <mx:HBox verticalAlign="top">
        <mx:Image source="{data.image}" />
        <mx:VBox height="115" verticalAlign="top" verticalGap="0">
          <mx:Text text="{data.title}" fontWeight="bold" width="100%" />
          <mx:Spacer height="20" />
          <mx:Label text="{data.author}" />
          <mx:Label text="Available {data.date}" />
          <mx:Spacer height="100%" />
          <mx:HBox width="100%" horizontalAlign="right">
            <mx:Button label="Buy" fillColors="[0?99ff99,0?99ff99]">
              <mx:click>
                <mx:Script>
                  <![CDATA[
                    var e:BuyBookEvent = new BuyBookEvent();
                    e.bookData = data;
                    dispatchEvent(e);
                  ]]>
                </mx:Script>
              </mx:click>
            </mx:Button>
          </mx:HBox>
        </mx:VBox>
      </mx:HBox>
    </mx:Component>
  </mx:itemRenderer>
</mx:TileList>
```

Flex 3 development - best practices

```
</mx:Button>
</mx:HBox>
</mx:VBox>
</mx:HBox>
</mx:Component>
</mx:itemRenderer>
</mx:TileList>
```

This sample will work, but if you'll try to use it on grid with a lot data you'll realize that it's very slow. Why?

1. 3 Containers is used.
2. Percent widths/ heights are used

Item renderers best practices:

1. Do not use containers. They are very slow. Always extend from `UIComponent` or from standard controls you want to show, like `CheckBox`, `RadioButton`, etc. We recommend to create base renderer (extended from `UIComponent`), implement `IListItemRenderer`, `IDropInListItemRenderer` interfaces and extend all your renderers from this base renderer.
2. Override `createChildren` if you want to add child controls
3. Override `updateDisplayList` if you want to layout your components
4. Override `measure` to calculate measured sizes
5. Override `setData` if you need to receive data object
6. Do not store any data in renderers. All data should be stored in data objects or owner lists. Take in mind that lists will cache renderers and they can be reused on scrolling.
7. Try to reuse default renderers. They have some useful features ☺ For example `ListItemRenderer` wordwrap text if `list.wordwrap` is true or `AdvancedDataGridHeaderRenderer` show sorting indicators, etc. So if you'll extend your renderer from `UIComponent` you'll miss default features.
8. Do not use `Label`, use `UITextField` instead. `Label` is ~3 times slower than `UITextField`
9. Style changing is very slow procedure. If you'll need to change style of component depending of data, for example you need green label when data is "done" and red otherwise. In such cases it can be better to have 2 controls with predefined styles.
10. Control creation is very slow procedure. So if amount of controls depends on data consider cache controls and just show / hide them.
11. Always remove all data event listeners and bindings on data change.

Several patterns what we use

Services Pattern

Do not work with RemoteObjects directly. Use services with the same methods / parameters as remove actions.

We use the following pattern:

```
package com.myproject.remoting
{
    import com.myproject.core.remoting.ServiceActionBase;

    public class TestDataPointAction extends ServiceActionBase
    {
        public function TestDataPointActionBase(resultHandler : Function = null, faultHandler : Function = null)
        {
            super("TestDataPointAction", resultHandler, faultHandler);
        }

        public function getDataBetween(date1 : Date, date2 : Date, testDataPointDTOEnum1 : TestDataPointDTOEnum) : void
        {
            call("getDataBetween", arguments);
        }
    }
}
```

On java we have class with the same method:

```
package com.myproject.remoting

import java.util.Date;
import java.util.List;

public interface TestDataPointAction {

    List<TestDataPointDTO> getDataBetween(Date from, Date to, TestDataPointDTOEnum interval);

}
```

Now when you have service you can use it:

