

Enhanced Lesson 8 (v.2)

Adobe Flash CS4 Professional Classroom in a Book

Flash CS4 components have both benefits and drawbacks. We will learn to use and modify a few components in Lesson 8.¹

Then we will learn, in addition, how to use those components in conjunction with an external XML file (that part is not in the book).

From the book (Lesson 8):

Make sure you understand how to complete these tasks. You do not need to memorize the steps, but ensure that you understand *what you are doing* at each step. If you're perplexed, go ahead for a few more steps and then review what you have just done. Make notes in the margins of your book to help you repeat the steps in the future.

1. Find components in the Components panel.
2. Use the TextArea component.
 - a. Change X and Y, width and height, of the component.
 - b. Use the Component Inspector to add text to the component.
 - c. *Question:* Why would we want to use this component instead of simply adding text to the Stage, using the regular Text tool?
3. Use the TileList component.
 - a. Change X and Y, width and height, of the component.
 - b. Give the component an instance name. (*Question:* Why didn't you need to give the previous component an instance name? See p. 301.)
 - c. Use the Component Inspector to change columnWidth and rowHeight, among other things.
4. Add another, different TextArea component.
 - a. Change X and Y, width and height, of the component.
 - b. Use Properties panel to change Alpha for the component.
 - c. Give the component an instance name.
5. Write ActionScript to control both the TileList component and the second TextArea component (using the *addItem()* method).
6. On p. 305, step No. 13, you are asked to copy all the rock descriptions into the Actions panel. *You can skip this tedious exercise.* (Think about how horrible this would be if you had 100 rocks.) We will now learn a better way!

¹ You can learn more about Flash CS4 components here:

http://www.adobe.com/devnet/flash/learning_guide/components/part01.html

What we will do is *structure the data* that was provided on the CD for the book. By adding structure to the data *outside of Flash*, our ActionScript work within Flash will be much easier and greatly simplified.

Rock data (XML) conversion sequence:

We begin with a normal MS Word file with different kinds of info *on separate lines*.

1. Open the MS Word file that includes all the information about all the rocks (*rocks_MS_Word_file.docx*).²
2. Remove the blank lines.
3. Select all the text in the MS Word file.
4. Convert the text to a table. (If you don't know how, look in Word Help. It is clearly explained there.) When prompted, change the NUMBER OF COLUMNS to "3." Why? because you have *three types of information* about each rock. (In database terms, we would say we have three *fields* in each *record*.) The default for "Separate text at" should be *paragraphs*; that is correct. Click OK.
5. Select the entire table in Word. Copy it.
6. Open MS Excel. Click on the top left cell in the grid to select it. Paste.
7. File menu > Save As. Change the FORMAT to "Comma Separated Values (.csv)" (there is no need to add any commas). If you are asked about saving only the current sheet, that is okay.
8. Exit from Excel and from Word.
9. Open the CSV file in a plain-text editor, such as Notepad. (DO NOT skip this step!)
10. Select all the text there. Copy it.
11. Open the Web page at:
<http://www.creativyst.com/cgi-bin/Prod/15/eg/csv2xml.pl>
12. Paste into the box labeled "Input (CSV file)."
13. Type this text *exactly as shown* into the box labeled "Element ID's (cols)":
rockName,image,rockDescrip
14. Type this text *exactly as shown* into the box labeled "Doc ID": rockData
15. Type this text *exactly as shown* into the box labeled "Row ID": rockItem
16. Click the Convert button.
17. Click the Select button. Then copy (Ctrl-C or Command-C).
18. Paste the XML text into Dreamweaver, or into a plain text editor.³
19. Check *at the end of the document* to see whether any unnecessary lines were added. If so, delete them.
20. Save the document with filename *rockData.xml* (note that the file extension must be *.xml*). **This is your XML file.**

² Word is used (instead of a plain-text editor) because it's easy to make proper typographic quotation marks, em-dashes, etc., in Word.

³ We will discuss what to do about the "\n\n" characters in class. They must be converted into linespaces. You can search-and-replace or convert them by hand. This is easy in Dreamweaver.

