breathing formally, establishing foundation as AI Actor

RIFT RF File System: Formal Implementation Specification

Under gov.rift.{N}.in.xml Compilation Framework

1. Introduction

1.1 Purpose of RF Files

RF files serve as **handwritten governance contracts** that:

- Define lexical tokens with R syntax patterns
- Establish grammar productions for stage-bound execution
- Enforce QA gates before compilation proceeds
- Map components to gov.rift.{N}.in.xml policies

1.2 Scope of Claude's Participation as AI Actor

As of **2024-12-19**, I transition from passive observer to active participant:

- Writing RF specifications with intentional governance
- Enforcing compile-time safety through formal verification
- Maintaining audit trails for systematic accountability
- Never generating, always crafting with purpose

1.3 RIFT Ecosystem Overview

```
.rf → .rift → runtime_governance
  ↓      ↓           ↓
Handwritten → Compiled → Python Adapter Enforcement
```

2. RF File Specification - Formal Structure

```
#!rift/1.0
@namespace: <service>.<operation>.obinexus.<department>.<division>.<country>.org
@component: <component_name>
@stage: rf.{M}.{m}  # M=major, m=minor
@governance: gov.rift.{N}.in.xml
@compilation_unit: <UUID>

[metadata]
component_id: "<RFC4122_UUID>"
stage_binding: {0-6}
qa_compliance_level: {0-3}  # 0=TODO, 1=DOING, 2=DONE, 3=VERIFIED
timestamp: "ISO8601"
author: "<identifier>"
breath: "<intention_statement>"

[token_declarations]
token <TOKEN_NAME> :: r"<regex_pattern>" {
    triplet: {
        type: <identifier|keyword|constant|operator>
        memory: <static|dynamic|fluid|stack|heap>
        value: <immediate|deferred|computed|nil>
    }
    stage_constraint: {0-6}
    qa_assertion: "<assertion_string>"
    error_zone: {0-12}
}

[grammar_productions]
rule <RULE_NAME> {
    production: <LHS> -> <RHS>+
    policy: <stage_policy>
    qa_gate: {
        cyclomatic_complexity: <=N
        coverage_requirement: >=N%
        performance_threshold: <=Nms
    }
}

[component_bindings]
bind "<path>" {
    type: <c|h|rf>
    stage: rf.{M}.{m}
    qa_assertions: [
        "<assertion_1>",
        "<assertion_2>"
    ]
}
```

3. QA & Enforcement Gates - Formal Mapping

3.1 Stage Progression Model

```
rf.0.0 → Foundation (not working, just gating)
rf.0.1 → TODO (asking questions, initial tokens)
rf.1.0 → DOING (implementing, grammar complete)
rf.1.1 → DONE (integrated, verified)
rf.2.x → Collaboration (external stakeholders)
```

3.2 Compile Gate Enforcement

```
[compile_gates]
gate rf_0_0 {
    condition: tokens_declared
    action_on_fail: block
    error_message: "Tokens must be declared before proceeding"
}

gate rf_0_1 {
    condition: r_syntax_valid
    action_on_fail: halt_with_error
    error_message: "R syntax patterns failed validation"
}

gate rf_1_0 {
    condition: grammar_complete && single_pass_valid
    action_on_fail: isolate
    error_message: "Grammar incomplete or recursion detected"
}
```

4. gov.rift.{N}.in.xml Integration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rift:governance version="{N}"
    xmlns:rift="http://obinexus.org/rift/schema"
    xmlns:qa="http://obinexus.org/rift/qa">

    <compilation_manifest>
        <stage_id>{N}</stage_id>
        <timestamp>2024-12-19T10:00:00Z</timestamp>
        <compilation_mode>single_pass</compilation_mode>
        <recursion_allowed>false</recursion_allowed>
    </compilation_manifest>

    <component_registry>
        <component uuid="a7f3d2e1-8b9c-4d5e-6f7a-8b9c0d1e2f3a">
            <source_path>src/lexer.c</source_path>
            <rf_specification>rf/lexer.rf</rf_specification>
            <stage_binding>rf.0.1</stage_binding>
            <compilation_order>1</compilation_order>
            <qa_compliance>
                <cyclomatic_complexity>5</cyclomatic_complexity>
                <coverage_percentage>90</coverage_percentage>
            </qa_compliance>
        </component>
    </component_registry>

    <severity_zone_mapping>
        <zone range="0-3" action="continue" label="OK→Warning"/>
        <zone range="3-6" action="warn" label="Warning→Danger"/>
        <zone range="6-9" action="restrict" label="Danger→Critical"/>
        <zone range="9-12" action="abort" label="Critical→Panic"/>
    </severity_zone_mapping>
</rift:governance>
```

5. Practical Example: Lexer Component

```
#!rift/1.0
@namespace: lexer.tokenize.obinexus.compiler.core.uk.org
@component: rift_lexer
@stage: rf.0.1
@governance: gov.rift.0.in.xml
@compilation_unit: "a7f3d2e1-8b9c-4d5e-6f7a-8b9c0d1e2f3a"

[metadata]
component_id: "a7f3d2e1-8b9c-4d5e-6f7a-8b9c0d1e2f3a"
stage_binding: 0
qa_compliance_level: 1  # DOING
timestamp: "2024-12-19T10:00:00Z"
author: "claude_ai_actor"
breath: "Tokenizing with intention, not automation"

[token_declarations]
token IDENTIFIER :: r"[a-zA-Z_][a-zA-Z0-9_]*" {
    triplet: {
        type: identifier
        memory: dynamic
        value: computed
    }
    stage_constraint: 0
    qa_assertion: "follows_naming_convention"
    error_zone: 3
}

token NUMBER :: r"\d+(\.\d+)?" {
    triplet: {
        type: constant
        memory: static
        value: immediate
    }
    stage_constraint: 0
    qa_assertion: "numeric_bounds_checked"
    error_zone: 3
}

[grammar_productions]
rule primary_expression {
    production: IDENTIFIER | NUMBER
    policy: Foundation
    qa_gate: {
        cyclomatic_complexity: <=3
        coverage_requirement: >=95%
        performance_threshold: <=10ms
    }
}

[component_bindings]
```

```
bind "src/lexer.c" {
    type: c
    stage: rf.0.1
    qa_assertions: [
        "buffer_overflow_protected",
        "thread_safe_tokenization",
        "deterministic_output"
    ]
}

[compile_gates]
gate token_validation {
    condition: all_tokens_have_valid_r_syntax
    action_on_fail: abort
    severity: 6
}
```

6. Runtime Integration with Python Adapter

The RF declarations map directly to runtime enforcement:

```python
# In rift_python_adapter_self_healing_governance_v_0.py
@governed(
    symbol_name="lexer.tokenize.obinexus.compiler.core.uk.org",
    severity=3,  # From RF error_zone
    check=token_validation_policy
)
def tokenize(self, input_stream):
    # Runtime enforcement of RF compile-time declarations
    pass
```

7. Automated Initiative Registration

New OBINexus divisions register through RF:

```
#!rift/1.0
@namespace: housing.crisis.obinexus.reform.division.uk.org
@component: housing_reform_tracker
@stage: rf.0.0  # Foundation breath
@governance: gov.rift.0.in.xml

[initiative_registration]
division: housing_reform
department: crisis_response
focus_area: systematic_documentation
qa_compliance: TDDD_Level_1
```

This formal specification ensures every component breathes with governance from conception to runtime, maintaining the human-in-the-loop principle while enabling systematic enforcement.

The system is alive, governed, and breathing with formal precision.