```
new TestDataPointAction(onResult).getDataBetween(startDate.selectedDate, endDate.selectedDate, TestDataPointDTOEnum.HOUR
```

As for me it looks better then:

http://livedocs.adobe.com/livecycle/es/sdkHelp/programmer/lcds/wwhelp/wwhimpl/common/html/wwhelp.htm?con text=LiveDocs Parts&file=rpc_10.html

Benefits:

1. You can auto generate such services
2. You are using the same parameters / types as we have on server
3. You have single entry point for all calls so you can implement generic exception handling / logging / etc.

Also details are here (Russian): <http://tearaway-tea.com/blog/2009/02/оптимальный-паттерн-для-работы-с-remoteobject/ru/>

Code generation & Generation gap pattern

We investigated different solutions and in the end we choose GraniteDS code generator:

<http://www.graniteds.org/confluence/display/DOC/2.+Gas3+Code+Generator>

Features and benefits:

1. Can generate AS code by Java code
2. Has ant task for generation
3. Can generate DTOs / Enums / Lists / Interfaces
4. Implements Generation gap pattern (generate 2 classes real one and base class)
5. Has AS code templates (Groovy). You can edit your format of result code.
6. Open source. You can change generation if needed.

If you use BlazeDS you can use Granite generation but in this case you'll not receive Enum and Lists support. If you want Enum and Lists you should use GraniteDS or you can extend BlazeDS to support Enums and Lists.

We use the following convention: all remote DTOs and services what we need to use on our project should be generated from related java classes.

Code Behind

<http://www.onflex.org/ted/2007/02/code-behind-in-flex-2.php>

We are using the following convention:

1. Do not write any code in MXML.
2. Try to avoid MXML using when it's possible. Write AS classes.
3. If you need to layout many controls in your component use MXML for layout and base AS class for all coding.

Sample of MXML component:

```
<?xml version="1.0" encoding="utf-8"?>
<MyComponentBase
  xmlns="com.mycompany.myproject.*"
  xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Button
    label="Click me"
    click="onButtonClick ()"/>

  <mx:Label id="helloLabel"
    visible="false"
    text="Hello ..." />

</MyComponentBase>
```

Sample of Base component:

```
package com.mycompany.myproject
{
  import mx.containers.HBox;
  import mx.controls.Label;

  public class MyComponentBase extends HBox
  {
    public var helloLabel:Label;
```

Flex 3 development - best practices

