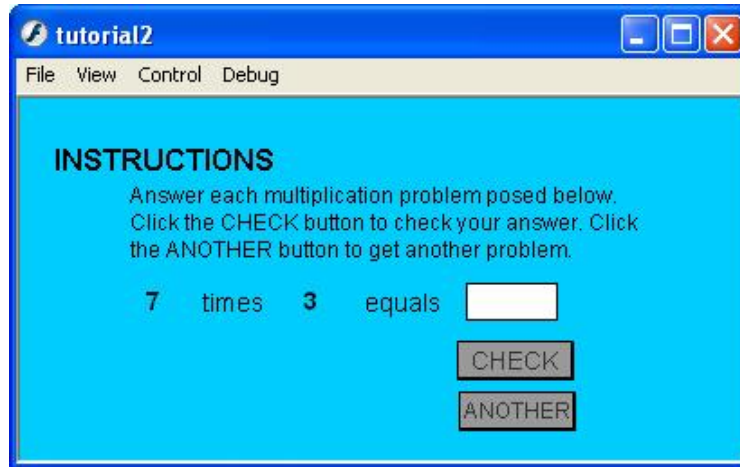


# Flash basics for mathematics applets

## Tutorial 2. Reading input text boxes and writing to dynamic text boxes

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

In this tutorial, we will make the Flash movie shown below on the left. This applet poses randomly generated arithmetic questions and responds appropriately to the user's answers. Flash elements used in this movie include dynamic textboxes, input textboxes and static textboxes as well as buttons. Actionscript elements used in this movie include random number generation, parsing integer input, and setting focus on an input box.



### Step 1. Add all textboxes to 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. Click on an empty section of the stage, and in the **Properties** panel set the size of the movie to be 400 by 200 pixels and choose a nice background color to appropriately contrast with your text. Even though there is little in your project yet, save your file<sup>1</sup> in the tutorials folder you have created.

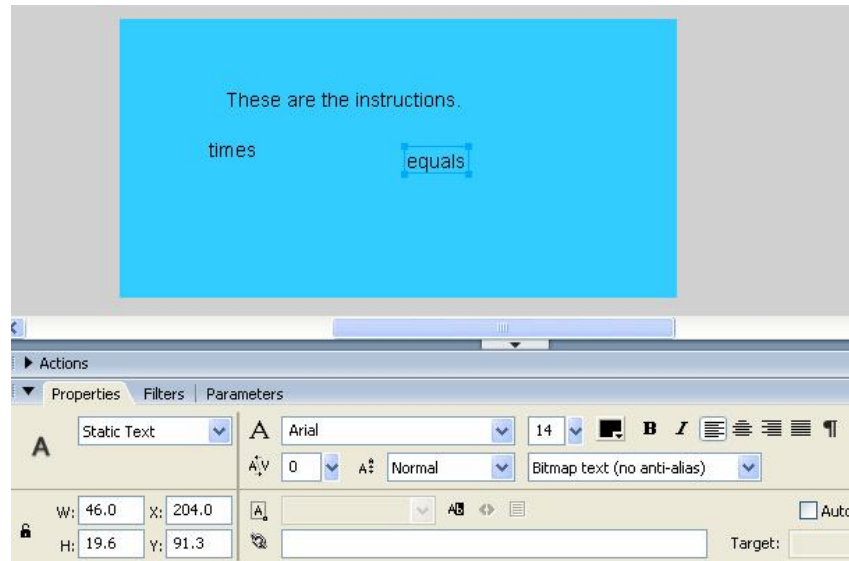


We will now add three different types of text to the stage. The first type (static text) is used for the parts of the stage that will never change. This includes the set of instructions and words “times” and “equals” in the example. To create this type of object, choose the Text tool from the **Tools** panel and drag a region on the screen where the instructions will go. (This can always be moved later, so don't dwell on the placement at this point.) The **Properties** panel should display properties of this textbox that can be modified at this time. The most important thing in the Properties panel is to choose Static Text as the type of textbox. (The other two choices are Dynamic Text and Input Text. We will use these shortly.) Something like 14 point, black, and Arial is fine for the font. Also, it is best to select “Bitmap text (no

<sup>1</sup> 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 Arithmetic or Tutorial2 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.

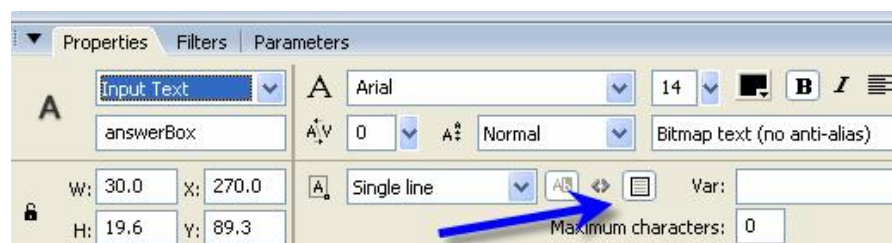
anti-aliasing) when working with these relatively small font sizes. Type a simple phrase like “The instructions go here.” in the textbox for now. We can worry about this later.

