Rogue Update
May 2018

Abe Pralle
May 2, 2018

Overview

About Rogue

- Language Designers
- Rogue Info

New Rogue Features

- Bootstrapping (Windows, Mac, Linux)
- VS Code Extension
- Rogo build system
- Introspection
- Miscellaneous new syntax
- Multithreading
- New Python extension support
- Modules

Little Languages

- ParseKit
- BitCalc Demo

Rogue Language Designers

Abe Pralle

- Created Rogue in 2015
- Evolution of earlier language projects dating back to 2004
- Ad tech developer at AppOnboard
- Indie game developer (Runegate, Plasmaworks)

Programming Interests

- Games, languages, APIs

Contact

- abe.pralle@gmail.com

Rogue Language Designers

Murphy McCauley

- Frequent collaborator & consultant
- Joined Rogue project in 2016
- Ph.D. student at Berkeley (Computer Science, 2018)
- Programmed "SENSE" packet-level network simulator in Rogue and Python

Major Contributions to Rogue

Automatic garbage collection, multithreading, two Python extension generators, tuples, method template type inference by parameter types, core Windows compatibility, the *Rose* prototype language, and the book-in-progress "Hacking with Rogue"

Rogue

A Powerful Language

- Elegant, efficient, full-featured, object-oriented, ergonomic

C++ Support

- Cross-compiles to and easily interoperates with C++
- A viable alternative or companion to C++

Links

- github.com/AbePralle/Rogue
- See Wiki for documentation

Bootstrapping

Windows

- Install Visual Studio 2017 with C++ support
- Install Git
- Clone Rogue repo: github.com/AbePralle/Rogue
- Always use Visual Studio Developer Command Prompt when working with Rogue
- From VS Dev Cmd Prompt:

```
> cd <Rogue Folder>
> rogo
```

- Add absolute path of <Rogue>\Programs\RogueC to PATH as prompted

Mac & Linux

- Clone Rogue repo: github.com/AbePralle/Rogue
- From Terminal:

```
> make
```

Visual Studio Code Extension

About

- Work-in-progress
- Syntax highlighting but no auto-indent (yet)

