

SPICYNODES - Skinning subsystem

Developer's documentation

Document version: 1.2
Author: Andrew Answer
Date: 17.12.2006

Table of contents

Tasks overview	3
Class-based and procedure-based code conjunction	3
ColorManager	4
Skin mixing	4
Transitions package.....	6
Simplifying new skins development.....	7

Tasks overview

SpicyNodes skin developing is a good example of applying class-based programming model. It have common task set, typical for start-up projects with huge amount of mid-quality code, hardcoded directly in Flash source file, without versioning and syntax checking. This set consists of:

- Move code under SCM system
- Create build script
- Make fonts substitution to get right-compiled file
- Moving out all code from movie instances and frames to separate plain-text files
- Checking syntax and delete all code errors and duplications (for example, MTASC can't compile construction "**a>3?a=1:null**", that should be change to "**if (a>3) {a=1;}**")
- Slice long methods
- Describe functions
- Implement new functionality
- Adding code profiling and unit testing
- Writing documentation

Class-based and procedure-based code conjunction

Because of some part of project have only procedural code and can't be changed, some trick used for conjunction class-based and procedure-based parts. Look at this:

1. Initial code

```
class Something {
    var mc:MovieClip;
    function Something() {
        mc.onEnterFrame = function() {
            this._x += 50;
        };
        _root.Node.prototype.cLMc = function() {
            this.type = 0;
        };
    }
}
```

2. Slicing code to methods

```
import mx.utils.Delegate;
class Something {
    var mc:MovieClip;
    function Something() {
        mc.onEnterFrame = Delegate.create(mc, inc); // using Delegate.create
        _root.Node.prototype.cLMc = cLMc;
    }
    function inc() {
        this._x += 50; // error: no _x property in class
    }
    function cLMc() {
        this.type = 0; // error: no type property in class
    }
}
```

3. Using context reassigning

```
import mx.utils.Delegate;
class Something {
    var mc:MovieClip;
    function Something() {
```

```

        mc.onEnterFrame = Delegate.create(mc, inc);
        _root.Node.prototype.cLMc = cLMc;
    }
    function inc() {
        var node = this;
        node._x += 50; // right: node have "mc" context
    }
    function cLMc() {
        var node = this;
        node.type = 0; // right: node have "node" context
    }
}

```

ColorManager

Class ColorManager load colors from settings only once, and use internal cache for repeated requests. For assigning colors it use flash.geom.ColorTransform class.

```

import InterfacesManager;
import flash.geom.Transform;
import flash.geom.ColorTransform;
class name.answer.andrew.ColorManager {
    // colors cache
    var colorMap:Array;
    var paramMap:Array;

    public function ColorManager() {
        colorMap = new Array();
        paramMap = new Array();
    }
    function getColor(index:Number):Number {
        if (!colorMap[index]) {
            var color =
InterfacesManager.getInstance().getInterface("Colors").getColorById(index);
            colorMap[index] = color.val;
        }
        return colorMap[index];
    }
    function getParam(index:Number):Number {
        if (!paramMap[index]) {
            var param =
InterfacesManager.getInstance().getInterface("Colors").getParamById(index);
            paramMap[index] = param.val;
        }
        return paramMap[index];
    }
    function setColor(index, item) {
        var t = new Transform(item);
        var c = new ColorTransform();
        c.rgb = getColor(index);
        t.colorTransform = c;
        // code using deprecated Color API
        //var color:Number = getColor(index);
        //var col:Color = new Color(item);
        //col.setRGB(color);
    }
}

```

Skin mixing

Another task in this project – create a combination of skin parts without skin re-programming. It's done with BaseEngine.convertXML() and load() functions (see code for additional details), and interface inheritance, implemented in every skin. Look at the common project architecture:

