

*ActionScript 3 basics in plain speak, for those coming from earlier versions of Flash. And who still like to use Flash not Flex or some other tool. And who are not programmers.*

## Part 3

### 'Object' classes and creating new items with code.

**duplicateMovieClip** and **attachMovie** no longer exist. Neither does **createEmptyMovieClip**. The downside of these were, mainly, the stacking order. Managing depth – or using `getNextHighestDepth`, became very cumbersome. Adding 'new (instance) names' for them also became tiresome.

The new method allows for a lot of **dynamic control** over what and how these new movie clips or sprites will behave. I am here talking to designers who want to have movie clips in the Library that are called by code when needed.

For now, I will not be looking at drawing object with code, nor using components. The word *object* refers to many things – color, date, etc. Here I am limiting the topic to **good old Movie Clips**.

We will need to consider 3 things for adding new movie clips to our movies.

1. Their Class
2. Adding them to the display list.
3. Controlling placement on screen.

#### 1. Class Definitions

All things in Flash have classes attached to them. Strings and Math have pre-built Flash classes, and as we have seen in some cases we need to import the classes in order to extend their usage in the movie we are working on; and in some cases it is a 'top level' class that is automatically included into any Flash movie.

Although Flash now gives one a great deal more **error** outputs if code is not correct – at times very helpful, at time very obscure - it handles classes in what I find to be a very peculiar fashion. [Here's an interesting if typically long-winded article by **Colin Moock** which covers error messages, amongst other things: <http://www.insideria.com/2008/07/the-charges-against-actionscri.html> ]

Make a new ActionScript 3 Flash movie. In the Property Inspector, type in any old nonsense for the 'Document class' and hit ENTER. You will get the following message:

*"A definition for the document class could not be found in the classpath, so one will be automatically generated in the SWF file upon export."*

Which means, it cannot find a class of that name (as it does not exist) so it will **just make up its own and continue happily without error**. Is it just me or does that seem a bit weird???

Now, in the case of a document class, this will probably cause problems – or be rectified by you – at some stage. By which I mean you either are doing an animation and it needs no code (in which case no problems will occur), or you will make a class and correctly link it in your own good time.

Knowing about this becomes very useful when we want to add movie clips to our movie though.

## **2. The Display List**

The 'display list' is the term for all visible elements in a Flash movie. These would be

- The Stage
- Movie Clips/Sprites, Text, Buttons, loaded SWFs and the loader object (into which SWFs are loaded).

It's really pretty much the 'nested' concept with a different name.

If we are drawing or placing these physically on the stage ourselves, they are automatically added in the stacking arrangement we intended (layers, 'bring to front', etc.) If we are adding them via code in the past we would need to specify the **depth/level** at which they are added. Flash now controls that so the only thing we do need to explicitly say is ADD THE ITEM.

For this we use the code '**addChild**'.

Note: Before we could set the depth of something to 70324; this is no longer legal. Flash adds things in chronological order; when something is moved the next item slips in to fill the gap – like tetris (if you are doing it correctly).

Children can be added in the 'base' movie as well as nested in other movie clips/sprites.

## **3. Placement**

Any attached movie clip would be added top and left of the stage according to the alignment point of that movie clip. Same thing here. We will need to set its x & y point.

### **Example Number 1**

Create a new Flash (ActionScript 3) movie.

Create a **square**, convert it to a Movie Clip symbol (aligned top left).

Delete the one on the stage.

Do the same with a **circle**.

As we would do in previous versions, in order to export something not on the stage or to gain access to an item in the library we need to set up **Linkage**.