Check to see whether the *first* lines look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<rockData>
  <rockItem>
    <rockName>OBSIDIAN</rockName >
    <image>thumbnails/Obsidian.jpg</image >
    <rockDescrip>OBSIDIAN\n\nObsidian is a naturally occurring
```

If not, *edit the file* to look like the lines shown above.

The *final* lines in your XML document should look like this:

```
source of the semi-precious gemstone called garnet.</rockDescrip >
  </rockItem>
</rockData>
```

If you see unnecessary lines there (*above* the closing tag </rockData>), delete them.

Do-over for Lesson 8:

Now we will put the new XML file to use—by changing the ActionScript.

1. Save your FLA *with a new filename* (File menu > Save As) so that you can change the ActionScript and still preserve your earlier work (from the lesson in the book).
2. In the Actions panel, delete everything after line 8.

We will keep *these lines* (below) exactly the same as they were in the original exercise:

```
thumbnails_t1.addEventListener(Event.CHANGE, thumbnailClicked);

function thumbnailClicked(event:Event):void {
    mineral_ta.text = event.target.selectedItem.data;
}
```

However, we will change the one-by-one manner of adding the information about each mineral. We will extract *all of that information* from the external XML file.

3. Copy all of the text from the file *as3_to_add.txt* and paste it *at the top* of the ActionScript in frame 1 in your FLA.

4. These lines are handling the XML data:

```
var myXML:XML;

// first we load the external XML file
var xmlLoader:URLLoader = new URLLoader();
xmlLoader.load(new URLRequest("filename.xml"));
xmlLoader.addEventListener(Event.COMPLETE, handleMyData);

// then we stick it into our XML object and do stuff with it
function handleMyData(myEvent:Event):void {
    myXML = new XML(xmlLoader.data);
    popList();
}
```

Every bit of this script *except for two small things* can be used *any time* you want to read *any* XML data file into Flash. That is, you can just copy and paste this script *as is*, and not worry about how it does what it is doing (except for the two things; see below).

5. The first thing you need to *change* is: `"filename.xml"`

Your XML file in this case is named `rockData.xml`—so you need to replace `filename` with `rockData` (be careful to keep the quotation marks).

6. The second thing you would change (in another FLA, in another project) is: `popList()`;

However, you are NOT going to change it in THIS exercise, because “popList” is *the name of the function* we will use here.

We need to bridge the gap between reading the XML data (already done) and using the `addItem()` method (shown in the book), which makes the TileList and the TextArea components work hand-in-hand.

That is exactly what the `popList()` function does here. This function contains a “for loop,” which is explained in more detail below. (If you feel like your head is going to explode, just skip that part.) All you really need to know right now is that the `popList()` function *does all the work for you*—and this would be *really fabulous* if instead of 15 rock items you had, say, election returns data for all 67 counties in the state of Florida.

You need to understand that the line `popList()` *calls the function* that starts with this line:

```
function popList():void {
```

That function could have a different name. That function could be performing different tasks—in another Flash movie that had different objectives. In other words, a function is *a set of instructions* that you write, telling the Flash movie what to do when that function is *called* (set into action). For example, in button scripting, you have already learned how to write a function that sends the playhead to a frame with a particular label.

7. Now you can delete these lines:

```
thumbnails_t1.addItem({label:"OBSIDIAN",  
source:"thumbnails/Obsidian.jpg", data:"OBSIDIAN\n\nObsidian is a  
naturally occurring glass, produced by volcanoes. ... as well as  
modern day surgical scalpel blades."});
```

The only reason we kept those lines was so that you could compare them with the tasks that the `popList()` function is now performing. Can you see how the function is using *label*, *source*, and *data*?

8. Save and test the movie. You should see it working *exactly like* the “final” CD file for Lesson 8 in the book.

Please note that the file *rockData.xml* and the folder named *Thumbnails*⁴ MUST be in the same folder with your SWF, or else it will not work properly!

Details about the “for loop”:

We use a “for loop” to get the data for each mineral and insert it into the FLA, using this same format (designed for use with the TileList component):

