/Flash/create an interactive map

Your essential CD
All the code from this month's tutorial can be found on this issue's CD.

Show off all the places you've visited with an interactive Flash-enabled map. Designer **Martin Dingley** (www.soapslurp.co.uk) explains how to go about it

Knowledge needed: Flash CS3, ActionScript, XML
Requires: Flash CS3
Project time: 1 hour

Most of us like to travel, and some of us have been lucky enough to have visited a good dozen places across the globe. What better way to show this than a Flash map that users can interact with to find out more about your travels?

On the CD, open **map-dashboard.fla**. The file has been set up, the frame rate has been pushed from 12fps to 30fps for smoother animations, and all the countries are individually grouped into movieclip symbols.

A closer inspection of the movieclip shows they've been given a unique instance name using the country codes. Later we'll use this as a hook to locate and map the data from our XML to the correct country.

The XML

Let's go over a sample of the xml structure we're going to be using in this tutorial. On the disc, open the file **mapdata.xml**.

It's a good rule of thumb to identify the document as being XML. XML gives us the ability to do this using the XML declaration.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

This declaration tells the parser which XML version we're using and the character encoding.

We're trying to keep the XML clean and readable: any country-specific information such as its name and code is formatted as an attribute to the node named country. If we were to have more information about the country – let's say we've visited France on two different occasions – then the details would be held within the node(s) named param for each of these visits.

You may notice the countries without any additional data look different – all elements must have an opening tag and a closing tag, with the exception of empty elements; these can be closed with a slash at the end.

Parsing our data

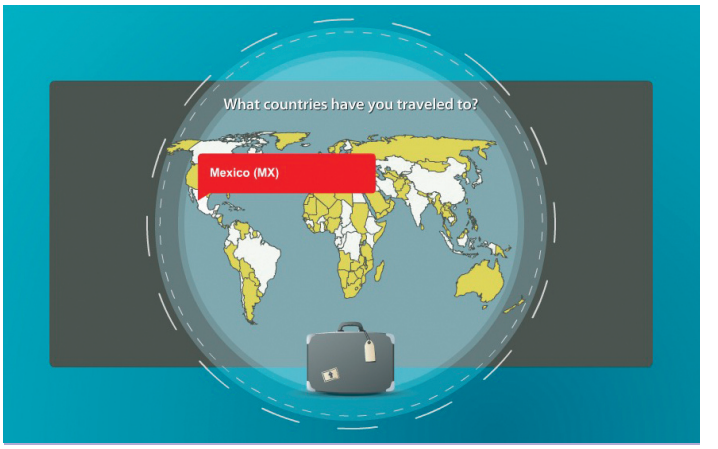
I find it useful to get the data into a manageable form inside Flash before getting bogged down with how and where you're going to display it. On the disc open up **partial-dashboard.fla**. Let's create a new XML object:

```
var mapDataXML: XML = new XML();
```

Next we need to tell Flash to ignore any white space inside the XML document. White space refers to characters that appear as blank but still include information that can confuse the Flash player.

```
mapDataXML.ignoreWhite = true;
```

The onLoad handler will tell the Flash movie what to do when the XML has been loaded into the Flash. In this instance we want it to run the function **parseXMLData**.



What you'll learn We're going to create a dynamic way of registering the different countries we've visited onto a Flash-enabled map

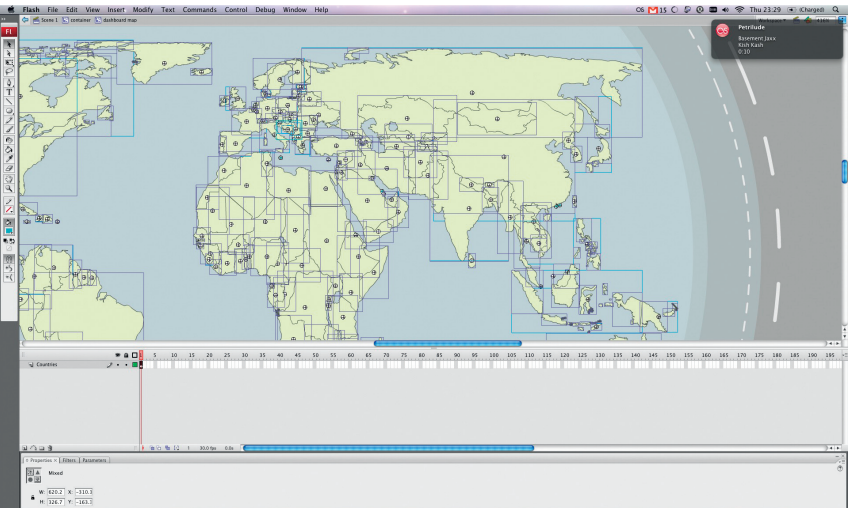
```
mapDataXML.onLoad = parseXMLData;
```

