

Home
Pages Classes Methods
Pages
LICENSE README
Class and Module Index
API AccumulateFilteredSegmentsTask CompressionTask CompressionTestAssessmentTask CompressionTestEvalTask CompressionTestTask Cursor DisplayResultsTask ExceptionAgent FileLoaderTask FileObject FilterSegmentsTask FinalReportTask Flowbots Flowbots::APIError Flowbots::AgentError Flowbots::BatchProcessor Flowbots::CLI Flowbots::ConfigurationError Flowbots::ExceptionAgent Flowbots::ExceptionHandler Flowbots::FileDiscovery Flowbots::FileLoader Flowbots::FileNotFoundError Flowbots::FlowbotError Flowbots::GrammarProcessor Flowbots::NLPPProcessor Flowbots::Task Flowbots::TaskNotFoundError Flowbots::TextProcessingWorkflow Flowbots::TextProcessor Flowbots::TextSegmentProcessor Flowbots::TextTaggerProcessor Flowbots::TextTokenizeProcessor Flowbots::TopicModelProcessor Flowbots::TopicModelTrainerWorkflow Flowbots::TopicModelTrainerWorkflowtest Flowbots::UnifiedFileProcessingPipeline Flowbots::WorkflowError Flowbots::Workflows FlowwiseApiClient InputRetrieval Jongleur Jongleur::WorkerTask Lemma LlmAnalysisTask LoadFileObjectTask LoadTextFilesTask Logging MarkdownYaml MarkdownYaml::Document0 MarkdownYaml::YamlFrontMatter0 MarkdownYaml::YamlFrontMatter1 MarkdownYamlParser MicroAgentTask MonadicError NlpAnalysisTask Object PreprocessFileObjectTask RedisConnection RedisKeys RunRubyTestsTask Segment

# Flowbots

Flowbots is an advanced text processing and analysis system that combines the power of nano-bots, workflow orchestration, and natural language processing to provide a flexible and powerful tool for document analysis and topic modeling.

## Features

- Text processing workflows for individual files and batch processing
- Advanced NLP capabilities including tokenization, part-of-speech tagging, and named entity recognition
- Topic modeling with dynamic model training and inference
- Flexible workflow system using Jongleur for task orchestration
- Redis-based data persistence using Ohm models
- Custom nano-bot cartridges for specialized AI-powered tasks
- Robust error handling and logging system
- User-friendly CLI interface

## System Architecture

### Class Diagram

```
classDiagram
    class CLI {
        +version()
        +workflows()
        +train_topic_model(folder)
        +process_text(file)
    }

    class Workflows {
```

Sublayer  
Sublayer::Actions  
Sublayer::Actions::RunTestCommandAction  
Sublayer::Actions::SpeechToTextAction  
Sublayer::Actions::TextToSpeechAction  
Sublayer::Actions::WriteFileAction  
TTY  
TTY::Markdown  
TTY::Markdown::Converter  
TTY::PromptX  
Task  
TextSegmentTask  
TextTaggerTask  
TextTokenizeTask  
TokenizeSegmentsTask  
Topic  
TopicModelingTask  
TrainTopicModelTask  
UI  
UI::Box  
UI::ScrollableBox  
Word  
WorkflowAgent  
WorkflowOrchestrator

```
-prompt: TTY::Prompt
+list_and_select()
+run(workflow_name)
-get_workflows()
-display_workflows(workflows)
-select_workflow(workflows)
-extract_workflow_description(file)
}

class WorkflowOrchestrator {
  -agents: Map
  +add_agent(role, cartridge_file)
  +define_workflow(workflow_definition)
  +run_workflow()
}

class WorkflowAgent {
  -role: String
  -state: Map
  -bot: NanoBot
  +process(input)
  +save_state()
  +load_state()
}

class Task {
  <<abstract>>
  +execute()
}

class TextProcessingWorkflow {
  -input_file_path: String
  -orchestrator: WorkflowOrchestrator
  +run()
}

class TopicModelTrainerWorkflow {
  -input_folder_path: String
  -orchestrator: WorkflowOrchestrator
  +run()
}

class TextProcessor {
  <<abstract>>
  +process(text)
}

class NLPProcessor {
  -nlp_model: Object
  +process(segment, options)
}

class TopicModelProcessor {
  -model_path: String
  -model: Object
  -model_params: Map
  +load_or_create_model()
  +train_model(documents, iterations)
  +infer_topics(document)
}
```

```

class FileLoader {
    -file_data: Textfile
    +initialize(file_path)
}

class Textfile {
    +path: String
    +name: String
    +content: String
    +preprocessed_content: String
    +metadata: Map
    +topics: Set~Topic~
    +segments: List~Segment~
    +lemmas: List~Lemma~
}

class Segment {
    +text: String
    +tokens: List
    +tagged: Map
    +words: List~Word~
}

class Word {
    +word: String
    +pos: String
    +tag: String
    +dep: String
    +ner: String
}

class Topic {
    +name: String
    +description: String
    +vector: List
}

CLI --> Workflows : uses
Workflows --> TextProcessingWorkflow : runs
Workflows --> TopicModelTrainerWorkflow : runs
TextProcessingWorkflow --> WorkflowOrchestrator
TopicModelTrainerWorkflow --> WorkflowOrchestrator
WorkflowOrchestrator --> WorkflowAgent : manages
WorkflowOrchestrator --> Task : executes
Task <|-- FileLoaderTask
Task <|-- PreprocessTextFileTask
Task <|-- TextSegmentTask
Task <|-- TokenizeSegmentsTask
Task <|-- NlpAnalysisTask
Task <|-- TopicModelingTask
Task <|-- LlmAnalysisTask
Task <|-- DisplayResultsTask
TextProcessor <|-- NLPProcessor
TextProcessor <|-- TopicModelProcessor
NlpAnalysisTask --> NLPProcessor : uses
TopicModelingTask --> TopicModelProcessor : uses
FileLoaderTask --> FileLoader : uses
Textfile "1" *-- "many" Segment
Segment "1" *-- "many" Word

```

```
Textfile "1" *-- "many" Topic
Textfile "1" *-- "many" Lemma
```

## Flowbots Project Overview

**Flowbots** is an advanced text processing and analysis system that combines the power of nano-bots, workflow orchestration, and natural language processing to provide a flexible and powerful tool for document analysis and topic modeling.

### Key Features

1. Text processing workflows for individual files and batch processing
2. Advanced NLP capabilities including tokenization, part-of-speech tagging, and named entity recognition
3. **Topic** modeling with dynamic model training and inference
4. Flexible workflow system using **Jongleur** for task orchestration
5. Redis-based data persistence using Ohm models
6. Custom nano-bot cartridges for specialized AI-powered tasks
7. Robust error handling and logging system
8. User-friendly CLI interface

### Project Structure

The **Flowbots** project is organized into several key directories:

- `/lib`: Main application code

- `/components`: Core system components
- `/processors`: Text and NLP processors
- `/tasks`: Individual workflow tasks
- `/workflows`: Workflow definitions
- `/ohm`: Ohm model definitions
- `/utils`: Utility functions and classes
- `/nano-bots/cartridges`: Nano-bot cartridge definitions
- `/test`: Test files and test helpers
- `/log`: Log files

## Key Components

1. **CLI**: The main entry point for user interaction, allowing users to select and run workflows.
2. **WorkflowOrchestrator**: Manages the execution of workflows and their constituent tasks.
3. **Task Processors**: Specialized classes for text processing, NLP analysis, and topic modeling.
4. **Ohm Models**: Data persistence layer for storing document information and workflow states.
5. **NanoBot Integration**: Utilizes nano-bot cartridges for specialized AI-powered tasks.
6. **Logging System**: Comprehensive logging for debugging and monitoring.

## Workflow Execution

1. User selects a workflow through the CLI.
2. The selected workflow is initialized

and configured.

3. The `WorkflowOrchestrator` sets up the task graph based on the workflow definition.
4. Tasks are executed in the defined order, with results passed between tasks as needed.
5. Results are stored in Redis and Ohm models for persistence.
6. The workflow completes, and final results are displayed or stored as appropriate.

This project demonstrates a sophisticated approach to text analysis and processing, combining multiple technologies and techniques to create a powerful and flexible system.

## Workflows

`Flowbots` uses a flexible workflow system to orchestrate various text processing and analysis tasks. The two main workflows defined in the project are:

1. `TextProcessingWorkflow`
2. `TopicModelTrainerWorkflow`

## TextProcessingWorkflow

This workflow is designed to process a single text file through a series of tasks.

```

stateDiagram-v2
    [*] --> Initialized
    Initialized --> PromptingForFile: No file path
    PromptingForFile --> FileSelected: User select
    FileSelected --> ProcessingFile: Start process
    ProcessingFile --> TextTagging: File processed
    TextTagging --> TopicModeling: Tagging complete
    TopicModeling --> LlmAnalysis: Modeling complete
    LlmAnalysis --> DisplayingResults: Analysis complete
    DisplayingResults --> [*]: Workflow complete

    state ProcessingFile {
        [*] --> LoadingFile
        LoadingFile --> PreprocessingFile
        PreprocessingFile --> SegmentingText
        SegmentingText --> TokenizingText
        TokenizingText --> [*]
    }

    DisplayingResults --> ErrorState: Error occurs
    ProcessingFile --> ErrorState: Error occurs
    TextTagging --> ErrorState: Error occurs
    TopicModeling --> ErrorState: Error occurs
    LlmAnalysis --> ErrorState: Error occurs
    ErrorState --> [*]: Log error and exit

```

## Key Steps:

1. **File Loading:** Loads the input file into the system.
2. **Preprocessing:** Extracts metadata and preprocesses the text content.
3. **Text Segmentation:** Splits the text into manageable segments.
4. **Tokenization:** Breaks down segments into individual tokens.
5. **NLP Analysis:** Performs part-of-speech tagging, dependency parsing, and named entity recognition.
6. **Topic Modeling:** Infers topics from the processed text.
7. **LLM Analysis:** Uses a language model to generate insights about the text.
8. **Result Display:** Presents the analysis results to the user.

# TopicModelTrainerWorkflow

This workflow is designed to process multiple files in batches and train a topic model.

## Key Steps:

1. **Batch Processing:** Processes files in batches of a defined size.
2. **File Loading:** Loads each file in the batch.
3. **Preprocessing:** Extracts metadata and preprocesses each file's content.
4. **Text Segmentation:** Splits each file's content into segments.
5. **Tokenization:** Breaks down segments into tokens.
6. **NLP Analysis:** Performs NLP tasks on the tokenized segments.
7. **Filtering:** Filters segments based on predefined criteria.
8. **Accumulation:** Accumulates filtered segments across all processed files.
9. **Topic Model Training:** Trains a topic model using the accumulated segments.

## Workflow Execution

Both workflows use the `WorkflowOrchestrator` class to manage task execution. The orchestrator:

1. Initializes the workflow and its tasks.
2. Sets up the task graph based on the workflow definition.
3. Executes tasks in the defined order.
4. Manages data flow between tasks using Redis for temporary storage.



5. Handles errors and exceptions during workflow execution.

## Workflow Flexibility

The workflow system is designed to be flexible and extensible:

- New workflows can be easily added by creating new workflow classes.
- Existing workflows can be modified by adding, removing, or reordering tasks.
- Tasks are modular and can be reused across different workflows.