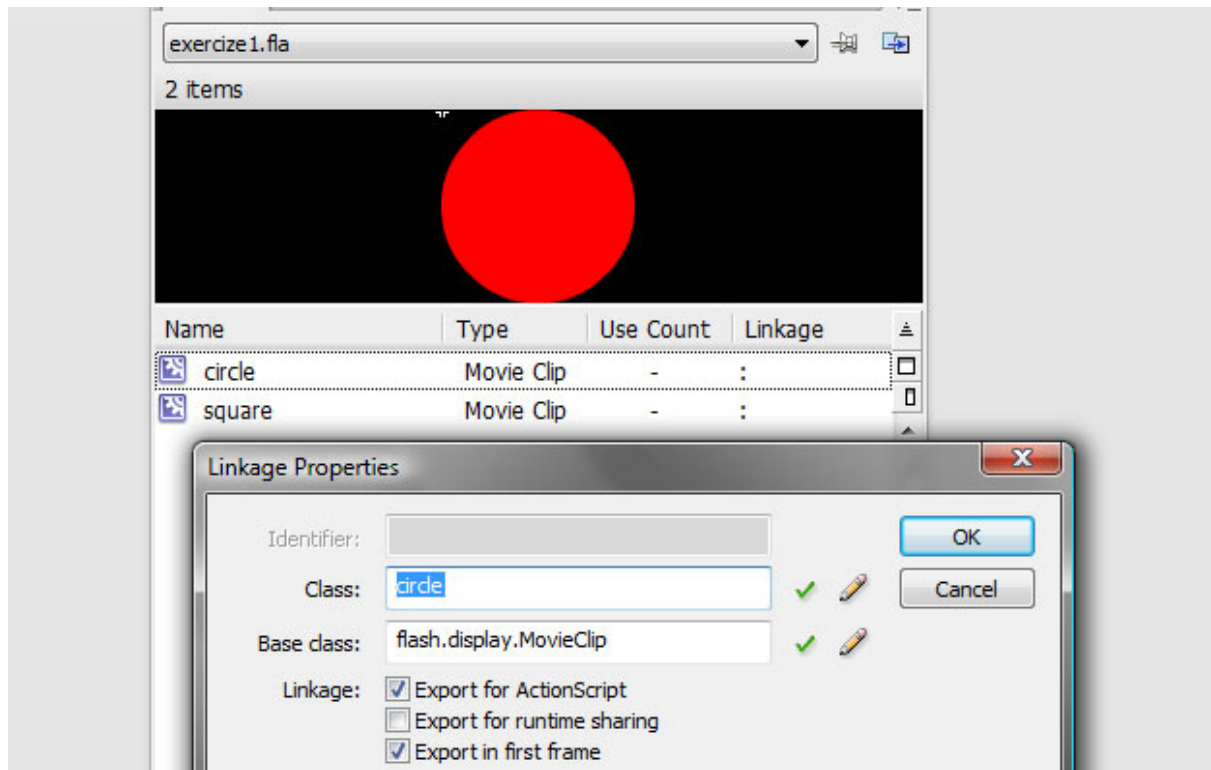
**Export for ActionScript & Export in First Frame** should be **checked**. (Export in First Frame will not give preloading issues as it used to as preloading is done in a different way – we will look at this later).

In the field labelled '**class**', leave the automatically entered name (will be same as the name of the item in the library).

Anything in the library will automatically have a **base class** of **flash.display.MovieClip**. It is the same as when we **extend** Movie Clip in our class code (*see Part 1*).

As previously mentioned, you will get an **alert** saying “A definition for the class could not be found in the classpath, so one will be automatically generated in the SWF file upon export.” when you say OK.

Not a problem, say **OK** again.



Do the same for both objects you drew.

Now, create a new **ActionScript file**, and save it into the **same** directory as the Flash movie you have just worked on. Call it “**MakeNew**”.

Set ‘MakeNew’ as the **document class** (in the Property Inspector, see Part 1 if in doubt).

Enter the following code into MakeNew:

```
package{

    import flash.display.Sprite; //or MovieClip

    public class MakeNew extends Sprite{ //or MovieClip - but must be same as above import

        public function MakeNew():void{

            addNewObjects(); //call function to add objects

        } //initialiser makeNew

        private function addNewObjects():void{
```

```

        var myCircle:circle = new circle;

//'circle' is name of linkage in library - note, no such class really exists and the name is not even
//Capitalized as is conventionally correct

        addChild(myCircle);

        myCircle.x = myCircle.y = 100;

        trace(getChildIndex(myCircle)); //traces 0

        var mySquare:square = new square;

        addChild(mySquare);

        mySquare.x = mySquare.y = myCircle.x + myCircle.width;

        trace(getChildIndex(mySquare)); //traces 1

    } //addNewObjects

} //class

} //package

```

The initial code should be comprehensible by now (if not read Part 1 & 2!), so all we need to look at is:

- 1] var myCircle:circle = new circle;
  - 2] addChild(myCircle);
  - 3] myCircle.x = myCircle.y = 100;
  - 4] trace(getChildIndex(myCircle)); //traces 0
- &
- 5] mySquare.x = mySquare.y = myCircle.x + myCircle.width;

Explanation of above:

Line 1] Make a new instance of 'circle' class and remember it by the name of myCircle. There is **no such class in reality**, but the object being exported has that '**name**' linked to it. So it magically works! (Please note, class names should be Capitalized.)