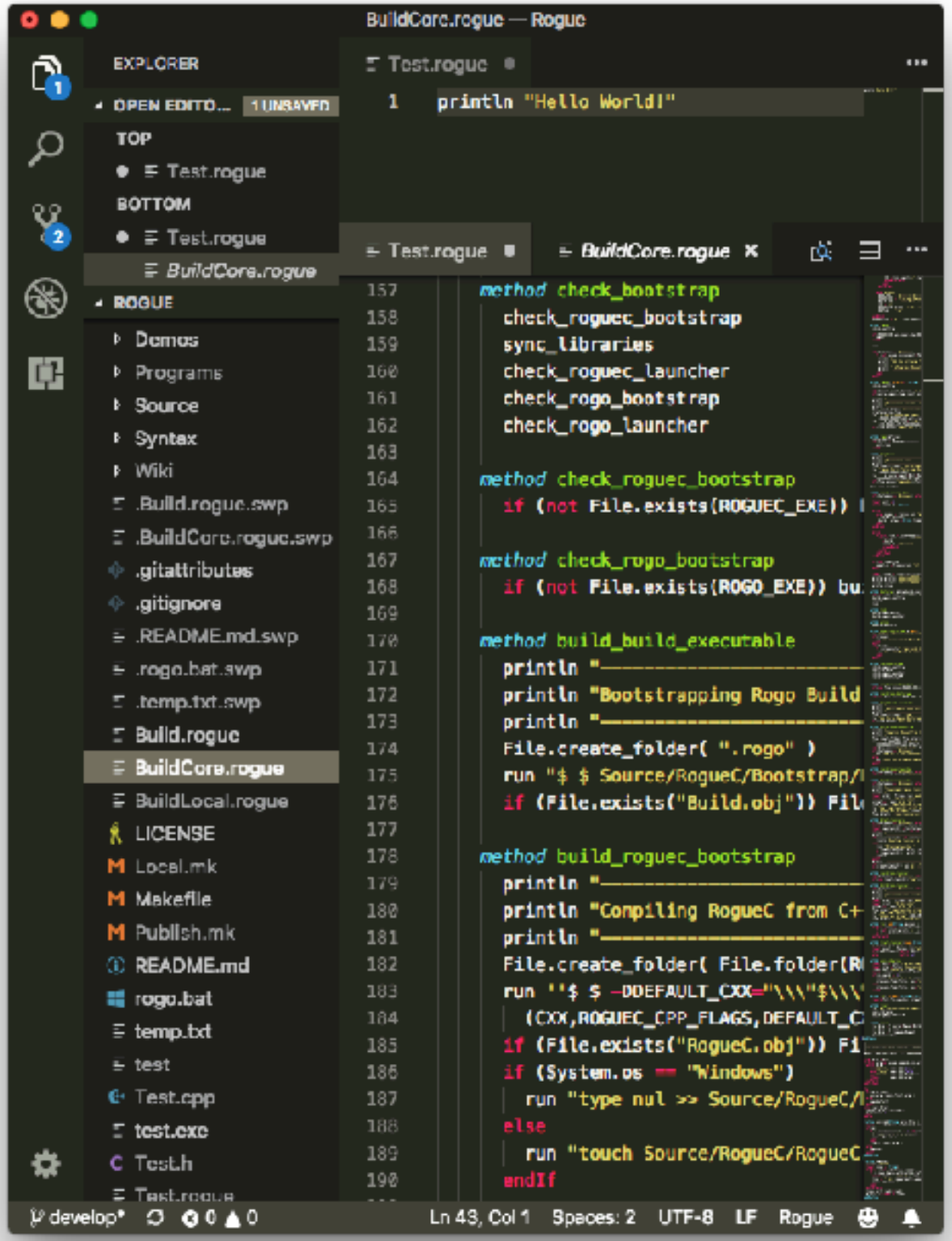
Installation

- Install VS Code

```
> cd <Rogue>
> rogo vscode
```

Uninstall

```
> cd <Rogue>
> rogo vscode uninstall
```



Rogo Build System

Overview

- Very simple concept
- Programmatic build system
- Rogo automatically compiles & runs build files written in Rogue, forwarding command line arguments
- Build files written entirely in Rogue with no restrictions on syntax
- Rogue repo Build files use a simple introspection-based framework to invoke routines based on command line arguments

Usage

- Run `rogo` in any folder
- Rogo looks for any or all of the following three build files:
  - Build.rogue
  - BuildCore.rogue
  - BuildLocal.rogue
- Any build files found are compiled together and executed with original command line arguments
- Executes Build without recompiling if no changes to Build files

Advanced Rogo

Comment Directives

- In Build files, lines beginning with #$ are treated as Rogo directives to control compilation of Build files

```
#$ ROGUEC       = roguec
#$ ROGUEC_ARGS = --whatever
#$ CPP          = g++ -Wall -std=gnu++11 ...
#$ CPP_ARGS     = -a -b -c
#$ LIBRARIES    = libalpha( header:"name.h" library:"libname.a" )
#$ LIBRARIES    = libbeta
#$ LINK         = -lalpha -lbeta
```

- Can define default option that applies to all platforms and then customize per-platform as follows:

```
#$ CPP(Windows) = cl /EHsc /nologo
```

- Certain escape sequences are replaced while compiling Build

```
$HEADER(library-name)  -> path/to/library-name/include
$LIBRARY(library-name) -> path/to/library-name/lib
```

Introspection

Overview

- AKA Reflection
- Uses Runtime Type Info generated by RogueC
- Basic approach is to use JSON-style "Value System" to make calls and access properties
- Template and pointer-based introspection methods available to avoid memory allocation overhead

Culling, [essential], and [api]

- By default RogueC culls unused classes and methods
- Add class attribute [essential] to keep class even if unused
- Add method attribute [api] to keep method if class is kept
- Add "METHODS [api]" to keep all methods in section if class is kept
- Add class attribute [api] to make all methods in class as [api] methods and keep them if class is kept

TypeInfo

TypeInfo

- Rogue-side class that holds RTTI for a specific type
- Basic mechanism for Introspection
- Retrieve TypeInfo for some object 'obj':
  local type = obj.type_info
- Get TypeInfo programmatically:
  local type = TypeInfo[name]
- Get TypeInfo for class String:
  local type = <<String>>
- Demonstration: Rogue Build file introspection mechanism

TypeInfo

PROPERTIES

```
name                : String
global_properties   : PropertyInfo[]
global_methods      : MethodInfo[]
properties          : PropertyInfo[]
methods             : MethodInfo[]
```

METHODS

```
call(Object,String,Value)->Value
create_object->Object
create_object<<$AsType>> -> $AsType
find_property(String)->PropertyInfo[]
find_method(String)->MethodInfo[]
property<<$T>>(Object,String)->$T
set_property<<$T>>(Obj,Str,$T)
...
```

use/endUse

use/endUse

- Control structure to acquire a resource from a provider for the duration of the *use* scope and automatically release it when the scope is exited via escape, return, or thrown exception

Acquisition Syntax

- use [resource =] provider
- ... # optional commands:
- escapeUse
- return resource
- throw Exception(...)
- endUse

