Using Arrays in ActionScript - Part 1

By: Paul Newman

Arrays are a staple of ActionScript, and for that matter, ECMAScript, upon which ActionScript is based. In this tutorial, we'll examine how to create different types of arrays, and how to loop through them using ActionScript.

NOTE: According to the Flash documentation, the current version of ActionScript is based on the ECMA-262 Edition 4 proposal.

Chances are you're already familiar with using variables in ActionScript to store data. If a variable holds a reference to a single piece of data, then we can think of an array as a variable that stores a grouping of related data. One analogy is that an array is like an audio CD. An audio CD contains a bunch of related data — its songs — and each song has a track number. As an example, we'll use the Beatles album *Help!*, which contains 14 tracks:

1. Help!
2. The Night Before
3. You've Got to Hide Your Love Away
4. I Need You
5. Another Girl
6. You're Going to Lose That Girl
7. Ticket to Ride
8. Act Naturally
9. It's Only Love
10. You Like Me Too Much
11. Tell Me What You See
12. I've Just Seen A Face
13. Yesterday
14. Dizzy Miss Lizzie

If you want to play "Ticket to Ride" on your stereo, you simply choose Track 7. If you press Shuffle on your stereo, the tracks are resorted in random order, but "Ticket to Ride" is still Track 7:

11. Tell Me What You See
3. You've Got to Hide Your Love Away
10. You Like Me Too Much
4. I Need You
7. Ticket to Ride
12. I've Just Seen A Face
9. It's Only Love
2. The Night Before
5. Another Girl
13. Yesterday
6. You're Going to Lose That Girl
1. Help!
14. Dizzy Miss Lizzie

In this sense, each track number represents an *index* for the audio CD, not unlike primary key columns in a database. Without track numbers, there would be no way for your stereo to identify which song to play. Arrays also have numeric indexes, which are used to identify each *element* of an array. Unlike CD track numbers, however, ActionScript arrays start counting from 0, rather than 1. If we were going to represent *Help!* as an ActionScript array, here is one way we might do it:

```
var help = new Array();
help[0] = "Help!";
help[1] = "The Night Before";
help[2] = "You've Got to Hide Your Love Away";
help[3] = "I Need You";
help[4] = "Another Girl";
help[5] = "You're Going to Lose That Girl";
help[6] = "Ticket to Ride";
help[7] = "Act Naturally";
help[8] = "It's Only Love";
help[9] = "You Like Me Too Much";
help[10] = "Tell Me What You See";
help[11] = "I've Just Seen A Face";
help[12] = "Yesterday";
help[13] = "Dizzy Miss Lizzie";
```

Listing 1 *Defining and populating the **help** array object in ActionScript*

On the first line, we define an array object called `help` using the `new Array()` constructor, and on the subsequent lines, we populate the array with elements using the array access operator (`[]`).

Looping Through the Array

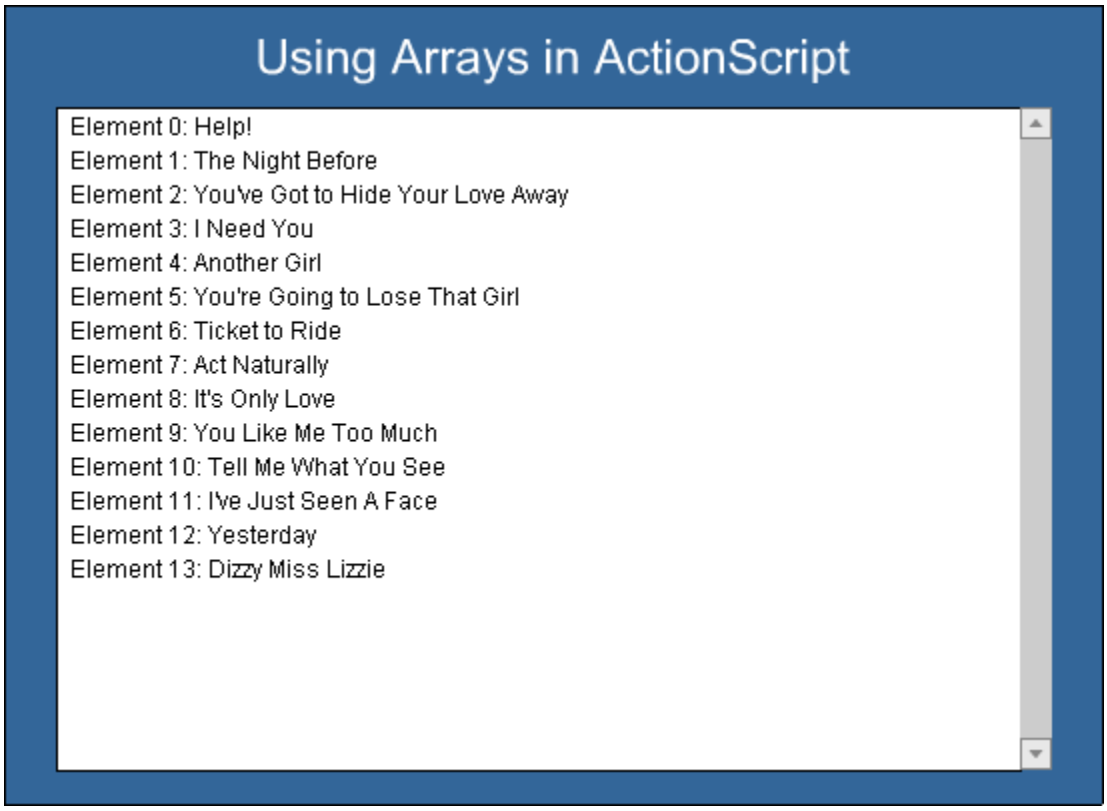
To see the `help` array in action, complete the following steps to display it in Flash:

1. Click **Download Support Files** at the bottom of this page.
2. Extract the zip file to a folder on your computer.
3. Open **arrays.fla** in Flash MX 2004 or Flash MX 2004 Professional.
4. Choose **File > New** and select **ActionScript File** on the General tab.
5. Click **OK** to create a blank AS file.
6. Copy and paste the following code into the AS file and save it as **arrays1.as**:

```
output_txt.htmlText = "";


function output(arr){
   len = arr.length;
   for(var i=0; i<len; i++){
     output_txt.htmlText += "Element " + i + ": " + arr[i] + "<br>";
   }
   output_txt.htmlText += "<br>";
}


var help = new Array();
help[0] = "Help!";
help[1] = "The Night Before";
help[2] = "You've Got to Hide Your Love Away";
help[3] = "I Need You";
help[4] = "Another Girl";
help[5] = "You're Going to Lose That Girl";
help[6] = "Ticket to Ride";
help[7] = "Act Naturally";
help[8] = "It's Only Love";
help[9] = "You Like Me Too Much";
help[10] = "Tell Me What You See";
help[11] = "I've Just Seen A Face";
help[12] = "Yesterday";
help[13] = "Dizzy Miss Lizzie";


output(help);
```