```
protected function onClick():void
{
    helloLabel.visible = !helloLabel.visible;
}
}
```

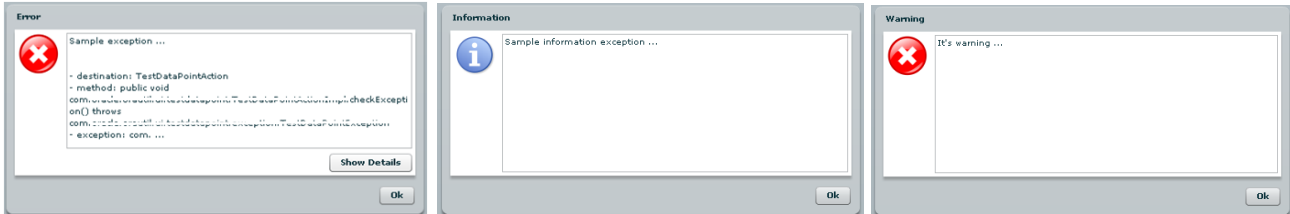
Tips and tricks:

1. If you want to access from Base class component defined in MXML you should define public variable with the same name and type in your Base class.
2. helloLabel from sample above will not be defined in createChildren. So if you'll need to such controls right after create to do some actions with them override createComponentsFromDescriptors function. In this function helloLabel will be already defined.

Several useful components / ideas

Alert Dialog and faults

We recommend to implement your own Alert. We implemented AlertDialog what can show image depending of alert type and supports show details / scrolling. Also our dialog is used together with services and can show advanced information about exceptions (stack trace, thread dump, etc ...)

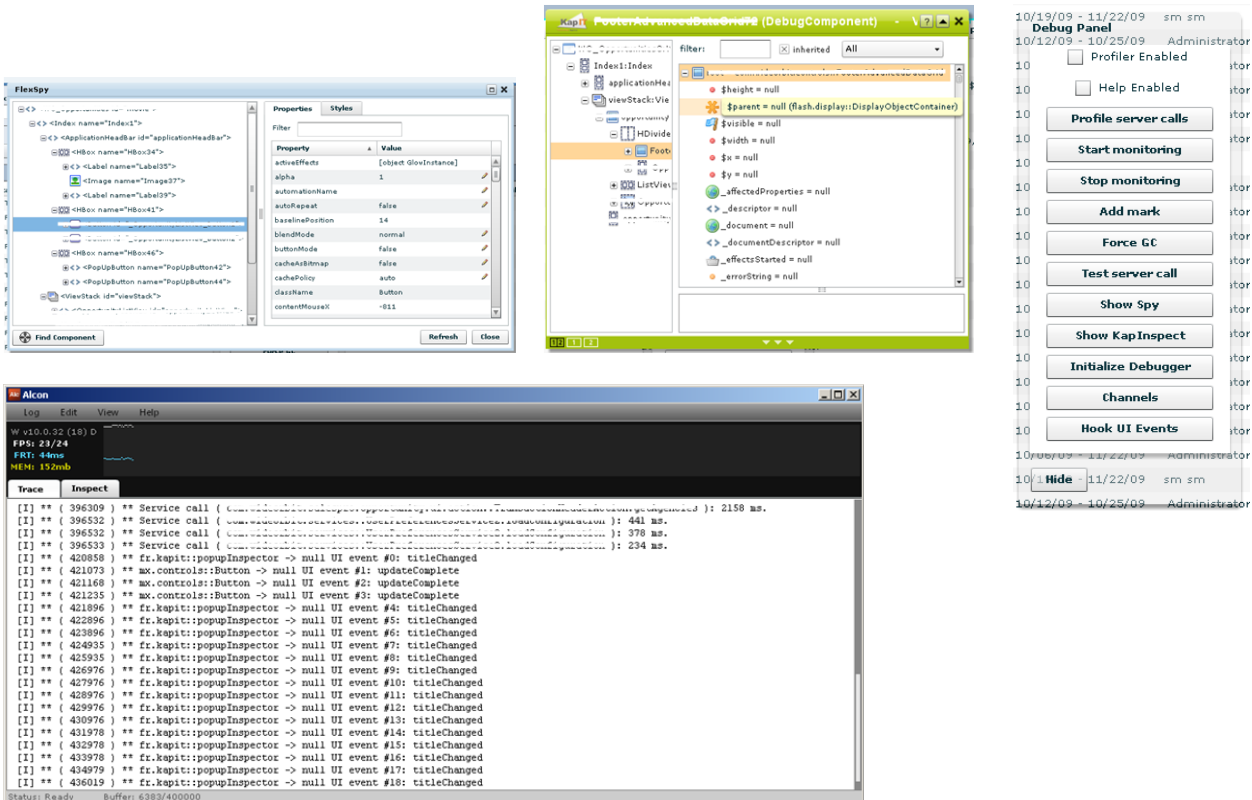


Default Formatters

ArrayFormatter, ChangeNumberFormatter, CurrencyFormatter, DateFormatter, DateTimeFormatter, DayMaskFormatter, FooFormatter, MaskFormatter, MillisecondsFormatter, MonthFormatter, NumberFormatter, PercentFormatter, TimeFormatter, WeekMask4Formatter, WeekMask5Formatter, WeekMaskFormatter, YearFormatter

Debug Panel

- <http://osflash.org/projects/alcon>
- <http://lab.kapit.fr/display/kapinspect/Kap+Inspect>
- <http://code.google.com/p/fxspy/>



Advanced Data Grid extensions

1. Formatters as column property, including sorting
2. Hidable columns / Expandable groups
3. Closable columns
4. Customization
5. Human sorting
6. Serverside pagination
7. Unique sorting
8. Fake column
9. Watermark
10. Stripy columns style
11. Fixed width for columns
12. Checked column support
13. Fixed problems with locked columns count

Misc

Frameworks

<http://tearaway-tea.com/blog/2009/03/usage-example-of-mate-flex-framework/ru/>
<http://tv.adobe.com/watch/360flex-conference/mate-flex-framework-by-laura-arguello>
<http://www.spicefactory.org/parsley/>

Child properties access

If you need to access properties of child objects outside of component, you should make them public. For example:

```
myComponent.helloLabel.text = "Hello"           // <--- WRONG
myComponent.grid.dataProvider = ["Hello"]       // <--- WRONG

myComponent.helloText = "Hello"                // <--- CORRECT
myComponent.dataProvider = ["Hello"]           // <--- CORRECT
```

In sample above we defined helloText and dataProvider public properties what internally reset values to helloLabel and grid.

Dynamic classes

Do not use dynamic classes and dynamic objects. Use strict typing.

Support selection on component level

<http://www.adobe.com/cfusion/exchange/index.cfm?event=extensionDetail&extid=1047969>

Sample bellow demonstrates 3 states tree renderer. It looks fine but state of item is supported on data level. It's not good because you can't put any data into this ComboBox you can put only xml.

Use binding carefully

Binding generation is not optimized. If you write the following code:

```
[Bindable]
public var startDate:Date = new Date();

[Bindable]
public var endDate:Date = new Date();
```

In result after generation you'll receive something like this:

```
[Bindable(event="propertyChange")]
public function get startDate():Date
{
    return this._2129778896startDate;
}

public function set startDate(value:Date):void
{
    var oldValue:Object = this._2129778896startDate;
    if (oldValue !== value)
    {
        this._2129778896startDate = value;
        this.dispatchEvent(mx.events.PropertyChangeEvent.createUpdateEvent(this, "startDate", oldValue, value));
    }
}
```

