SYDJS July 2011

What is HaXe?

- Multi-platform language
- Open source (www.haxe.org)
- Community driven
- Version 2.07 (around since 2005)
- Single syntax for client-side, server-side & command line programming
  - ECMAScript based syntax is easy for JavaScript and ActionScript developers to work with
- Strictly typed (compile time checking) across all targets
  - Even for non compiled languages like JavaScript and PHP
- Lightning fast compilation
  - Even for large applications with multiple target/device outputs
- Excellent runtime performance on target platforms

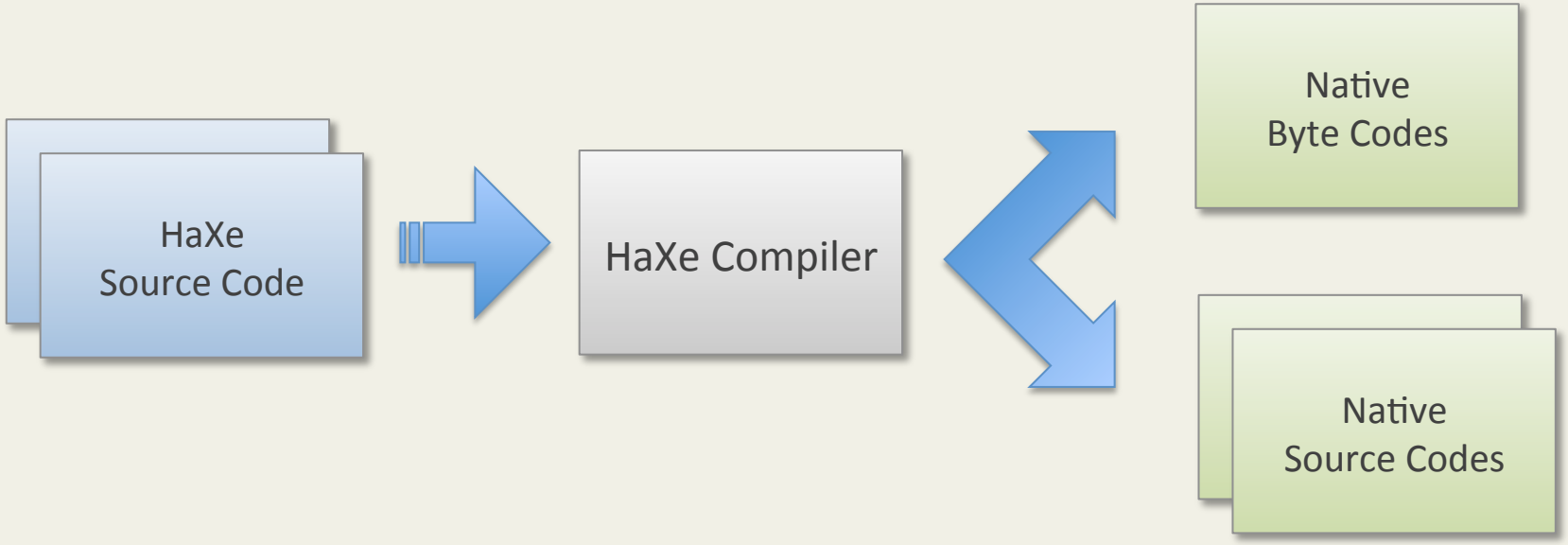
What platforms can HaXe target?

- Client side
  - JavaScript (HTML, Canvas, WebGL)
  - Flash (AS2/3, AIR, Molehill)
  - C++ (desktop, mobile platforms like Android and iOS)
- Server side
  - PHP
  - Neko
  - NodeJS
- Command Line
  - Neko
  - C++

- Java and C# (.NET) support coming soon!

How HaXe works

- The HaXe compiler translates source code into native source and/or bytecode
  - Source code like JavaScript, ActionScript, C++, PHP
  - Byte code like Flash, NekoVM



Hello World

HelloWorld.hx

```
class HelloWorld
{
    static function main()
    {
        return new HelloWorld();
    }

    public function new()
    {
        trace("Hello World!");
    }
}
```

```
haxe -x HelloWorld
```

Hello [JavaScript] World

```
haxe -main -js HelloWorld.js
```

index.html

```
<html>
<head>
    <title>Hello World</title>

</head>
<body>
<div id="haxe:trace"></div>
<script type="text/javascript" src="HelloWorld.js"></script>
</body>
</html>
```

