

# THE COBRA PROGRAMMING LANGUAGE

San Diego .NET User Group

June 2009

[cobra-language.com](http://cobra-language.com)

# YOUR SPEAKER



- Chuck Nor^H^H^H Esterbrook
- AKA “Cobra Commander”
- Independent contractor / consultant
- Based in Los Angeles
- [Chuck.Esterbrook@gmail.com](mailto:Chuck.Esterbrook@gmail.com)



# INTRO

- Cobra is a fairly new language (sub 1.0)
- Object-oriented, imperative
- Embraces unit tests, contracts and more
- General purpose. Open source.
- Runs on .NET & Mono. JVM later this year
- Windows, Mac, Linux, Solaris, etc.

# WHY?

- It's a **HUGE** amount of work to create a language
- Especially one with a rich feature set
- So why do it?

# MOTIVATION

- Clean, expressive syntax (Python, Ruby)
- Run-time performance (C#, C++)
- Static and dynamic typing (Objective-C, VB)
- Contracts (Eiffel, Spec#)
- Nil tracking (Spec#, iihtdioa.C#)
- *Productivity boosters are scattered across languages*
- Not mutually exclusive! Yet, must decide per project.

# GET IT ALL

- Clean, expressive syntax (Cobra, Python, Ruby)
- Run-time performance (Cobra, C#, C++)
- Static and dynamic typing (Cobra, Objective-C, VB)
- Contracts (Cobra, Eiffel, Spec#)
- Nil tracking (Cobra, Spec#)
- *Now in one place: Cobra*
- Goal is maximum productivity

# INFLUENCES

- The “Big Four”
  - Python, C#, Eiffel, Objective-C
- Others
  - Visual Basic, D, Boo, Smalltalk
- Originally conceived of as a cross between Python and Objective-C

- *show code* -

# NO NIL UNLESS I SAY SO

- Problems:
  - `NullPointerException` happen *one at a time at run-time*
  - Methods don't indicate if they return or accept it
- `def nodeFor(name as String) as Node?`
- `def nodeFor(name as String?) as Node?`
- Compile-time detection happens *many times at compile-time*

- show code -

Anders H, C#, iihtdioa...



# SQUEAKY CLEAN SYNTAX

- Python-like
- Light on symbols; strong on indentation, keywords
- list literals, dict literals, set literals
- in / not in, is vs. ==
- But even cleaner!
  - Straight forward properties
  - Other tweaks. Ex: `/# ... #/` comments

- show code -

# DYNAMIC OR STATIC? BOTH!

- Programmers should choose, not language designers
- Objective-C has been doing it for ~20 years  
Others include Visual Basic and Boo. Upcoming C#
- `def add(a as int, b as int) as int`
- `def add(a, b) as dynamic`
- There are pros and cons to both
- Don't have to switch languages to switch approaches

# DYNAMIC IS CLEARLY BEST!

- ```
def add(a, b) as dynamic  
  return a + b
```
- Flexible
- Fast coding and prototyping
- Less brittle w.r.t. changes
- More reusable

# STATIC IS CLEARLY BEST!

- `def nodeFor(name as String) as INode?`
- Compile-time detection of errors
- Multiple errors reported at once
- Fast at run-time
- Slim too (no boxing)
- Easy Intellisense. More self-documenting.

- *show code* -

# PERFORMANCE

- Performance can be very important
- ... financial analysis, video games, compilers, AI, ...
- Performance can *become* important
  - Yahoo Mail: Python, then C++
  - AI company: Ruby prototype, then C++
- Cobra compiles and leans towards static (~C#/Java)
- “i = 5” infers “i” as an “int”

# SCRIPTING CONVENIENCE

- Compile and run in one command:  
    > cobra foo.cobra
- #! line on Unix-like systems
- Clean syntax is a hallmark of *some* scripting languages
- Dynamic binding is a hallmark of scripting languages



# CONTRACTS

- ```
def nodeFor(name as String) as INode?  
  require name.length  
  ensure  
    result.name.toLowerCase == name.toLowerCase
```

...

- Supports invariant, old, result and implies
- Inheritance works
- Eiffel-style: the “real thing”
- Future? Integrate with Spec# backend

- *show code* -

# UNIT TESTS

- `def capped(s as String) as String is shared test`

```
    assert Utils.capped('aoeu') == 'Aoeu'  
    assert Utils.capped('') == ''  
    expect NullPointerException  
        Utils.capped(nil) # ahem  
body  
...
```
- Same motivations as doc strings:  
 localized, encourage use, get people on same page

- *show code* -



# MIX-INS ARE NEW

- Break out of single inheritance
- VM does not support so this is all compile-time magic
- + Save time
- + Reduce coding
- + Catch errors
- “Alpha” feature - not complete

# ACCURATE MATH ALREADY

- 0.1 added ten times is what?  
In most languages: not 1.0!
- Python:

```
>>> .1+.1+.1+.1+.1+.1+.1+.1+.1+.1
0.999999999999999999999989
>>> assert 1.0 == .1+.1+.1+.1+.1+.1+.1+.1+.1+.1
AssertionError
```
- Cobra supports both decimal and float (64/32-bit)
- Defaults to decimal because it's 2009 for Turing's sake

# CHANGE DEFAULT

- With *-number* option, you can choose float64 or float32 instead
- *number* is a built-in type that represents this default

```
def add(a as number, b as number) as number  
    return a + b
```

- I rarely use *decimal*, *float* or *float32* anymore.