Line 2] Add this new item to the stage – make it visible

Line 3] Position it.

Line 4] For interest sake, find out '**depth**' it is added at (if you go to **Debug > List Objects** after Testing you may get some odd values, for these 2 items I got 'instance 1' and 'instance 3', with no number 2!

Line 5] We have not supplied instance names. You can, if you want by adding:

```
myCircle.name = "myInstanceName_mc";
```

Adobe recommends against using instance names. I am sorry, I find it useful at times. But in most cases, it really is *not necessary*, besides keeping track of things if you use **Debug > List Objects** a lot, as I do.

Here, you can see it's not necessary as the **variable name makes the movie clip** – and its properties – **accessible**. (Placing it next to the previous objects position.)

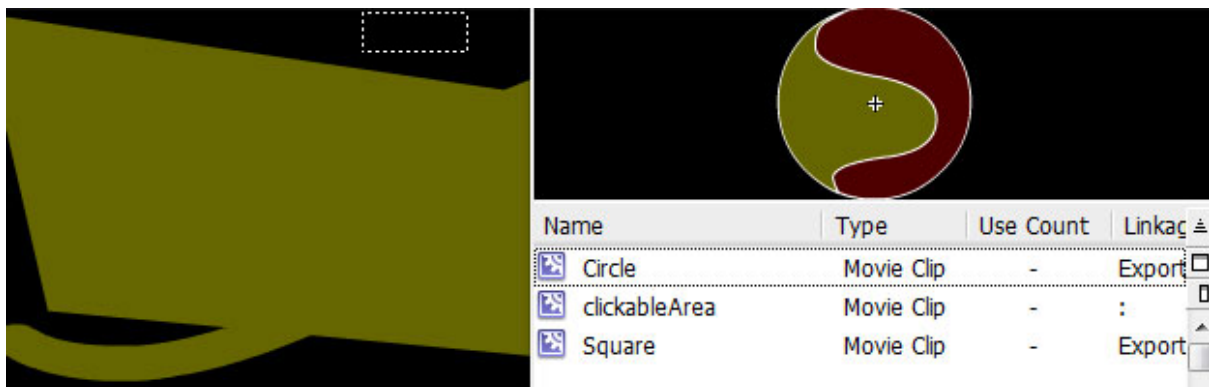
Now, isn't that easier and clearer than adding and setting depth and instance naming and so forth?

### **Example 2 – adding Object Classes correctly, and making our newly added sprites DO something.**

[Download the files for this example.](#)

If you wish to make it yourself from scratch, you will need to have the following in a Flash movie:

- A **dynamic text field**, instance named 'index\_txt'
- A **movie clip** that fills most of the stage area (irregular shape will make the result more interesting). No instance name required (but it must be a movie clip).
- **2 movie clips** in the library that you will notice rotate – due to names I have used I suggest a circle and a square. For the circle, draw something in it so you can see it spin!



*Stage, with 'clickableArea' (green) and text field (dotted line). | Library with Circle shown.*

Now, we will make 3 ActionScript files; 1 for the Document Class and one for each of the items to be added by code (Circle and Square). The Document class will add the objects; the object classes will make the objects do something.

New ActionScript file.

Save to same folder as FLA.

Name "MouseTrail".

Link to FLA as Document Class.

Add the following script:

```

package {

    import flash.display.*; //movie clip and stage
    import flash.events.*;
    import flash.text.*; //we have a text field

    public class MouseTrail extends Sprite{

    public function MouseTrail():void{

        addEventListener(MouseEvent.CLICK, addNewShapes);
        //will only respond to "filled" area of the movie – where the movie clip is!
        buttonMode = true;
        // see a hand cursor over clickable area

    } //initializer

    private function addNewShapes(event:Event):void{

        if(mouseY < (stage.stageHeight/2)){ //if mouse is in top half of movie

            var newBox:Square = new Square(); //add new Square class object

            addChild(newBox); //add to display - make it visible

            newBox.x = mouseX; //position

            newBox.y = mouseY;

            index_txt.text = String(getChildIndex(newBox)); //find out its "depth"

        }else{ //if mouse is in lower half of movie

            var newCircle:Circle = new Circle(); //add new Circle class object

            addChild(newCircle);

            newCircle.x = mouseX;

            newCircle.y = mouseY;

            index_txt.text = String(getChildIndex(newCircle));

        }

    } //addNewShapes

    } //class

} //package

```

Save. Do not test the movie yet – we must make the Object classes first.