This flexibility allows **Flowbots** to adapt to various text processing and analysis needs.

```
sequenceDiagram
    participant User
    participant TextProcessingWorkflow
    participant UnifiedFileProcessingPipeline
    participant WorkflowOrchestrator
    participant TextTaggerTask
    participant TopicModelingTask
    participant LlmAnalysisTask
    participant DisplayResultsTask
    participant Logger
    participant UI

    User->>TextProcessingWorkflow: Initialize with
    alt No file path provided
        TextProcessingWorkflow->>User: Prompt for
        User->>TextProcessingWorkflow: Provide file path
    end
    TextProcessingWorkflow->>UnifiedFileProcessingPipeline: Start processing
    TextProcessingWorkflow->>Logger: Log workflow start
    TextProcessingWorkflow->>UI: Display workflow progress
    TextProcessingWorkflow->>UnifiedFileProcessingPipeline: End processing
    UnifiedFileProcessingPipeline-->>TextProcessingWorkflow: Return results
    TextProcessingWorkflow->>WorkflowOrchestrator: Start tasks
    TextProcessingWorkflow->>WorkflowOrchestrator: End tasks
    WorkflowOrchestrator->>TextTaggerTask: Execute
    TextTaggerTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>TopicModelingTask: Execute
    TopicModelingTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>LlmAnalysisTask: Execute
    LlmAnalysisTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>DisplayResultsTask: Execute
    DisplayResultsTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator-->>TextProcessingWorkflow: Return results
    TextProcessingWorkflow->>Logger: Log workflow end
    TextProcessingWorkflow->>UI: Display completion
    TextProcessingWorkflow-->>User: Workflow finished
```

```
sequenceDiagram
    actor User
    participant CLI
    participant TextProcessingWorkflow
    participant WorkflowOrchestrator
    participant FileLoaderTask
    participant PreprocessTextFileTask
    participant TextSegmentTask
    participant TokenizeSegmentsTask
    participant NlpAnalysisTask
    participant TopicModelingTask
    participant LlmAnalysisTask
    participant DisplayResultsTask
    participant Redis
    participant Textfile

    User->>CLI: process_text(file)
    activate CLI
    CLI->>TextProcessingWorkflow: new(input_file_path)
    activate TextProcessingWorkflow
```

```
TextProcessingWorkflow->>WorkflowOrchestrator: activate
TextProcessingWorkflow->>WorkflowOrchestrator: deactivate
WorkflowOrchestrator->>FileLoaderTask: execute
WorkflowOrchestrator->>FileLoaderTask: activate
FileLoaderTask->>Redis: set("current_textfile", "current_textfile.txt")
FileLoaderTask->>Textfile: create
FileLoaderTask->>Textfile: deactivate
WorkflowOrchestrator->>PreprocessTextFileTask: execute
WorkflowOrchestrator->>PreprocessTextFileTask: activate
PreprocessTextFileTask->>Textfile: update(preprocessed_text)
PreprocessTextFileTask->>Textfile: deactivate
WorkflowOrchestrator->>TextSegmentTask: execute
WorkflowOrchestrator->>TextSegmentTask: activate
TextSegmentTask->>Textfile: add_segments()
TextSegmentTask->>Textfile: deactivate
WorkflowOrchestrator->>TokenizeSegmentsTask: execute
WorkflowOrchestrator->>TokenizeSegmentsTask: activate
TokenizeSegmentsTask->>Textfile: update_segments(tokens)
TokenizeSegmentsTask->>Textfile: deactivate
WorkflowOrchestrator->>NlpAnalysisTask: execute
WorkflowOrchestrator->>NlpAnalysisTask: activate
NlpAnalysisTask->>Textfile: update_segments_with_nlp_analysis()
NlpAnalysisTask->>Textfile: deactivate
WorkflowOrchestrator->>TopicModelingTask: execute
WorkflowOrchestrator->>TopicModelingTask: activate
TopicModelingTask->>Textfile: add_topics()
TopicModelingTask->>Textfile: deactivate
WorkflowOrchestrator->>LlmAnalysisTask: execute
WorkflowOrchestrator->>LlmAnalysisTask: activate
LlmAnalysisTask->>Textfile: update(analysis)
LlmAnalysisTask->>Textfile: deactivate
WorkflowOrchestrator->>DisplayResultsTask: execute
WorkflowOrchestrator->>DisplayResultsTask: activate
DisplayResultsTask->>Textfile: retrieve_data()
DisplayResultsTask->>User: display_results()
DisplayResultsTask->>User: deactivate
WorkflowOrchestrator->>WorkflowOrchestrator: deactivate
TextProcessingWorkflow-->>CLI: workflow complete
TextProcessingWorkflow-->>CLI: deactivate
CLI-->>User: display completion message
CLI-->>User: deactivate
```

## Task Processors

**Flowbots** uses a variety of task processors to handle different aspects of text processing and analysis. These processors are modular and can be combined in workflows to create complex text processing pipelines.

## Key Task Processors

1. FileLoaderTask
2. Loads input files into the system.
3. Stores file content in Ohm models for further processing.
4. PreprocessTextFileTask
5. Extracts metadata from file content (e.g., YAML front matter in Markdown files).
6. Preprocesses the main content for further analysis.
7. TextSegmentTask
8. Splits preprocessed text into manageable segments.
9. Uses the TextSegmentProcessor for actual segmentation logic.
10. TokenizeSegmentsTask
11. Breaks down text segments into individual tokens.
12. Uses the TextTokenizeProcessor for tokenization.
13. NlpAnalysisTask
14. Performs various NLP tasks on tokenized segments.
15. Includes part-of-speech tagging, dependency parsing, and named entity recognition.
16. Uses the NLPPProcessor which wraps the Spacy library for NLP operations.
17. FilterSegmentsTask
18. Filters processed segments based on predefined criteria.
19. Removes irrelevant or low-quality segments to improve analysis quality.
20. TopicModelingTask
21. Infers topics from processed text

- segments.
22. Uses the TopicModelProcessor which implements topic modeling algorithms.
  23. LlmAnalysisTask
  24. Utilizes a language model (via NanoBot) to generate insights about the text.
  25. Provides high-level analysis and summarization of the processed content.
  26. DisplayResultsTask
  27. Formats and displays the results of the text processing and analysis pipeline.

## Task Processor Architecture

Each task processor:

1. Inherits from `Jongleur::WorkerTask` or `Flowbots::BaseTask`.
2. Implements an `execute` method that performs the core task logic.
3. Uses Redis for temporary data storage and passing data between tasks.
4. Interacts with Ohm models for persistent data storage.
5. Includes error handling and logging for robust execution.

## Extensibility

The task processor system is designed to be easily extensible:

- New task processors can be added by creating new classes inheriting from `Jongleur::WorkerTask` or `Flowbots::BaseTask`.

- Existing task processors can be modified or extended to support new functionality.
- **Task** processors can be combined in different ways within workflows to create custom text processing pipelines.

This modular design allows **Flowbots** to adapt to various text processing and analysis requirements.

## **Flowbots** Detailed Operation

### 1. Workflow Initialization

When a user selects a workflow through the CLI, the system initializes the chosen workflow (e.g., `TextProcessingWorkflow` or `TopicModelTrainerWorkflow`). The **WorkflowOrchestrator** sets up the task graph based on the workflow definition.

### 2. Task Execution

The **WorkflowOrchestrator** executes tasks in the defined order. Each task follows a similar pattern:

1. Retrieve necessary data from Redis or Ohm models.
2. Process the data using specialized processors (e.g., `NLPProcessor`, `TopicModelProcessor`).
3. Store the results back in Redis (for temporary storage) or Ohm models (for persistence).

### 3. Data Flow

- Redis is used for storing temporary data and passing information between tasks. This includes file IDs, current batch information, and intermediate processing results.
- Ohm models, backed by Redis, are used for persistent storage of document information, segments, tokens, and analysis results.

### 4. NLP and **Topic** Modeling

- The `NlpAnalysisTask` uses the ruby-spacy gem to perform tasks like tokenization, part-of-speech tagging, and named entity recognition.
- The `TopicModelingTask` uses the tomoto gem to implement topic modeling algorithms.

### 5. LLM Integration

The `LlmAnalysisTask` integrates with external language models through the NanoBot system. This allows for high-level analysis and insights generation based on the processed text data.

### 6. Error Handling and **Logging**

Each task and the `WorkflowOrchestrator` include error handling mechanisms. Errors are caught, logged, and in some cases, trigger the `ExceptionAgent` for detailed error analysis.

### 7. Batch Processing

For the `TopicModelTrainerWorkflow`, files are processed in batches. The `WorkflowOrchestrator` manages the batch state, ensuring all files in a batch are processed before moving to the next batch.

## 8. Result Presentation

The `DisplayResultsTask` formats the analysis results and presents them to the user through the CLI. This may include summaries, topic distributions, and insights generated by the LLM.

## Key Interactions

1. `CLI <-> WorkflowOrchestrator`: The CLI initiates workflow execution and receives final results.
2. `WorkflowOrchestrator <-> Tasks`: The orchestrator manages task execution order and handles task results.
3. `Tasks <-> Redis`: Tasks use Redis for short-term storage and inter-task communication.
4. `Tasks <-> Ohm Models`: Tasks interact with Ohm models for persistent storage of document data and analysis results.
5. `NLP and Topic Modeling Tasks <-> External Libraries`: These tasks utilize external Ruby gems for specialized processing.
6. `LlmAnalysisTask <-> NanoBot`: This task interacts with the NanoBot system to leverage external language models.

This architecture allows `Flowbots` to process text data through a series of specialized tasks, each building upon the



results of previous tasks, to provide comprehensive text analysis and insights.

## Ruby Gems Used in **Flowbots**

**Flowbots** leverages a variety of Ruby gems to provide its functionality. Here's a comprehensive list of the gems used in the project, along with their purposes:

1. `jongleur`
2. Purpose: Workflow orchestration and task management
3. Usage: Core component for defining and executing task workflows
4. `ohm`
5. Purpose: **Object**-hash mapping for Redis
6. Usage: Data persistence layer for storing document information and workflow states
7. `redis`
8. Purpose: In-memory data structure store
9. Usage: Temporary data storage and passing data between tasks
10. `json`
11. Purpose: JSON parsing and generation
12. Usage: Handling JSON data throughout the application
13. `parallel`
14. Purpose: Parallel processing
15. Usage: Potential use for parallel execution of tasks (not prominently used in the current implementation)
16. `pry` and `pry-stack_explorer`
17. Purpose: Enhanced REPL and debugging

tools

18. Usage: Development and debugging
19. `ruby-spacy`
20. Purpose: Ruby bindings for the Spacy NLP library
21. Usage: Natural Language Processing tasks
22. `thor`
23. Purpose: Building command-line interfaces
24. Usage: Creating the CLI for `Flowbots`
25. `treetop`
26. Purpose: parsing expression grammar (PEG) parser generator
27. Usage: Custom grammar parsing, particularly for Markdown with YAML front matter
28. `yaml`
  - Purpose: YAML parsing and generation
  - Usage: Handling YAML data, particularly in configuration files and document front matter
29. `faraday` and `faraday/multipart`
  - Purpose: HTTP client library
  - Usage: Making HTTP requests, potentially for integrations with external services
30. `logging`
  - Purpose: Flexible logging
  - Usage: Comprehensive logging system throughout the application
31. `tty-box`, `tty-cursor`, `tty-prompt`, `tty-`

screen, tty-spinner, tty-table

- Purpose: Various terminal output formatting and interaction tools
- Usage: Creating rich command-line interfaces and displaying formatted output

### 32. pastel

- Purpose: Terminal output styling
- Usage: Adding colors and styles to terminal output

### 33. highline

- Purpose: High-level command-line interface building
- Usage: Additional CLI features and user input handling

### 34. cli-ui

- Purpose: CLI user interface components
- Usage: Enhancing the command-line interface with advanced **UI** elements

### 35. kramdown

- Purpose: Markdown parsing and conversion
- Usage: Handling Markdown content in documents

### 36. lingua

- Purpose: Natural language detection and processing
- Usage: Additional NLP capabilities

### 37. pragmatic\_segmenter

- Purpose: Text segmentation
- Usage: Splitting text into meaningful segments

#### 38. `pragmatic_tokenizer`

- Purpose: Text tokenization
- Usage: Breaking text into individual tokens

#### 39. `tomoto`

- Purpose: **Topic** modeling
- Usage: Implementing topic modeling algorithms

#### 40. `minitest` and `minitest/rg`

- Purpose: Testing framework
- Usage: Writing and running tests for the application

These gems provide a robust foundation for **Flowbots**, covering areas such as data persistence, natural language processing, command-line interfaces, HTTP communications, and more. The combination of these tools allows **Flowbots** to offer a comprehensive text processing and analysis system with a user-friendly interface.

