Adobe® Flash®
Player ActionScript
Virtual Machine (Tamarin)

Rick Reitmaier
Sr. Scientist
Adobe Systems

December 6, 2006
Adobe Today

Worldwide Offices

Corporate Headquarters – San Jose, California

Key Statistics

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Adobe FY 2005 Revenue</td>
<td>$1.966B</td>
</tr>
<tr>
<td>Macromedia FY 2005 Revenue</td>
<td>$436M</td>
</tr>
<tr>
<td>Years in Business</td>
<td>23</td>
</tr>
<tr>
<td>Employees</td>
<td>5,000</td>
</tr>
</tbody>
</table>
Widest Reach in the World

600 million
PCs and devices

Source: NPD December 2005
Copyright 2006 Adobe Systems Incorporated.
Device shipment growth is explosive

- 150+ handset models
- 300+ device models
- Over 100 million devices shipped
Plug-In Technology Penetration: Mature Markets

Internet Connected Desktops

- Flash Player typically reaches 80% adoption in < 1yr each release.
- Version 8 (Fall 2005) achieved a higher adoption in less time.
- Version 9 (Summer 2006) appears to be following same curve as version 8.

---

* MillwardBrown, September 2006
Flash Video Momentum

Flash Video Growth

- MySpace
- Yahoo
- YouTube
- Viacom
- ABC/Disney
- NBC
- CNET
- Comcast
- Showtime
- E! OnLine
- Ralph Lauren/Polo
- Red Bull
- ClickTV
- Motionbox

Copyright 2006 Adobe Systems Incorporated.
Anatomy of a Rich Internet Application

- **Menus & Navigation Controls**
- **Bi-Directional Audio & Video**
- **Data Visualization & Collaboration**
- **Real Time Data Push & Alerting**
- **Resizable Views/Effects/Transitions**
- **Chat**
- **Mapping, Geo-Coding, Annotations & White boarding**
- **Rich Data Entry**
- **Data Synchronization & Conflict Resolution**
- **Offline**

Copyright 2006 Adobe Systems Incorporated.
All this using a scripting language?

- **Strengths**
  - low barrier to entry
  - rapid, flexible development
  - familiar
  - pervasive (the web wouldn’t compile)

- **Opportunities**
  - safe, predictable and efficient
  - debugging
# A Tale of Names

<table>
<thead>
<tr>
<th>ECMA</th>
<th>Mozilla</th>
<th>Adobe</th>
</tr>
</thead>
<tbody>
<tr>
<td>262 edition 2</td>
<td>JavaScript 1.3</td>
<td>ActionScript 1.0</td>
</tr>
<tr>
<td>262 edition 3</td>
<td>JavaScript 1.4</td>
<td>ActionScript 2.0</td>
</tr>
<tr>
<td>262 edition 4</td>
<td>JavaScript 1.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JavaScript 1.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JavaScript 1.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JavaScript 2.0</td>
<td>ActionScript 3.0</td>
</tr>
</tbody>
</table>

* JavaScript and ActionScript are trademarked names by Sun Microsystems and Adobe Systems respectively.
A bit of history

- Prototype-based scripting language
  - no classes
  - instances created by cloning other instances
  - Self (Ungar, Smith 1987) influenced design

- Object
  - consists of a collection of properties, also called ‘slots’
  - accessing a slot returns a value and may execute code.
Prototype example

var myObj = {};

myObj.value = "this is string"

myObj.addProperty("foo", function() { return "foo"; }, null);

trace(myObj.value); // <= "this is string"
trace(myObj.foo); // <= "foo"
Objectifying

- Customers requested “Class”
  - added support in Actionscript 2.0
  - stitched together using prototypes
  - pure compiler only change
  - based on unreleased ECMAScript edition 4 proposal

- Explicit data types
  - used as annotation for the compiler to perform error checking
  - ignored when executing
Object Example

class A {
    var value:String;
    function get foo():String { return "foo"; }
}

var myObj:A = new A();

myObj.value = "this is string"

trace(myObj.value); // <= "this is string"
trace(myObj.foo); // <= "foo"
Workflow