Provider Protocol

- Implement the following two methods:
- on_use->ResourceType
- on_end_use(...)->Exception
- on_end_use args are any or all:
  - res:ResType # from on_use
  - ex:Exception # if thrown
- Returning an exception will throw it, return null to suppress

Macros

$macro

- Similar to C macros
- Can overload by number of parameters
- $macro name(a,b) a+b
- $macro name(a,b)
  multi-line stuff with a & b
  $endMacro

$localMacro

- Like $macro but exists from point of definition or until end of scope
- End of scope for a local macro is end of file or end of metablock
- $block ... $endBlock defines a metablock to contain local macros
- $localDefine also supported

Enums

Example

```
enum Color(hex="000":String)
  CATEGORIES # optional
    RED("F00")
    GREEN("0F0") = 2
    BLACK
endEnum

local c = Color.GREEN
# c is an Int32
println c->Int32  # 2
println c->String # GREEN
println c.hex  # 0F0
println Color.names
# ["RED","GREEN","BLACK"]
println Color.values #[0,2,3]
println Color(3)  # BLACK
```

Description

- enums are fast, efficient integers underneath
- Enumeration categories can be given various immutable properties
- Properties are actually retrieved via call to global method
- Color.RED.hex is internally translated to Color.hex(Color.RED) -> Color.hex(0) and a switch returns the correct string

Tuples

Example

```
local t = (5,"Five")
println t      # (5,Five)
println t._1   # 5
println t._2   # Five
t = adjust( t )
local value    : Int32
local name     : String
(value,name)   = t
println value # 55
println name  # Five!

routine adjust(
    t:(Int32,String) )
    ->(Int32,String)
  return ...
    (t._1*10+t._1, t._2+'!')
endRoutine
```

Description

- A tuple is an ad-hoc container class
- No separate class definition required (automatically generated)
- A tuple's type name is a parenthesized list of its element types
- Tuples may have any number of elements
- Destructuring assignment supported

Miscellaneous Syntax

?:{...}

- select{...} is Rogue's decision/conditional/ternary operator
- ?:{...} is shorthand for select{...}
- Styled after the syntax for C's decision operator
- The following are equivalent:
  - println select{ n<0:"negative" || n>0:"positive" || "zero" }
  - println ?:{ n<0:"negative" || n>0:"positive" || "zero" }

#FIXME

- Any lines with comments containing the word FIXME (in all-caps) are automatically printed out to the console during compilation
- Compile with --todo to also print out lines with TODO comments
- Compile with --todo="KEYWORD" to print lines with arbitrary keywords

Miscellaneous Syntax

try-expression

- Syntax

  result = try alpha else beta

- Equivalent to

```
try
  result = alpha
catch (Exception)
  result = beta
endTry
```

Metacode

Description

- Metacode is Rogue code that is executed by the compiler as it is compiling the source file containing the metacode
- Can run "inline" (instantly) or hook into a later phase o the compile process
- Uses the same parsing engine and AST classes as Rogue plus introspection
- Inspired by Murphy's "Rose" prototype language

Example

```
local size =
$metacode<inline>
source File.size(filepath)
$endMetacode
println "This program's source
  code is $ bytes" (size)
```

Output

```
This program's source code is
  126 bytes
```

Convenience Syntax

```
local size = ${ source
  File.size(filepath) }$
```

...

Multithreading

Garbled Output Example

```
# Must compile with:
# --threads --gc=auto-mt
ThreadTest()

class ThreadTest
 METHODS
    method init
     println "Program start"
     local thread = Thread(
        this=>count, 1, 10 )
     println "Started counting"
     thread.join
     println "All done"

   method count( first:Int32, last:Int32 )
     println (forEach in first..last)
endClass
```

Synchronized Example

```
ThreadTest()
class ThreadTest
 PROPERTIES
    mutex = Mutex()
 METHODS
    method init
      log "Program start"
      local thread = ...
      log "Started counting"
      thread.join
      log "All done"
    method count( first:Int32, last:Int32 )
      log (forEach in first..last)
    method log( message:String )
      use mutex
        println message
      endUse
endClass
```

Modules

Overview

- module X
- class ABC # becomes class X::ABC
- module # switch back to default namespace
- uses X
- ABC() # becomes X::ABC()
- module Y<<$Type>>
- module
- uses Y<<Int32>>
- # implicit use of types in Y<<Int32>>
- uses Y<<Int32>> as Int32Y
- # use Int32Y::XYZ
- module [essential api]

New Python Extension

Overview

- Uses CTypes
- See description in Hacking with Rogue
- Use [api] & [essential] flags
- roguec --target=C++,Python --output=test
- g++ ... test_module.cpp -o test_module.so
- python
- import test
- x = test.RogueClass()
- x.rogue_method()
- ...

ParseKit

Overview

- BitCalc Demo