Let's get the ball rolling and load our XML into our movie.

```
mapDataXML.load("../xml/mapdata.xml");
```

OK, now we're going perform the actions to parse our data into objects we can easily access in our Flash movie.

```
function parseXMLData(success: Boolean) : Void {
 if (!success) return;
 xmlRoot = this.firstChild;
 xmlRootTotal = xmlRoot.childNodes.length;
 for (var k: Number = 0; k < xmlRootTotal; k++) {
   module = xmlRoot.childNodes[k].nodeName.toLowerCase();
   if (module == 'countries') {
     dashboardRoot = xmlRoot.childNodes[k];
     dashboardTotal = dashboardRoot.childNodes.length;
     dashboardInfo = new Array();
     for (var i: Number = 0; i < dashboardTotal; i++) {
       var DBCountryCode: String = dashboardRoot.childNodes[i].attributes.
code;
       dashboardInfo[DBCountryCode] = new Object();
       dashboardInfo[DBCountryCode].name = dashboardRoot.childNodes[i].
attributes.name;
       dashboardInfo[DBCountryCode].code = dashboardRoot.childNodes[i].
attributes.code;
       dashboardInfo[DBCountryCode].params = new Array();
       dashboardParamLength = dashboardRoot.childNodes[i].childNodes.
length;
       for (var n: Number = 0; n < dashboardParamLength; n++) {
```



Movieclips Each country has been converted into a movieclip and given an instance name. All country outlines have been grouped

```
        dashboardInfo[DBCountryCode].params[n] = new Object();
        dashboardInfo[DBCountryCode].params[n].title = dashboardRoot.
childNodes[i].childNodes[n].childNodes[0].firstChild.nodeValue;
        dashboardInfo[DBCountryCode].params[n].desc = dashboardRoot.
childNodes[i].childNodes[n].childNodes[1].firstChild.nodeValue;
      }
    }
    getDashboardCountriesMC();
  }
 }
}
```

You can find a commented version of the code above on the CD for a more step-by-step detailed explanation of what everything is doing.

What we've done is create an associative array, which is an array that has named indices, not numeric ones. For us to access country specific information, all we need to use is the country code. Finally, once our array has been built, we want to run the function **getDashboardCountriesMC**.

Mapping everything together

We're going to start by creating ourselves some little helpers. The **getCountryCode function** will return the country code we added to all our country movieclips when used with **_name** property.

```
function getCountryCode(str:String) { return str.substr(8);                    }
```

The second function will adjust the colour property of any movieclip we pass to it.

```
this.stop();

// CONSTANTS
var BUT_ROLL_OUT = 0xe0d453;
var BUT_ROLL_OVER = 0xFFFFFF;
var BUT_DEFAULT = 0xF8FBF5;


var stageWidth:Number = Stage.width;
var stageHeight:Number = Stage.height;

var count:Number = 0;
var duration:Number = 50;



var mapDataXML:XML = new XML();

mapDataXML.ignoreWhite = true;
mapDataXML.load("xml/mapdata.xml");
mapDataXML.onLoad = parseXMLData;


function parseXMLData(success:Boolean):Void {

  if (success) {
```

New XML Object Before we can start working with the XML, we first need to create a new XML object and then import the XML into our Flash movie

Flash alternatives

Don't feel confined to the Flash IDE

Some people find the Flash IDE more of a hindrance than a help, and some people just feel more comfortable working with ActionScript in an editor that they can use day in and day out, when working with HTML.

Macromates Textmate is my weapon of choice. (Unfortunately for those of you using Windows this is only available for Mac and the future doesn't hold much hope of this changing any time soon – check out e-editor.) Textmate is an extremely powerful text editor specialising in saving you time. The program is extendable and has a huge and active community behind it.