7. Click the **arrays.fla** tab on the Document window and select **Control > Test Movie** (or press **Ctrl+Enter)**.

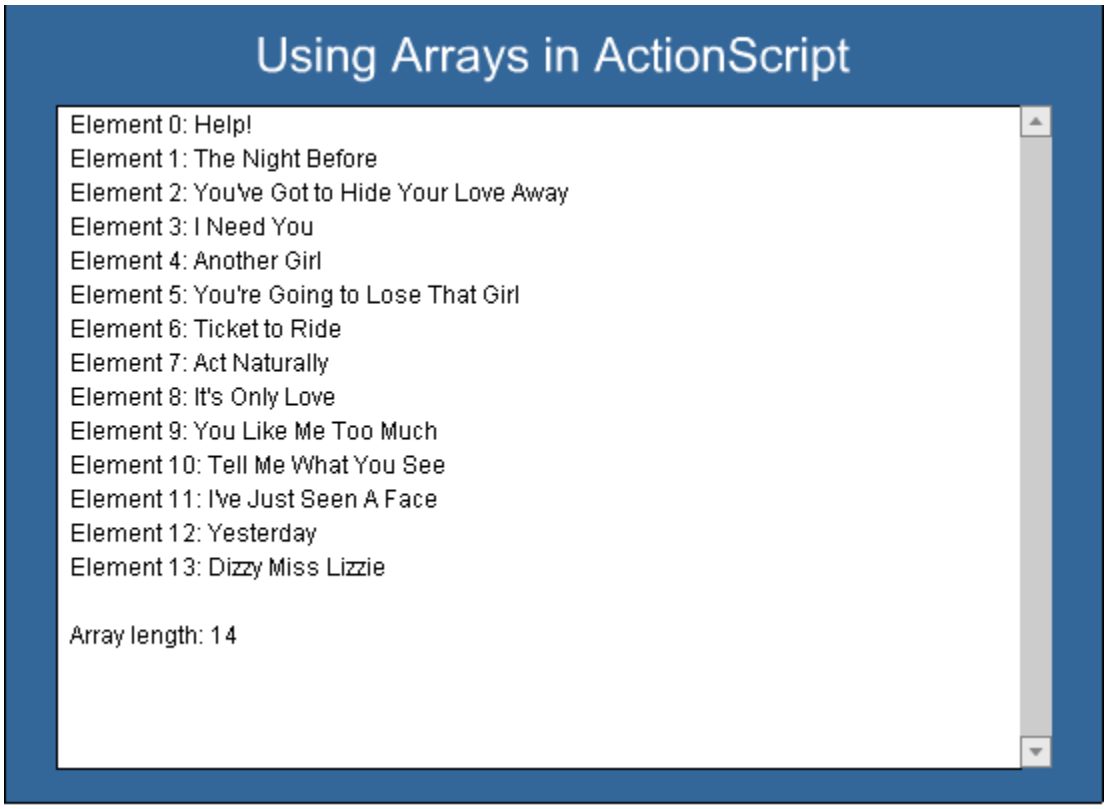
You should get the following result:



Notice that the last index number is 13, even though *Help!* has 14 tracks. This is because ActionScript's array index starts at zero. However, if we use the `Array.length` property, we can determine how many elements are contained in the array. To do this, add the following code to the end of arrays1.as and test the movie:

```
output_txt.htmlText += "Array length: " + help.length;
```

This should produce the following result:



At this point, the completed ActionScript file looks like this:

```
output_txt.htmlText = "";


function output(arr){
  len = arr.length;
  for(var i=0; i<len; i++){
    output_txt.htmlText += "Element " + i + ": " + arr[i] + "<br>";
  }
  output_txt.htmlText += "<br>";
}


var help = new Array();
help[0] = "Help!";
help[1] = "The Night Before";
help[2] = "You've Got to Hide Your Love Away";
help[3] = "I Need You";
help[4] = "Another Girl";
help[5] = "You're Going to Lose That Girl";
help[6] = "Ticket to Ride";
help[7] = "Act Naturally";
help[8] = "It's Only Love";
help[9] = "You Like Me Too Much";
help[10] = "Tell Me What You See";
help[11] = "I've Just Seen A Face";
help[12] = "Yesterday";
help[13] = "Dizzy Miss Lizzie";


output(help);


output_txt.htmlText += "Array length: " + help.length;
```

Listing 2 *The final version of arrays1.as*

If you didn't get the same results, refer to the final version of arrays1.as in the **completed** folder.

How the Code Works

Let's examine the code from Listing 2 in detail. The first line simply sets the value of `output_txt` to an empty string (`""`) so that we can use the addition assignment operator (`+=`) to add lines to the text field. The `output` function uses a `for` loop to loop through the array and add each array element to the text field:

```
for(var i=0; i<len; i++){
  output_txt.htmlText += "Element " + i + ": " + arr[i] + "<br>";
}
```

This is why array indexes are so useful. If we were storing the track listings as variables, we would have to output them like this:

```
output_txt.htmlText += track1_str;
output_txt.htmlText += track2_str;
output_txt.htmlText += track3_str;
...
```

Because `help` is an array, we can use its *index* to loop through it. In the `for` loop, the variable `i` represents the index number of each array element. Since we know that ActionScript uses zero-based arrays, we set the initial value of `i` to 0. Then we tell ActionScript to loop through the array while `i` is *less* than the length of the array (in this case, 14). After each `for` loop is executed, ActionScript increments the `i` variable by 1 (`i++`).

TIP: In case you're wondering, ++ is known as a post-increment operator. When you write i++, you increment the value of i by 1. It's equivalent to writing i = i + 1. And yes, -- is a post-decrement operator.

Just as we used the array access operator (`[]`) to populate the array, we can also use it to display elements of the array:

```
output_txt.htmlText += "Element " + i + ": " + arr[i] + "<br>";
```

At run-time, `arr[i]` evaluates to `arr[0]`, `arr[1]`, `arr[2]`, etc., depending on how many times the `for` loop has executed. If it were `arr[6]`, for example, its value would be "Ticket to Ride."