Validate

Generated by RDoc 6.4.0.

Based on **Darkfish** by Michael Granger.

# Table of Contents

Table of Contents	2
Home	11
class API	11
Parent	11
Methods	11
Private Instance Methods	11
Home	12
Flowbots	12
Table of Contents	12
Features	12
Pages	12
System Architecture	12
Class Diagram	12
Flowbots Project Overview	15
Key Features	15
Project Structure	15
Key Components	16
Workflow Execution	16
Workflows	17
TextProcessingWorkflow	17
Key Steps:	18
TopicModelTrainerWorkflow	19
Key Steps:	19
Workflow Execution	19
Workflow Flexibility	20
Task Processors	22
Key Task Processors	23
Task Processor Architecture	24
Extensibility	24
Flowbots Detailed Operation	25
1. Workflow Initialization	25
2. Task Execution	25
3. Data Flow	26
4. NLP and Topic Modeling	26
5. LLM Integration	26
6. Error Handling and Logging	26
7. Batch Processing	26
8. Result Presentation	27
Key Interactions	27
Ruby Gems Used in Flowbots	28
Home	32
module Flowbots	32
Methods	32
Constants	32
Public Class Methods	32
Private Class Methods	32
Home	34
class Flowbots::AgentError	34
Parent	34
Home	35
class Flowbots::APIError	35
Parent	35
Home	36
class Flowbots::BatchProcessor	36
Parent	36
Methods	36
Attributes	36
Public Class Methods	36
Public Instance Methods	37
Private Instance Methods	37
Home	39
class Flowbots::CLI	39
Parent	39
Extended With Modules	39
Methods	39
Public Class Methods	39
Public Instance Methods	39
Home	42
class Flowbots::ConfigurationError	42
Parent	42
Home	43

class Flowbots::ExceptionAgent	43
Parent	43
Methods	43
Public Class Methods	43
Public Instance Methods	43
Private Instance Methods	44
Home	47
class Flowbots::ExceptionHandler	47
Parent	47
Methods	47
Public Class Methods	47
Home	49
module Flowbots::FileDiscovery	49
Methods	49
Constants	49
Public Class Methods	49
Home	51
class Flowbots::FileLoader	51
Parent	51
Methods	51
Attributes	51
Public Class Methods	51
Private Instance Methods	51
Home	53
class Flowbots::FileNotFoundError	53
Parent	53
Home	54
class Flowbots::FlowbotError	54
Parent	54
Methods	54
Attributes	54
Public Class Methods	54
Home	55
class Flowbots::GrammarProcessor	55
Parent	55
Methods	55
Public Class Methods	55
Public Instance Methods	55
Private Instance Methods	56
Home	57
class Flowbots::NLPPProcessor	57
Parent	57
Methods	57
Public Class Methods	57
Public Instance Methods	57
Private Instance Methods	58
Home	59
class Flowbots::Task	59
Parent	59
Methods	59
Public Class Methods	59
Public Instance Methods	59
Home	60
class Flowbots::TaskNotFoundError	60
Parent	60
Home	61
class Flowbots::TextProcessingWorkflow	61
Parent	61
Methods	61
Attributes	61
Public Class Methods	61
Public Instance Methods	61
Private Instance Methods	62
Home	64
class Flowbots::TextProcessor	64
Parent	64
Included Modules	64
Methods	64
Public Class Methods	64
Public Instance Methods	64
Home	65
class Flowbots::TextSegmentProcessor	65
Parent	65
Methods	65

Constants	65
Attributes	65
Public Class Methods	65
Public Instance Methods	65
Private Instance Methods	66
Home	67
class Flowbots::TextTaggerProcessor	67
Parent	67
Included Modules	67
Methods	67
Public Class Methods	67
Public Instance Methods	67
Private Instance Methods	68
Home	69
class Flowbots::TextTokenizeProcessor	69
Parent	69
Methods	69
Constants	69
Attributes	69
Public Class Methods	69
Public Instance Methods	69
Private Instance Methods	70
Home	71
class Flowbots::TopicModelProcessor	71
Parent	71
Methods	71
Attributes	71
Public Class Methods	71
Public Instance Methods	71
Private Instance Methods	73
Home	75
class Flowbots::TopicModelTrainerWorkflow	75
Parent	75
Methods	75
Attributes	75
Public Class Methods	75
Public Instance Methods	75
Private Instance Methods	75
Home	77
class Flowbots::TopicModelTrainerWorkflowtest	77
Parent	77
Methods	77
Constants	77
Attributes	77
Public Class Methods	77
Public Instance Methods	77
Private Instance Methods	77
Home	79
class Flowbots::UnifiedFileProcessingPipeline	79
Parent	79
Methods	79
Attributes	79
Public Class Methods	79
Public Instance Methods	79
Private Instance Methods	79
Home	81
class Flowbots::WorkflowError	81
Parent	81
Home	82
class Flowbots::Workflows	82
Parent	82
Methods	82
Public Class Methods	82
Public Instance Methods	82
Private Instance Methods	83
Home	85
module Jongleur	85
Home	86
class Jongleur::WorkerTask	86
Parent	86
Home	87
module MarkdownYaml	87
Included Modules	87
Methods	87

Public Instance Methods	87
Home	89
module MarkdownYaml::Document0	89
Methods	89
Public Instance Methods	89
Home	90
module MarkdownYaml::YamlFrontMatter0	90
Home	91
module MarkdownYaml::YamlFrontMatter1	91
Methods	91
Public Instance Methods	91
Home	92
module Sublayer	92
Home	93
module Sublayer::Actions	93
Home	94
class Sublayer::Actions::RunTestCommandAction	94
Parent	94
Methods	94
Public Class Methods	94
Public Instance Methods	94
Home	95
class Sublayer::Actions::SpeechToTextAction	95
Parent	95
Methods	95
Public Class Methods	95
Public Instance Methods	95
Home	96
class Sublayer::Actions::TextToSpeechAction	96
Parent	96
Methods	96
Public Class Methods	96
Public Instance Methods	96
Home	97
class Sublayer::Actions::WriteFileAction	97
Parent	97
Methods	97
Public Class Methods	97
Public Instance Methods	97
Home	98
module TTY	98
Home	99
module TTY::Markdown	99
Home	100
class TTY::PromptX	100
Parent	100
Attributes	100
Methods	100
Public Class Methods	100
Public Instance Methods	100
Home	101
module TTY::Markdown	101
Home	102
class TTY::Markdown::Converter	102
Parent	102
Methods	102
Public Instance Methods	102
Home	103
module UI	103
Methods	103
Constants	103
Public Instance Methods	103
Home	106
module UI::Box	106
Methods	106
Public Instance Methods	106
Home	109
module UI::ScrollableBox	109
Methods	109
Private Class Methods	109
Public Instance Methods	110
Home	112
class AccumulateFilteredSegmentsTask	112
Parent	112



Included Modules	112
Methods	112
Public Instance Methods	112
Private Instance Methods	112
Home	114
class API	114
Parent	114
Methods	114
Private Instance Methods	114
Home	115
class CompressionTask	115
Parent	115
Methods	115
Public Instance Methods	115
Home	116
class CompressionTestAssessmentTask	116
Parent	116
Methods	116
Public Instance Methods	116
Home	117
class CompressionTestEvalTask	117
Parent	117
Methods	117
Public Instance Methods	117
Home	118
class CompressionTestTask	118
Parent	118
Methods	118
Public Instance Methods	118
Home	119
class Cursor	119
Parent	119
Public Class Methods	119
Methods	119
Home	120
class DisplayResultsTask	120
Parent	120
Included Modules	120
Public Instance Methods	120
Methods	120
Private Instance Methods	120
Home	122
class ExceptionAgent	122
Parent	122
Public Class Methods	122
Methods	122
Public Instance Methods	122
Private Instance Methods	122
Home	123
class FileLoaderTask	123
Parent	123
Methods	123
Public Instance Methods	123
Private Instance Methods	123
Home	124
class FileObject	124
Parent	124
Public Class Methods	124
Included Modules	124
Methods	124
Public Instance Methods	124
Protected Instance Methods	125
Home	126
class FilterSegmentsTask	126
Parent	126
Included Modules	126
Public Instance Methods	126
Methods	126
Private Instance Methods	126
Home	127
class FinalReportTask	127
Parent	127
Methods	127
Public Instance Methods	127

Home	129
module Flowbots	129
Methods	129
Constants	129
Public Class Methods	129
Private Class Methods	129
Home	131
class FlowiseApiClient	131
Parent	131
Methods	131
Public Class Methods	131
Public Instance Methods	131
Private Instance Methods	132
Home	133
Flowbots	133
Pages	133
Class and Module Index	133
Features	133
System Architecture	133
Class Diagram	133
Flowbots Project Overview	136
Key Features	136
Project Structure	136
Key Components	137
Workflow Execution	137
Workflows	138
TextProcessingWorkflow	138
Key Steps:	139
TopicModelTrainerWorkflow	140
Key Steps:	140
Workflow Execution	140
Workflow Flexibility	141
Task Processors	143
Key Task Processors	144
Task Processor Architecture	145
Extensibility	145
Flowbots Detailed Operation	146
1. Workflow Initialization	146
2. Task Execution	146
3. Data Flow	147
4. NLP and Topic Modeling	147
5. LLM Integration	147
6. Error Handling and Logging	147
7. Batch Processing	147
8. Result Presentation	148
Key Interactions	148
Ruby Gems Used in Flowbots	149
Home	153
module InputRetrieval	153
Methods	153
Public Instance Methods	153
Home	154
module Jongleur	154
Home	155
class Lemma	155
Parent	155
Included Modules	155
Home	156
Pages	156
Home	157
class LlmAnalysisTask	157
Parent	157
Included Modules	157
Methods	157
Public Instance Methods	157
Private Instance Methods	158
Home	160
class LoadFileObjectTask	160
Parent	160
Included Modules	160
Public Instance Methods	160
Methods	160
Private Instance Methods	161
Home	162

class LoadTextFilesTask	162
Parent	162
Included Modules	162
Public Instance Methods	162
Methods	162
Private Instance Methods	163
Home	164
module Logging	164
Methods	164
Constants	164
Public Class Methods	164
Public Instance Methods	165
Home	166
module MarkdownYaml	166
Included Modules	166
Methods	166
Public Instance Methods	166
Home	168
class MarkdownYamlParser	168
Parent	168
Included Modules	168
Home	169
class MicroAgentTask	169
Parent	169
Public Class Methods	169
Methods	169
Public Instance Methods	169
Private Instance Methods	169
Home	170
class MonadicError	170
Parent	170
Home	171
class NlpAnalysisTask	171
Parent	171
Included Modules	171
Methods	171
Public Instance Methods	171
Private Instance Methods	171
Home	174
class Object	174
Parent	174
Constants	174
Included Modules	174
Methods	174
Public Instance Methods	174
Home	176
class PreprocessFileObjectTask	176
Parent	176
Included Modules	176
Methods	176
Public Instance Methods	176
Private Instance Methods	177
Home	180
Flowbots	180
Table of Contents	180
Features	180
Pages	180
System Architecture	180
Class Diagram	180
Flowbots Project Overview	183
Key Features	183
Project Structure	183
Key Components	184
Workflow Execution	184
Workflows	185
TextProcessingWorkflow	185
Key Steps:	186
TopicModelTrainerWorkflow	187
Key Steps:	187
Workflow Execution	187
Workflow Flexibility	188
Task Processors	190
Key Task Processors	191
Task Processor Architecture	192