Don't worry though, Windows users, we haven't forgotten about you. FlashDevelop, is a .NET open source editor for Flash and web developers which is sure to fit the needs of many, with full support for ActionScript 2 and ActionScript 3 development, offering a host of tools at the ready, more noticeably for some intelligent code completion, which is a gem if you ever suffer from a lazy finger or two.

Similar to Textmate, running and compiling your Flash movie can be set up with the same key combination you're used to using in Flash; this saves you one trip.

Importing any external ActionScript 2 file into your main timeline involves adding a single line of ActionScript code to your timeline; just remember not to close off the include with a semi colon.

```
#include "init_script.as"
```

```
function changeColour(mc: MovieClip, colour: String) : Void {
 var newColour = new Color(mc);
 newColour.setRGB(colour);
}
```

We're telling Flash to create a new colour object based upon the movieclip. The colour object has several methods available to it; we're only interested in setRGB. This will give our colour object its new colour value based on the hexadecimal value we pass to it. Next create a new array, which will act a temporary container for our country movieclips on the Flash stage.

```
function getDashboardCountriesMC():Void {
```

The frame rate has been pushed from 12fps to 30fps for smoother animations

Iterate through all instances in the movieclip **dashBoardMap_mc**, which contains all our individual countries.

```
for (i in container_mc.dashboardMap_mc) {
```

We only want movieclips; we can check this using the type of operator. This can return any of the following as a string: string, movieclip, object, number, boolean, object, function.

```
if (typeof (container_mc.dashboardMap_mc[i]) == "movieclip") {
```

Good: now we know they're all movieclips, we need to iterate through all the items in our array we created when parsing the XML data.

```
for (j in dashboardInfo) {
```

Check that we have a country movieclip on the stage, which shares the same country code in its instance name, and then push that into the temporary array we created earlier.

>>

Extra interaction

Add some buttons for users to play with

An obvious limitation of the map we're creating is the scale. Some of the harder to reach countries such as Malta and Jamaica are really hard to interact with. We could add some buttons to the movie to increase the scale, giving the user further tools to interact with our map. To achieve this we could just add three constants to our ActionScript, which will be used to set the maximum, minimum and incremental scale values of the map.

```
var MAX_SCALE:Number = 600;
var MIN_SCALE:Number = 100;
var INC_SCALE:Number = 20;
```

Once you've created yourself some spiffy little buttons and assigned them both instance names you can then create a function to handle the scale. In this example we have a button on the stage with an instance name of scaleMapDown.

```
scaleMapDown_btn.onPress = function() {
 if (container_mc._xscale > MIN_SCALE) {
   container_mc._xscale = container_mc._yscale -= INC_SCALE;
 } else {
 //do something
}
```

When the user invokes the onPress event handler for our button, we first check to see that our map is greater than the minimum scale. If this condition is met and returned true, then we adjust the scale of the map by 20, by subtracting this value from the movieclips _xscale and _yscale. Otherwise we can provide the user with some suitable feedback to say "Hey! you are unable to scale the map any further".

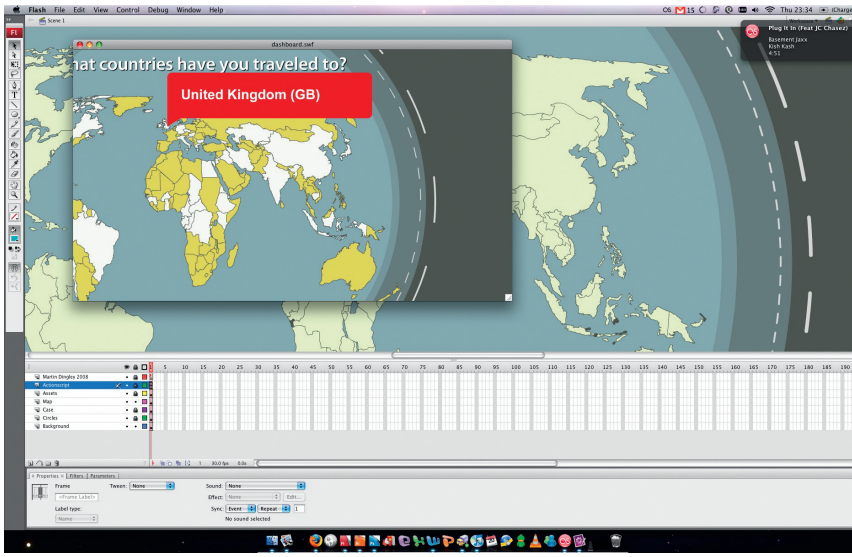
```
>>  if (dashboardInfo[j].code == getCountryCode(container_mc.dashboardMap_
    mc[i]._name)) {
newClipArr.push(container_mc.dashboardMap_mc[i]);
}
}
}
}
```