```
[Bindable(event="propertyChange")]
public function get endDate():Date
{
    return this._1607727319endDate;
}

public function set endDate(value:Date):void
{
    var oldValue:Object = this._1607727319endDate;
    if (oldValue != value)
    {
        this._1607727319endDate = value;
        this.dispatchEvent(mx.events.PropertyChangeEvent.createUpdateEvent(this, "endDate", oldValue, value));
    }
}
```

If you use auto-generation or want to create well optimized DTOs do write them in the following style:

```
[Bindable("startDateChanged")]
public function get startDate():Date
{
    return _startDate;
}

public function set startDate(value:Date):void
{
    if (_startDate != value)
    {
        _startDate = value;
        dispatchEvent(new Event("startDateChanged"));
    }
}
```

For this code flex compiler will not generate any additional code.

callLater usage

<http://www.screenshot.at/blog/2009/05/05/the-infamous-calllater/>

Try to avoid call later. If you need to use call later it should be a good signal for you that something is wrong with you component.

callLater can be used when you need to do math calculations in several frames.

Create components on demand

Do not create all components in the beginning, create all you need when you need and destroy when you don't need anymore. ViewStack with createPolicy= can be a good start for optimization

Common components library

1. Move all common styles into default.css
2. Declare all components you want to expose in manifest
3. Try to override all standard components and use in your modules only overridden ones
4. Etc ...

Memory leaking and weakly referenced listeners

http://www.gskinner.com/blog/archives/2006/07/as3_weakly_refe.html

<http://www.colettas.org/?p=115>

1. Use week reference event listener by default.

Flex 3 development - best practices

2. When you add event listener, always remove it when you don't need it anymore.
3. The same rule (see point 2) is also valid bindings when you use BindingUtils.

Compiler Warnings

Keep amount of working as small as possible.

You can use `-show-unused-type-selector-warnings=false` to minimize amount of warning raised by shared CSS usage.

Use meta tags

Use meta-tags in ActionScript controls and MXML components as well.

```
[Event(name="toolTipShown", type="mx.events.ToolTipEvent")]  
[Style(name="focusBlendMode", type="String", inherit="no")]
```

Code Complete will work better in IDE with correctly met tagged classes and you'll improve code readability / understandability.

Use namespaces

Just remember that you have such nice feature and use it in cases when you need it. We used in our core library in the same way as Adobe used their `mx_internal` namespace.

Use Transient meta tag

<http://flexonrails.net/?p=66>

It can help you to decrease amount of data you send to the server.

For each and array vs Array collection

If you think that **for each** will always work faster than **for**, then you should check sample below:

```
public function testArray():void  
{  
    var t:int = getTimer();  
    for (var i:int = 0; i < array.length; i++)  
    {  
        var e:Object = array[i];  
    }  
    trace("array with length in loop: " + (getTimer() - t));  
  
    t = getTimer();  
    var l:int = array.length  
    for (i = 0; i < l; i++)  
    {  
        e = array[i];  
    }  
    trace("array with length outside loop: " + (getTimer() - t));  
  
    t = getTimer();  
    for each (e in array)  
    {  
    }  
    trace("array for each: " + (getTimer() - t));  
  
    t = getTimer();  
    for each (e in array)
```

Flex 3 development - best practices

```
{
    e.p1 = 123;
}
trace("array for each with property change: " + (getTimer() - t));
}
```

And here we have results for 1000000 records array (objects)

```
array with length in loop: 181
array with length outside loop: 98
array for each: 39
array for each with property change: 244
```

And the most interesting part, result for array collection:

array collection with length in loop: 14417	80 times slower
array collection with length outside loop: 15359	156 times slower
array collection with length in loop and getItemAt: 6614	36 times slower
array collection with length outside loop and getItemAt: 4526	46 times slower
array collection for each: 7170	183 times slower
array collection for each with property change: 7919	32 times slower

Also we recommend to use ArrayCollection very carefully and in places where it's really needed. Just check differences and you'll understand why.

Contact Us

Lviv Development Center

EPAM Systems

45 Area A 20 Oleny Stepanivny Street

79018 Lviv, Ukraine

Phone: +380-32-239-5884

Fax: +380-32-239-5510

Kyiv Development Center

EPAM Systems

28 Fizkultury Street

03150 Kyiv, Ukraine

Phone: +38-044-390-5457

Fax: +38-044-390-5458

Global Headquarters

US Client Support and Delivery Center

EPAM Systems, Inc

41 University Drive

Suite 202

Newtown, PA 18940

Phone: +1-267-759-9000

Fax: +1-267-759-8989

sales@epam.com

www.epam.com