Extensibility	192
Flowbots Detailed Operation	193
1. Workflow Initialization	193
2. Task Execution	193
3. Data Flow	194
4. NLP and Topic Modeling	194
5. LLM Integration	194
6. Error Handling and Logging	194
7. Batch Processing	194
8. Result Presentation	195
Key Interactions	195
Ruby Gems Used in Flowbots	196
Home	200
class RedisConnection	200
Parent	200
Methods	200
Constants	200
Attributes	200
Public Class Methods	200
Home	201
module RedisKeys	201
Methods	201
Constants	201
Public Class Methods	201
Home	203
class RunRubyTestsTask	203
Parent	203
Methods	203
Public Instance Methods	203
Home	204
class Segment	204
Parent	204
Public Instance Methods	204
Included Modules	204
Methods	204
Home	205
module Sublayer	205
Table of Contents - flowbots v0.1	206
Pages	206
Classes and Modules	207
Methods	209
Home	217
class Task	217
Parent	217
Public Class Methods	217
Included Modules	217
Methods	217
Public Instance Methods	217
Home	219
class TextSegmentTask	219
Parent	219
Included Modules	219
Public Instance Methods	219
Methods	219
Private Instance Methods	219
Home	221
class TextTaggerTask	221
Parent	221
Included Modules	221
Public Instance Methods	221
Methods	221
Private Instance Methods	221
Home	223
class TextTokenizeTask	223
Parent	223
Methods	223
Public Instance Methods	223
Home	224
class TokenizeSegmentsTask	224
Parent	224
Included Modules	224
Methods	224
Public Instance Methods	224
Private Instance Methods	224

Home	225
class Topic	225
Parent	225
Included Modules	225
Home	226
class TopicModelingTask	226
Parent	226
Included Modules	226
Public Class Methods	226
Methods	226
Public Instance Methods	226
Private Instance Methods	226
Home	228
class TrainTopicModelTask	228
Parent	228
Methods	228
Public Instance Methods	228
Home	230
module TTY	230
Home	231
module UI	231
Methods	231
Constants	231
Public Instance Methods	231
Home	234
class Word	234
Parent	234
Included Modules	234
Home	235
class WorkflowAgent	235
Parent	235
Methods	235
Constants	236
Attributes	236
Public Class Methods	236
Public Instance Methods	237
Private Instance Methods	237
Home	238
class WorkflowOrchestrator	238
Parent	238
Methods	238
Constants	238
Public Class Methods	238
Public Instance Methods	238

# class API

... (Ohm model definitions from OhmModels.rb)  
...

---

## Private Instance Methods

### splat\_sort(splat\_vals)

... (Add routes for other object buckets and their attributes) ...

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

Home
Pages Classes Methods
Table of Contents
Flowbots
Features
System Architecture
Class Diagram
Flowbots Project Overview
Key Features
Project Structure
Key Components
Workflow Execution
Workflows
TextProcessingWorkflow
Key Steps:
TopicModelTrainerWorkflow
Key Steps:
Workflow Execution
Workflow Flexibility
Task Processors
Key Task Processors
Task Processor Architecture
Extensibility
Flowbots Detailed Operation
1. Workflow Initialization
2. Task Execution
3. Data Flow
4. NLP and Topic Modeling
5. LLM Integration
6. Error Handling and Logging
7. Batch Processing
8. Result Presentation
Key Interactions
Ruby Gems Used in Flowbots
Pages
LICENSE
README

# Flowbots

Flowbots is an advanced text processing and analysis system that combines the power of nano-bots, workflow orchestration, and natural language processing to provide a flexible and powerful tool for document analysis and topic modeling.

## Features

- Text processing workflows for individual files and batch processing
- Advanced NLP capabilities including tokenization, part-of-speech tagging, and named entity recognition
- Topic modeling with dynamic model training and inference
- Flexible workflow system using Jongleur for task orchestration
- Redis-based data persistence using Ohm models
- Custom nano-bot cartridges for specialized AI-powered tasks
- Robust error handling and logging system
- User-friendly CLI interface

## System Architecture

### Class Diagram

```
classDiagram
  class CLI {
    +version()
    +workflows()
    +train_topic_model(folder)
    +process_text(file)
  }

  class Workflows {
```

```

        -prompt: TTY::Prompt
        +list_and_select()
        +run(workflow_name)
        -get_workflows()
        -display_workflows(workflows)
        -select_workflow(workflows)
        -extract_workflow_description(file)
    }

    class WorkflowOrchestrator {
        -agents: Map
        +add_agent(role, cartridge_file)
        +define_workflow(workflow_definition)
        +run_workflow()
    }

    class WorkflowAgent {
        -role: String
        -state: Map
        -bot: NanoBot
        +process(input)
        +save_state()
        +load_state()
    }

    class Task {
        <<abstract>>
        +execute()
    }

    class TextProcessingWorkflow {
        -input_file_path: String
        -orchestrator: WorkflowOrchestrator
        +run()
    }

    class TopicModelTrainerWorkflow {
        -input_folder_path: String
        -orchestrator: WorkflowOrchestrator
        +run()
    }

    class TextProcessor {
        <<abstract>>
        +process(text)
    }

    class NLPProcessor {
        -nlp_model: Object
        +process(segment, options)
    }

    class TopicModelProcessor {
        -model_path: String
        -model: Object
        -model_params: Map
        +load_or_create_model()
        +train_model(documents, iterations)
        +infer_topics(document)
    }

```



```

class FileLoader {
    -file_data: Textfile
    +initialize(file_path)
}

class Textfile {
    +path: String
    +name: String
    +content: String
    +preprocessed_content: String
    +metadata: Map
    +topics: Set~Topic~
    +segments: List~Segment~
    +lemmas: List~Lemma~
}

class Segment {
    +text: String
    +tokens: List
    +tagged: Map
    +words: List~Word~
}

class Word {
    +word: String
    +pos: String
    +tag: String
    +dep: String
    +ner: String
}

class Topic {
    +name: String
    +description: String
    +vector: List
}

CLI --> Workflows : uses
Workflows --> TextProcessingWorkflow : runs
Workflows --> TopicModelTrainerWorkflow : runs
TextProcessingWorkflow --> WorkflowOrchestrator
TopicModelTrainerWorkflow --> WorkflowOrchestrator
WorkflowOrchestrator --> WorkflowAgent : manages
WorkflowOrchestrator --> Task : executes
Task <|-- FileLoaderTask
Task <|-- PreprocessTextFileTask
Task <|-- TextSegmentTask
Task <|-- TokenizeSegmentsTask
Task <|-- NlpAnalysisTask
Task <|-- TopicModelingTask
Task <|-- LlmAnalysisTask
Task <|-- DisplayResultsTask
TextProcessor <|-- NLPProcessor
TextProcessor <|-- TopicModelProcessor
NlpAnalysisTask --> NLPProcessor : uses
TopicModelingTask --> TopicModelProcessor : uses
FileLoaderTask --> FileLoader : uses
Textfile "1" *-- "many" Segment
Segment "1" *-- "many" Word

```

```
Textfile "1" *-- "many" Topic
Textfile "1" *-- "many" Lemma
```

## Flowbots Project Overview

**Flowbots** is an advanced text processing and analysis system that combines the power of nano-bots, workflow orchestration, and natural language processing to provide a flexible and powerful tool for document analysis and topic modeling.

### Key Features

1. Text processing workflows for individual files and batch processing
2. Advanced NLP capabilities including tokenization, part-of-speech tagging, and named entity recognition
3. **Topic** modeling with dynamic model training and inference
4. Flexible workflow system using **Jongleur** for task orchestration
5. Redis-based data persistence using Ohm models
6. Custom nano-bot cartridges for specialized AI-powered tasks
7. Robust error handling and logging system
8. User-friendly CLI interface

### Project Structure

The **Flowbots** project is organized into several key directories:

- `/lib`: Main application code

- /components: Core system components
- /processors: Text and NLP processors
- /tasks: Individual workflow tasks
- /workflows: Workflow definitions
- /ohm: Ohm model definitions
- /utils: Utility functions and classes
- /nano-bots/cartridges: Nano-bot cartridge definitions
- /test: Test files and test helpers
- /log: Log files

## Key Components

1. CLI: The main entry point for user interaction, allowing users to select and run workflows.
2. WorkflowOrchestrator: Manages the execution of workflows and their constituent tasks.
3. Task Processors: Specialized classes for text processing, NLP analysis, and topic modeling.
4. Ohm Models: Data persistence layer for storing document information and workflow states.
5. NanoBot Integration: Utilizes nano-bot cartridges for specialized AI-powered tasks.
6. Logging System: Comprehensive logging for debugging and monitoring.

## Workflow Execution

1. User selects a workflow through the CLI.
2. The selected workflow is initialized

and configured.

3. The `WorkflowOrchestrator` sets up the task graph based on the workflow definition.
4. Tasks are executed in the defined order, with results passed between tasks as needed.
5. Results are stored in Redis and Ohm models for persistence.
6. The workflow completes, and final results are displayed or stored as appropriate.

This project demonstrates a sophisticated approach to text analysis and processing, combining multiple technologies and techniques to create a powerful and flexible system.

## Workflows

`Flowbots` uses a flexible workflow system to orchestrate various text processing and analysis tasks. The two main workflows defined in the project are:

1. `TextProcessingWorkflow`
2. `TopicModelTrainerWorkflow`

## TextProcessingWorkflow

This workflow is designed to process a single text file through a series of tasks.

```

stateDiagram-v2
    [*] --> Initialized
    Initialized --> PromptingForFile: No file path
    PromptingForFile --> FileSelected: User select
    Initialized --> FileSelected: File path provid
    FileSelected --> ProcessingFile: Start process
    ProcessingFile --> TextTagging: File processe
    TextTagging --> TopicModeling: Tagging complet
    TopicModeling --> LlmAnalysis: Modeling complet
    LlmAnalysis --> DisplayingResults: Analysis co
    DisplayingResults --> [*]: Workflow complete

    state ProcessingFile {
        [*] --> LoadingFile
        LoadingFile --> PreprocessingFile
        PreprocessingFile --> SegmentingText
        SegmentingText --> TokenizingText
        TokenizingText --> [*]
    }

    DisplayingResults --> ErrorState: Error occurs
    ProcessingFile --> ErrorState: Error occurs
    TextTagging --> ErrorState: Error occurs
    TopicModeling --> ErrorState: Error occurs
    LlmAnalysis --> ErrorState: Error occurs
    ErrorState --> [*]: Log error and exit

```

## Key Steps:

1. **File Loading:** Loads the input file into the system.
2. **Preprocessing:** Extracts metadata and preprocesses the text content.
3. **Text Segmentation:** Splits the text into manageable segments.
4. **Tokenization:** Breaks down segments into individual tokens.
5. **NLP Analysis:** Performs part-of-speech tagging, dependency parsing, and named entity recognition.
6. **Topic Modeling:** Infers topics from the processed text.
7. **LLM Analysis:** Uses a language model to generate insights about the text.
8. **Result Display:** Presents the analysis results to the user.

# TopicModelTrainerWorkflow

This workflow is designed to process multiple files in batches and train a topic model.

## Key Steps:

1. **Batch Processing:** Processes files in batches of a defined size.
2. **File Loading:** Loads each file in the batch.
3. **Preprocessing:** Extracts metadata and preprocesses each file's content.
4. **Text Segmentation:** Splits each file's content into segments.
5. **Tokenization:** Breaks down segments into tokens.
6. **NLP Analysis:** Performs NLP tasks on the tokenized segments.
7. **Filtering:** Filters segments based on predefined criteria.
8. **Accumulation:** Accumulates filtered segments across all processed files.
9. **Topic Model Training:** Trains a topic model using the accumulated segments.