Creating an Array Using Literal Notation

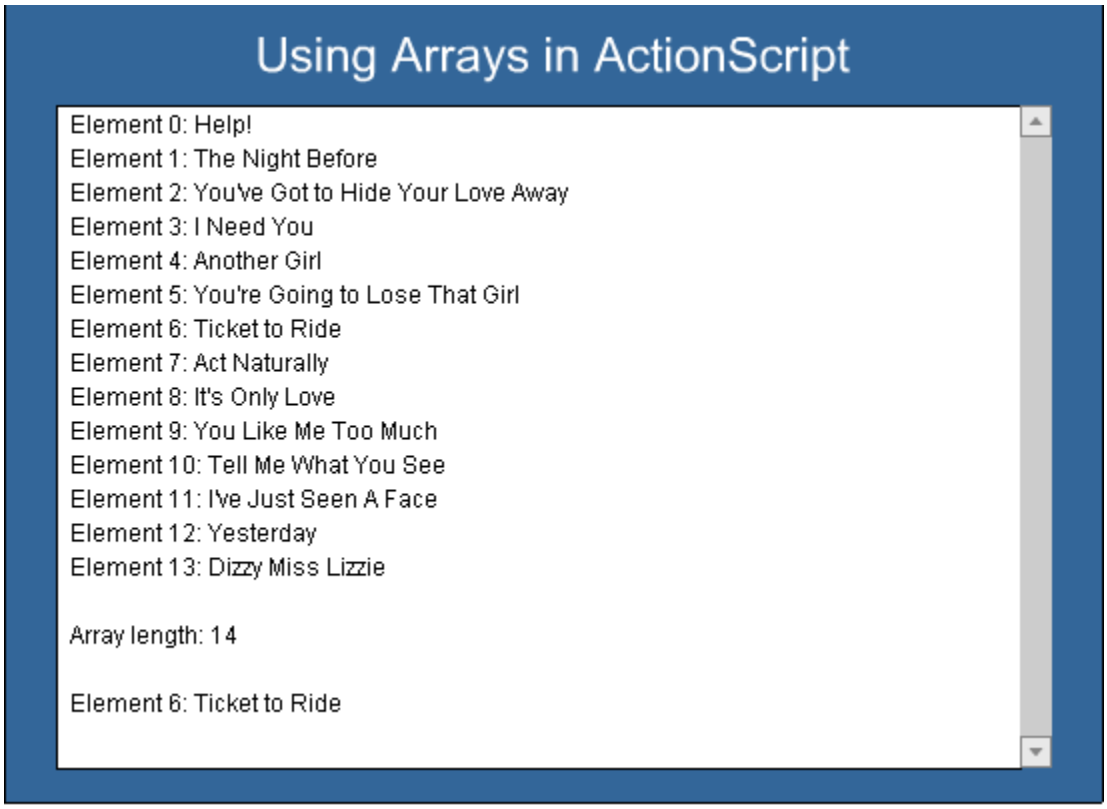
Another way to create an array is to use array literal syntax. Here's an example using the same tracks from *Help!* (this code should appear on a single line, even though it wraps in the listing below):

```
var help = ["Help!", "The Night Before", "You've Got to Hide Your
Love Away", "I Need You", "Another Girl", "You're Going to Lose That
Girl", "Ticket to Ride", "Act Naturally", "It's Only Love", "You
Like Me Too Much", "Tell Me What You See", "I've Just Seen A Face",
"Yesterday", "Dizzy Miss Lizzie"];
```

If you replace Listing 1 with the code above, you should get the same result when you publish the SWF file. The main difference between this code and Listing 1, aside from brevity, is that this version doesn't use the `new Array()` constructor to create the array object. Instead, we set the value of `help` to a comma-separated list in brackets (`[]`). ActionScript recognizes the array literal syntax, implicitly creates a new array object, and populates it with the comma-separated list. The order in which the items appear in the list determines their index numbers. For example, enter the following code after the new version of the `help` array:

```
output_txt.htmlText += "<br>Element 6: " + help[6];
```

You should get the following result:



You can even nest arrays inside arrays, to simulate a *multidimensional array*. For example, let's say we wanted to include the composer and running time of each track on *Help!*. Here's one way to do it:

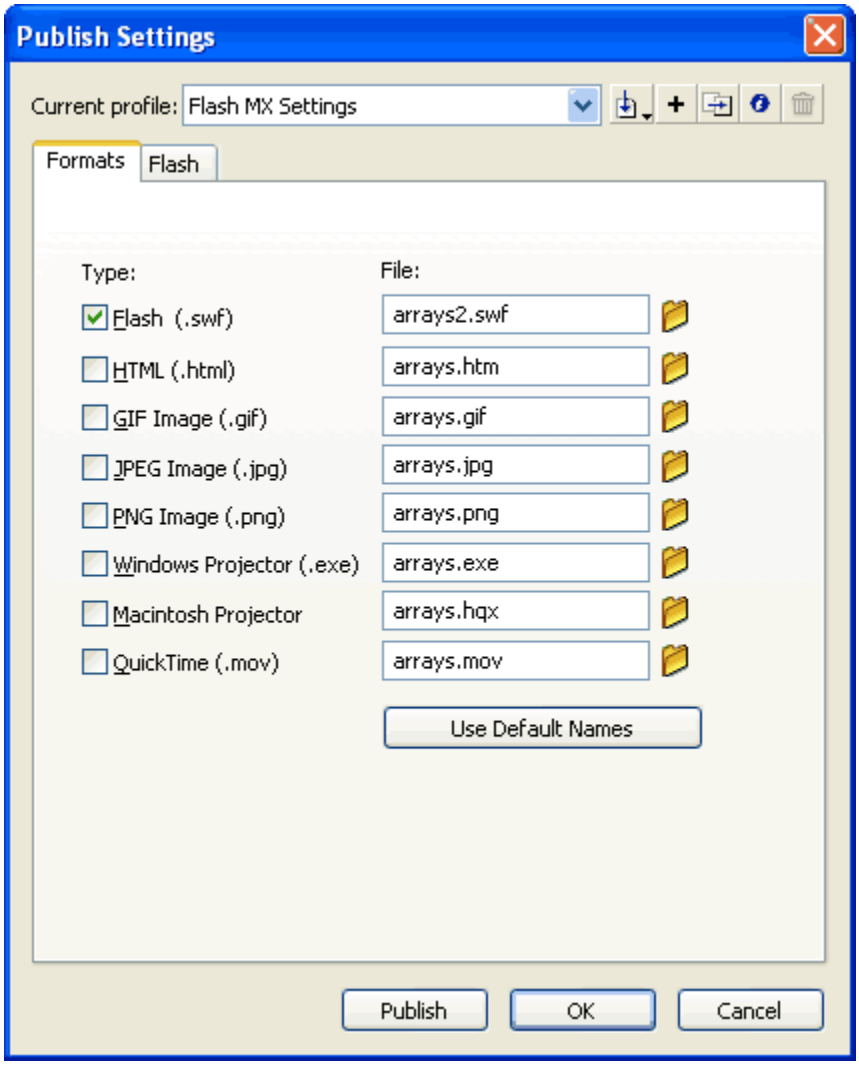
```
output_txt.htmlText = "";


// Version 1
function output(arr){
  var len = arr.length;
  for(var i=0; i<len; i++){
    output_txt.htmlText += "<b>Track " + (i+1) + "</b>";
    var len2 = arr[i].length;
    for(var j=0; j<len2; j++){
      output_txt.htmlText += "Element " + j + ": " + arr[i][j];
    }
  output_txt.htmlText += "<br>";
  }
}


// create a nested array using array literal syntax
var help = [
    ["Help!", "2:20", "Lennon/McCartney"],
    ["The Night Before", "2:36", "Lennon/McCartney"],
    ["You've Got to Hide Your Love Away", "2:11",
"Lennon/McCartney"],
    ["I Need You", "2:31", "Harrison"],
    ["Another Girl", "2:07", "Lennon/McCartney"],
    ["You're Going to Lose That Girl", "2:19", "Lennon/McCartney"],
    ["Ticket to Ride", "3:12", "Lennon/McCartney"],
    ["Act Naturally", "2:32", "Morrison/Russell"],
    ["It's Only Love", "1:58", "Lennon/McCartney"],
    ["You Like Me Too Much", "2:38", "Harrison"],
    ["Tell Me What You See", "2:39", "Lennon/McCartney"],
    ["I've Just Seen A Face", "2:06", "Lennon/McCartney"],
    ["Yesterday", "2:06", "Lennon/McCartney"],
    ["Dizzy Miss Lizzie", "2:53", "Williams"]
];


output(help);
```

Listing 3 *Simulating a multidimensional array by nesting arrays in ActionScript*

Complete the following steps to display the new array in Flash:

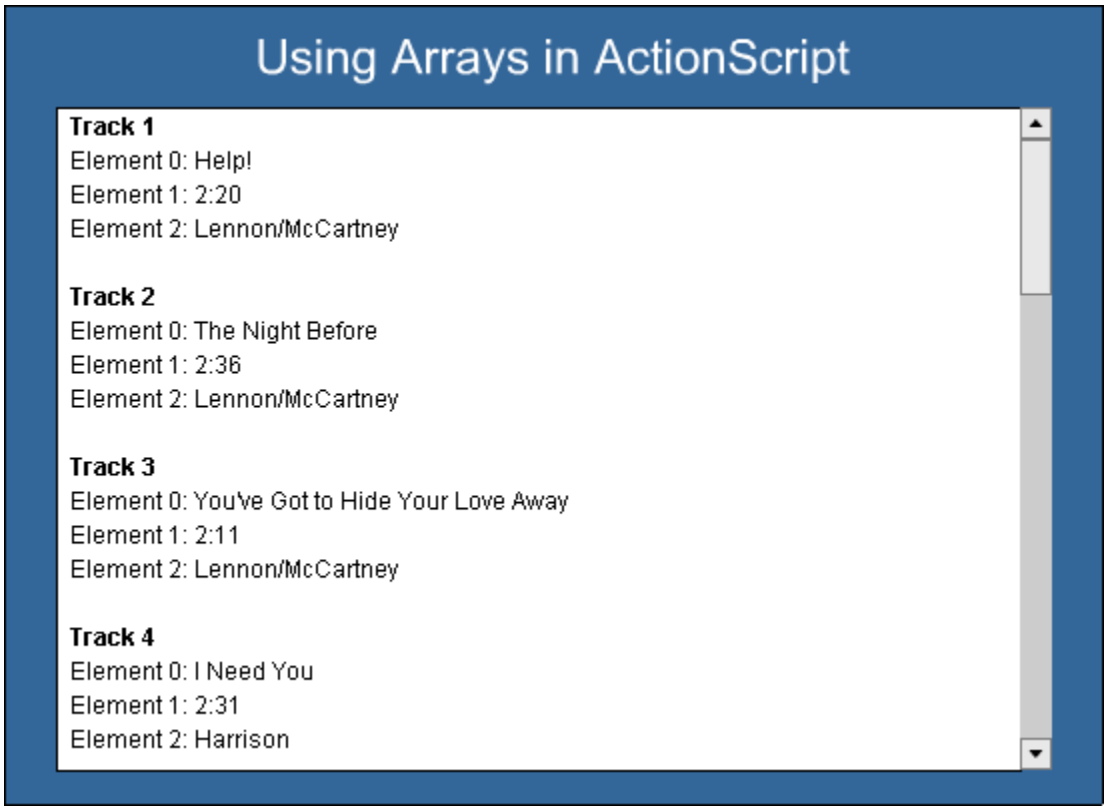
1. Create a new ActionScript file in Flash 2004.
2. Insert the code from Listing 3 into the AS file and save it as **arrays2.as**.
3. Click the **arrays.fla** tab on the Document window.

4. Select the **actions** layer and choose **Window > Development Panels > Actions** (or press **F9**) to open the Actions panel.
5. Change the first line to **#include "arrays2.as"** and save the file.
6. Choose **File > Publish Settings** (or press **Ctrl+Shift+F12**) and change the filename to **arrays2.swf**.



