Yankovinator: Reference Manual

API Documentation and Usage Guide

Shyamal Suhana Chandra

Contents

1 Introduction

This reference manual provides comprehensive documentation for the Yankovinator Swift library. Yankovinator is a parody generation system that converts songs into parodies while maintaining syllable structure, rhyming schemes, and semantic coherence.

2 Overview

Yankovinator consists of several key components:

- `Yankovinator` - Main library interface
- `ParodyGenerator` - Core parody generation engine
- `OllamaClient` - Interface to Ollama LLM API
- `SyllableCounter` - Syllable counting utilities
- `RhymeSchemeAnalyzer` - Rhyme detection and analysis

3 Yankovinator

The main entry point for the Yankovinator library.

3.1 generateParody(originalLyrics:keywords:ollamaURL:ollamaModel:)

Main function to generate a parody from original lyrics.

Signature:

```
public static func generateParody(
    originalLyrics: [String],
    keywords: [String: String],
    ollamaURL: String = "http://localhost:11434",
    ollamaModel: String = "llama3.2:3b"
) async throws -> [String]
```

Parameters:

- `originalLyrics: [String]` - Array of original song lines (one line per element)
- `keywords: [String: String]` - Dictionary mapping theme keywords to their definitions
- `ollamaURL: String` - Base URL for Ollama API (default: `"http://localhost:11434"`)
- `ollamaModel: String` - Ollama model name (default: `"llama3.2:3b"`)

Returns: `[String]` - Array of generated parody lines matching the original structure
Throws: Various errors including network errors, model not found errors, and generation errors
Example:

```
let lyrics = [
    "Twinkle twinkle little star",
    "How I wonder what you are"
]

let keywords = [
    "space": "the physical universe beyond Earth",
    "stars": "luminous celestial bodies"
]

let parody = try await Yankovinator.generateParody(
    originalLyrics: lyrics,
    keywords: keywords
)
```

3.2 countSyllables(_:)

Count syllables in a text string.

Signature:

```
public static func countSyllables(_ text: String) -> Int
```

Parameters:

- `text: String` - Text to analyze

Returns: `Int` - Total syllable count

Example:

```
let count = Yankovinator.countSyllables("Hello world")
// Returns: 2
```

3.3 analyzeStructure(_:)

Analyze syllable structure of a complete song.

Signature:

```
public static func analyzeStructure(_ lyrics: [String]) -> [Int]
```

Parameters:

- `lyrics: [String]` - Array of lyric lines

Returns: `[Int]` - Array of syllable counts per line

Example:

```
let lyrics = ["Hello world", "How are you"]
let structure = Yankovinator.analyzeStructure(lyrics)
// Returns: [2, 3]
```

4 ParodyGenerator

The core parody generation engine that orchestrates the conversion process.

4.1 Initialization

Signature:

```
public init(
    ollamaBaseURL: String = "http://localhost:11434",
    ollamaModel: String = "llama3.2:3b"
)
```

Parameters:

- `ollamaBaseURL: String` - Base URL for Ollama API
- `ollamaModel: String` - Model name to use

Example:

```
let generator = ParodyGenerator(
    ollamaBaseURL: "http://localhost:11434",
    ollamaModel: "llama3.2:3b"
)
```

4.2 generateParody(originalLyrics:keywords:progressCallback:refinementPasses:verbose:)

Generate a parody matching the original song's structure.

Signature:

```
public func generateParody(
    originalLyrics: [String],
    keywords: [String: String],
    progressCallback: ((Int, Int) -> Void)? = nil,
    refinementPasses: Int = 2,
    verbose: Bool = false
) async throws -> [String]
```

Parameters:

- `originalLyrics: [String]` - Array of original song lines
- `keywords: [String: String]` - Theme keywords and definitions
- `progressCallback: ((Int, Int) -> Void)?` - Optional callback for progress updates (current line, total lines)
- `refinementPasses: Int` - Number of refinement passes (default: 2)
- `verbose: Bool` - Whether to print verbose messages (default: false)

Returns: `[String]` - Generated parody lines with preserved empty lines
Throws: `OllamaError` and other errors during generation
Example:

```
let generator = ParodyGenerator()
let parody = try await generator.generateParody(
    originalLyrics: lyrics,
    keywords: keywords,
    progressCallback: { current, total in
        print("Progress: \(current)/\(total)")
    },
    refinementPasses: 2,
    verbose: true
)
```

4.3 extractKeywords(from:)

Extract keywords and definitions from text.

