

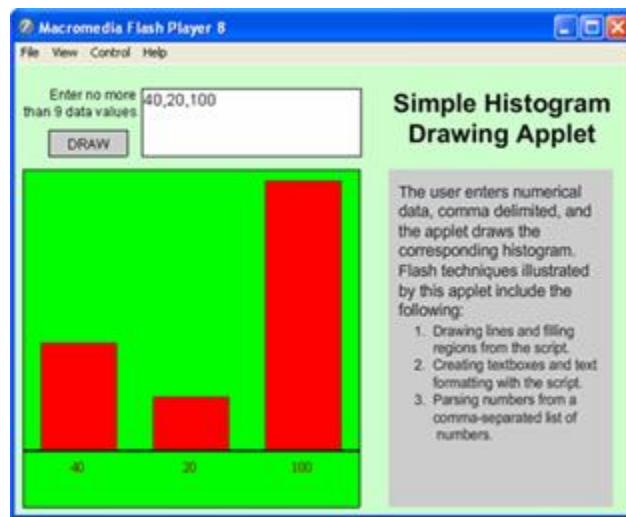
Flash basics for mathematics applets

Tutorial 6. Creating a histogram with dynamic drawing

by Doug Ensley, Shippensburg University and
Barbara Kaskosz, University of Rhode Island

The design of successful, flexible applets depends largely on the ability to manipulate objects at runtime. We have already seen how to create clips on the stage and change their properties with a script. We will now see *how to create clips* (via simple drawing methods) with the script. This will allow us greater flexibility.

In this tutorial, we will make the application shown below. This applet produces a simple histogram for a list of comma delimited numbers. Not only do we need to be able to draw the filled rectangles with our script, but we must also discuss a way to “parse” out numbers from such a comma delimited list. These are the two new ideas illustrated in this tutorial.



Step 1. Create background and objects on the stage

Start with a new project in the *Flash* authoring environment. You will need the **Tools** panel, the **Properties** panel and the **Stage** to be visible. All other panels can be closed if you wish. Just as you did in the previous tutorials, click on an empty section of the stage, and in the **Properties** panel choose a nice background color to contrast appropriately with your text. As before, save your file¹ in the tutorials folder you have created.

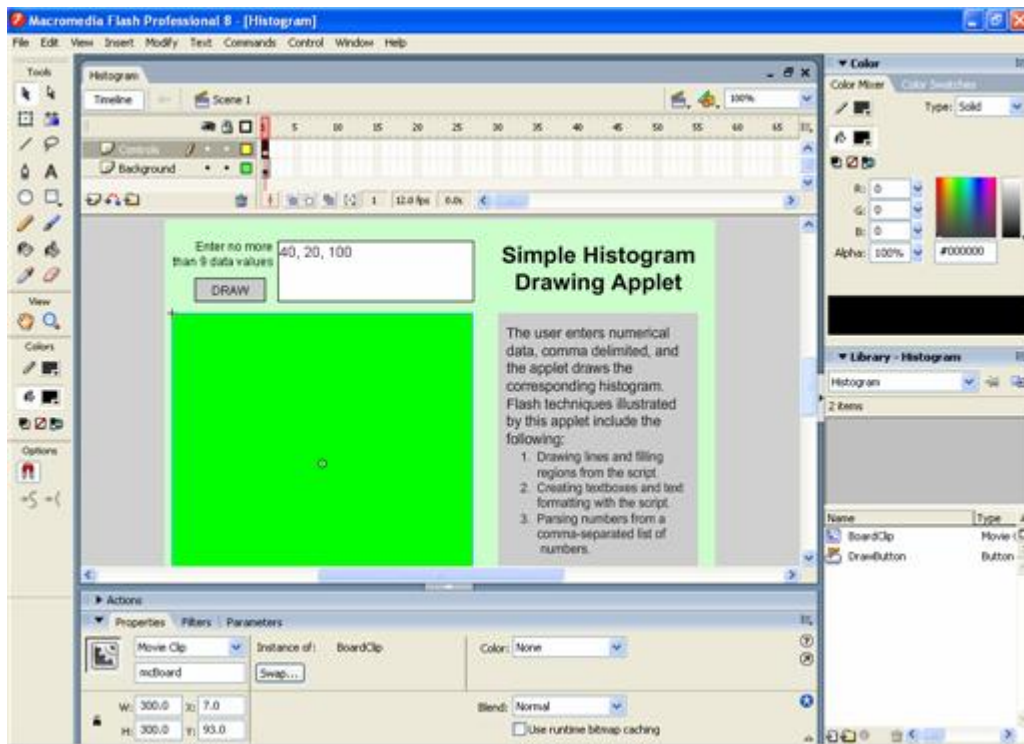
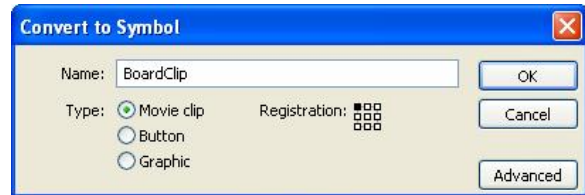
Start by creating the static textboxes to hold the title, the instructions, and the prompt “Enter no more than 9 data values” all positioned roughly as shown in the example above. If you are unsure how to create static text boxes, see Tutorial 2, “Reading input text boxes and writing to dynamic text boxes.” In the example above, we have drawn a gray rectangle to highlight the instructions text. In the **Timeline** panel, change the name of Layer 1 to “Background,” and insert a new layer to be named “Controls.”