7. Click **OK** to close the Publish Settings dialog box.
8. Press **Ctrl+Enter** to test the movie.

You should get the following result:



As you can see, Listing 3 loops through the outer array (the track numbers), then it loops through the inner array (the title, running time, and composer of each track). Just as the arrays are nested, so are the `for` loops (edited here for clarity):

```
for(var i=0; i<arr.length; i++){
  output_txt.htmlText += "<b>Track " + (i+1) + "</b>";
  for(var j=0; j<arr[i].length; j++){
    output_txt.htmlText += "Element " + j + ": " + arr[i][j];
  }
}
```

Notice the strange pairing of array access operators (`[]`) at the end of the fourth line: `arr[i][j]`. In this example, the first set of brackets identifies an element of the outer array, and the second set identifies an element of the nested array. The first time the `for` loop is executed, this is how the variables are evaluated at run-time:

```
for(var i=0; i<14; i++){
  output_txt.htmlText += "<b>Track " + 1 + "</b>";
  for(var j=0; j<3; j++){
    output_txt.htmlText += "Element " + 0 + ": " + arr[0][0];
  }
}
```

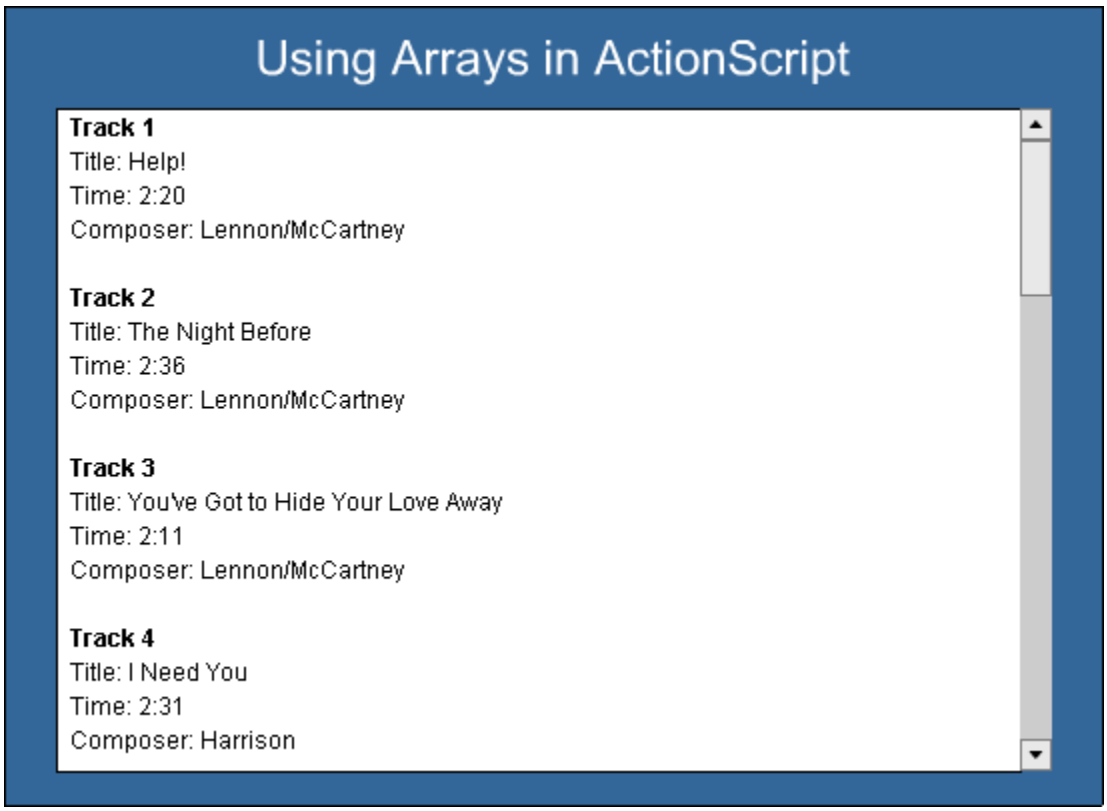
On line 4, `arr[0][0]` evaluates to "Help!", the title of the first element in the first nested array. It might help to think of nested arrays like rows and columns of a database, or an Excel spreadsheet:

	Column 0	Column 1	Column 2
Row 0	arr[0][0]	arr[0][1]	arr[0][2]
Row 1	arr[1][0]	arr[1][1]	arr[1][2]
Row 2	arr[2][0]	arr[2][1]	arr[2][2]

The problem with Listing 3 is that the title, running time, and composer elements are identified merely as Element 0, Element 1, and Element 2. To remedy this, replace the `output` function in Listing 3 with the following code:

```
// version 2
function output(arr){
  var len = arr.length;
  for(var i=0; i<len; i++){
    output_txt.htmlText += "<b>Track " + (i+1) + "</b>";
    output_txt.htmlText += "Title: " + arr[i][0];
    output_txt.htmlText += "Time: " + arr[i][1];
    output_txt.htmlText += "Composer: " + arr[i][2];
    output_txt.htmlText += "<br>";
  }
}
```

This should produce the following result:



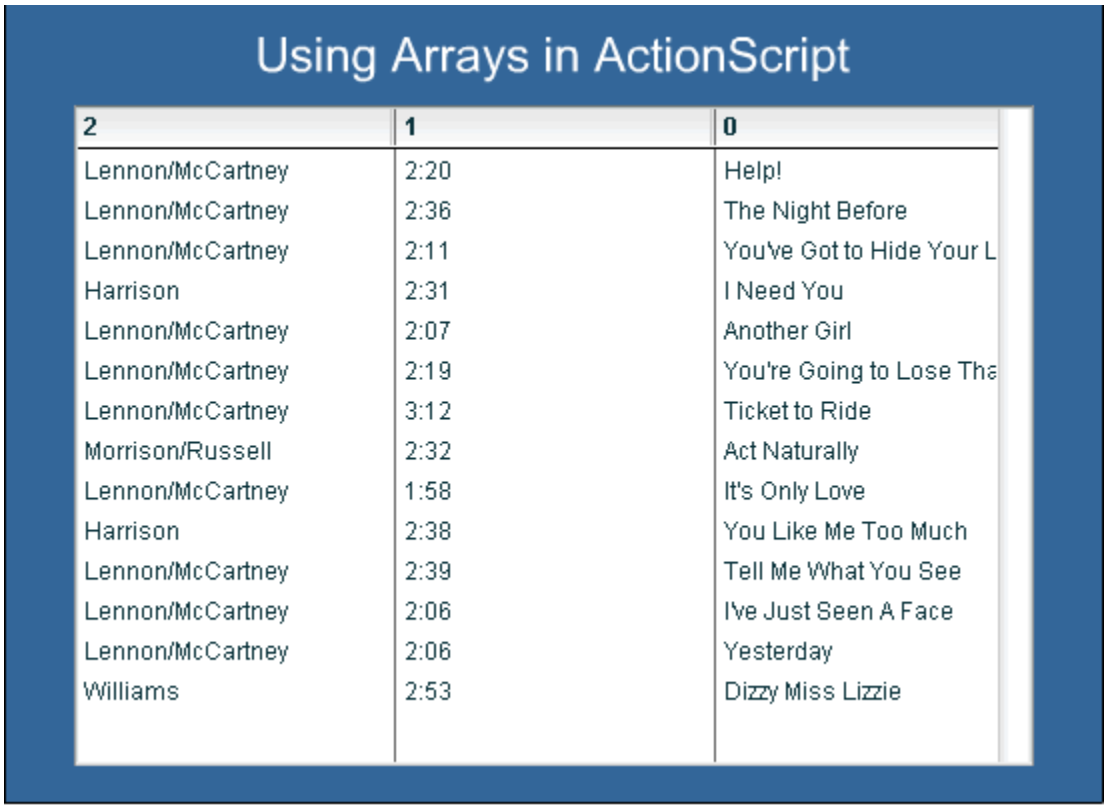
Now that's more like it! However, there is still an important limitation to Version 2: the index numbers of the nested array are hard-coded. This means that if we later decide to add additional elements to the nested array — publisher, recording date, etc. — we will have to revise the `output` function. The labels in the nested array — Title, Time, Composer — are also hard-coded. If only there were a way to make arrays self-documenting, so that the array could identify its elements by name, rather than by index numbers. Enter associative arrays.