Signature:

```
public func extractKeywords(from text: String) -> [String: String]
```

Parameters:

- `text: String` - Text containing keywords in format `"keyword: definition"`

Returns: `[String: String]` - Dictionary of keywords and their definitions
Example:

```
let text = """
space: the physical universe beyond Earth
stars: luminous celestial bodies
"""
let keywords = generator.extractKeywords(from: text)
```

4.4 validateOllamaConnection()

Validate that Ollama is available and model exists.

Signature:

```
public func validateOllamaConnection() async throws -> Bool
```

Returns: `Bool` - True if Ollama is reachable and model is available

Throws: `OllamaError` if connection fails

Example:

```
let isValid = try await generator.validateOllamaConnection()
if !isValid {
    print("Ollama connection failed")
}
```

4.5 verifyModel()

Verify model is available before generation.

Signature:

```
public func verifyModel() async throws
```

Throws: `OllamaError.modelNotFound` if model is not available

Example:

```
try await generator.verifyModel()
```

5 OllamaClient

Client for interacting with the Ollama API.

5.1 Initialization

Signature:

```
public init(
    baseURL: String = "http://localhost:11434",
    model: String = "llama3.2:3b"
)
```

Parameters:

- `baseURL: String` - Base URL for Ollama API
- `model: String` - Model name to use

5.2 generateParodyLine(originalLine:syllableCount:keywords:previousLines:customPrompt:rhy

Generate a single parody line matching constraints.

Signature:

```
public func generateParodyLine(
    originalLine: String,
    syllableCount: Int,
    keywords: [String: String],
    previousLines: [String] = [],
    customPrompt: String? = nil,
    rhymeGroup: String? = nil,
    rhymingLines: [String] = [],
    rhymeScheme: String? = nil,
    wordSyllablePattern: String? = nil,
    wordSyllables: [Int]? = nil
) async throws -> String
```

Parameters:

- `originalLine: String` - Original song line
- `syllableCount: Int` - Target syllable count
- `keywords: [String: String]` - Theme keywords and definitions
- `previousLines: [String]` - Previous lines for context (default: empty)
- `customPrompt: String?` - Optional custom prompt (default: nil)
- `rhymeGroup: String?` - Rhyme group identifier (A, B, C, etc.)
- `rhymingLines: [String]` - Lines that should rhyme with this one
- `rhymeScheme: String?` - Overall rhyme scheme pattern (e.g., "ABAB")
- `wordSyllablePattern: String?` - Pattern showing word-by-word syllable counts
- `wordSyllables: [Int]?` - Array of syllable counts per word position

Returns: `String` - Generated parody line
Throws: `OllamaError` for various error conditions

5.3 checkAvailability()

Check if Ollama is available and model exists.

Signature:

```
public func checkAvailability() async throws -> Bool
```

Returns: `Bool` - True if Ollama is reachable and model is available

5.4 verifyModel()

Verify model exists and is available.

Signature:

```
public func verifyModel() async throws
```

Throws: `OllamaError.modelNotFound` if model is not available

5.5 generateKeywords(from:count:)

Generate keywords with definitions from subjects.

Signature:

```
public func generateKeywords(
    from subjects: [String],
    count: Int = 10
) async throws -> [String: String]
```

Parameters:

- `subjects: [String]` - Array of subjects or topics
- `count: Int` - Number of keyword pairs to generate (default: 10, max: 100)

Returns: `[String: String]` - Dictionary mapping keywords to definitions
Example:

```
let keywords = try await client.generateKeywords(
    from: ["artificial intelligence", "machine learning"],
    count: 15
)
```

6 SyllableCounter

Utility for counting syllables in words and lines.

6.1 countSyllables(in:)

Count syllables in a single word.

Signature:

```
public static func countSyllables(in word: String) -> Int
```

Parameters:

- `word: String` - The word to analyze

Returns: `Int` - Number of syllables

Example:

```
let count = SyllableCounter.countSyllables(in: "hello")
// Returns: 2
```

6.2 countSyllablesInLine(_:)

Count total syllables in a line of text.