## Workflow Execution

Both workflows use the `WorkflowOrchestrator` class to manage task execution. The orchestrator:

1. Initializes the workflow and its tasks.
2. Sets up the task graph based on the workflow definition.
3. Executes tasks in the defined order.
4. Manages data flow between tasks using Redis for temporary storage.

5. Handles errors and exceptions during workflow execution.

## Workflow Flexibility

The workflow system is designed to be flexible and extensible:

- New workflows can be easily added by creating new workflow classes.
- Existing workflows can be modified by adding, removing, or reordering tasks.
- Tasks are modular and can be reused across different workflows.

This flexibility allows **Flowbots** to adapt to various text processing and analysis needs.

```
sequenceDiagram
    participant User
    participant TextProcessingWorkflow
    participant UnifiedFileProcessingPipeline
    participant WorkflowOrchestrator
    participant TextTaggerTask
    participant TopicModelingTask
    participant LlmAnalysisTask
    participant DisplayResultsTask
    participant Logger
    participant UI

    User->>TextProcessingWorkflow: Initialize with
    alt No file path provided
        TextProcessingWorkflow->>User: Prompt for
        User->>TextProcessingWorkflow: Provide file path
    end
    TextProcessingWorkflow->>UnifiedFileProcessingPipeline: Initialize
    TextProcessingWorkflow->>Logger: Log workflow
    TextProcessingWorkflow->>UI: Display workflow
    TextProcessingWorkflow->>UnifiedFileProcessingPipeline: Process file
    UnifiedFileProcessingPipeline-->>TextProcessingWorkflow: Return results
    TextProcessingWorkflow->>WorkflowOrchestrator: Execute tasks
    TextProcessingWorkflow->>WorkflowOrchestrator: Execute tasks
    WorkflowOrchestrator->>TextTaggerTask: Execute task
    TextTaggerTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>TopicModelingTask: Execute task
    TopicModelingTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>LlmAnalysisTask: Execute task
    LlmAnalysisTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>DisplayResultsTask: Execute task
    DisplayResultsTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator-->>TextProcessingWorkflow: Return results
    TextProcessingWorkflow->>Logger: Log workflow
    TextProcessingWorkflow->>UI: Display completion
    TextProcessingWorkflow-->>User: Workflow finished
```

```
sequenceDiagram
    actor User
    participant CLI
    participant TextProcessingWorkflow
    participant WorkflowOrchestrator
    participant FileLoaderTask
    participant PreprocessTextFileTask
    participant TextSegmentTask
    participant TokenizeSegmentsTask
    participant NlpAnalysisTask
    participant TopicModelingTask
    participant LlmAnalysisTask
    participant DisplayResultsTask
    participant Redis
    participant Textfile

    User->>CLI: process_text(file)
    activate CLI
    CLI->>TextProcessingWorkflow: new(input_file_path)
    activate TextProcessingWorkflow
```



```

TextProcessingWorkflow->>WorkflowOrchestrator: activate
TextProcessingWorkflow->>WorkflowOrchestrator: activate WorkflowOrchestrator
WorkflowOrchestrator->>FileLoaderTask: execute
activate FileLoaderTask
FileLoaderTask->>Redis: set("current_textfile")
FileLoaderTask->>Textfile: create
deactivate FileLoaderTask
WorkflowOrchestrator->>PreprocessTextFileTask: activate
activate PreprocessTextFileTask
PreprocessTextFileTask->>Textfile: update(preprocessed_text)
deactivate PreprocessTextFileTask
WorkflowOrchestrator->>TextSegmentTask: execute
activate TextSegmentTask
TextSegmentTask->>Textfile: add_segments()
deactivate TextSegmentTask
WorkflowOrchestrator->>TokenizeSegmentsTask: activate
activate TokenizeSegmentsTask
TokenizeSegmentsTask->>Textfile: update_segments()
deactivate TokenizeSegmentsTask
WorkflowOrchestrator->>NlpAnalysisTask: execute
activate NlpAnalysisTask
NlpAnalysisTask->>Textfile: update_segments_with_nlp()
deactivate NlpAnalysisTask
WorkflowOrchestrator->>TopicModelingTask: execute
activate TopicModelingTask
TopicModelingTask->>Textfile: add_topics()
deactivate TopicModelingTask
WorkflowOrchestrator->>LlmAnalysisTask: execute
activate LlmAnalysisTask
LlmAnalysisTask->>Textfile: update(analysis)
deactivate LlmAnalysisTask
WorkflowOrchestrator->>DisplayResultsTask: execute
activate DisplayResultsTask
DisplayResultsTask->>Textfile: retrieve_data()
DisplayResultsTask-->>User: display results
deactivate DisplayResultsTask
deactivate WorkflowOrchestrator
TextProcessingWorkflow-->>CLI: workflow complete
deactivate TextProcessingWorkflow
CLI-->>User: display completion message
deactivate CLI

```

## Task Processors

**Flowbots** uses a variety of task processors to handle different aspects of text processing and analysis. These processors are modular and can be combined in workflows to create complex text processing pipelines.

## Key Task Processors

1. FileLoaderTask
2. Loads input files into the system.
3. Stores file content in Ohm models for further processing.
4. PreprocessTextFileTask
5. Extracts metadata from file content (e.g., YAML front matter in Markdown files).
6. Preprocesses the main content for further analysis.
7. TextSegmentTask
8. Splits preprocessed text into manageable segments.
9. Uses the TextSegmentProcessor for actual segmentation logic.
10. TokenizeSegmentsTask
11. Breaks down text segments into individual tokens.
12. Uses the TextTokenizeProcessor for tokenization.
13. NlpAnalysisTask
14. Performs various NLP tasks on tokenized segments.
15. Includes part-of-speech tagging, dependency parsing, and named entity recognition.
16. Uses the NLPProcessor which wraps the Spacy library for NLP operations.
17. FilterSegmentsTask
18. Filters processed segments based on predefined criteria.
19. Removes irrelevant or low-quality segments to improve analysis quality.
20. TopicModelingTask
21. Infers topics from processed text

- segments.
22. Uses the TopicModelProcessor which implements topic modeling algorithms.
  23. LlmAnalysisTask
  24. Utilizes a language model (via NanoBot) to generate insights about the text.
  25. Provides high-level analysis and summarization of the processed content.
  26. DisplayResultsTask
  27. Formats and displays the results of the text processing and analysis pipeline.

## Task Processor Architecture

Each task processor:

1. Inherits from `Jongleur::WorkerTask` or `Flowbots::BaseTask`.
2. Implements an `execute` method that performs the core task logic.
3. Uses Redis for temporary data storage and passing data between tasks.
4. Interacts with Ohm models for persistent data storage.
5. Includes error handling and logging for robust execution.

## Extensibility

The task processor system is designed to be easily extensible:

- New task processors can be added by creating new classes inheriting from `Jongleur::WorkerTask` or `Flowbots::BaseTask`.

- Existing task processors can be modified or extended to support new functionality.
- **Task** processors can be combined in different ways within workflows to create custom text processing pipelines.

This modular design allows **Flowbots** to adapt to various text processing and analysis requirements.

## **Flowbots** Detailed Operation

### 1. Workflow Initialization

When a user selects a workflow through the CLI, the system initializes the chosen workflow (e.g., `TextProcessingWorkflow` or `TopicModelTrainerWorkflow`). The **WorkflowOrchestrator** sets up the task graph based on the workflow definition.

### 2. Task Execution

The **WorkflowOrchestrator** executes tasks in the defined order. Each task follows a similar pattern:

1. Retrieve necessary data from Redis or Ohm models.
2. Process the data using specialized processors (e.g., `NLPProcessor`, `TopicModelProcessor`).
3. Store the results back in Redis (for temporary storage) or Ohm models (for persistence).

### 3. Data Flow

- Redis is used for storing temporary data and passing information between tasks. This includes file IDs, current batch information, and intermediate processing results.
- Ohm models, backed by Redis, are used for persistent storage of document information, segments, tokens, and analysis results.

### 4. NLP and **Topic** Modeling

- The `NlpAnalysisTask` uses the ruby-spacy gem to perform tasks like tokenization, part-of-speech tagging, and named entity recognition.
- The `TopicModelingTask` uses the tomoto gem to implement topic modeling algorithms.

### 5. LLM Integration

The `LlmAnalysisTask` integrates with external language models through the NanoBot system. This allows for high-level analysis and insights generation based on the processed text data.

### 6. Error Handling and **Logging**

Each task and the `WorkflowOrchestrator` include error handling mechanisms. Errors are caught, logged, and in some cases, trigger the `ExceptionAgent` for detailed error analysis.

### 7. Batch Processing

For the `TopicModelTrainerWorkflow`, files are processed in batches. The `WorkflowOrchestrator` manages the batch state, ensuring all files in a batch are processed before moving to the next batch.

## 8. Result Presentation

The `DisplayResultsTask` formats the analysis results and presents them to the user through the CLI. This may include summaries, topic distributions, and insights generated by the LLM.

## Key Interactions

1. `CLI <-> WorkflowOrchestrator`: The CLI initiates workflow execution and receives final results.
2. `WorkflowOrchestrator <-> Tasks`: The orchestrator manages task execution order and handles task results.
3. `Tasks <-> Redis`: Tasks use Redis for short-term storage and inter-task communication.
4. `Tasks <-> Ohm Models`: Tasks interact with Ohm models for persistent storage of document data and analysis results.
5. `NLP and Topic Modeling Tasks <-> External Libraries`: These tasks utilize external Ruby gems for specialized processing.
6. `LlmAnalysisTask <-> NanoBot`: This task interacts with the NanoBot system to leverage external language models.

This architecture allows `Flowbots` to process text data through a series of specialized tasks, each building upon the

results of previous tasks, to provide comprehensive text analysis and insights.

## Ruby Gems Used in **Flowbots**

**Flowbots** leverages a variety of Ruby gems to provide its functionality. Here's a comprehensive list of the gems used in the project, along with their purposes:

1. `jongleur`
2. Purpose: Workflow orchestration and task management
3. Usage: Core component for defining and executing task workflows
4. `ohm`
5. Purpose: **Object**-hash mapping for Redis
6. Usage: Data persistence layer for storing document information and workflow states
7. `redis`
8. Purpose: In-memory data structure store
9. Usage: Temporary data storage and passing data between tasks
10. `json`
11. Purpose: JSON parsing and generation
12. Usage: Handling JSON data throughout the application
13. `parallel`
14. Purpose: Parallel processing
15. Usage: Potential use for parallel execution of tasks (not prominently used in the current implementation)
16. `pry` and `pry-stack_explorer`
17. Purpose: Enhanced REPL and debugging

- tools
- 18. Usage: Development and debugging
- 19. ruby-spacy
- 20. Purpose: Ruby bindings for the Spacy NLP library
- 21. Usage: Natural Language Processing tasks
- 22. thor
- 23. Purpose: Building command-line interfaces
- 24. Usage: Creating the CLI for **Flowbots**
- 25. treetop
- 26. Purpose: parsing expression grammar (PEG) parser generator
- 27. Usage: Custom grammar parsing, particularly for Markdown with YAML front matter
- 28. yaml
  - Purpose: YAML parsing and generation
  - Usage: Handling YAML data, particularly in configuration files and document front matter
- 29. faraday and faraday/multipart
  - Purpose: HTTP client library
  - Usage: Making HTTP requests, potentially for integrations with external services
- 30. logging
  - Purpose: Flexible logging
  - Usage: Comprehensive logging system throughout the application
- 31. tty-box, tty-cursor, tty-prompt, tty-



screen, tty-spinner, tty-table

- Purpose: Various terminal output formatting and interaction tools
- Usage: Creating rich command-line interfaces and displaying formatted output

### 32. pastel

- Purpose: Terminal output styling
- Usage: Adding colors and styles to terminal output

### 33. highline

- Purpose: High-level command-line interface building
- Usage: Additional CLI features and user input handling

### 34. cli-ui

- Purpose: CLI user interface components
- Usage: Enhancing the command-line interface with advanced **UI** elements

### 35. kramdown

- Purpose: Markdown parsing and conversion
- Usage: Handling Markdown content in documents

### 36. lingua

- Purpose: Natural language detection and processing
- Usage: Additional NLP capabilities

### 37. pragmatic\_segmenter

- Purpose: Text segmentation
- Usage: Splitting text into meaningful segments

#### 38. `pragmatic_tokenizer`

- Purpose: Text tokenization
- Usage: Breaking text into individual tokens

#### 39. `tomoto`

- Purpose: **Topic** modeling
- Usage: Implementing topic modeling algorithms

#### 40. `minitest` and `minitest/rg`

- Purpose: Testing framework
- Usage: Writing and running tests for the application

These gems provide a robust foundation for **Flowbots**, covering areas such as data persistence, natural language processing, command-line interfaces, HTTP communications, and more. The combination of these tools allows **Flowbots** to offer a comprehensive text processing and analysis system with a user-friendly interface.