- show code -

# INTEGRATIONS

- Today
  - Various editors (see wiki)
  - Any .NET tool for byte code: profilers, analysis, obfuscation, etc.
  - Reflector, Nant, Pygments
- Tomorrow
  - MSBuild, Visual Studio, DLR, MS Contracts, Pex

# VEND TO C# AND VB

- You can vend class libraries to C# and VB, both technically and practically.
- Super-C# features like non-nil degrade gracefully
- Technically: .NET/Mono DLLs and CLI-style classes
- Practically
  - Cobra favors .NETisms like generic lists
  - Can embed Cobra run-time (avoid Cobra.Lang.dll)

# THEME: CODER'S CHOICE

- This is in keeping with the “coder’s choice” theme:
  - Choose static or dynamic
  - Choose default numeric representation
  - Unit tests or not
  - Contracts or not
  - In the future: .NET, JVM or Obj-C

# THEME: QDD

- Quality Driven Development  
(because we're do for another XDD)
- Doc Strings
- Unit Tests
- Nil/Null Tracking
- Assertions
- Contracts

# THEME: PRODUCTIVITY

- Better error checking => Fewer trips to run-time
- Static and Dynamic => Flexibility
- Unit tests and Contracts => Specify what's easy
- Clean syntax => Fast to read, write and maintain
- Note: Concerned with medium+ sized programs.



# THEME: PLAYS NICE

- Consumes other binaries (.dll, .exe, .class) with no extra steps
- Uses standard library classes like List<>, Dictionary<>, etc.
- Produces VM-standard binaries/byte-code that can be consumed by other languages (C#, VB, Java, etc.)

# THE COMPILER

- Self-implemented a.k.a “self-hosted”
- Usual phases:  
tokenize, parse, AST nodes, analysis, code gen
- Something different: chose C# as backend over IL
  - Growing number of “super-VM” features in C#
  - Faster implementation
  - Piggy back on error checking and cmd line options

- *show code* -

# OPEN SOURCE FTW

- MIT license
- Typical pros: contribs, transparency, early access to new fixes and features, cannot disappear on you
- Typical cons: um, any cons?  
maybe: no full-timers on this project
- self hosted + open source = you can read compiler!
- install-from-workspace
- Discussion boards, Wiki, Tickets, Subversion

# WEAKNESSES

- Mix-ins feature not ready yet
- JVM back-end not done yet
- No IDE plug-ins, but we do have editor plug-ins.
- No interactive prompt

# COMPARED TO PYTHON

- Best place: <http://cobra-language.com/docs/python/>
- Better error checking, Compile-time nil tracking
- First class contracts and unit tests
- Speed, Default to accurate math
- Syntax, Self-hosted
- Disadvantages: Maturity, Docs, Less malleable

# ONGOING WORK

- JVM back-end
- Always refinements and fixes
- Apply patches
- Monthly updates
- Next release: 0.9
  - Should be close to final feature set and syntax of Cobra 1.0

# COMMERCIALISM

- In 2007 Q3+Q4, I worked full time on Cobra.  
Paid rent with savings (and a poker tournament).
- In 2008, return to contracting.  
Less time for Cobra. :-)
- Ideas:
  - IDE or VS plug-in, Book
  - App Server, Web Ads
- Bad idea: Corporate sponsors

# FUTURE FEATURES

- Context: Be the best, most productive, high-level, general-purpose OO language. Be popular.
- JVM, Objective-C, Python?, Parrot?
- Full LINQ
- traits / subtypes ...
- DLR integration
- Language level reg-ex (maybe)



# MORE FUTURE FEATURES

- More sophisticated unit test features
- Units of measurement (feet, meters, ...)
- Compile-time analysis of contracts

```
def foo(thing)
  require
    thing responds to (get name as String)
```

# THE FAR FUTURE

- Parallel programming
- Futures / lazy arguments
- Macros?
- Would be nice to leverage .NET advances as with generics, LINQ, etc.

# THE FAR, FAR FUTURE

- Cobra has compile-time nil tracking and contracts
- Microsoft has Pex and Spec# / Boogie
- Could we eventually get here:
  - Detect all technical errors at compile-time in  $< 5$  secs
  - Leave slower run-time tests and round-tripping to domain logic issues only

# JOIN THE FUN

- You can help!
- Participate in the forums, wiki and issue tickets
- Write sample code
- **Blog**, discuss, write
- Write a cool app or library
- Patch the open source compiler

# FIN

- [cobra-language.com](http://cobra-language.com)
- [cobra-language.com/docs/why](http://cobra-language.com/docs/why)
- [cobra-language.com/docs/python](http://cobra-language.com/docs/python)
- Sample programs, How To, Documentation, Forums
- [cobralang.blogspot.com](http://cobralang.blogspot.com)
- <http://cobra-language.com/docs/contact/>