Signature:

```
public static func countSyllablesInLine(_ line: String) -> Int
```

Parameters:

- `line: String` - The line of text

Returns: `Int` - Total syllable count

Example:

```
let count = SyllableCounter.countSyllablesInLine("Hello world")
// Returns: 2
```

6.3 analyzeSongStructure(_:)

Analyze syllable structure of a complete song.

Signature:

```
public static func analyzeSongStructure(_ lyrics: [String]) -> [Int]
```

Parameters:

- `lyrics: [String]` - Array of lyric lines

Returns: `[Int]` - Array of syllable counts per line

6.4 analyzeWordSyllables(in:)

Analyze word-by-word syllable structure of a line.

Signature:

```
public static func analyzeWordSyllables(in line: String) -> [(word: String,
    syllables: Int)]
```

Parameters:

- `line: String` - The line of text

Returns: `[(word: String, syllables: Int)]` - Array of tuples with word and syllable count

Example:

```
let analysis = SyllableCounter.analyzeWordSyllables(in: "Hello world")
// Returns: [("Hello", 2), ("world", 1)]
```

7 RhymeSchemeAnalyzer

Utility for detecting and analyzing rhyming schemes.

7.1 detectRhymeScheme(from:)

Detect rhyming scheme from lyrics.

Signature:

```
public static func detectRhymeScheme(from lyrics: [String]) -> (rhymeGroups: [
    String], scheme: String)
```

Parameters:

- `lyrics: [String]` - Array of lyric lines (non-empty lines only)

Returns: `(rhymeGroups: [String], scheme: String)` - Tuple with rhyme groups and scheme pattern

Example:

```
let lyrics = [
    "Twinkle twinkle little star",
    "How I wonder what you are",
    "Up above the world so high",
    "Like a diamond in the sky"
]
let (groups, scheme) = RhymeSchemeAnalyzer.detectRhymeScheme(from: lyrics)
// Returns: (["A", "A", "B", "B"], "AABB")
```

7.2 getRhymeGroup(for:in:)

Determine the rhyme group for a line position.

Signature:

```
public static func getRhymeGroup(
    for index: Int,
    in rhymeGroups: [String]
) -> String
```

Parameters:

- `index: Int` - Line index (0-based)
- `rhymeGroups: [String]` - Array of rhyme group identifiers

Returns: `String` - The rhyme group identifier for this line

7.3 getRhymingLineIndices(for:in:)

Get all line indices that should rhyme with the given line.

Signature:

```
public static func getRhymingLineIndices(
    for index: Int,
    in rhymeGroups: [String]
) -> [Int]
```

Parameters:

- `index: Int` - Current line index
- `rhymeGroups: [String]` - Array of rhyme group identifiers

Returns: `[Int]` - Array of indices that should rhyme with this line

8 Error Handling

8.1 OllamaError

Enumeration of errors that can occur when interacting with Ollama.

Cases:

- `invalidURL` - Invalid Ollama URL
- `httpError(statusCode:message:)` - HTTP error with status code and message
- `invalidResponse` - Invalid response from Ollama API
- `networkError(Error)` - Network-level error
- `modelNotFound(model:)` - Model not found error

Example:

```
do {
    let parody = try await Yankovinator.generateParody(
        originalLyrics: lyrics,
        keywords: keywords
    )
} catch OllamaError.modelNotFound(let model) {
    print("Model \(model) not found. Please install it.")
} catch OllamaError.networkError(let error) {
    print("Network error: \(error.localizedDescription)")
} catch {
    print("Unexpected error: \(error)")
}
```

9 Installation

9.1 Homebrew Installation (Recommended)

The easiest way to install Yankovinator is using Homebrew:

```
# Add the tap
brew tap shyamalschandra/yankovinator

# Install Yankovinator
brew install yankovinator

# Verify installation
yankovinator --help
keyword-generator --help
```

The Homebrew formula automatically handles all Swift Package Manager dependencies and builds the package in release mode.