Validate

Generated by **RDoc** 6.4.0.

Based on **Darkfish** by **Michael Granger**.

```
::initialize  
::load_components  
::setup_redis  
::shutdown  
::stop_running_workflows
```

# module Flowbots

Module for `Flowbots` application.

---

## Constants

### BATCH

Constant indicating whether the application is running in batch mode.

### IN\_CONTAINER

Constant indicating whether the application is running in a container.

---

## Public Class Methods

### `initialize()`

Initializes the `Flowbots` application.  
@return [void]

### `shutdown()`

Shuts down the `Flowbots` application.  
@return [void]

---

## Private Class Methods

### `load_components()`

Loads the necessary components for the application.  
@return [void]

### **setup\_redis()**

Sets up the Redis connection for Ohm.

@return [void] @raise [Ohm::Error] If there is an error connecting to Redis.

### **stop\_running\_workflows()**

Stops any running workflows.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class Flowbots::AgentError

Error raised when there is a problem with an agent.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class Flowbots::APIError

Error raised when there is a problem with an `API` call.

Validate

Generated by `RDoc 6.4.0`.

Based on `Darkfish` by `Michael Granger`.

```
::new  
#discover_files  
#file_types_pattern  
#process_batch  
#process_files  
#prompt_for_folder
```

# class Flowbots::BatchProcessor

The BatchProcessor class provides a mechanism for processing files in batches. It is particularly useful when dealing with a large number of files, as it prevents potential memory issues and allows for more controlled and efficient processing.

---

## Attributes

**batch\_size** [R]

@return [Integer] The number of files to process in each batch.

**file\_types** [R]

@return [Array<String>] An array of file extensions to process.

**input\_folder\_path** [R]

@return [String] The path to the folder containing the files to be processed.

---

## Public Class Methods

**new(input\_folder\_path, batch\_size=10, file\_types=nil)**

Initializes a new BatchProcessor instance.

@param input\_folder\_path [String] The path to the folder containing the files to be processed. If not provided, it prompts the user to select a folder. @param batch\_size [Integer] The number of files to process in each batch. Defaults to 10. @param file\_types [Array<String>] An array of file

extensions to process. Defaults to text files: ['.txt', '.md', '.markdown'].

---

## Public Instance Methods

### **process\_files(&block)**

Processes the files in batches.

This method iterates through the files in the input folder, divides them into batches, and yields each file path to the provided block for processing.

@yieldparam file\_path [String] The path to the file being processed. @return [void]

---

## Private Instance Methods

### **discover\_files()**

Discovers all files within the input folder that match the specified file types.

@return [Array<String>] An array of file paths.

### **file\_types\_pattern()**

Constructs a regular expression pattern from the file\_types array.

@return [String] The regular expression pattern.

### **process\_batch(batch\_files) { |file\_path| ... }**

Processes a single batch of files.

@param batch\_files [Array<String>] An array of file paths for the current batch.

@yieldparam file\_path [String] The path to the file being processed. @return [void]



## `prompt_for_folder()`

Prompts the user to select a folder using the `gum file` command.

@return [String] The path to the selected folder. @raises [FlowbotError] If the selected folder does not exist.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::exit_on_failure?  
#process_text  
#train_topic_model  
#version  
#workflows
```

# class Flowbots::CLI

This class provides a command-line interface (CLI) for interacting with the Flowbots application.

---

## Public Class Methods

### exit\_on\_failure?()

Defines whether the CLI should exit with a non-zero status code when an error occurs.

@return [Boolean] True if the CLI should exit on errors, false otherwise.

---

## Public Instance Methods

### process\_text(file)

Processes a text file using the text processing workflow.

@param file [String] The path to the text file.

@return [void]

### **train\_topic\_model(folder)**

Trains a topic model using text files in the specified folder.

@param folder [String] The path to the folder containing the text files.

@return [void]

### **version()**

Displays the **Flowbots version** and Ruby environment information.

@return [void]

### **workflows()**

Lists available **workflows**, allows the user to select one, and runs it.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class Flowbots::ConfigurationError

Error raised when there is a problem with the configuration.

Validate

Generated by [RDoc](#) 6.4.0.

Based on [Darkfish](#) by [Michael Granger](#).

```
::new  
#extract_relevant_files  
#fallback_exception_report  
#format_exception_report  
#generate_exception_prompt  
#load_file_structure  
#process_exception  
#write_markdown_report
```



# class Flowbots::ExceptionAgent

This class handles exceptions in the **Flowbots** application.

---

## Public Class Methods

### **new()**

Initializes a **new** instance of the **ExceptionAgent** class.

@return [void]

**Calls superclass method **WorkflowAgent::new****

---

## Public Instance Methods

### **process\_exception(classname, exception)**

Processes an exception and generates a report.

@param classname [String] The name of the class where the exception occurred. @param exception [Exception] The exception object.

@return [String] The formatted exception report.

## Private Instance Methods

### `extract_relevant_files(exception)`

Extracts relevant files from the exception backtrace.

@param exception [Exception] The exception object.

@return [Hash] A hash of relevant file names and their content.

### `fallback_exception_report(exception_details)`

Generates a fallback exception report if the agent fails to generate a report.

@param exception\_details [Hash] A hash containing exception details.

@return [String] The fallback exception report.

**format\_exception\_report(agent\_response, exception\_details)**

Formats the exception report based on the agent's response.

@param agent\_response [String] The response from the exception handler agent. @param exception\_details [Hash] A hash containing exception details.

@return [String] The formatted exception report.

**generate\_exception\_prompt(exception\_details)**

Generates a prompt for the exception handler agent.

@param exception\_details [Hash] A hash containing exception details.

@return [String] The prompt for the agent.

**load\_file\_structure()**

Loads the file structure from the flowbots.json file.

@return [Hash] The file structure.

**write\_markdown\_report(report, exception\_details)**

Writes the exception report to a markdown file.

@param report [String] The exception report. @param exception\_details [Hash] A hash containing exception details.

@return [void]



Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class Flowbots::ExceptionHandler

This class handles exceptions in the **Flowbots** application.

---

## Public Class Methods

### handle\_exception(classname=nil, exception)

Handles an exception by generating a report and notifying relevant parties.

@param classname [String] The name of the class where the exception occurred. @param exception [Exception] The exception object.

@return [String] The formatted exception report.

### log\_exception(exception)

Logs an exception to the application's logger.

@param exception [Exception] The exception object.

@return [void]

### notify\_exception(report)

Notifies relevant parties about an exception.

@param report [String] The formatted exception report.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# module Flowbots::FileDiscovery

This module provides file discovery utilities for `Flowbots`.

---

## Constants

### `FILE_TYPES`

A constant hash defining file extensions grouped by their types.

---

## Public Class Methods

### `discover_files(directory)`

Discovers files in the given directory and groups them by type.

@param directory [String] The directory to search for files. @return [Hash] A hash where keys are file types and values are arrays of file paths.

### `file_count(files)`

Counts the number of files for each file type.

@param files [Hash] A hash where keys are file types and values are arrays of file paths. @return [Hash] A hash where keys are file types and values are the number of files of that type.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

---

```
::new  
#classify_file  
#extract_text  
#extract_text_json  
#parse_pdf  
#store_file_data
```

# class Flowbots::FileLoader

This class handles loading and processing text files.

---

## Attributes

**file\_data** [RW]

The **FileObject** object representing the loaded file.

---

## Public Class Methods

**new(file\_path)**

Initializes a new **FileLoader** instance.

@param file\_path [String] The path to the file to be loaded.

@return [void]

---

## Private Instance Methods

**classify\_file(file\_path)**

Classifies the file type based on its MIME type.

@param file\_path [String] The path to the file.

@return [Symbol] The file type, e.g., :text, :pdf, :image, etc.

### **extract\_text(file\_type, file\_path)**

Extracts the text content from a file based on its type.

@param file\_type [Symbol] The file type.

@param file\_path [String] The path to the file.

@return [String] The extracted text content.

### **extract\_text\_json(file\_path)**

### **parse\_pdf(file\_path)**

Parses a PDF file and extracts its text content.

@param file\_path [String] The path to the PDF file.

@return [String] The extracted text content.

### **store\_file\_data(file\_path, extracted\_text)**

Stores the file data in the database.

@param file\_path [String] The path to the file. @param extracted\_text [String] The extracted text content.

@return [FileObject] The **FileObject** object representing the stored file data.

Validate

Generated by **RDoc** 6.4.0.

Based on **Darkfish** by **Michael Granger**.

# class Flowbots::FileNotFoundError

Custom error class for workflow file not found.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.



# class Flowbots::FlowbotError

Base class for all `Flowbots` errors.

---

## Attributes

**details** [R]

@return [Hash] Additional `details` about the error.

**error\_code** [R]

@return [String] The error code.

---

## Public Class Methods

**new(message, error\_code, details={})**

Initializes a new `FlowbotError`.

@param message [String] The error message.

@param `error_code` [String] The error code.

@param `details` [Hash] Additional `details` about the error.

**Calls superclass method**

Validate

Generated by `RDoc 6.4.0`.

Based on `Darkfish` by `Michael Granger`.

```
..new  
#extract_markdown_content  
#extract_yaml_front_matter  
#load_grammar  
#parse
```

# class Flowbots::GrammarProcessor

This class handles parsing text using a specified grammar.

---

## Public Class Methods

### `new(grammar_name)`

Initializes a `new GrammarProcessor` instance.

@param grammar\_name [String] The name of the grammar to use for parsing.

@return [void]

---

## Public Instance Methods

### `parse(text)`

Parses the given text using the specified grammar.

@param text [String] The text to `parse`.

@return [Hash, nil] A hash containing the parsed YAML front matter and Markdown content, or nil if parsing fails.

## Private Instance Methods

### `extract_markdown_content(parse_result)`

Extracts the Markdown content from the `parse result`.

@param parse\_result  
[Treetop::Runtime::SyntaxNode] The `parse result`.

@return [String] The extracted Markdown content.

### `extract_yaml_front_matter(parse_result)`

Extracts the YAML front matter from the `parse result`.

@param parse\_result  
[Treetop::Runtime::SyntaxNode] The `parse result`.

@return [String] The extracted YAML front matter.

### `load_grammar()`

Loads the grammar file and creates a parser instance.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::new  
#create_doc  
#load_model  
#process
```

# class Flowbots::NLPProcessor

This class provides functionality for performing natural language processing (NLP) analysis on text.

---

## Public Class Methods

### `new()`

Initializes a `new NLPProcessor` instance.

@return [void]

---

## Public Instance Methods

### `process(segment, options={})`

Processes the given segment using the loaded NLP model and returns a hash of processed tokens.

@param segment [Segment] The `Segment` object to be processed. @param options [Hash] A hash of options for the NLP processing.