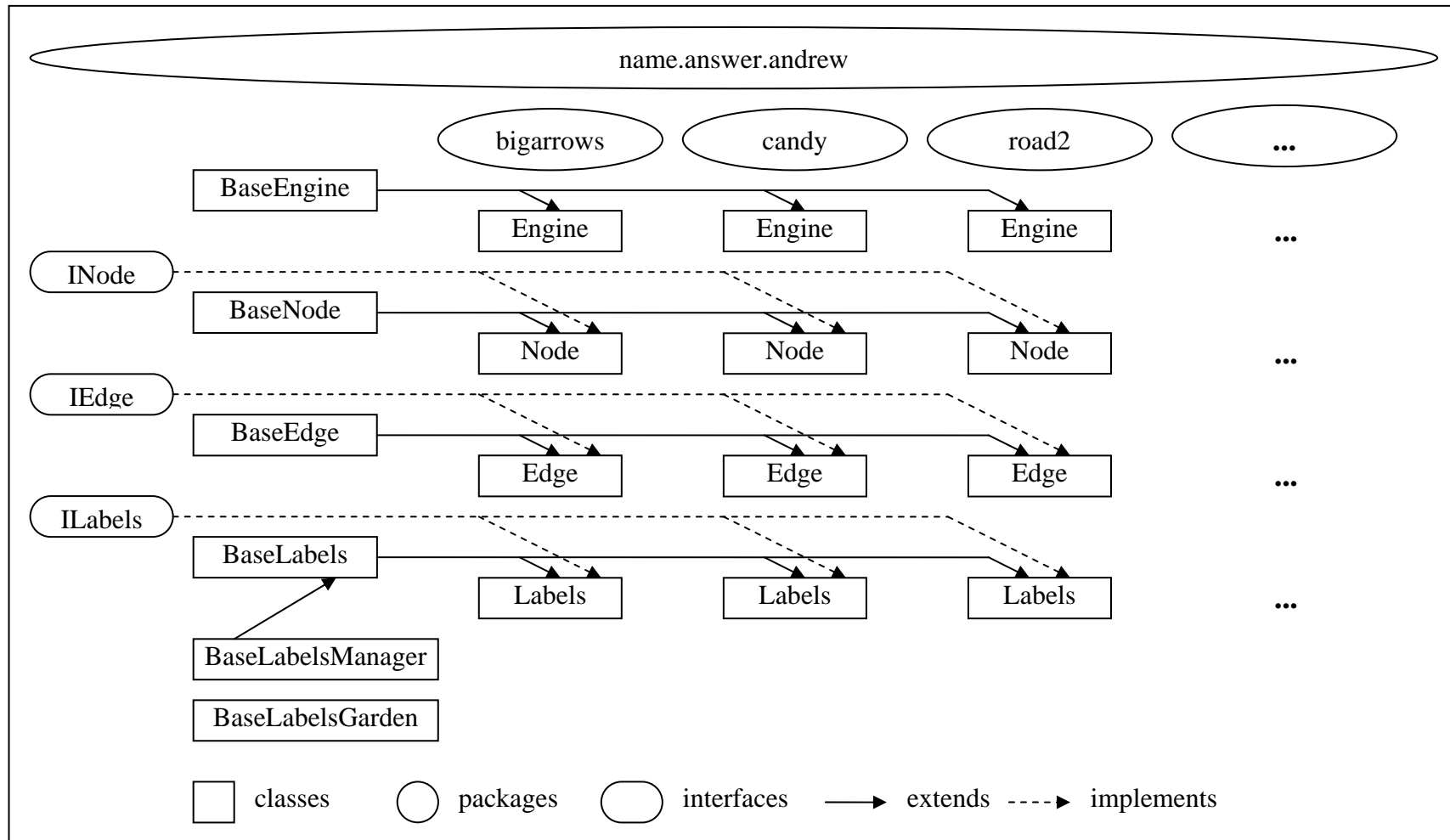


Figure 1. Common project architecture.

Also special way used for support calls of other class functions from different contexts (i.e. from mc.onEnterFrame handlers). Every Engine.as has static function getEngine which return Engine instance:

```
class name.answer.andrew.bigarrows.Engine extends BaseEngine {
    static var Base:Engine;
    public static function getEngine(): Engine {
        return Base;
    }
    public static function init(): Void {
        Base = new Engine();
        _root.getEngine = getEngine;
        _root.BaseEngine = Base;
    }
}
```

And BaseEngine have getters for currently loaded Node/Edge/Labels instance:

```
class name.answer.andrew.BaseEngine {
    public var CM:ColorManager;
    public var G:BaseLabelsGarden;
    public var N;
    public var E;
    public var L;
    function getColorManager():ColorManager {
        return CM;
    }
    function getLabelsGarden():BaseLabelsGarden {
        return G;
    }
    function getNode() {
        return N;
    }
    function getEdge() {
        return E;
    }
    function getLabels() {
        return L;
    }
    function getLabelsManager() {
        return L;
    }
}
```

So, when you call anywhere in the code `_root.getEngine().getNode().getActiveArea()`, you get actual result because a) format of this function described in INode, b) `getNode()` return right Node instance.

Transitions package

For arrows smooth animation special mx.transitions.Transition implementation was written, called Shape. It allow to use TransitionManager class with custom drawing function (only BaseEdge.arrowEdge now used, but it can be extended later) passed as parameter:

```
class name.answer.andrew.BaseEdge {
    function animateEdge(node:Object, d: Number, edgeColor:Number, p:Array,
effect) {
        edgeColor = _root.getEngine().getColorManager().getColor(edgeColor);
        if (effect==undefined) effect = None.easeNone;
        node.edge.transition = TransitionManager.start(node.edge, {type:Shape,
direction:Transition.IN, duration:d, easing:effect,
                obj: node, toColor: edgeColor, to: p, drawF: _root.arrowEdge});
    }
}
```

```
}  
}
```

To support other effects, new transition types can be implemented inside this package. Additional details about transitions you can find in Flash Professional 8 Help.

Simplifying new skins development

For new skins creation you could use next:

- create new package in classes/name/answer/andrew directory
- copy Engine/Node/Edge/Labels.as from other skin to that directory, for ex. from BigArrows
- create new FLA with graphic on “classes” level
- add path to “classes” to AS classpath in Flash IDE
- include “name.answer.andrew.[skin].Engine.init();” call in first FLA frame (for compiling with Flash MMC)
- modify your code by your needs.