Using Associative Arrays

The advantage of an *associative array* is that it uses *keys*, or alphanumeric names, rather than numeric indexes, to identify its elements. Why is this important? Let's say, for example, we want to output the results of the `help` array using Flash's DataGrid component. We could easily do this using the following code (see **grid.as**):

```
// create a nested array using array literal syntax
var help = [
    ["Help!", "2:20", "Lennon/McCartney"],
    ["The Night Before", "2:36", "Lennon/McCartney"],
    ["You've Got to Hide Your Love Away", "2:11",
"Lennon/McCartney"],
    ["I Need You", "2:31", "Harrison"],
    ["Another Girl", "2:07", "Lennon/McCartney"],
    ["You're Going to Lose That Girl", "2:19", "Lennon/McCartney"],
    ["Ticket to Ride", "3:12", "Lennon/McCartney"],
    ["Act Naturally", "2:32", "Morrison/Russell"],
    ["It's Only Love", "1:58", "Lennon/McCartney"],
    ["You Like Me Too Much", "2:38", "Harrison"],
    ["Tell Me What You See", "2:39", "Lennon/McCartney"],
    ["I've Just Seen A Face", "2:06", "Lennon/McCartney"],
    ["Yesterday", "2:06", "Lennon/McCartney"],
    ["Dizzy Miss Lizzie", "2:53", "Williams"]
];


// populate DataGrid component using the dataProvider property
myDG.dataProvider = help
```

This produces the following result:

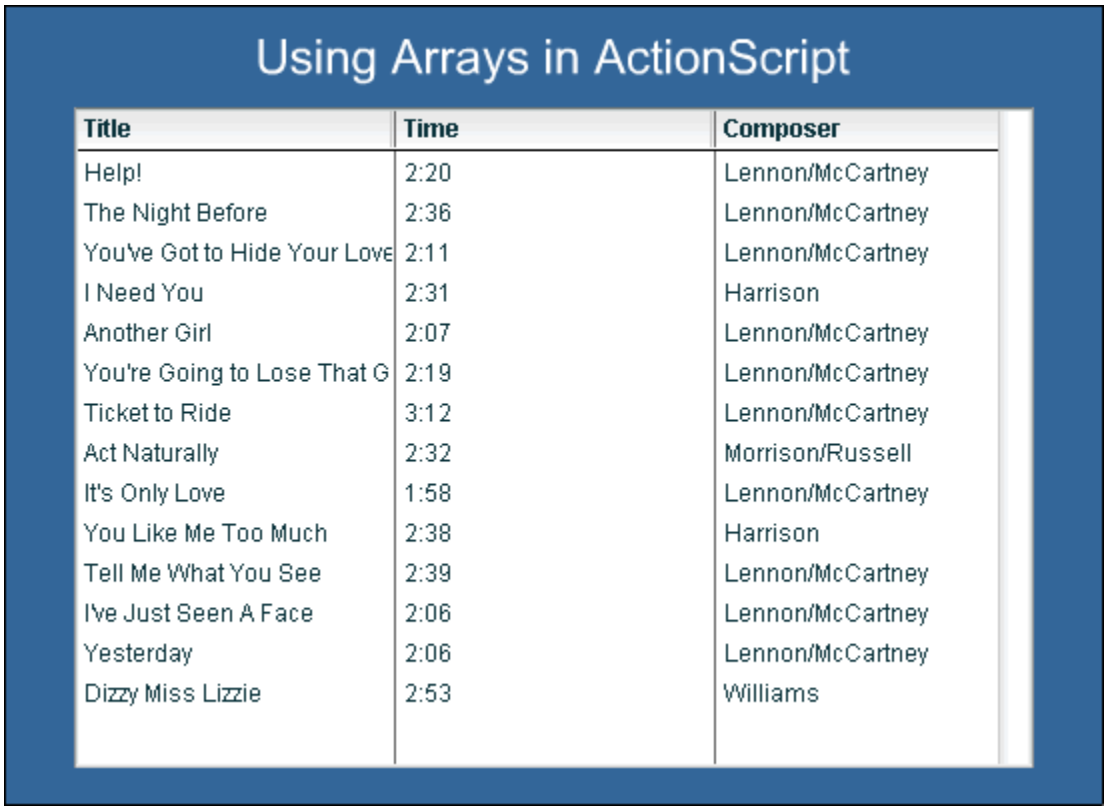


Already there's a problem. You and I know what this grid means because we've been working with the `help` array, but it won't make much sense to anyone else (except a Beatlemaniac). One workaround is to loop through the array and populate the DataGrid using the `DataGrid.addItem` method. However, this still requires hard-coding the names of the DataGrid columns. The solution, as you've probably guessed, is to create an associative array. To illustrate this, open **grid.fla** and **grid.as** in Flash and replace the contents of grid.as with the following code:

```
// create an associative array using object literal syntax
var help = [
    {Title: "Help!", Time: "2:20", Composer: "Lennon/McCartney"},
    {Title: "The Night Before", Time: "2:36", Composer:
"Lennon/McCartney"},
    {Title: "You've Got to Hide Your Love Away", Time: "2:11",
Composer: "Lennon/McCartney"},
    {Title: "I Need You", Time: "2:31", Composer: "Harrison"},
    {Title: "Another Girl", Time: "2:07", Composer:
"Lennon/McCartney"},
    {Title: "You're Going to Lose That Girl", Time: "2:19",
Composer: "Lennon/McCartney"},
    {Title: "Ticket to Ride", Time: "3:12", Composer:
"Lennon/McCartney"},
    {Title: "Act Naturally", Time: "2:32", Composer:
"Morrison/Russell"},
    {Title: "It's Only Love", Time: "1:58", Composer:
"Lennon/McCartney"},
    {Title: "You Like Me Too Much", Time: "2:38", Composer:
"Harrison"},
    {Title: "Tell Me What You See", Time: "2:39", Composer:
"Lennon/McCartney"},
    {Title: "I've Just Seen A Face", Time: "2:06", Composer:
"Lennon/McCartney"},
    {Title: "Yesterday", Time: "2:06", Composer:
"Lennon/McCartney"},
    {Title: "Dizzy Miss Lizzie", Time: "2:53", Composer: "Williams"}
];


// populate DataGrid component using the dataProvider property
myDG.dataProvider = help;
```

Listing 4 *The **help** array now contains 14 associative arrays*

This produces the following result:



Now we're cooking with gas! You can even sort the results by clicking on the column headers. To loop through the new array, insert the following code at the end of grid.as and test the movie:

```
// loop through associative array
for(var i in help){
  for(var j in help[i]){
    trace(j + ": " + help[i][j]);
  }
trace(newline);
}
```

This should produce the following results:

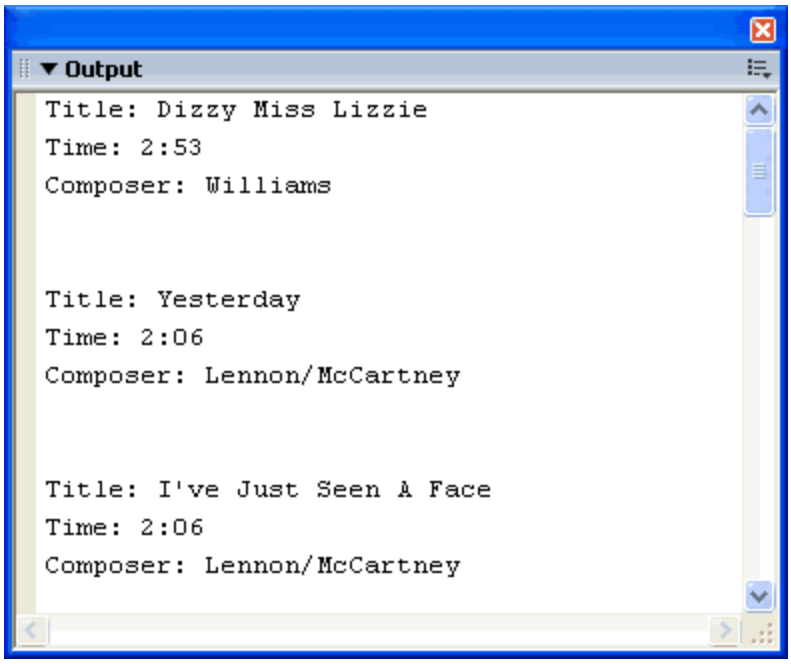


Figure 2 *Tracing the new **help** array in the Output panel*

Notice that this time, `j` evaluates to Title, Time, or Composer, rather than a number (0, 1, 2). The nested associative arrays are actually using names, rather than numbers, to identify their elements. To illustrate this, enter the following code at the end of grid.as and test the movie:

```
trace(help[6]["Title"]);
```

This should print "Ticket to Ride" to Flash's Output panel.

Understanding Associative Arrays

All right, let's backtrack a little. We've introduced two new concepts here: *associative arrays* and *object literals*. Let's examine each one more closely.

In my opinion, working with an associative array is much more intuitive than using numeric arrays. In particular, dot notation is already a familiar concept to Flash, ASP, and ColdFusion developers:

```
Flash Target Path
myMovieClip.Comments_txt.text;


ASP Recordset
<%= rsGuestbook.Fields.Item("Comments").Value %>


ColdFusion Query
<cfoutput>#qGuestbook.Comments#</cfoutput>
```

Creating and manipulating associative arrays in ActionScript is a lot like working with ColdFusion

structures. Here's an example of an associative array containing Track 1 of *Help!*:

```
var track1 = new Object();
track1.Title = "Help!";
track1.Time = "2:20";
track1.Composer = "Lennon/McCartney";

trace(track1.Title); // output: "Help!"
```

Listing 5 *Creating an associative array in ActionScript*

That's all there is to it.

Understanding Object Literals

Now that you know what an associative array is, you may be wondering about the curly braces (`{}`) in Listing 4. Object literal syntax is a shorthand method for creating associative arrays, just as the array literal syntax is a shortcut for writing normal arrays.

```
Array Literal
var beatles = ["John", "Paul", "George", "Ringo"];
trace(beatles[0]); // output: "John"


Object Literal
var track1 = {Title: "Help!", Time: "2:20", Composer:
"Lennon/McCartney"};
trace(track1.Title); // output: "Help!"
```

In the first example, Flash creates a new array object named `beatles`. In the second example, Flash creates an associative array named `track1`, using the `{key: "property"}` syntax. Despite the different techniques, the object literal example and Listing 5 create exactly the same associative array. The main advantage of using object-literal syntax is that it's less verbose, and once you're familiar with it, easier to read.

But what about Listing 4? It creates unnamed associative arrays:

```
var help = [
    {Title: "Help!", Time: "2:20", Composer: "Lennon/McCartney"},
    {Title: "The Night Before", Time: "2:36", Composer:
"Lennon/McCartney"},
    ...
];
```

How does this work? In the case of Listing 4, Flash creates what is known as *anonymous objects*. Each of the 14 "rows" of the `help` array is an associative array, and each associative array is an anonymous object.

