B

ActionScript Optimization Measurements

Code optimization is an evidence-based discipline. All optimizations detailed in *Chapter 15, ActionScript Optimization* were verified across different versions of AIR. The results from various tests carried out are provided here along with links to their source code.

At the time of writing, the most recent version of the AIR SDK (3.1) was overlaid onto Flash Professional CS5.5 and used. The tests were also carried out using AIR 2.0 to accommodate Flash Professional CS5 users.

There are no guarantees that these optimizations will apply to future versions of AIR. However, the source code for these tests has been provided if you wish to run them yourself.

The table below shows the four devices used for testing:

Device and model	iOS version	CPU	RAM
First-generation iPod touch	3.1.3	412 MHz	128 MB
Third-generation iPod touch	5.0.1	600 MHz	256 MB
iPhone 4	5.0.1	800 MHz	512 MB
iPad 2	5.0.1	1 GHz	512 MB

Each device was running iOS 5 with the exception of the first-generation iPod touch, which had iOS 3.1 installed. As older ARMv6 devices are not supported by more recent versions of AIR, the first-generation iPod touch was omitted from the AIR 3.1 tests.

The execution time of each test was used to compare the performance benefits of each optimization. The results shown throughout this chapter are an average taken from eight runs of each test.

Execution times are measured in milliseconds and shown throughout this Appendix.

Declaring data types

Two tests were run to compare the performance benefits of using typed variables against un-typed variables.

Test A performs a series of operations on a collection of un-typed variables, whereas Test B performs the same operations using typed versions of those variables.

The following code snippet shows some of the un-typed variables from Test A being declared:

```
var a = 1;
var c = 1.5;
var e = "hullo ";
```

You can see the same variables in Test B but with a data type being explicitly declared for each:

```
var a:int = 1;
var c:Number = 1.5;
var e:String = "hullo ";
```

Each test performs 100,000 iterations.

The complete source code for both tests is available from the book's accompanying code bundle at `appendix-b\declaring-data-types\`.

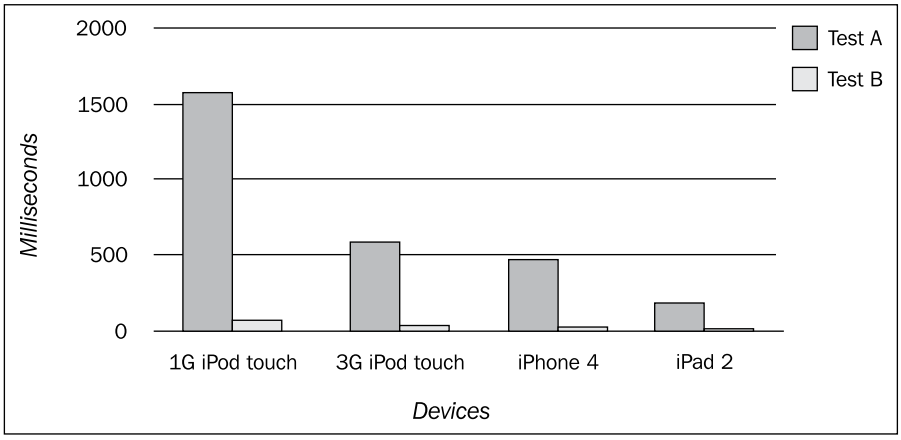
Let us take a look at the performance measurements, starting with AIR 2.0 used by Flash Professional CS5.

AIR 2.0

The following table shows the times taken for each test to run:

	Test A (ms)	Test B (ms)
1G iPod touch	1576	72
3G iPod touch	590	38
iPhone 4	477	28
iPad 2	177	14

Displaying the same data within the following chart really highlights the performance advantage of declaring each variable's type:

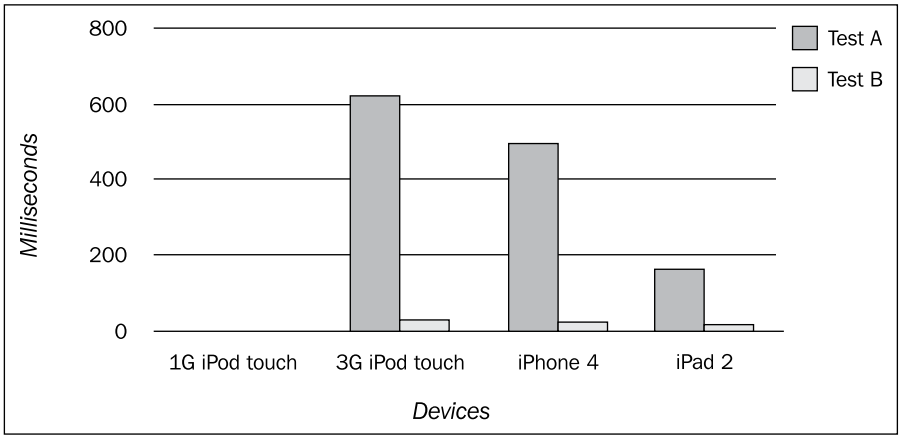


AIR 3.1

Similarly impressive gains are to be had when using AIR 3.1:

	Test A (ms)	Test B (ms)
1G iPod touch		
3G iPod touch	623	32
iPhone 4	498	25
iPad 2	167	18

The test results are also shown in the following chart:



Replacing objects with custom classes

Two tests were run to compare the difference in performance between accessing properties of an `Object` instance and properties of a custom class.

Test A accesses and updates properties belonging to an `Object`, whereas Test B performs the same operations on an equivalent custom class.

The following code snippet shows the `Object` from Test A being defined:

```
var player:Object = {
  lives:  3,
  energy: 100,
  score:  0
};
```

Here's the equivalent custom class used by Test B:

```
public class Player {
  public var lives :uint = 3;
  public var energy:uint = 100;
  public var score :uint = 0;
}
```

Each test performs 100,000 iterations.

The complete source code for each test is available from the book's accompanying code bundle at `appendix-b\object-v-class\`.

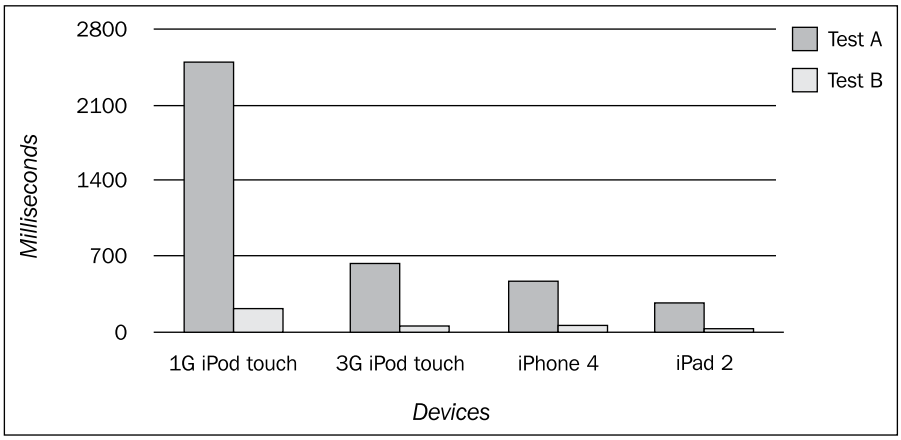
Let us take a look at the performance measurements from using both AIR 2.0 and AIR 3.1.

AIR 2.0

From the following table, it is clear that accessing a dynamic object is significantly slower than accessing an instance of a custom class:

	Test A (ms)	Test B (ms)
1G iPod touch	2537	225
3G iPod touch	646	95
iPhone 4	483	74
iPad 2	299	36

The following chart shows the same data:

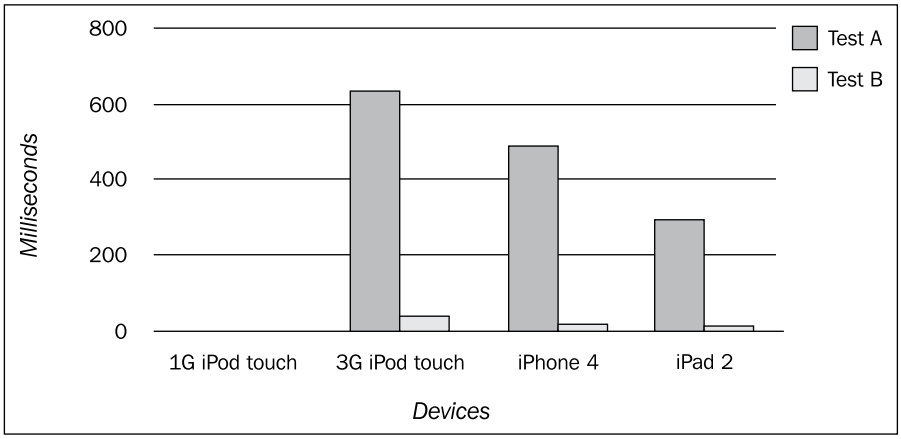


AIR 3.1

Now let us take a look at the results of the tests compiled from Flash Professional CS5.5 using AIR 3.1:

	Test A (ms)	Test B (ms)
1G iPod touch		
3G iPod touch	640	26
iPhone 4	485	20
iPad 2	318	16

Again, there is a huge performance increase when accessing a custom class as opposed to a dynamic equivalent. The test results are also shown in the following chart:



Optimizing loops

Three tests were run to check for performance benefits when applying various optimizations to `for` loops. Each test iterates over an array of 100,000 elements.

Test A uses an iterator of type `Number` and queries the array's length when evaluating the loop's condition expression. The following code snippet illustrates this:

```
for(var i:Number=0; i<array.length; i++)
{
  // Do something
}
```

Test B uses an iterator of type `int` and queries the array's length when evaluating the loop's condition expression:

```
for(var i:int=0; i<array.length; i++)
{
  // Do something
}
```

Test C uses an iterator of type `int` and caches the array's length within a local variable, avoiding the need to query the array's length as part of the loop's condition expression:

```
var len:int = array.length;
for(var i:int=0; i<len; i++)
{
  // Do something
}
```

The complete source code for each test is available from the book's accompanying code bundle at `appendix-b\loop-optimizations\`.

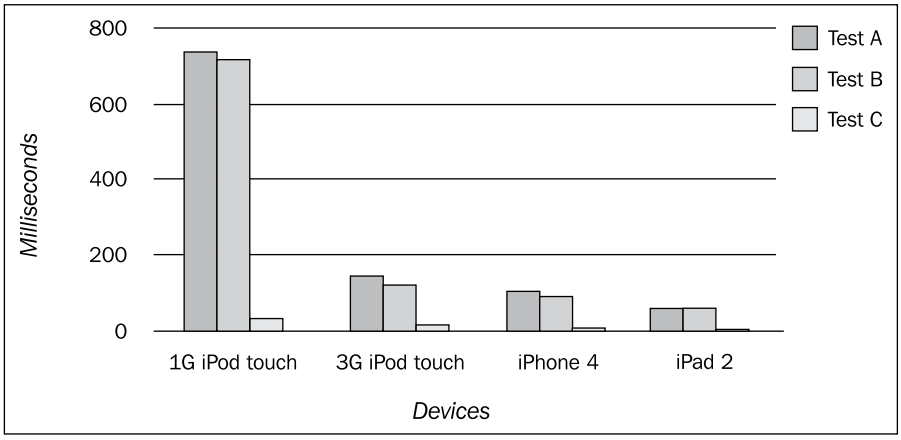
We will start by taking a look at the performance measurements from running the three tests using AIR 2.0, before moving onto AIR 3.1.

AIR 2.0

The following table shows the times taken for each of the three tests to run:

	Test A (ms)	Test B (ms)	Test C (ms)
1G iPod touch	746	723	34
3G iPod touch	148	122	17
iPhone 4	109	91	11
iPad 2	62	63	7

Whereas Test B provides relatively small gains in performance, Test C shows a significant improvement. This is visualized in the following chart:

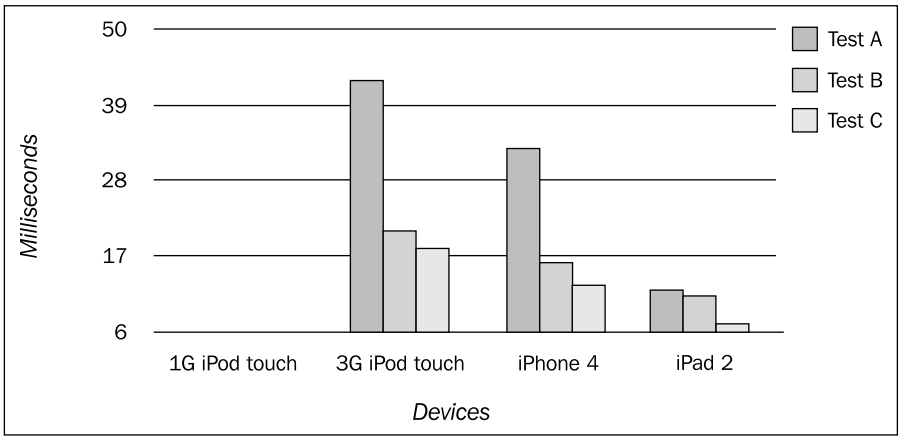


AIR 3.1

Using AIR 3.1, Tests A and B perform significantly faster than their AIR 2.0 counterparts. It is clear that ADT is applying optimizations to the loops that aren't being applied by AIR 2.0's PFI tool. Test C shows that ActionScript optimization is still beneficial in AIR 3.1, but expect the gains to be modest.