On the “Controls” layer we will create three objects: An input textbox called **txtDataList**, a button called **DrawButton** with instance name **btnDraw**, and a movie clip called **BoardClip** with instance name **mcBoard**. The input textbox and button should be created just as we have done before (See Tutorial 2, for example.) with the proviso that the input textbox must be large enough to hold nine numbers in a

¹ It is a good idea to create a separate folder for all of these tutorial exercises so they are easy to find later. Giving your project a meaningful name like Histogram, will also be helpful. At the end of Tutorial 1, we talked about the files created by Flash and how this is relevant to posting your movies on a website.

comma-delimited list. When the textbox is selected, there is a “Multiline” option in the **Properties** panel that allows the input string to wrap around in the textbox. For the movie clip, we will be more specific: Choose the rectangle tool from the **Tools** panel, set the border and fill colors to something nice and drag a rectangle on the stage; use the **Properties** panel to make the rectangle measure 300 by 300, and position the box on the stage.

With the square selected, choose **Modify > Convert to Symbol** to turn this graphic into the movie clip **BoardClip** with registration point in the upper left hand corner as shown on the right. Finally, give this clip the instance name **mcBoard** in the **Properties** panel. At this point, your project should look like the screen shot below.



Step 2. Add the script

To create the scripts for this applet, first click on the layer called **Controls** in the **Timeline** panel, and choose **Insert > Timeline > Layer**. Double click the new layer's label to give the new layer the name **Scripts**. Click Frame 1 of the **Scripts** layer, and open the **Actions** panel in preparation of writing some code. The full script for this movie appears below (in **Courier** font) with advice included for each section in a **dark red Times Roman** typeface. As always, there is no reason to type the comments that appear below. And once again, if you do not wish to type *any* of this, you can open the file **FullScript.txt**, and copy and paste the entire script at this time.

Script Block 1. Set parameters to match properties of stage objects

The first block of code sets parameters for the applet so they can easily be found and modified if the instructor would like to change the layout of the graphing board or the maximum number of data values. These tie in directly with objects on the stage, so a change on the stage may require a corresponding change to these parameters.

```

var bSize:Number = 300;           // The size in pixels of the square graphing board.
var axisHeight:Number = 50;      // The horizontal axis will be 50 pixels from the
                                // bottom of the board.
                                // The value of axisHeight should always be less than
                                // the value of bSize.
var highest:Number = 240;        // The highest bar will be 240 pixels high.
                                // This should always be less than
                                // bSize - axisHeight.

var maxDataNumber:Number = 9;    // The maximum number of points you will allow.
                                // Requires consideration of bSize and font sizes.

```

Script Block 2. Global variables and settings

This block of code completes the set up of the variables and structures to be used. Of particular interest is the “restrict” property for the input textbox, a wonderful way to cut down on checking user input errors – this prevents the keyboard from typing any characters other than the digits, the comma, and the space into the input box.

```

// To avoid a lot of error-checking of input, allow the user to only type integers,
// commas and blank spaces into txtDataList.

txtDataList.restrict = "0123456789, ";

// Declaration of other variables that need global scope.

var dataList:Array = new Array();
var maxHeight:Number;
var numBoxes:Number;

```

We create a TextFormat object that will be applied later to the dynamically created textboxes that are used to display a data value underneath a histogram bar. When the program creates those textboxes, we will set the SetTextFormat property of the new textbox to be this fmtLabel object, and the textboxes will then be have centered, 12 point, dark red font.

```

var fmtLabel:TextFormat = new TextFormat();
with(fmtLabel) {
    align = "center";
    size = 12;
    color = 0x990000;
}

```

Script Block 3. Clearing and drawing the histogram boxes

These two functions manage the creation and destruction of the dynamically created objects within the mcBoard clip. Note in the clearBoxes function that there are two different ways to refer to a clip within a clip. A reference to mcBoard.axis is exactly the same as a reference to mcBoard[“axis”]. The latter “array style” reference is extremely useful when we want to reference clips within a loop, as is seen in both functions. For example, mcBoard[“box”+String(i)] refers to mcBoard[“box0”] when i=0, which is the same thing as mcBoard.box0 but is possible to access within a loop using string concatenation.

```

// This function clears all dynamically created movie clips and text fields.

var clearBoxes = function():Void {
    mcBoard.axis.clear();
    for (var i=0; i<numBoxes;i++) {
        mcBoard["box"+ String(i)].clear();
        mcBoard["txtData"+ String(i)].removeTextField();
    }
}