You can confirm this by testing grid.fla in Flash (**Control > Test Movie**) and selecting **Debug > List Variables**. If you scroll down the list in the Output panel, eventually you'll find the associative arrays:

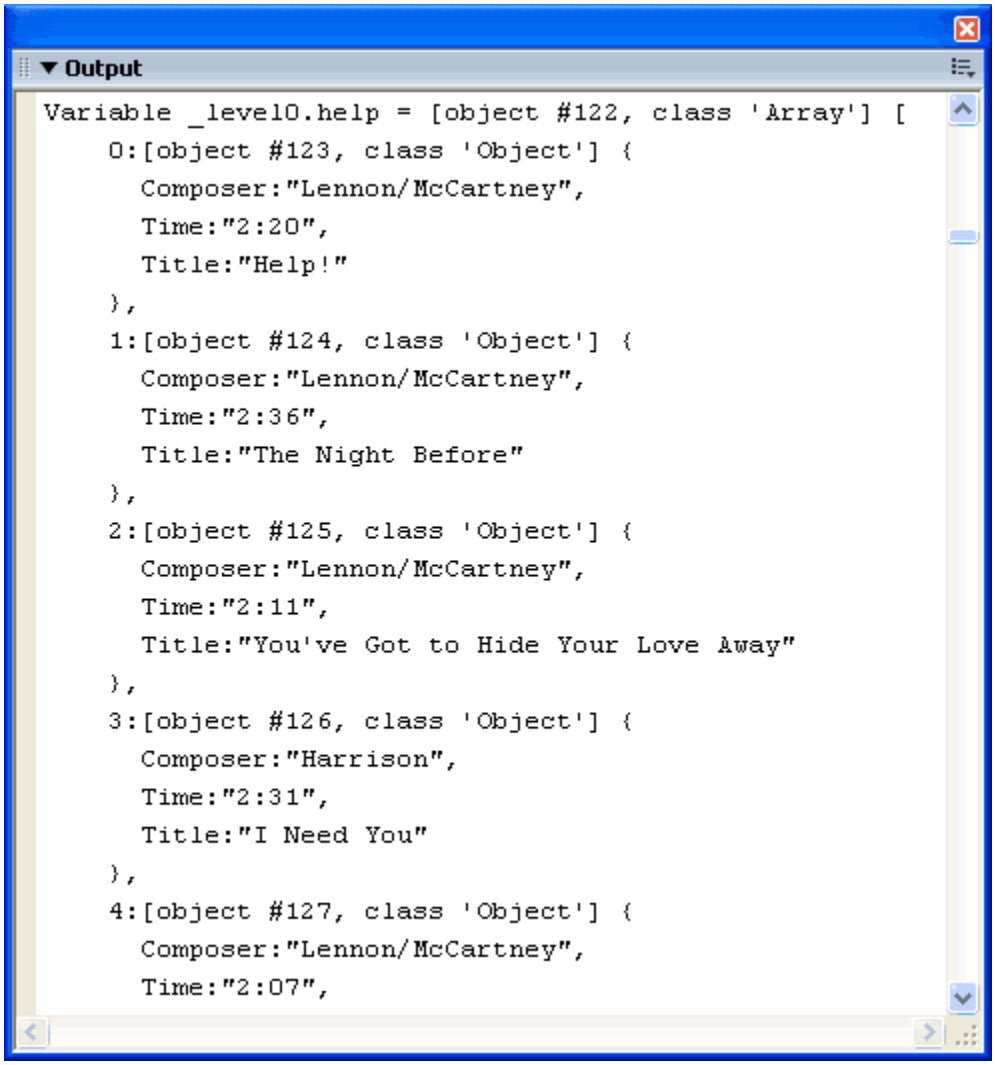
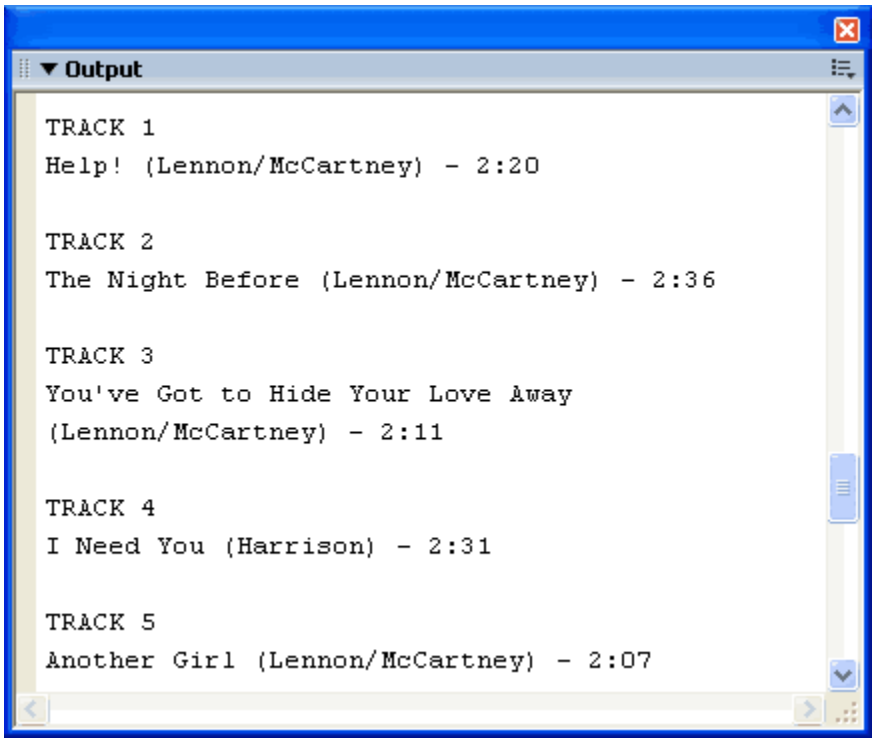


Figure 3 *The associative arrays in Flash's Output panel*

Whereas `help` is clearly identified on the first line, the associative arrays nested inside `help` are unnamed. How is this possible? The code from Listing 4 is legal in Flash because the index numbers of the outer array (0-13) can be used to identify each nested array, or "row":

```
len = help.length;
for(var i=0; i<len; i++){
  trace("TRACK " + (i+1));
  trace(help[i].Title + " (" + help[i].Composer + ") - " +
help[i].Time + newline);
}
```

This results in the following output:



Compare this with the `output` function in Listing 3 and you get an immediate appreciation for how much easier it is to work with associative arrays.

Conclusion

I hope this tutorial gave you a sense of the power and versatility of arrays in ActionScript. In Part 2, we will examine how to add, edit, sort, and remove array elements using the built-in methods of the `Array` class.

Keywords
flash mx 2004, actionscript, arrays, object literal, array literal, associative array