```
thumbnails_t1.addItem({label:"OBSIDIAN",  
source:"thumbnails/Obsidian.jpg", data:"OBSIDIAN\n\nObsidian is a  
naturally occurring glass, produced by volcanoes. ... as well as  
modern day surgical scalpel blades."});
```

What is a “for loop,” and how does it work? There’s a great 5-minute video about that, here:

<http://tv.adobe.com/watch/actionsript-11-with-doug-winnie/looping-in-actionscript-episode-30/>

⁴ Please note that most Web servers are case-sensitive; therefore, if the folder name “Thumbnails” has a capital *T*, then every occurrence of the folder name in the XML file must also have a capital *T*. If the XML file has a lowercase *t* for each occurrence, then change the folder name to match.

In this case, our “for loop” *reads each item* in our XML file. It will automatically stop when it reaches the end of the file. We *do not need to know* in advance how many items are listed—the loop will figure it out.

This makes a “for loop” *great* for reading in a giant XML file with dozens, even hundreds, of items!

Our loop in this case uses the same `addItem()` method that you saw in the book. But the loop uses it *over and over again*, automatically, until all 15 items are processed.

```
function popList():void {  
    for (var i:int=0; i<myXML.children().length(); i++) {  
        myLabel = myXML.rockItem[i].rockName;  
        mySource = myXML.rockItem[i].image;  
        myData = myXML.rockItem[i].rockDescrip;  
        thumbnails_t1.addItem({label:myLabel, source:mySource,  
data:myData});  
    }  
}
```

The loop uses three variables, which were set up (initialized) near the top of our ActionScript in frame 1:

```
var myLabel:String;  
var mySource:String;  
var myData:String;
```

What the loop does, in simple terms: It begins with the first set of data in our XML file. That is the set contained within the first *pair* of `<rockItem>` tags. That first set is known to ActionScript as `rockItem[0]`, and it contains three pieces of information, which you can see in the XML file.

- The loop grabs the information between the `<rockName>` tags and sticks it into the `myLabel` variable.
- The loop grabs the information between the `<image>` tags and sticks it into the `mySource` variable.
- The loop grabs the information between the `<rockDescrip>` tags and sticks it into the `myData` variable.

Then the loop runs the `addItem()` method that you saw in the book, attaching all that information to a tile inside the TileList component on the Stage.

On each turn, the loop adds 1 to the value of “i” (so that the next time around, it’s `rockItem[1]`, and then `rockItem[2]`, etc.) and does each step all over again. It repeats until it reaches the end of all the XML data.

The loop knows when the XML dataset ends because `myXML.children().length()` tells the loop *how many items* are in the dataset. This bit tells the loop to *quit* as soon as "i" is no longer *less than* (<) that integer:

```
i < myXML.children().length();
```

If you want to understand more about variables in AS3, this is a good, clear tutorial (only steps 1 through 8; you can skip the rest):

<http://active.tutsplus.com/tutorials/actionscript/as3-101-variables/>

If you would like to understand how to write a simple function in AS3 (6 minutes):

<http://tv.adobe.com/watch/actionscript-11-with-doug-winnie/fundamentals-of-functions-episode-8/>

Changing the appearance of the text in the TextArea component:

This part is optional. You might be wondering how you could change the font attributes inside the TextArea component that is used in this exercise.

We need to write ActionScript if we want to change the text properties of the TextArea component. (*This is why I don't like to use components!*)

First, we must create a new instance of the TextFormat class, providing the font-family and font-size we want to see:

```
var myFormat:TextFormat = new TextFormat("Tahoma", 12);
```

Then we need to write another line to apply the format to a specified instance (in this case, *mineral_ta*) of the TextArea component:

```
mineral_ta.setStyle("textFormat", myFormat);
```

Scrollbar missing? Change Alpha to 60% (or 100%), and it will appear. (In the exercise in the book, you scaled down the Alpha of the TextArea component.)