Hello [Multi-Platform] World

build.hxml

```
# js
-main HelloWorld
-js HelloWorld.js

--next
# swf 9
-main HelloWorld
-swf9 HelloWorld_9.swf

--next
# swf 8
-main HelloWorld
-swf HelloWorld_8.swf

--next
# php
-main HelloWorld
-php build/php
-cmd echo "----- PHP-----"
-cmd php build/php/index.php

--next
# neko
-main HelloWorld
-neko build/HelloWorld.n
-cmd echo "----- NEKO-----"
-cmd neko build/HelloWorld.n

--next
# c++
-main HelloWorld
-cpp build/cpp
-cmd echo
-cmd echo "----- CPP-----"
-cmd build/cpp/HelloWorld
```

```
haxe build.hxml
```

What HaXe has to offer

- **Single Language and syntax** enabled across all platforms
- **Standard cross platform libraries** work across all client and server targets (like Date, Xml, Math, Http, etc)
- **Access full target platform APIs** (JavaScript, Flash, etc)
- **Interface with existing platform libraries** (e.g. JQuery, Raphael, SWCs)
- **Conditional compilation** flags for targeting specific platforms (#if js ….)
- **3rd Party HaXe libraries** via HaxeLib (terrific open source community)

What HaXe has to offer JavaScript Developers?

- **Compile time checking** of your javascript code
- Advanced language features allow for **beautiful, readable code**
- **Single syntax** for client, server and command line applications
- Targeting new platforms is just looking up additional APIs, rather than learning entirely new languages
- Admittedly, the generated JavaScript is fast but not always pretty… use **macros** to easily extend or replace the default JS generator (using haXe of course!)

Cross Platform APIs

```
var string = new String("foo");
var bool = new Bool();
var date = new Date();

var array = ["a", "b", "c"];

for(s in array)
{
    trace(s);
}

var xml = Xml.parse("<foo id="bar" />");

var o:Dynamic = {};

var http:haxe.Http = new haxe.Http("http://www.example.org/
foo.bar");
```

Platform Specific APIs

Example.hx

```
#if js
import js.Dom;
#else if flash
import flash.display.Sprite;
#end

class Example
{
    public function new()
    {
        #if js
            js.Lib.alert("foo");
        #elseif flash
            flash.Lib.trace("foo");
        #else
            trace("foo");
        #end
    }
}
```

Leveraging 3rd Party APIs

- Use **Extern Classes** to define functions in external JavaScript libraries (or any other language)
- Compile time type checking on non typed runtime JavaScript APIs
- Many popular JS libraries already already have externs out there
  - e.g. jQuery, raphael, nodeJS, google-js
- Check out standard HaXe JS classes as examples

Foo.hx

```
extern class Foo
{
    public function new(id:String):Void;
    public function bar():Void;
    public var visible(default, null):Bool
}
```

Using HaxeLib Libraries

```
haxelib install jeash
haxelib install nme
haxelib install hxcpp
```

build.hxml

```
#flash
-main Example
-swf9 Example.swf

--next

# js
--remap flash:jeash
-main Example
-lib jeash
-js Example.js
```

```
--next

# cpp
--remap flash:nme
-main Example
-lib nme
-lib hxcpp
-cpp example-cpp


-cmd example-cpp/Example
```

Multi Platform UI

Example.hx

```
import flash.display.Shape;
import flash.display.Sprite;
import flash.Lib;

class Example extends Example
{
	public static function main()
	{
		var example = new Example ();
		Lib.current.addChild(example);
	}

	public function new()
	{
		super();
		var circle:Shape = new Shape( );
		circle.graphics.beginFill( 0x7A9AA9 , 1 );
		circle.graphics.drawCircle( 0 , 0 , 40 );
		addChild(circle);
	}
```

Cross Platform Considerations

- Libraries like Jeash and NME make it simple to target multiple platforms using a single API
- But sometimes the flash API is overkill, and each library implements different subsets of it
- Your application will still require plenty of conditional compilation and package remapping
- The larger the project the more complex it is to manage cross platform implementations

Next Steps

- Download HaXe from http://haxe.org
- Browse through http://haxe.org/haxelib
- Subscribe to the HaXe mailing list (http://lists.motion-twin.com/mailman/listinfo/haxe)
- Enable your IDEs (http://haxe.org/com/ide)
- Follow people on twitter (@skial @ncannasse @DavidPeek @andy_li)
- Start coding!

What we've been up to!

- Raxe (multi-platform build/release tool)
- MUnit (cross-platform unit testing framework/runner)
- Sublime Text Plugins (auto-completion, raxe integration)
- Massivision UI Framework

Thank You

Any Questions?

Massive is currently hiring: www.massiveinteractive.com/hiring/UIDeveloper

Follow Us on Twitter:

@DavidPeek @DeanBurge @misprintt