- **Compiler**
  - local to developers machine
  - generates bytecode for a stack based machine

- **Bytecode**
  - bytecode is representative of evolution
  - crosses levels of abstraction; eg. actionPush and actionGetTime

- **Run-time**
  - simple interpreter
AVM2

- Trends
  - more code being written
  - complexity of web applications increasing
  - componentization is a growing market

- ActionScript Virtual Machine 2
  - addresses performance
  - maintains backward compatibility; indirectly
  - allows revolutionary innovation to code base
Flash Player 9

Enterprise-Class Runtime for Data-Rich Applications

- New Virtual Machine
  - Terrific performance improvements
  - Reduced memory consumption
- ActionScript 3.0
  - ECMAScript Edition 4 compliant
  - Dynamic and typed programming language
  - External API integrates w/ AJAX & native apps
  - ECMAScript for XML (E4X)
- Seamless Deployment
  - One-click upgrade
  - Full backward compatibility

Copyright 2006 Adobe Systems Incorporated.
### Challenges

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Player size is ~1MBytes compressed</td>
<td>tight budget for growth</td>
</tr>
<tr>
<td>amount of bytecode required for ‘typical’</td>
<td>language specific operations</td>
</tr>
<tr>
<td>application must remain small</td>
<td>variable length encoding</td>
</tr>
<tr>
<td>startup speed critical for interactive</td>
<td>compression</td>
</tr>
<tr>
<td>applications</td>
<td>novel just-in-timer compiler</td>
</tr>
<tr>
<td></td>
<td>novel garbage collector</td>
</tr>
</tbody>
</table>
AVM2 Design

- **Bytecode**
  - constant data; strings, numbers, etc
  - type and method descriptors
  - exception tables
  - stack oriented abstract machine
  - object creation, slot access, property search

- **Run-time engine**
  - separate verification from execution
  - leaner interpreter
  - 2-pass just-in-time compiler
  - revamped garbage collector
AVM2 Architecture
AVM2 Garbage collector

- Reusable library
  - nothing specific to AVM2
  - new / delete (unmanaged memory)
  - new only (i.e. garbage collected)
  - debugging aids, profiling

- Characteristics
  - deferred reference counting (DRC)
  - backed by incremental, conservative mark and sweep
    - incremental = interruptible
    - conservative = random bit pattern in memory may be pointer
AVM2 Verifier

- **Structural Integrity**
  - ensure branches land on valid instructions
  - can’t fall off end of code
  - constant references are valid

- **Type Safety**
  - dataflow analysis to track types
  - early binding
AVM2 Interpreter

- Stack oriented
  - no surprises here; non-threaded
  - all values boxed; 32bit encoding
  - executes directly from verified buffer
  - no bytecode modification
AVM2 Just-in-Time Compiler (JIT)

- First pass
  - intermediate representation (MIR) produced concurrent with verification
  - early binding
  - constant folding
  - copy and constant propagation
  - common sub-expression elimination (CSE)

- Second pass
  - native code generation
  - instruction selection
  - register allocation
  - dead code elimination (DCE)
Tamarin

- AVM2 released to open source community

  The goal of the "Tamarin" project is to implement a high-performance, open source implementation of the ECMAScript 4th edition (ES4) language specification. The Tamarin virtual machine will be used by Mozilla within SpiderMonkey, the core JavaScript engine embedded in Firefox®, and other products based on Mozilla technology. The code will continue to be used by Adobe as part of the ActionScript™ Virtual Machine within Adobe® Flash® Player.

- Working intimately with Mozilla to integrate with JavaScript engine
- See for yourself
  - http://www.mozilla.org/projects/tamarin/
Interesting papers

- M. Chen, K. Olukotun, “Targeting Dynamic Compilation for Embedded Environments”
  - Computer System Lab, Stanford University

- M. Poletto, V. Sarkar, “Linear Scan Register Allocation”

- U. Hölzle, D. Ungar, “Optimizing Dyamically-Dispatched Calls with Run-Time Type Feedback”
  - Computer System Lab, Stanford University / Sun Microsystems

- J. Aycock, “A Brief History of Just-in-Time”