Create additional static textboxes, one containing just the word “times” and another containing just the word “equals.” Do not worry about placing these boxes yet – we will wait until we have all of the relevant text elements on stage before we line things up – but be sure that the size of the box (controlled by dragging the corner of the box after you type in it) is as small as possible. This will help when it is time to align the text.



We will now add an Input Text box to our application. We use this type of textbox to get keyboard input from the user. In this application, we need this for the user to type the answer to an arithmetic question. Once again choose the Text tool from the **Tools** panel, and drag a rectangle on the right side of the stage. When the **Properties** panel shows text box properties, this time choose Input Text instead of Static Text. In the **Properties** panel, you will see a new field appear into which you must type a name to the Input Text box so that it can be properly referenced within our *Actionscript* code later. For consistency, let’s agree to give our Input Text box the name “answerBox.” Set the font in this textbox to be bold, and type the number 999 into the box to make sure that you have made it an appropriate size for a user to enter a three-digit answer. (You can leave the 999 in the box on the stage. We will programmatically clear the **answerBox** with every new question anyway.)

Finally, we will include a border around the input box to be displayed so the user will easily see where to enter the answer. Click on the icon indicated at the right by the arrow to show the border around the box.



The last type of text box, Dynamic Text, will be used for output to our user. More specifically, any text that is programmatically generated will be displayed to the user using a Dynamic Text box. This will include the two numbers that comprise the arithmetic question as well as the messages that the user sees when the answer is checked. To create the first Dynamic Text box, choose the Text tool from the **Properties** panel and drag a small rectangle large enough for a two digit number. Once again, you can actually type the number 99 into the box to be sure it is no bigger than necessary. Select Dynamic Text from the pull down menu in the Properties panel, and give this box the name **mult1Box**. Turn off the “show border” feature (by clicking on the icon identified by the arrow above) if it is still on from when you created the Input Box. After you create another Dynamic Text box called **mult2Box**, we will be ready to line things up.

Use the Selection tool (the black arrow on the **Tools** panel) to select and move the textboxes (all except the “Instructions” text) into alignment so that they look like the screenshot below. There is an Align option in the Modify menu that may help with this process.



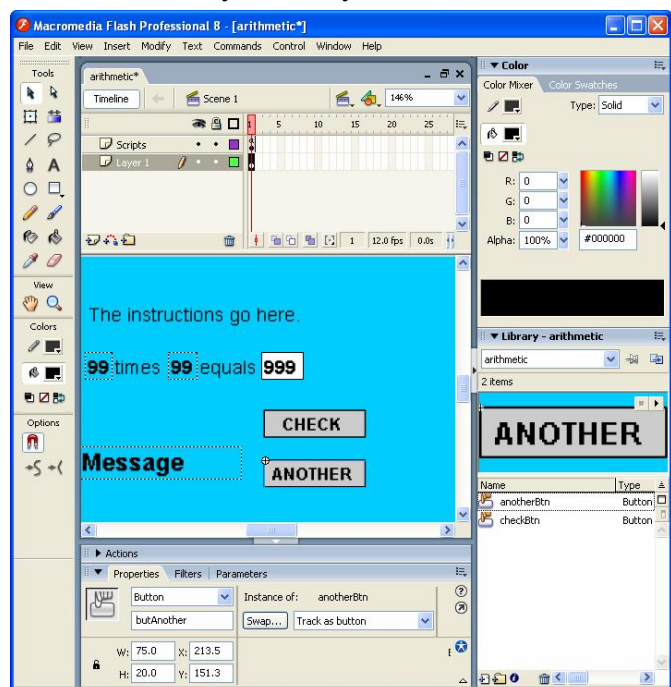
Finally, create a Dynamic Textbox with instance name **messageBox** to be used to display messages to the user when she checks her answer. In our example, this box appears at the bottom of the screen with 18 pt font and initially contains the text “MESSAGE” to assure that the size is appropriate.

## Step 2. Create the buttons

We will make two buttons using the process described in Tutorial 1, “Making a button.” If you do not know how to make a button, you should complete that tutorial first. The description below is very brief. The relevant new idea here is the duplication of one button to get a second one that matches in appearance.

First create a rectangle with black border and filled with light gray. Size the rectangle in the **Properties** panel so that it is 75 pixels wide and 20 pixels high. Select the rectangle and convert it into a button symbol called “checkBtn.” Double click on this button, and set the over, down and hit frames as you did in Tutorial 1. Add a second layer to the **checkBtn** timeline so that you can lay the static text “CHECK” on top of the rectangle. Return to the main timeline.