Create new **AS file**, save as **Circle**. Add the following into it:

```
package {

    import flash.display.Sprite; //it may be movie clip but only has one frame
    import flash.utils.Timer; //required for timer
    import flash.events.*; //timer event

    public class Circle extends Sprite{

        public function Circle():void{

            var myCircleTimer:Timer; //declare new timer object

            myCircleTimer = new Timer(300, 0);
            //amount of milliseconds is first value
            //amount of times to repeat is second parameter - 0 is forever until told to stop

            myCircleTimer.addEventListener("timer", timerEvent);
            //tell it what to do each time timer executes

            myCircleTimer.start(); //timer is told to start

        } //initializer

        private function timerEvent(event:Event):void{

            rotation += 5;

        } //timerEvent

    } //class

} //package
```

The only thing new here is the **Timer object** – a preferable way to loop than **enterFrame** or **SetInterval**, which executes code a specified amount of times at specified intervals. See code comments above for the structure.

Remember, any function called by an event – mouse, timer, whatever – will have **event:Event** in its parameters or an error will display.

As mentioned before, you could do (event:MouseEvent) but that would mean **only** mouse events could execute it. 'Event' is more flexible.

Let's create the second Object Class, then we can link them to the objects and we're done.

New **AS file**. Name it **Square**, saved into same folder as all the others. Add code:

```

package{

    import flash.display.Sprite; //it may be movie clip but only has one frame
    import flash.events.*; //mouse event

    public class Square extends Sprite{

        public function Square():void{

            addEventListener(MouseEvent.MOUSE_MOVE, flipSquare);
//mouse move is when in the objects area, unlike previous actionscript
//where mouse move was anywhere in stage area

        } //initializer

        private function flipSquare(event:Event):void{

            rotation += 45;

        }

    } //class

} //package

```

No big deal there.

### **Link the Object Classes.**

Right click on the circle or whatever its library name is, go to **Linkage**, check on Export for ActionScript. For class name, it will automatically insert the symbols name. **Type in the correct class name as per the one you created for the object – in this case ‘Circle’**. Do the same for the other object (but make the class to ‘Square’). Remember, just the name of the AS file, do not include the extension.

Save and test the movie.

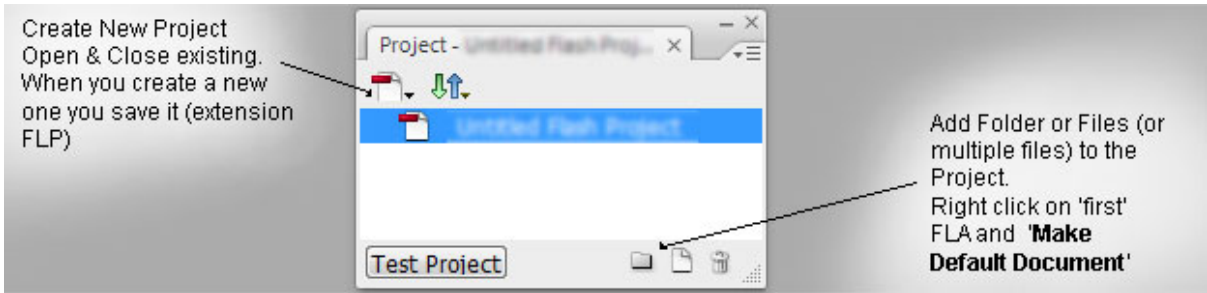
Note that the hand cursor and clickable area is not the stage (though the document class does not specify what is clickable) but the filled area. It will not work if the filled area is a graphic-type symbol though. Very clever. Potentially a way to make a very easy maze?

Finally, we will look at a way to make having multiple files in a ‘project’ more manageable. That’s right - the Project Window.

### **Project Window**

Go to **Window>Project**.

Here we can add files to project, set the ‘default’ file and use this to open and work with all our files. Much like defining a site in Dreamweaver.



Now you will be able to just open the FLP to gain quick access to all your files. You will still need to save changes to AS files before testing.

**In Part 4: Object Class Parameters – making them really dynamic! Sharing classes between more than 1 movie clip.**