9.2 Build from Source

For development or if you prefer building from source:

```
# Clone repository
git clone https://github.com/shyamalschandra/Yankovinator.git
cd yankovinator

# Build
swift build

# Run CLI
swift run yankovinator --help
```

9.3 Requirements

- Swift 6.2 or later
- macOS 13+ or iOS 16+
- Ollama installed and running
- Model: llama3.2:3b (install with `ollama pull llama3.2:3b`)

10 Usage Patterns

10.1 Basic Usage

The simplest way to generate a parody:

```
let lyrics = ["Line 1", "Line 2", "Line 3"]
let keywords = ["theme": "definition"]

let parody = try await Yankovinator.generateParody(
    originalLyrics: lyrics,
    keywords: keywords
)
```

10.2 With Progress Tracking

Track generation progress:

```
let generator = ParodyGenerator()
let parody = try await generator.generateParody(
    originalLyrics: lyrics,
    keywords: keywords,
    progressCallback: { current, total in
        let percentage = Double(current) / Double(total) * 100
        print("Progress: \(Int(percentage))%")
    }
)
```

10.3 With Custom Ollama Configuration

Use a custom Ollama instance:

```
let parody = try await Yankovinator.generateParody(
    originalLyrics: lyrics,
    keywords: keywords,
    ollamaURL: "http://192.168.1.100:11434",
    ollamaModel: "llama3.2:3b"
)
```

10.4 Error Handling Best Practices

Always handle errors appropriately:

```
do {
    // Verify connection first
    let generator = ParodyGenerator()
    try await generator.verifyModel()

    // Generate parody
    let parody = try await generator.generateParody(
        originalLyrics: lyrics,
        keywords: keywords
    )
```

```
11
12      // Process results
13      for line in parody {
14          print(line)
15      }
16  } catch OllamaError.modelNotFound(let model) {
17      print("Error: Model '\(model)' not found.")
18      print("Install it with: ollama pull \(model)")
19  } catch OllamaError.networkError(let error) {
20      print("Network error: \(error.localizedDescription)")
21      print("Ensure Ollama is running: ollama serve")
22  } catch {
23      print("Unexpected error: \(error.localizedDescription)")
24  }
```

11 Configuration

11.1 Refinement Passes

Control the number of refinement passes:

```
let generator = ParodyGenerator()
let parody = try await generator.generateParody(
    originalLyrics: lyrics,
    keywords: keywords,
    refinementPasses: 3  // More passes = better quality, slower
)
```

Default: 2 passes

- Pass 1: Word-by-word syllable matching
- Pass 2: Semantic coherence refinement
- Pass 3+: Punctuation and capitalization matching

11.2 Verbose Mode

Enable verbose output for debugging:

```
let generator = ParodyGenerator()
let parody = try await generator.generateParody(
    originalLyrics: lyrics,
    keywords: keywords,
    verbose: true  // Prints detailed progress information
)
```

12 Best Practices

12.1 Keyword Selection

- Use specific, meaningful keywords related to your theme
- Provide clear definitions for each keyword
- Use 5-15 keywords for best results
- Ensure keywords are relevant to the desired theme

12.2 Input Format

- One line per verse in the lyrics array
- Preserve empty lines to maintain song structure
- Use consistent formatting
- Include punctuation in original lyrics for better matching

12.3 Performance

- Use appropriate refinement passes (2-3 is usually sufficient)
- Monitor progress with callbacks for long songs
- Cache keywords if generating multiple parodies with same theme
- Verify Ollama connection before batch processing

13 Examples

13.1 Complete Example

```
import Yankovinator

// Prepare input
let originalLyrics = [
    "I want you to stay",
    "'Til I'm in the grave",
    "'Til I rot away, dead and buried",
    "'Til I'm in the casket you carry"
]

let keywords = [
    "love": "deep affection and attachment",
    "eternity": "infinite or unending time",
    "commitment": "dedication to a cause or relationship"
]

// Generate parody
do {
    let parody = try await Yankovinator.generateParody(
        originalLyrics: originalLyrics,
        keywords: keywords
    )

    // Output results
    for line in parody {
        print(line)
    }
} catch {
    print("Error: \(error)")
}
```

13.2 With Keyword Generation

```
let generator = ParodyGenerator()
let client = OllamaClient()

// Generate keywords from subjects
let keywords = try await client.generateKeywords(
    from: ["space exploration", "NASA", "astronomy"],
```

```
    count: 10
)

// Use generated keywords
let parody = try await generator.generateParody(
    originalLyrics: lyrics,
    keywords: keywords
)
```

14 Appendix: Type Definitions

14.1 Type Aliases

None currently defined.

14.2 Protocols

None currently defined.

15 Version History

- **v1.0** - Initial release with basic parody generation
- **v1.1** - Added semantic coherence features
- **v1.2** - Added capitalization and punctuation matching

16 License