haXe

A Web-oriented
Programming Language

OSCON 2006
Nicolas Cannasse
ncannasse@motion-twin.com

http://haxe.org

First Rule

haXe == Simple

What is haXe ?

It's simple !

Crossplatform : Linux, Windows, OSX, BSD

Multiplatform : JS, Neko, SWF

Easy to get started : let's do it

Hello World

```
class Hello {
	static function main() {
		trace("Hello World !");
	}
}
```

Outputs

Javascript : hello.js

Flash SWF : hello.swf

Neko bytecode : hello.n

Hello World

Let's try it !

haXe == Simple

One single language for your whole website...

Simplify the whole development process.

...if you need it.

haXe == Simple

One standard crossplatform library

+ Platform-specific libraries :

(root package) : core library

js.* / flash.* / neko.* : platform-specific libraries

haxe.* : standard library components, crossplatform

haXe == Simple

Created to simplify your life

Anything complicated ?

...fill a bug-report

Simple does not mean featureless

Second Rule

haXe == Powerful

haXe == Powerful

QUESTION :

« What is your favorite programming language »

haXe == Powerful

Dynamicly typed ? This is good

Some people like staticly typed

They have some points too...

- typos, documentation, refactoring...

But people don't like to write...

haXe == Powerful

```
public static HashMap attributesHash( String xmlData ) {
    try {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        Document doc = factory.newDocumentBuilder().parse(xmlData);
        Element element = doc.getElementById("id");
        NamedNodeMap attrs = element.getAttributes();
        int numAttrs = attrs.getLength();
        HashMap hattribs = new HashMap();
        for(int i=0; i<numAttrs; i++) {
            Attr attr = (Attr)attrs.item(i);
            String attrName = attr.getNodeName();
            String attrValue = attr.getNodeValue();
            h.put(attrName,attrValue);
        }
        return hattribs;
    } catch (SAXException e) {
    } catch (ParserConfigurationException e) {
    } catch (IOException e) {
    }
    return null;

}
```

haXe == Powerful

You would prefer instead :

```
function attributesHash( xml ) {
    var x = Xml.parse(xml);
    var h = new Hash();
    for( att in x.attributes() )
        h.set(att,x.get(att));
    return h;
}
```

haXe == Powerful

Everybody can understand why...

But « what if.... » ?

« Best of both worlds ? »

haXe == Powerful

Let's play a game...

```
var x = "Hello";

var y = x.indexOf("l");

var z = Math.sqrt(y);
```

haXe == Powerful

Back to the sample :

```
function attributesHash( xml ) {
    var x = Xml.parse(xml);
    var h = new Hash();
    for( att in x.attributes() )
        h.set(att,x.get(att));
    return h;
}
```

haXe == Powerful

Back to the sample :

```
function attributesHash( xml : String ) {
    var x = Xml.parse(xml);
    var h = new Hash();
    for( att in x.attributes() )
        h.set(att,x.get(att));
    return h;
}
```

haXe == Powerful

Back to the sample :

```
function attributesHash( xml : String ) {
    var x : Xml = Xml.parse(xml);
    var h = new Hash();
    for( att in x.attributes() )
        h.set(att,x.get(att));
    return h;
}
```

haXe == Powerful

Back to the sample :

```
function attributesHash( xml : String ) {
    var x : Xml = Xml.parse(xml);
    var h : Hash<???> = new Hash();
    for( att in x.attributes() )
        h.set(att,x.get(att));
    return h;
}
```

haXe == Powerful

Back to the sample :

```
function attributesHash( xml : String ) {
    var x : Xml = Xml.parse(xml);
    var h : Hash<???> = new Hash();
    for( att in x.attributes() )
        h.set(att,x.get(att));
    return h;
}
```

haXe == Powerful

Back to the sample :

```
function attributesHash( xml : String ) {
    var x : Xml = Xml.parse(xml);
    var h : Hash<???> = new Hash();
    for( att : String in x.attributes() )
        h.set(att,x.get(att));
    return h : Hash<String>;
}
```

haXe == Powerful

It's called « type inference »

It works with haXe !

