HAXE LANDS ON LUA

SHOOTING FOR THE MOON

HAXE LANDS ON LUA
WRITE ONCE, TARGET MANY
Your humble presenter

Salesforce quietly spent hundreds of millions of dollars to build a team of 175 data scientists.
Why Haxe?

Use Cases for Haxe

Because the Haxe Language can compile to many different platforms, it is useful in a wide variety of domains. Take a look at who is using Haxe, or explore some of the use cases below:

Games
Haxe is popular with game creators because it is fast, has many useful libraries, and can target iOS, Android, Web and Desktop easily.
» Haxe for Game Development

Web
Haxe gives you a powerful, type-safe language that can target JavaScript on the client and PHP, NodeJS or Neko on the server. Share code and APIs between the client and server seamlessly.
» Haxe for Web Development

Mobile
Share code between key platforms. Access native functionality without sacrificing performance.
» Haxe for Mobile Development

Desktop
Build cross platform desktop apps using WX Widgets, Node Webkit, Java Swing or custom UI libraries.
» Haxe for Desktop Development

Command Line
Take advantage of easy-to-use libraries to write powerful, cross platform CLI applications.
» Haxe for CLI Development

Cross-Platform APIs
Write cross platform APIs in Haxe that can be exported and shared with other languages and environments.
» Haxe for API Development

http://haxe.org/use-cases/
Why Haxe?

4123  Command exited with 0 in 27s: haxe [compile-java.hxml,-D,travis]
4124  Command: java [-jar,bin/java/TestMain-Debug.jar]
4125  TestMain.hx:36: Generated at: 2016-10-10 11:47:57
4126  TestMain.hx:38: START
4127  Test.hx:220: DONE [7511 tests]
4128  Test.hx:221: SUCCESS: true

6672  Command exited with 0 in 3s: haxe [compile-lua.hxml,-D,travis]
6673  Command: lua [bin/unit.lua]
6674  TestMain.hx:36: Generated at: 2016-10-10 11:54:45
6675  TestMain.hx:38: START
6676  Test.hx:220: DONE [6838 tests]
6677  Test.hx:221: SUCCESS: true
A Taste of Haxe

```haxe
class Test {
    static function main() {
        var people = [
            "Elizabeth" => "Programming",
            "Joel" => "Design"
        ];
        for (name in people.keys()) {
            var job = people[name];
            trace('{$name does $job for a living!}');
        }
    }
}
```

```bash
$> haxe -main Test -lua out.lua
```
Haxe Features

- Abstract Types
- Anonymous Types
- Array Comprehension
- Classes, Interfaces, and Inheritance
- Conditional Compilation
- (Generalized) Algebraic Data Types
- Inlined Calls
- Iterators
- Local functions and closures
- Metadata
- Static Extensions

- String Interpolation
- Partial function application
- Pattern matching
- Properties
- Type parameters, constraints, variance
- Reflection
- AST macros
- Static Analysis
  - Const propagation
  - Copy propagation
  - Local dead code elimination
  - Fusion
  - Purity Inference

https://haxe.org/documentation/introduction/language-features.html
Why Haxe and Lua?

1. Why not?
2. LuaJit (Nginx, Torch, etc.)
3. Scripting for editors (Neovim, vim)
4. Scripting for games (WoW, Factorio)
5. Community match (game + webdev)
6. Boredom/Hubris
Which Lua?

1. Lua 5.1
2. Lua 5.2
3. LuaJit 2.0
4. LuaJit 2.1
5. Lua 5.3*

* Partial support backwards compatibility flags

NEW! Announcement!
Related Work

1. [Unfinished Lua target](#) by Russel Weir (2008) - Partial support for Lua 5.1 in Haxe 2
2. [hx-lua](#) by Matt Tuttle (2012) - Run Lua code inside C++/Neko targets
3. [LuaXe](#) by Peyty (2014) - Partial support for Lua 5.1 in Haxe 3 as a custom javascript target*
4. [hxpico8](#) by Vadim Dyachenko (2015) - Run an experimental/limited version of Lua for a virtual console.
5. [linc-luajit](#) by RudenkoArts (2016) - @:native bindings for hxcpp/linc
6. [A Comparison of Neko and Lua](#) by Nicolas Canasse

* Peyty/Oleg provided much needed support and ideas for this project, thanks!
Hello World

- Simple main()
- Trace == print
- All classes local
- Objects use special _hx_o helper
- __name__ for reflection
BitOps

- Bit operators turn into bit methods
- `var =~ local`
Unops

- Transform unary operators to one or more statements
Extern

- `@:native` binds to native or non-conformingly named interface
- `@:expose` binds class/method body to global metatable
- `@:selfCall` allows methods to call the module/class name as a function
- `includeFile` adds helper methods in lua
Extern

- `@:multiReturn` allows specification of extern-only classes that represent multiple returns.
NEW!!! Announcement!
Still some kinks to work out

- Cannot declare more than 200 local variables in single scope
- Sys api is incomplete*
- Null (nil) in string concatenation throws errors

* Progress on libuv/luv backend
Avoiding Pain And Humiliation

- Don’t use more than 200 local variables (even when workaround is in place).
  - Avoid abstracts/inlines that result in temporary variable creation
- Avoid assigning instance/static methods unnecessarily (e.g. dynamic methods or as fields).
- Avoid using “Lua.arg” or “haxe.extern.Rest” (defeating jit optimizations)
- Use unique variable names in any lua include/__init__ code.
Haxe Love

- Love-haxe-wrappergen
- Released ~24 hours after official Haxe Lua announcement

This project uses the awesome love-api project, which provides a lua tables representation of the love documentation, to generate Haxe wrappers. To use this project, make sure to checkout the submodule (`git submodule update --init love-api`).

Fair warning, the code is awful, and full of hacks. Oh yeah, and unless you have `find . -p`, it won’t run. Look, it was easy.
Nginhx

https://github.com/jdonaldson/nginhx
HaxeCraft

https://github.com/jdonaldson/haxecraft
How to get started

2. Haxe cookbook: http://code.haxe.org/
3. Haxe mailing list: https://groups.google.com/forum/#!forum/haxelang
4. Haxe discord group: https://discord.gg/znfNW
5. Haxe IRC: (freenode #haxe) http://webchat.freenode.net/?channels=haxe
Recap/Conclusion

- Haxe and Lua communities are similar: creative, independent, mindful
  - (even though languages are different)
- Haxe as a language is very “standard”
  - EcmaScript based, multi paradigm language
- Haxe provides a way to leverage an existing language ecosystem, while expanding towards other targets/platforms.
  - You don’t leave the Lua community by joining the Haxe community
- Haxe avoids impedance mismatch by supporting target specific extern features (e.g. `@:multiReturn`)
- Haxe provides convenient and powerful static typing features on dynamic languages
  - I’ve learned more from Haxe than any other programming language community
THE END!
QUESTIONS?

jdonaldson@gmail.com
twitter @omgjjd