Initiate a setInterval

Finally we want to initiate a setInterval. This will in turn fire off another function that will give the user feedback on each of the countries loading progressively, rather than all at once.

```
    },
    TM:[object #178, class 'Object'] {
      name:"Turkmenistan",
      code:"TM",
      params:[object #179, class 'Array'] []
    },
    UG:[object #180, class 'Object'] {
      name:"Uganda",
      code:"UG",
      params:[object #181, class 'Array'] []
    },
    UA:[object #182, class 'Object'] {
      name:"Ukraine",
      code:"UA",
      params:[object #183, class 'Array'] []
    },
    AE:[object #184, class 'Object'] {
      name:"United Arab Emirates",
      code:"AE",
      params:[object #185, class 'Array'] []
    },
    GB:[object #186, class 'Object'] {
      name:"United Kingdom",
      code:"GB",
```

Output Our output window is a crucial tool in the development. It allows us to debug the movie and check that all the necessary data has been successfully imported



Testing Throughout the build of this project we can continually check our progress by previewing the movie within Flash

```
 intervalId = setInterval(callBack, duration);
}
```

The **callBack** function checks that the count is less than the total amount of countries. If this condition is returned true then it will call our **addDashboardMovieClipProps**, otherwise it will terminate the setInterval using **clearInterval();**.

```
function addDashboardMovieClipProps(mc: MovieClip) : Void {
 changeColour(mc, BUT_ROLL_OUT);
 mc.onRollOver = function() {
   changeColour(this, BUT_ROLL_OVER);
   countryCode = getCountryCode(this._name);
   displayCountryInfo(dashboardInfo[countryCode].name + " (" +
dashboardInfo[countryCode].code + ")");
 };
 mc.onRollOut = function() {
   changeColour(this, BUT_ROLL_OUT);
   removeTip();
 };
}
```

We pass into this function the movieclip (country) that we want to add the events to. We start by setting the default colour.

This is different from the initial load of the movie, so the user can tell what country(s) they can interact with.

Creating the tooltip

So everything's great, we can interact with our map now, but we could give the user some more information.

Let's give them some feedback by displaying the name of the country they've hovered over in the form of a tooltip.

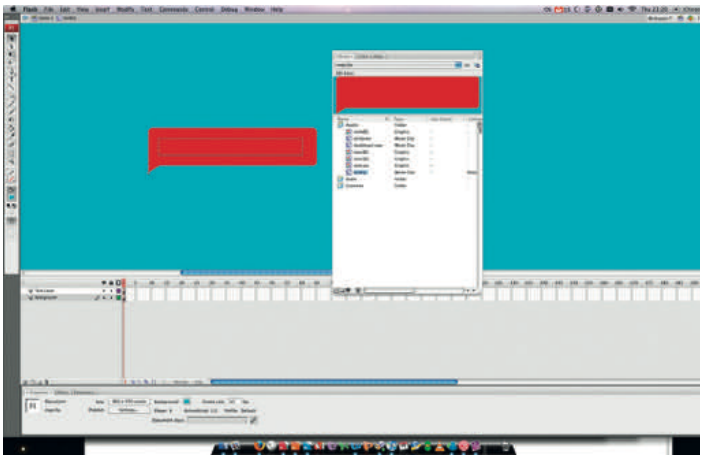
Start by creating a new movielclip symbol: **insert > new symbol**. You should be prompted with a dialogue box asking you to select the name and type. Give your new movieclip symbol a name of **tooltip** and double-check the type has been set to **movieclip**.

Next we want to expand the options available to us for this symbol. Click on the **advanced** button.