```

```

    }

    // Reset the number of boxes and the dataList array.
    numBoxes = 0;
    dataList = new Array();
}

var drawBox = function(which:Number):Void {
    // Determine the dimensions of the current box to be drawn.
    var boxWidth:Number = bSize/(1.5*numBoxes);
    var boxHeight:Number = dataList[which]/maxHeight*highest;

    // The coordinates (x1,y1) and (x2,y2) are diagonally opposite corners
    // of the current box to be drawn.
    var x1:Number = boxWidth/4 + 3/2*boxWidth*which;
    var y1:Number = bSize - axisHeight;
    var x2:Number = x1 + boxWidth;
    var y2:Number = y1 - boxHeight;
}

```

The first line below creates an empty clip called “box0”, “box1”, “box2”, etc. at depth 1, 2, 3, etc., respectively. The first “+” here is “string concatenation” and the second “+” is “integer addition.” The second line simply says to use a dark red pen color with width 1.

```

mcBoard.createEmptyMovieClip("box"+String(which), which+1);
mcBoard["box"+ String(which)].lineStyle(1, 0x990000);

```

The block below creates a filled rectangle (fill color is pure red 0xFF0000) with lower left coordinates (x1,y1) and upper right coordinates (x2,y2). The moveTo and.lineTo methods can be visualize by thinking of a pen being moved or dragged, respectively, with each call.

```

mcBoard["box"+ String(which)].beginFill(0xFF0000);
    mcBoard["box"+ String(which)].moveTo(x1,y1);
    mcBoard["box"+ String(which)].lineTo(x1,y2);
    mcBoard["box"+ String(which)].lineTo(x2,y2);
    mcBoard["box"+ String(which)].lineTo(x2,y1);
mcBoard["box"+ String(which)].endFill();

// The following block creates dynamic textboxes and puts the
// correct data entry into the box. We place these boxes at depths
// 50, 51, 52, etc. and make them have the same width as the histogram bars.
// We set the format of these labels to be the format object fmtLabel
// that we created earlier.

```

```

mcBoard.createTextField("txtData"+ String(which), 50+which, x1, y1 + 5, boxWidth, 20);
mcBoard["txtData"+ String(which)].type = "dynamic";
mcBoard["txtData"+ String(which)].text = String(dataList[which]);
mcBoard["txtData"+ String(which)].setTextFormat(fmtLabel);
}

```

The following method is executed when the mouse is released over the **btnDraw** button. It is here that we parse the input textbox contents into substring based on the comma delimiters, place those substrings representing numbers into the array `dataList`, ignore data values beyond the maximum number that are allowed to be input, and call the `drawBox` function for each value to produce the histogram.

```

btnDraw.onRelease = function():Void {
    clearBoxes(); // Clear all previous drawings
    var strList:String = txtDataList.text; // Get user input list as a string.
    var strData:Array = strList.split(","); // substrings. At the commas,
}

```

```

//      split string into substrings,
//      and place substrings into array.

// The next block puts valid (numbers) from the input list into another array
// and disregards bad input. We also ignore any inputs beyond the maximum
// number allowed.

var countValidValues:Number = 0;
for (var i=0; i<strData.length; i++) {
    if ((!isNaN(parseInt(strData[i]))) && (countValidValues < maxDataNumber)) {
        dataList.push(parseInt(strData[i]));
        countValidValues++;
    }
}
txtDataList.text = String(dataList); // Put the data that is really being
// used back into the user input box
// This assures that the input box
// data matches the histogram.

numBoxes = dataList.length; // This is the number of boxes to be drawn.

// The next block finds the maximum user input value as the variable name maxHeight.

maxHeight = 0;
for (var i=0; i<numBoxes;i++) {
    if (dataList[i] > maxHeight) {
        maxHeight = dataList[i];
    }
}

// The next block puts draws the axis at depth 100.

mcBoard.createEmptyMovieClip("axis", 0);
mcBoard["axis"].lineStyle(2, 0x000000);
mcBoard["axis"].moveTo(1,bSize - axisHeight);
mcBoard["axis"].lineTo(bSize,bSize - axisHeight);

// The next block calls the function that draws the boxes.

for (var i=0; i<numBoxes;i++) {
    drawBox(i);
}
}

Selection.setFocus(txtDataList); // Place focus on the user input textbox.

```

Save and test your movie, and you will see a full working version of the applet we described at the beginning of this tutorial.

Additional resources

To see the source code for the applet we made for this tutorial, open the file **tutorial6 fla** in Flash. Another idea for accomplishing the functionality of this applet is to create a generic “filled box” clip, and resize and attach instances of it to the graphing board at run-time. Since this clip would need to exist in the library and not the stage, there will be a subtle step of linking it to the library. To learn more about this idea, see the tutorial “The Line Through Two Points” in the *Creating Applets from Scratch* section of the Flash Forum Learning Center.