	Test A (ms)	Test B (ms)	Test C (ms)
1G iPod touch			
3G iPod touch	42	21	18
iPhone 4	33	16	13
iPad 2	12	11	7

The test results are also shown in the following chart:



Replacing arrays with vectors

Four tests were run to evaluate the strength of using vectors over arrays. Each test creates and initializes a collection of 100,000 values.

Test A creates an empty array and uses a `for` loop to populate each index of the array. The following code block shows this:

```
const SIZE:int = 100000;
var collection:Array = [];
for(var i:int=0; i<SIZE; i++)
{
  collection[i] = 0;
}
```

Test B creates the array's size ahead of time. It then uses a `for` loop to initialize each element:

```
const SIZE:int = 100000;
var collection:Array = new Array(SIZE);
for(var i:int=0; i<SIZE; i++)
{
  collection[i] = 0;
}
```

Test C creates an empty vector and uses a `for` loop to populate each index of the vector:

```
const SIZE:int = 100000;
var collection:Vector.<int> = new Vector.<int>();
for(var i:int=0; i<SIZE; i++)
{
  collection[i] = 0;
}
```

Test D creates a fixed-sized vector containing 100,000 elements. It then uses a `for` loop to initialize each element:

```
const SIZE:int = 100000;
var collection:Vector.<int> = new Vector.<int>(SIZE, true);
for(var i:int=0; i<SIZE; i++)
{
  collection[i] = 0;
}
```

The complete source code for each test is available from the book's accompanying code bundle at `appendix-b\arrays-v-vectors\`.

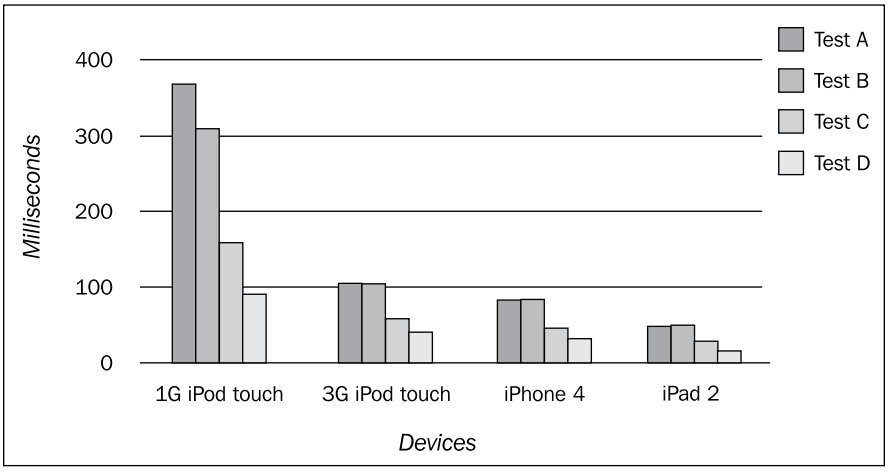
Let us see how each test performs, starting with the results using Flash Professional CS5 and AIR 2.0.

AIR 2.0

The following table shows that vectors (Tests C and D) have faster read and write access than arrays (Tests A and B). A fixed-size vector is particularly fast as it doesn't grow in size, preventing the need for additional memory allocations during its lifetime.

	Test A (ms)	Test B (ms)	Test C (ms)	Test D (ms)
1G iPod touch	341	312	161	92
3G iPod touch	109	106	61	44
iPhone 4	85	86	47	33
iPad 2	49	50	29	17

The test results are also shown in the following chart:



AIR 3.1

When compiling with AIR 3.1, fixed-size vectors still have a significant performance benefit. However, it is clear that the gap has been bridged when comparing arrays (Tests A and B), and vectors that can dynamically grow in size (Test C).

	Test A (ms)	Test B (ms)	Test C (ms)	Test D (ms)
1G iPod touch				
3G iPod touch	99	84	78	37
iPhone 4	76	66	58	28
iPad 2	39	39	40	16

You can also see the execution times presented in the following chart: