

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 2

This is where we left off:

```
package{

    import flash.display.*;

    import flash.events.*;

    public class StartPoint extends MovieClip{

        function StartPoint(){

            trace("StartPoint started");

            //initialiser

        }//class

    }//package
```

When you test in now, all that should happen is it should trace the string we asked it to.

Variables

Variables should have **data types** (you may have been doing this already). For example:

```
var myText:String;
var myNumber:Number;
or an object type
var myNewMC:MovieClip;
```

The data type comes after the variable name, and is only done once – when you declare the variable.

ActionScript 3 adds in a few new ones, notable:

```
var myNumber:uint;
```

Uint represents a **NON-negative** number (if it gets a negative value expect Flash to freeze!) You also use uint to declare the type for a **colour**.

There is also **Sprite** type, **SimpleButton** type and your own defined types. And ones you may be familiar with – Array, Date, Number, String, int (non-decimal number that can be negative) and **Boolean** (true or false – **NOTE: Flash will no longer accept use of 0 or 1 for Booleans**).

Finally, you can set it to not having a specific type by using the **wildcard**, e.g.

```
var myUnspecifiedVar:*;
```

Classes, variables and function can – and should (it depends on how **strict** your ActionScrip setting are – see *Part 1*) have **attributes** that decide how - and where - the information can be **accessed**. Its kindof confusing, but think of something like (the no longer existing) **global** variable – which can be read and accessed at any point in a movie vs. a local variable.

If you are not familiar with variable scope, here’s a very potted version.

variable A

```
function whatever(){
variableB;
}
```

```
function other(){
    at this point I can access variableA as it was declared outside of the functions.

    I cannot access variableB as it was declared inside a different function.
}
```

Here’s a table laying out what it means, copied from the Adobe Flash CS3 Help File.

| Attribute | Definition |
|----------------------|---|
| internal (default) | Visible to references inside the same package. |
| private | Visible to references in the same class. |
| protected | Visible to references in the same class and derived classes. |
| public | Visible to references everywhere. |
| static | Specifies that a property belongs to the class, as opposed to instances of the class. |
| UserDefinedNamespace | Custom namespace name defined by user. |

For the most part, private is good to use while public used all the time is bad practise. I think this is something that makes more sense the more you do it; no one ever got anywhere by being perfect from the start. Try to think in terms of if you will need to access something again (public) or not (private). It should be indicated for all functions EXCEPT the one that initialises the class – has the same name as the class. But if you do include it there it causes no problems I know of.

Functions

Not much new to look at here, except for the void bit. Here’s an example.

```
private function myExample():void{
    trace (“This function has no return in it”);
}
```

```
private function myOtherExample():String{  
    return("This function has a return in it – which returns a string");  
}
```

In other words, if your function has the **return** method in it, work out what type of data will be returned and set the type after the function name. In all other cases, type the word **void**.

And because curly brackets look pretty with colons, you also type any parameters passed to your function 😊

```
public function lastOne(thisValue:uint):void{  
    trace(thisValue);  
}
```

Let's make something.

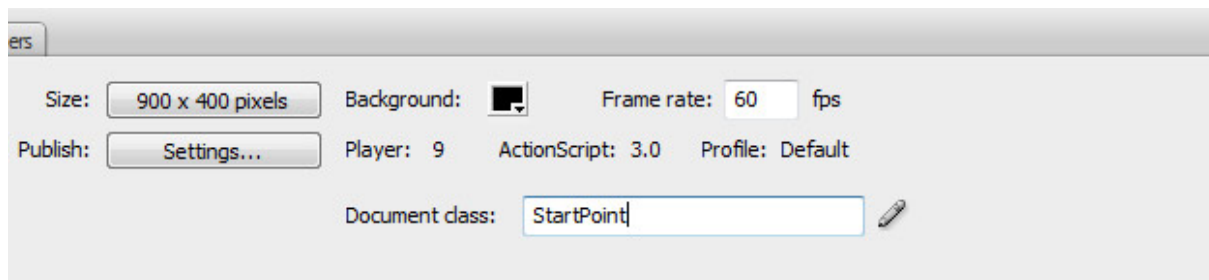
While we do this, we will also look at mouse events as well as changes to properties.

If you have already made the files mentioned in Part 1, use those. If not, we need to do the following before we get started:

Create a **Flash file**, name it whatever you wish. Draw a box and convert to movie clip symbol. Give it an instance name of "box_mc".

Create an **Action Script** file, and save it as **StartPoint** into the same folder as the fla.

Link the as file to the flash movie (in the Property Inspector, where it says document class, just type in the name of the as file, not the extension).



Type (or paste) the following code into the as file. (It was all explained in Part 1, with added bits from what was discussed above).

```
package{  
    import flash.display.*;  
    import flash.events.*;  
    public class StartPoint extends MovieClip{
```

```

    public function StartPoint():void{

        trace("StartPoint started");

    }//initialiser

} //class

} //package

```

Save.

Now, I want to be able to click the movie clip and move it across the stage each time I do so. Simple in previous ActionScript versions, trickier now.

MAY I JUST SAY at this point – I know it looks complicated and ‘unnecessary’ but once things are set up it really is a timesaver and makes a lot of tasks much easier (and some harder!). So hang in there.

As mentioned before, buttons and movie clips cannot have actions put “on” them any longer. This is a good thing – it makes everything tidier. So we need to basically do two things now.

1. **Tell the movie clip to “listen” for a mouse event.**
2. **Tell the movie clip what to do when the mouse event happens.**

So, firstly, as soon as my ActionScript file (or class really) executes I want to say that I can click the movie clip. I will add into my ‘initialiser’ the following:

```

public function StartPoint():void{

    trace("StartPoint started");

    box_mc.addEventListener(MouseEvent.CLICK , moveClickedBox);

```

There are a few parameters of the addEventListener function – many of which you will seldom if ever need to use or specify. We can look at that another time.

We will need to specify the mouse event type – once you type the dot after **MouseEvent** you will see them listed.

(Note: dragOver and dragOut no longer exist).

After the mouse event, we put in the name of the function that will be executed. Next to write that function.

We only have one class in a package, but we can have many functions in that class. The initialising function has the same name as the class (and as file) and is capitalised by convention, the rest are not.

After the close curly bracket for the StartPoint function add **(not the numbers in square brackets!!!)**:

```

    private function moveClickedBox(event:MouseEvent):void{    [1]

        trace("clicked box:" + event.target.name);            [2]

```

```
    this.x += 10;
```

```
    [3]
```

```
    }
```

Save the file, test the fla. It will work – maybe does something odd but you may not notice that right now.

Let's examine the above code.

Line [1]: The function does not get accessed outside of the class, hence 'private'.

In the functions parameters we have to specify that it is responding to an event, of the type MouseEvent. You can also just say 'Event' (handy if it will also be executed by a non-mouse event). If you exclude this, Flash will output an error about expecting 1 and getting 0.

Note: *it will often occur that Flash will notify you if you do not have the correct number of parameters for a function.*

Line [2]: I want to find out what I have clicked on, and I do this by finding the name of the clicked "target". I will use this information later.

Line [3]: I move "this" across by 10 pixels. But who is this? **Duplicate** the box on your stage, change the instance name to "**box2_mc**" and you will see "box_mc" is not moving – the entire stage is. So **this** refers to the *object that the script is linked to* – which is our document.

We need to fix this, so let's alter line [3] to

```
    event.target.x += 10;
```

So the object responding to the event now moves.

Changes to properties.

All properties lose their underscores.

_x becomes **x** and so on.

_xscale becomes **scaleX** (and the same for scaleY)

_xmouse becomes **mouseX**

Alpha is now 0 for 0% and **1 for 100%**. So 50% is 0.5. Just divide your required value by hundred every time.

Visible, being Boolean, must be **true or false**. 0 and 1 will be rejected.

And finally – the bumper.

For reasons known to Higher Powers, the above method means that I cannot send through my own parameters to the function. I cannot say:

```
box_mc.addEventListener(MouseEvent.CLICK , moveClickedBox(leftOnly));
```

I have seen some long and complicated ways around this. For now I just want both boxes on the stage to be clickable.

The cursor, you enquire?

Ah yes, 'simpleButtons' have automatic hand cursors. Movie Clips and Sprites do not. It's like Director all over again!

Here is the **final code**, with both boxes moving and hand cursors showing.

```
package{

    import flash.display.*;

    import flash.events.*;

    public class StartPoint extends MovieClip{

        public function StartPoint ():void{

            trace("StartPoint started");

            box_mc.addEventListener(MouseEvent.CLICK , moveClickedBox);

            box_mc.buttonMode = true;

            box2_mc.addEventListener(MouseEvent.CLICK , moveClickedBox);

            box2_mc.buttonMode = true;

        } //initialiser

        private function moveClickedBox():void{

            trace("clicked box:" + event.target.name);

            if(event.target.name == "box_mc"){

                event.target.x += 10;

            } else if(event.target.name == "box2_mc"){

                event.target.x -= 10;

            }//if

        }//moveClickedBox

    } //class

} //package
```

So, after adding the event listener, we set **buttonMode** to true to get the hand cursor.

We use the instance name of the movie clip to determine how it will respond to the function – one box moves left and the other right.

In Part 3: 'Object' classes. Adding new object via code.