@return [Array] An array of processed tokens, or nil if the processing fails.

## Private Instance Methods

### `create_doc(segment)`

Creates a `Spacy::Doc` object from the given segment's tokens.

@param segment [Segment] The `Segment` object.

@return [Spacy::Doc] The `Spacy::Doc` object.

### `load_model()`

Loads the NLP model from the specified environment variable.

@return [void]

Validate

Generated by `RDoc` 6.4.0.

Based on `Darkfish` by Michael Granger.

# class Flowbots::Task

This module encapsulates tasks used in `Flowbots` workflows.

---

## Public Class Methods

### `load_tasks()`

Loads all task files from the `TASK_DIR` directory.

@return [void]

### `new(options={})`

Initializes a new `Task` instance.

@param options [Hash] A hash of options for the task.

@return [void]

---

## Public Instance Methods

### `execute()`

Executes the task.

This method must be implemented in subclasses.

@return [void] @raise [NotImplementedError]  
If the method is not implemented in a subclass.

Validate

Generated by `RDoc` 6.4.0.

Based on `Darkfish` by Michael Granger.

# class Flowbots::TaskNotFoundError

Custom error class for task not found.

Validate

Generated by [RDoc](#) 6.4.0.

Based on [Darkfish](#) by [Michael Granger](#).

Home
Pages Classes Methods
Parent
Object
Methods
<code>new</code> <code>#create_or_fetch_file_object</code> <code>#fetch_unprocessed_file_ids</code> <code>#perform_additional_tasks</code> <code>#process_batch</code> <code>#process_single_file</code> <code>#prompt_for_file</code> <code>#run</code>

# class Flowbots::TextProcessingWorkflow

This class defines a workflow for processing text files, either individually or in batch mode. It utilizes a `UnifiedFileProcessingPipeline` to handle the initial processing steps and then performs additional tasks like text tagging, topic modeling, LLM analysis, and result display.

## Attributes

`pipeline` [R]

@return [UnifiedFileProcessingPipeline] The `pipeline` responsible for the initial processing steps.

## Public Class Methods

`new(input_file_path=nil, batch_mode=false)`

Initializes a new `TextProcessingWorkflow` instance.

@param input\_file\_path [String, nil] The path to the input file. If nil, the user will be prompted to select a file. @param batch\_mode [Boolean] Whether to process files in batch mode (default: false).

@return [void]

## Public Instance Methods

`run()`

Runs the text processing workflow.

Sets up the workflow, processes the file(s), and performs additional tasks based on the batch mode.

@return [void]



## Private Instance Methods

### `create_or_fetch_file_object(file_path)`

Creates or fetches a `FileObject` for the given file path.

@param file\_path [String, Hash] The path to the file or a hash containing the path.

@return [FileObject] The created or fetched `FileObject`.

### `fetch_unprocessed_file_ids()`

Fetches the IDs of unprocessed files.

@return [Array<Integer>] An array of unprocessed file IDs.

### `perform_additional_tasks(file_id)`

Performs additional tasks for the given file ID.

Defines the workflow for additional tasks and runs the workflow using the orchestrator.

@param file\_id [Integer] The ID of the file to process.

@return [void]

### `process_batch()`

Processes files in batch mode.

Fetches unprocessed file IDs and performs additional tasks for each file.

@return [void]

### `process_single_file()`

Processes a single file.

Creates or fetches the `FileObject` for the input file and performs additional tasks.

@return [void]

## `prompt_for_file()`

Prompts the user to select a file using the `gum file` command.

@return [String] The path to the selected file.

@raises [FlowbotError] If the selected file does not exist.

Validate

Generated by `RDoc 6.4.0`.

Based on `Darkfish` by `Michael Granger`.

# class Flowbots::TextProcessor

This class provides a base class for text processors in the `Flowbots` application.

---

## Public Class Methods

### `new()`

Initializes a `new TextProcessor` instance.  
@return [void]

---

## Public Instance Methods

### `process(text)`

Processes the given text.

This method must be implemented in subclasses.

@param text [String] The text to be processed.

@return [void] @raise [NotImplementedError]  
If the method is not implemented in a subclass.

Validate

Generated by `RDoc 6.4.0`.

Based on `Darkfish` by `Michael Granger`.

# class Flowbots::TextSegmentProcessor

This class provides functionality for segmenting `text` into smaller units.

---

## Constants

### DEFAULT\_OPTIONS

Default `options` for the segmenter.

---

## Attributes

### `options` [RW]

The `options` for the segmenter.

### `text` [RW]

The `text` to be segmented.

---

## Public Class Methods

### `new()`

Initializes a new `TextSegmentProcessor` instance.

@return [void]

**Calls superclass method**

**Flowbots::TextProcessor::new**

---

## Public Instance Methods

### `process(text, opts={})`

Segments the given `text` using the specified `options`.

@param `text` [String, Array] The `text` to be segmented. @param `opts` [Hash] A hash of `options` for the segmenter.

@return [Array] An array of segments.

## Private Instance Methods

### `segment_array()`

Segments an array of `text`.

@return [Array] An array of segments.

### `segment_string(txt)`

Segments a single string.

@param txt [String] The `text` to be segmented.

@return [Array] An array of segments.

Validate

Generated by `RDoc 6.4.0`.

Based on `Darkfish` by `Michael Granger`.

```
new  
#analyze_transitivity  
#extract_main_topics  
#identify_speech_acts  
#load_engtagger  
#process
```

# class Flowbots::TextTaggerProcessor

This class provides functionality for tagging text using the EngTagger library.

---

## Public Class Methods

### new()

Initializes a new `TextTaggerProcessor` instance.

@return [void]

---

## Public Instance Methods

### analyze\_transitivity(text)

Analyzes the transitivity of sentences in the given text.

@param text [String] The text to analyze.

@return [Array] An array of hashes containing transitivity information for each sentence.

### extract\_main\_topics(text, limit=5)

Extracts the main topics from the given text.

@param text [String] The text to extract topics from. @param limit [Integer] The maximum number of topics to extract.

@return [Array] An array of main topics.

### **identify\_speech\_acts(text)**

Identifies the speech acts in the given text.

@param text [String] The text to analyze.

@return [Array] An array of speech act classifications for each sentence.

### **process(text, options={})**

Processes the given text using the EngTagger library and returns a hash of tagged results.

@param text [String] The text to be tagged.

@param options [Hash] A hash of options for the tagging **process**.

@return [Hash] A hash containing the tagged results.

---

## **Private Instance Methods**

### **load\_engtagger()**

Loads the EngTagger library.

@return [void]

**Validate**

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class Flowbots::TextTokenizeProcessor

This class provides functionality for tokenizing `text`.

---

## Constants

### DEFAULT\_OPTIONS

Default `options` for the tokenizer.

---

## Attributes

`options` [RW]

The `options` for the tokenizer.

`text` [RW]

The `text` to be tokenized.

---

## Public Class Methods

`new()`

Initializes a new `TextTokenizeProcessor` instance.

@return [void]

**Calls superclass method**

`Flowbots::TextProcessor::new`

---

## Public Instance Methods

`process(text, opts={})`

Tokenizes the given `text` using the specified `options`.

@param `text` [String, Array] The `text` to be tokenized. @param `opts` [Hash] A hash of `options` for the tokenizer.

@return [Array] An array of tokens.



## Private Instance Methods

### `tokenize_array()`

Tokenizes an array of strings.

@return [Array] An array of tokens.

### `tokenize_string(str)`

Tokenizes a single string.

@param str [String] The string to be tokenized.

@return [Array] An array of tokens.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::new  
#create_new_model  
#ensure_model_exists  
#infer_topics  
#load_existing_model  
#load_or_create_model  
#save_model  
#train_model
```



# class Flowbots::TopicModelProcessor

This class provides functionality for processing text using a topic `model`.

---

## Attributes

`model` [RW]

The Tomoto::LDA `model` object.

`model_params` [RW]

The parameters for the topic `model`.

`model_path` [RW]

The path to the topic `model` file.

---

## Public Class Methods

`new()`

Initializes a new `TopicModelProcessor` instance.

@return [void]

Calls superclass method  
`Flowbots::TextProcessor::new`

---

## Public Instance Methods

`infer_topics(document)`

Infers the topics for a given document.

@param document [String] The document to infer topics for.

@return [Hash] A hash containing the most probable topic, topic distribution, and top words for the document.

**load\_or\_create\_model()**

Loads an existing topic `model` or creates a `new` one if it doesn't exist.

@return [void]

**train\_model(documents, iterations=100)**

Trains a topic `model` using the provided documents.

@param documents [Array] An array of documents to train the `model` on. @param iterations [Integer] The number of iterations to train the `model` for.

@return [void]

## Private Instance Methods

### `create_new_model()`

Creates a `new` topic `model` with the specified parameters.

@return [void]

### `ensure_model_exists()`

Ensures that the topic `model` exists, loading or creating it if necessary.

@return [void]

### `load_existing_model()`

Loads an existing topic `model` from the specified path.

@return [void]

### `save_model()`

Saves the topic `model` to the specified path.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::new  
#clean_segments_for_modeling  
#prompt_for_folder  
#run  
#train_topic_model
```

# class Flowbots::TopicModelTrainerWorkflow

This class defines a workflow for training a topic model using a collection of text files. It utilizes a `UnifiedFileProcessingPipeline` to handle the file processing and segment filtering, and then trains a topic model using the filtered segments.

---

## Attributes

**pipeline** [R]

@return [UnifiedFileProcessingPipeline] The `pipeline` responsible for file processing and segment filtering.

---

## Public Class Methods

**new**(input\_folder\_path=nil)

Initializes a new `TopicModelTrainerWorkflow` instance.

@param input\_folder\_path [String, nil] The path to the folder containing the input files. If nil, the user will be prompted to select a folder.

@return [void]

---

## Public Instance Methods

**run**()

Runs the topic model trainer workflow.

Sets up the workflow, processes the files, and trains the topic model.

@return [void]

---

## Private Instance Methods

**clean\_segments\_for\_modeling**(segments)

Cleans the segments for topic modeling by removing unwanted segments and words.

@param segments [Array<Array<String>>] The segments to clean.

@return [Array<Array<String>>] The cleaned segments.

### `prompt_for_folder()`

Prompts the user to select a folder using the `gum file` command.

@return [String] The path to the selected folder. @raises [FlowbotError] If the selected folder does not exist.

### `train_topic_model()`

Trains the topic model using the filtered segments from the processed files.

Retrieves the filtered segments from Redis, cleans them, trains the topic model, and logs the progress.

@return [void]

Validate

Generated by `RDoc` 6.4.0.

Based on `Darkfish` by Michael Granger.

