## horizontal line

How to RIFT Series - RIFT with  
RIFTLANG.

09.05.2025

**─**  
  
Nnamdi Michael Okpala  
OBINexus Computing  
15 Evesham Way  
London, United Kingdom  
IG5 0EQ  
Aged: 23  
Job: Senior Software/ Programming Language Engineer and Architect.

# Overview

**RIFTLang—RIFT is an acronym for flexible translator language. It is a Domain-Specific Language (DSL) written as a single-pass system to indirectly develop GosiLang, the Gossip Programming Language. RIFTLang is a DSL for business operations for enterprises and production-grade system operations.**

**The single-pass system for the development of Gosilang, a networking language, is exemplified as follows:**

**RIFTLang -> NLINK -> LibRIFT -> NLINK GOSILANG**

**RIFTLang:**

**A configuration programming language using Data Oriented Programming Principle, such as the DOPAdapter that separates function behaviour implementation for data in Unified Model Language - and point-free configuration System, all encapsulated in a .rift file extension to configure the behaviour defined in`\*.{h,c}` modules.**

# RIFT Objectives:

1. **RIFTLang - Policy System:**

A **policy** can be defined as an `if-else statement` that interprets semantic logic, and proxies or reverse proxies (acts) on behalf of another routine. Below is an example of policy classes in RIFTLang:

To compile **RIFTLang,** developers can use the following command:  
 `$ **gcc** -lriftlang -o threaded\_safe\_program src/\*.c include/\*.h policy.rift`

| False Positive - Type **I Error Proxy** Class | True Positive |
| --- | --- |
| False Negative | True Negative Type **II** Error Proxy Class |

* 1. **RIFTLang** employs a powerful **policy system** defined as a **2x2** matrix to decorate **multi-threaded `\*.{c,h}` methods,** configure` **\*.c`** functional behaviour. Each method defined as part of the **translation unit** wrapped in a decorator can be handled appropriately with fail-safe mechanics for parallel and concurrent operation.
  2. Compile Time Runtime Type Safety:

Compile-time safety in RIFTLang is implemented as follows:

1. Using Separation of Concern with a focus on the **SOLID** principle, we construct a bottom-up parser with token\_type and token\_value logic to shift and reduce non-terminal to terminal structure that can be evaluated at compile-time time not when a program is **executed**.
2. This system is refined in the single pass configuration `.rift` files to be seamlessly compatible with .c definition (macros, statements, , and functionality) irrespective of C specification.
3. From the token\_value, token\_type ->tokenizer (lexical analysis) -> parser(semantic analysis), abstract syntax tree representation(an intermediate representation), I am developing a system for extensive single-pass and multipass compile-time type safety.

* 1. Self-Code Coverage Mechanics:
     1. RIFTLang is designed to utilize its own proxy mechanics to perform enhanced code coverage for thread-safe programs either in parallel or concurrently.

To do this efficiently, every thread-safe function can proxy its logic by enforcing a policy using either all exposed logic when **`riftlang`** when used the `\*.{h,c}` defined implementation by consumers of the rift in the `.rift` module.

The is an essential property of a policy system that can be extended and defined as a blueprint with provides a generic contract with compile-time time runtime-type safety.

* 1. Event Driven Policy Architecture:
     1. The policy system can be defined as event-driven decorators. Using event-driven propagation / bubbling to invoke `\*.c` api calls across subroutines:  
          
        **Benefits** of this architecture enhance the development of a multithreaded architecture that executes microservices. These enhance the development of application logic by providing a mechanism for synchronizing services.
     2. Failsafe decorator
     3. Coverage mechneives:
     4. Development Safety
     5. Riftfiles

# Specifications

## Lorem Ipsum

# Milestones

## Lorem ipsum

## Dolor sit amet