From the Window menu, select Library and you will see your button is the sole occupant of the library. Right click on the string **checkBtn** in the **Library** panel, and select Duplicate. You will now be prompted to name your duplicated button – you should name it “anotherBtn” and notice that it now shows up as a second item in the **Library** panel. Drag an instance of it onto the stage, and double click on it to open its timeline. Select that layer with text, and change the text to read “ANOTHER” instead of “CHECK.” Return to the main timeline and select each of your two buttons in turn in order to give them the instance names (in the **Properties** panel) “butCheck” and “butAnother”, respectively. Your screen should look similar to the one shown at the right:



### Step 3. Add the scripts

In the main timeline, add a second layer<sup>2</sup> and name it “Scripts.” (If you are bothered by lack of symmetry, name the existing layer something descriptive like “On Stage.”) Select frame 1 of the Scripts layer, and open the **Actions** panel so that we can write some code. If you do not wish to type this code, you can copy the contents of the file **FullScript.txt** and paste it into the **Actions** panel for this frame.

We will first write the function that generates a problem to be asked. To do this, we need to choose two random numbers and put them into the Dynamic Text boxes **mult1Box** and **mult2Box**. We do this by setting the text property of these boxes to be the result of converting the random numbers to strings. Here is the code that will accomplish this task. (The comments do not have to be typed of course.)

```
var setUpProblem:Function = function():Void {
    var r:Number;

    // Let r be a randomly chosen real number between 0 and 1.
    r = Math.random();

    // Ceiling of 12*r produces an integer from {1,2,...,12}. Set the text
    // property of mult1Box equal to this number converted to a string.
    mult1Box.text = String(Math.ceil(12*r));

    // Repeat the previous two steps to fill mult2Box
    r = Math.random();
    mult2Box.text = String(Math.ceil(12*r));

    // Clear the answerBox and the messageBox and place focus on answerBox
    answerBox.text = "";
    messageBox.text = "";
    Selection.setFocus("answerBox");
}
```

We make special note of the `setFocus` method of the `Selection` object since this is bit less intuitive than the rest of the script. Using this method, we can give any object on the stage (buttons, clips, etc.) the focus. For a textbox, this means either selecting the text in the box or putting the blinking cursor in an empty box. This is a good way to help your user navigate your application, especially if there is a lot on the stage.

Next we add the `onRelease` methods for our two buttons as follows:

```
// When the "ANOTHER" button is pressed, call the setUpProblem() function.
butAnother.onRelease = function():Void {
    setUpProblem();
}

// When the "CHECK" button is pressed, check the user's answer.
butCheck.onRelease = function():Void {

    // Let x be the integer in mult1Box and y be the integer in mult2Box.
    var x:Number = parseInt(mult1Box.text);
    var y:Number = parseInt(mult2Box.text);

    // Let z be the integer that the user typed in answerBox.
    var z:Number = parseInt(answerBox.text);

    if ((x*y) == z) { // If z is the product of x and y,
```

---

<sup>2</sup> The exact process for adding a new layer is covered in Tutorial 1, “Making a button.”

```

        messageBox.text = "Good job!!";           // pat user on the head.
    }
    else {
        messageBox.text = "Try again.";           // Otherwise invite another try,
        Selection.setFocus("answerBox");         // and select bad answer to make
    }                                           // it easier to replace.
}

```

The `onRelease` method for the ANOTHER button has nothing to it since we did all of the work in the previously defined function. On the other hand, there is one thing in this `onRelease` method for the CHECK button that is worth noting.

The `parseInt` function converts a string (which is what the contents of the text property of a textbox is) to an integer. If the contents of the textbox is a floating point (i. e., having a decimal point) number, then `parseInt` truncates the decimal part leaving just the integer part behind. If the contents of the textbox do not constitute a well-formed number (e.g., if the user types the word “ten”), then `parseInt` returns “NaN” which stands for “Not a Number.” Handling this kind of input error is important when writing applications for student use. In this case there is a simple solution: On the very first line of the entire script, write the line `answerBox.restrict = "0123456789"`. This restricts the user input into the **answerBox** to strings of these ten characters. Hence, it is impossible for the user to enter anything other than a positive integer value. Note that if the user types nothing at all into **answerBox** and hits the CHECK button, then technically the value of `z` is NaN which is not equal to  $x*y$  (We are guaranteed each of these is an integer since our program put them there!) so the program responds that this is wrong and invites another try.

If you save and test this movie, you will find that it works perfectly except there is no problem given when the program first starts up. This is because our program only produces a new problem when the “ANOTHER” button is pressed. To alleviate this, we simply end our script with a simple call to the `setUpProblem()` function so that this function is called when the movie is loaded. Add this to the end of your script.

```

// Call the setUpProblem function to get the first problem set up
// without needing a button click.
setUpProblem();

```

Now save and test the movie again, and you will see the final movie functioning as we described.

### **Additional resources**

To see the source code for the applet we made for this tutorial, open the file **tutorial2 fla** in Flash.