We're only interested in the linkage properties. Tick **Export for ActionScript**. This will enable us to give our symbol a unique identifier, which will be available to us through ActionScript.

Next we want to press **R** or select the rectangle tool from the tools palette. We're going to draw ourselves a rectangle.

Don't worry about the size and placement; we can adjust that from our Properties Inspector. We're going to adjust the width and height of our



Library All our movie assets are available direct from our library. You can access this at any point by going to **Window > Library** or **Command** (Mac)/**Ctrl**(PC) + **L**.

rectangle to 250x50, and we want to ensure its X and Y position is sitting comfortably on 0.

The rectangle will form the basic shape and outline for our tooltip. All we need now is a textfield to display the information.

Hit **T** on your keyboard or select the text tool from your tools palette. Draw yourself a single line text area slightly less wide than our rectangle.

If it's not already, we want to make sure this text area is set to dynamic and give it an instance name of **label_txt**. Let's now build the function to bring this puppy to life.

```
function displayCountryInfo(name: String) : Void {
```

We want to attach the tooltip from our library to a variable called theTip.

```
theTip = attachMovie('tooltip', 'tooltip', 999);
```

Set its initial display to **false** until we successfully assign its position on the stage:

```
theTip._visible = false;
Update the movieclip X and Y position to match our cursor position
 theTip._x = _root._xmouse;
 theTip._y = _root._ymouse;
Make it visible
 theTip._visible = true;
```

Because our mouse will be moving over the countries, the initial X/Y positions we assigned to the movie clip will have changed and will constantly be changing. So we need to be updating these values at every movement of the mouse.

We can achieve this by invoking the event handler onMouseMove.

```
 this.theTip.onMouseMove = function() {
   this._x = _root._xmouse;
   this._y = _root._ymouse;
   updateAfterEvent();
 };
```

Assign the country name to the textfield inside our tooltip:

```
 theTip.label_txt.text = name;
}
```

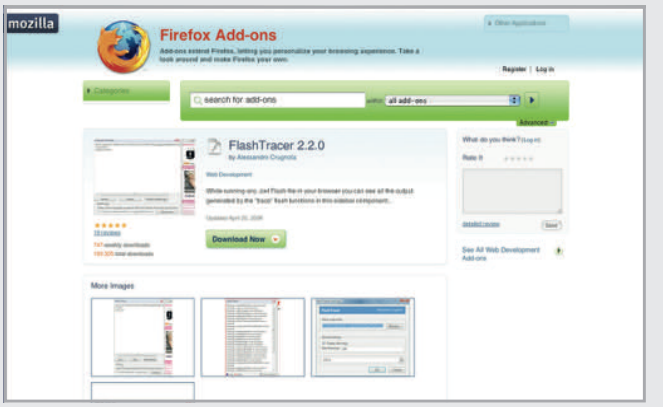
With that complete, we should be ready to rock and roll.

We just need to provide a way of clearing the tooltip when it's not needed. We'll do this by calling another function called **removeTip();** which will hide the tooltip.

Go to **Control>Test Movie** to preview the movie and check our output window for any errors.

Resources Where to find more

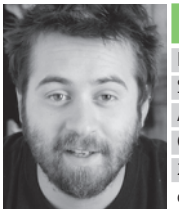
Vector maps
Free vector world maps cannot be found easily as there are only few around. The web resource depot lists a small handful of free maps in an array of styles, which are available for download to use in your projects.
www.webresourcesdepot.com/free-vector-world-maps-collection



Flash debugging
If Firefox is your primary browser, then Flash tracer by Alessandro of sephiroth.it is a handy little add-on. It relies on you having the debug player installed on your system. Flash tracer will output your Flash trace methods inside your browser window.
https://addons.mozilla.org/en-US/firefox/addon/3469

Where to go from here

We've only just touched the surface on this tutorial; the things you can do to expand on this are endless. We can move away from the idea of using the map as aesthetic navigational tool/toy and look at using it as a way to demonstrate website statistics. You could track your visitor's IP address and use geo-locating to look up the origin of that address. Using this data you can easily come up with a simple legend for the map, which can colour the maps based on the visitor count.



About the author

Name Martin Dingley
Site www.soapslurp.co.uk
Areas of expertise Front-end, interactive design
Clients Grants, Regus
2009 resolution Lose the overhang by exercising at least once a week!