One requirement :

declare and type you object member variables

Anonymous

Are type-safe :

```
var o = { name : "John", age : 29, city : "Portland" };
trace(o.name);
trace(o.aje); // ERROR
```

Because :

o : { name : String, age : Int, city : String }

Class

Class Sample :

```
class User {
    public var name : String;
    public var age : Int;
    public function new(n,a) { name = n; age = a; }
    public function foo() { }
}
var u = new User("John",29);
```

Structural

Display Function :

```
function display(o) {
    return "My name is " + o.name;
}
```

Structural :

```
display(new User("John",29));
```

Functional

An simple function :

```
function f(x,y) {
   return "Sum"+(x + y);
}
```

Type(f) ?

```
var myf : Int -> Int -> String = f;
```

Duck Typing

```
class Duck {
    public function new( ){ }
    public function say() { trace("Coin!"); }
}
class Dog {
    public function new() { }
    public function say() { trace("Waf!"); }
}

....
    var l = new List();
    l.add(new Duck());
    l.add(new Dog());
    for( a in animals )
        a.say();
```

Duck Typing

```
class Duck {
    public function new( ){ }
    public function say() { trace("Coin!"); }
}
class Dog {
    public function new() { }
    public function say() { trace("Waf!"); }
}

....
    var l = new List();
    l.add(new Duck());
    l.add(new Dog());  // ERROR
    for( a in animals )
        a.say();
```

Duck Typing

```
class Duck {
    public function new( ){ }
    public function say() { trace("Coin!"); }
}
class Dog {
    public function new() { }
    public function say() { trace("Waf!"); }
}
....
    var l = new List<{ say : Void -> Void }>(); // WORKS
    l.add(new Duck());
    l.add(new Dog());
    for( a in animals )
        a.say();
```

Duck Typing

A small requirement...

Brings you security...

...in the case your Dog doesn't « say » anymore.

Type inference greatly helps refactoring.

Dynamic / untyped

Everything is Dynamic :

```
var o : Dynamic = "Hello";
```

Dynamic is Everything :

```
var v : User = o;  // use with care
```

Untyped :

```
untyped {
   .... // don't bother me
}
```

haXe == Powerful

haXe is strictly typed

...but you (almost) don't write types

everything is checked... unless you don't want to

the compiler is here to help you, not to bother you

Third Rule

haXe == Extensible

haXe == Extensible

« Swiss knife of the web developer »

You can use different tools available...

Like you need

NO Framework : Small Components

haXe == Extensible

Crossplatform Libraries ?

Conditional compilation :

```
#if js
    // js code
#else flash
    // flash code
#else neko
   // neko code
#end
```

An Example

From Std.hx :

```
public static function parseFloat( x : String ) : Float {
    untyped {
        #if flash
            return _global["parseFloat"](x);
        #else neko
            return __dollar__float(x.__s);
        #else js
            return __js__("parseFloat(x)");
        #else error
        #end
    }
}
```

haXe == Extensible

All the platform-specific code is written in haXe !

... using untyped and other magic

You can redefine everything... if you want

All is transparent

Getting « low-level »

Want to get « low-level » ?

In JS : __js__

In Flash : haXe syntax (plus _global)

In Neko : C libraries / Neko API

haXe == Extensible

Everything is written in haXe

You can add your own libraries

Even crossplatform

... if you need it !

Fourth Rule

haXe == Ready

Features

Powerful, top-class, compiler

- fine error reporting
- no « internal compiler error ». ever.

Great language features

- type inference / polymorphism / duck typing
- iterators / optional arguments / conditional compilation
- enums / functional types / type parameters / properties

Libraries

Standard :

- haxe.Http , serialization , remoting (RPC)
- template system
- unit testing

JS :

- complete DOM

Neko :

- mysql, sqlite, regexp, zlib, filesystem, sockets, utf8
- persistent database objects : SPOD

NekoVM

Lightweight

- 220K incl. std libary

Embeddable

- secure, easy to use API, multithread

Mod_neko

- for Apache 1.3.x and Apache 2.2.x
- allow application caching (save statics)

NekoVM

Very Fast

- x86 JIT
- low memory usage

Computer language Shootout

- recursive(7) : x2.5 over Python , x7 over PHP
- bintrees(16) : x9 over Python, x120+ over PHP

Open to other languages

- Ruby on Neko soon ?

Community

http://haxe.org Wiki

- documentation, tutorials

Different people, different usages

- RIA, Websites, Games

Enthousiasm

- Just join today and feel free to ask any question

Conclusion

haXe = {

Simple,

Powerful,

Extensible,

Ready,

... and much much more

}

EOF

QUESTIONS ?

Visit http://haxe.org