whither eero?

Andy Arvanitis

What is Eero?

- A dialect of Objective-C
    - Streamlined syntax
    - Binary- and header-compatible with Objective-C
    - Built using a fork of LLVM/Clang

Why?

- Readability matters!
  - You read code many more times than you write it
  - Even your own code looks foreign to you over time
  - Easier to read => easier to maintain

What kind of name is that?

- Finnish. It’s pronounced like “aero.”
- Named after Eero Saarinen, the Finnish-American architect and industrial designer

Inspiration — the Tulip chair

- He wanted to clear up the “slum of legs”
- I wanted to clear up the “slum of braces”

An example

```
import Foundation

int main()
  parts := ['hello', ' ', 'world']
  greeting := ''

  for String part in parts
    locale := Locale.currentLocale
    greeting << part.capitalizedStringWithLocale:locale

  Log('%@', greeting + '!')

  return 0
```

Key features

- Offside rule
  - Sort of like Python, but also Haskell, et al.
  - You already indent, don’t you?
  - WYSIWYG

Key features

- Local type inference
    - ‘:=’ operator (like Google Go)
    - Like C++11’s ‘auto’ keyword

```
counter := 0  // infers an int
const title := ”hello, world”
```

Key features

- Dot notation everywhere
  - Consistency
  - End the debate!
  - Started off like Smalltalk (object-space-message), but didn't play nice with C heritage

Key features

- Hardly any semicolons are needed
    - Tired of the compiler saying, "Hey, you missed a semicolon here"?
    - If the machine can figure it out, let <u>it</u> do the work

Key features

- Pseudo namespaces, based on prefixes
  - "NS" is built-in
  - Can also be user-defined

```
String s = "hello, world"

using prefix AB
addressBook := AddressBook.sharedAddressBook
```

Key features

- Fewer @s — keywords, string and object literals
- Objects are always pointers
- Optional parentheses around conditions

Key features

- Compact blocks (lambdas)
- Concise method declarations
- Optional and default parameters
- And…operator overloading

Example: fun with blocks

```
// A normal (multiline) block

foo.doThingWithBlock: (id object)
  Log(‘Object is %@', object)
  object.doSomething


// A compact block

bar.map:(String s | return s.lowercaseString)
```

C++ interoperability

- With a few restrictions, you can use C++ in your Eero code

```
#pragma eero "C++"

#import <string>
#import <iostream>

int main()
  std::string str = "abc"
  for char c in str                      // unified for-in syntax
    std::cout << c << std::endl
  return 0
```

C++ interoperability

- Unified C++ and ObjC exception handling

```
try
  str.substr(1000, 1001) // C++ exception will be thrown

catch std::exception& e  // handle STL exceptions
  Log('caught C++ exception: %s', e.what())

catch Exception e  // handle objc exceptions
  Log('Caught %@ => %@', e.name, e.reason)

catch ... // handle exceptions of any sort
  Log('caught "...!"')
```

Anything new?

- Adding a "pipe-period" operator
    - Message passing with low precedence
    - Allows concise message chaining

```
desc := Array.arrayWithArray:myList |. description

length := doThingWithObject:foo|.description|.length
```

Anything new?

- Ideas under consideration
  - Blocks not capturing ‘self’ by default
  - Type-safe collections?
  - GCD syntax?

Tools

- It’s a fork of modern clang, so you get its goodies
    - ARC
    - Static analyzer
    - LLVM optimizations
    - Platforms (sadly, no arm64 yet)
    - Eero-to-Objective-C translation

Tools

- Xcode plugin
  - Get it through Alcatraz!
- Command-line binaries available via MacPorts
- Build it yourself from source

Volunteers welcome!

- Until now, it's been mostly me working on it
- Everything's on github:
  - github.com/eerolanguage
- Warning: it's mostly a C++ project (except for the Xcode plugin)
  - But learning LLVM can be useful