Home
Pages Classes Methods
Parent
Object
Methods
<code>__new__</code> <code>#clean_segments_for_modeling</code> <code>#flush_redis_cache</code> <code>#process_batch</code> <code>#process_files</code> <code>#prompt_for_folder</code> <code>#run</code> <code>#setup_workflow</code> <code>#train_topic_model</code>

# class Flowbots::TopicModelTrainerWorkflowtest

## Constants

`BATCH_SIZE`

## Attributes

`input_folder_path` [R]

`orchestrator` [RW]

## Public Class Methods

`new(input_folder_path=nil)`

## Public Instance Methods

`run()`

## Private Instance Methods

`clean_segments_for_modeling(segments)`

`flush_redis_cache()`

`process_batch(batch_files)`

`process_files()`



```
prompt_for_folder()
```

```
setup_workflow()
```

```
train_topic_model()
```

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
new  
#flush_redis_cache  
#process  
#process_batch  
#process_file  
#process_single_file  
#setup_workflow
```

# class Flowbots::UnifiedFileProcessingPipeline

This class defines a pipeline for processing files, either individually or in batches. It utilizes a `WorkflowOrchestrator` to manage the execution of tasks related to file processing.

## Attributes

`batch_processor` [R]

@return [BatchProcessor] The batch processor for handling multiple files.

`orchestrator` [R]

@return [WorkflowOrchestrator] The `orchestrator` responsible for managing the workflow.

## Public Class Methods

`new(input_path, batch_size: 10, file_types: %w[md markdown txt pdf json])`

Initializes a new `UnifiedFileProcessingPipeline` instance.

@param input\_path [String] The path to the file or directory to be processed. @param batch\_size [Integer] The number of files to `process` in each batch (default: 10). @param file\_types [Array<String>] An array of file extensions to `process` (default: ['md', 'markdown', 'txt', 'pdf', 'json']).

@return [void]

## Public Instance Methods

`process()`

Processes the file(s) specified in the input path.

@return [void]

## Private Instance Methods

`flush_redis_cache()`

Flushes the Redis cache.

@return [void]

### **process\_batch()**

Processes a batch of files using the batch processor.

@return [void]

### **process\_file(file\_path)**

Processes a single file by setting the current file path in Redis and running the workflow.

@param file\_path [String] The path to the file to be processed.

@return [void]

### **process\_single\_file()**

Processes a single file.

@return [void]

### **setup\_workflow()**

Sets up the workflow by defining the task graph and adding agents to the **orchestrator**.

@return [void]

Validate

Generated by **RDoc** 6.4.0.

Based on **Darkfish** by **Michael Granger**.

# class Flowbots::WorkflowError

Error raised when there is a problem with a workflow.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::load_workflows
::new
#display_workflows
#extract_workflow_description
#get_workflows
#list_and_select
#run
#select_workflow
```

# class Flowbots::Workflows

This class manages workflows in the **Flowbots** application.

---

## Public Class Methods

### load\_workflows()

Class method to load all workflow files from the WORKFLOW\_DIR directory. It also checks for user-defined workflows in a custom directory.

@return [void]

### new()

Initializes a new **Workflows** instance.

@return [void]

---

## Public Instance Methods

### list\_and\_select()

Lists available workflows and allows the user to select one.

@return [String, nil] The name of the selected workflow, or nil if no workflow is selected.

### run(workflow\_name)

Runs the specified workflow.

@param workflow\_name [String] The name of the workflow to **run**.

@return [void] @raise [FileNotFoundError] If the workflow file is not found.

## Private Instance Methods

### `display_workflows(workflows)`

Displays a list of available workflows in a table format.

@param workflows [Array<Array(String, String)>] An array of arrays, where each inner array contains the workflow name and its description.

@return [void]

### `extract_workflow_description(file)`

Extracts the description of a workflow from its file. The description is assumed to be the first line of the file starting with “# Description:”.

@param file [String] The path to the workflow file.

@return [String] The workflow description, or “No description available” if not found.

### `get_workflows()`

Retrieves a list of available workflows from the WORKFLOW\_DIR directory.

@return [Array<Array(String, String)>] An array of arrays, where each inner array contains the workflow name and its description.

### `select_workflow(workflows)`

Prompts the user to select a workflow from the list of available workflows.

@param workflows [Array<Array(String, String)>] An array of arrays, where each inner array contains the workflow name and its description.

```
@return [String, nil] The name of the  
selected workflow, or nil if no workflow is  
selected.
```

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.



# class Jongleur::WorkerTask

Define a Redis connection for  
`Jongleur::WorkerTask`

`Jongleur::WorkerTask` is a class that  
defines a task to be executed by `Jongleur`.

Validate

Generated by `RDoc 6.4.0`.

Based on `Darkfish` by `Michael Granger`.

Home
Pages Classes Methods
Included Modules
Treetop::Runtime
Methods
<div>#_nt_document</div> <div>#_nt_markdown_content</div> <div>#_nt_newline</div> <div>#_nt_yaml_front_matter</div> <div>#root</div>

# module MarkdownYaml

Autogenerated from a Treetop grammar. Edits may be lost.

## Public Instance Methods

### \_nt\_document()

Parses the document node.

@return [Treetop::Runtime::SyntaxNode] The parsed document node.

### \_nt\_markdown\_content()

Parses the Markdown content node.

@return [Treetop::Runtime::SyntaxNode] The parsed Markdown content node.

**\_nt\_newline()**

Parses the newline node.

@return [Treetop::Runtime::SyntaxNode] The  
parsed newline node.

**\_nt\_yaml\_front\_matter()**

Parses the YAML front matter node.

@return [Treetop::Runtime::SyntaxNode] The  
parsed YAML front matter node.

**root()**

The **root** node of the grammar.

@return [Treetop::Runtime::SyntaxNode] The  
**root** node of the grammar.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

#markdown\_content  
#yaml\_front\_matter

# module MarkdownYaml::Document0

The Document node represents the entire document structure. It contains the YAML front matter and the Markdown content.

---

## Public Instance Methods

### markdown\_content()

The Markdown content section of the document.

@return [Treetop::Runtime::SyntaxNode] The Markdown content node.

### yaml\_front\_matter()

The YAML front matter section of the document.

@return [Treetop::Runtime::SyntaxNode] The YAML front matter node.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# module MarkdownYaml::YamlFrontMatter0

The YamlFrontMatter node represents the YAML front matter section.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

#newline1  
#newline2

# module MarkdownYaml::YamlFrontMatter1

The YamlFrontMatter node represents the YAML front matter section.

---

## Public Instance Methods

### newline1()

The first newline character after the “—” delimiter.

@return [Treetop::Runtime::SyntaxNode] The first newline node.

### newline2()

The second newline character after the “—” delimiter.

@return [Treetop::Runtime::SyntaxNode] The second newline node.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

[blueprints.sublayer.com/blueprints/70562717-70c5-4406-a792-358d169f9f0b](https://blueprints.sublayer.com/blueprints/70562717-70c5-4406-a792-358d169f9f0b)

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.



Home
Pages Classes Methods
Parent
Base
Methods
<code>new</code> <code>#call</code>

class

Sublayer::Actions::RunTestCommandAction

Public Class Methods

`new(test_command:)`

Public Instance Methods

`call()`

Validate

Generated by `RDoc` 6.4.0.

Based on `Darkfish` by Michael Granger.

Home
Pages Classes Methods
Parent
Base
Methods
<code>new</code> <code>#call</code>

# class Sublayer::Actions::SpeechToTextAction

---

## Public Class Methods

`new(audio_data)`

---

## Public Instance Methods

`call()`

Validate  
Generated by `RDoc` 6.4.0.  
Based on `Darkfish` by Michael Granger.

Home
Pages Classes Methods
Parent
Base
Methods
<code>new</code> <code>#call</code>

# class Sublayer::Actions::TextToSpeechAction

---

## Public Class Methods

`new(text)`

---

## Public Instance Methods

`call()`

Validate  
Generated by `RDoc` 6.4.0.  
Based on `Darkfish` by Michael Granger.

# class Sublayer::Actions::WriteFileAction

---

## Public Class Methods

**new(file\_contents:, file\_path:)**

Initializes the action with the contents to write and the target file path @param [String] file\_contents the contents to write to the file @param [String] file\_path the file path where contents will be written

---

## Public Instance Methods

**call()**

Writes the contents to the file in binary mode @return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

Validate

Generated by [RDoc](#) 6.4.0.

Based on [Darkfish](#) by [Michael Granger](#).

Home
Pages Classes Methods
Parent
Prompt
Methods
<code>new</code> <code>#readline</code>

# class TTY::PromptX

## Attributes

`prefix` [R]

## Public Class Methods

`new(active_color:, prefix:, history: true)`

Calls superclass method

## Public Instance Methods

`readline(text = "")`

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.



# class TTY::Markdown::Converter

Converts a Kramdown::Document tree to a terminal friendly output

---

## Public Instance Methods

`convert_p(ell, opts)`

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

#footer  
#header  
#info  
#main\_menu  
#prompt  
#say  
#spinner

# module UI

This module provides user interface (**UI**) elements and functions for the **Flowbots** application.

This module provides methods for creating and displaying boxes in the **UI**.

This module provides methods for creating and displaying scrollable boxes in the **UI**.

---

## Constants

### PASTEL

An instance of the `Pastel` gem for colorizing text.

### TITLE\_WIDTH

The width of the title box.

---

## Public Instance Methods

### footer()

Displays the **Flowbots footer** in a framed box.

@return [void]

### header()

Displays the **Flowbots header** in a framed box.

@return [void]

### **info(text)**

Displays an information message in a framed box.

@param text [String] The text to display in the **info** box.

@return [void]

### **main\_menu()**

Displays the main menu and prompts the user for a choice.

@return [Symbol] The value of the selected choice.

### **prompt()**

Returns the TTY::Prompt instance used for user interaction.

@return [TTY::Prompt] The TTY::Prompt instance.

### **say(type, statement)**

Displays a message to the user with the specified type and logs it.

@param type [Symbol] The type of message (:ok, :warn, :error, or nil). @param statement [String] The message to display.

@return [void]

## `spinner(text)`

Creates and returns a `TTY::Spinner` instance with the specified text.

@param text [String] The text to display next to the `spinner`.

@return [TTY::Spinner] The `TTY::Spinner` instance.

Validate

Generated by `RDoc` 6.4.0.

Based on `Darkfish` by Michael Granger.

#comparison\_box  
#eval\_result\_box  
#exception\_box  
#info\_box  
#multi\_column\_box

# module UI::Box

This module provides methods for creating and displaying boxes in the **UI**.

## Public Instance Methods

**comparison\_box(text1, text2, title1: "Text 1", title2: "Text 2")**

Creates a box containing two texts side-by-side for comparison.

@param text1 [String] The first text to display. @param text2 [String] The second text to display. @param title1 [String] The title for the first text box (default: "Text 1"). @param title2 [String] The title for the second text box (default: "Text 2").

@return [String] The combined box containing both texts.

**eval\_result\_box(result, title: "Evaluation Result")**

Creates a box displaying the evaluation result with a success style.

@param result [String] The evaluation result to display. @param title [String] The title for the box (default: "Evaluation Result").

@return [String] The box containing the evaluation result.

### **exception\_box(message)**

Creates a box displaying an exception message with an error style.

@param message [String] The exception message to display.

@return [String] The box containing the exception message.

### **info\_box(message, title: "Info")**

Creates a box displaying an information message with an info style.

@param message [String] The information message to display. @param title [String] The title for the box (default: "Info").

@return [String] The box containing the information message.

### **multi\_column\_box(data, titles)**

Creates a box displaying data in multiple columns with headers.

@param data [Array<Array>] A 2D array of data to display in the columns. @param titles [Array<String>] An array of titles for the columns.

@return [String] The box containing the multi-column data.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::create_scrollable_box  
::display_boxes  
::print_boxes  
::print_navigation_info  
#side_by_side_boxes
```

# module UI::ScrollableBox

This module provides methods for creating and displaying scrollable boxes in the [UI](#).

---

## Private Class Methods

**create\_scrollable\_box(text, width, height, title)**

Creates a scrollable box data structure.

@param text [String] The text to display in the box. @param width [Integer] The width of the box. @param height [Integer] The height of the box. @param title [String] The title of the box.

@return [Hash] A hash containing the box data.

**display\_boxes(box1, box2, box\_height)**

Displays the scrollable boxes and handles user navigation.

@param box1 [Hash] The data for the first box. @param box2 [Hash] The data for the second box. @param box\_height [Integer] The height of the boxes.

@return [void]



```
print_boxes(box1, box2, box_height)
```

Prints the scrollable boxes to the console.

@param box1 [Hash] The data for the first box. @param box2 [Hash] The data for the second box. @param box\_height [Integer] The height of the boxes.

```
@return [void]
```

```
print_navigation_info(box1, box2)
```

Prints navigation information for the scrollable boxes.

@param box1 [Hash] The data for the first box. @param box2 [Hash] The data for the second box.

```
@return [void]
```

---

## Public Instance Methods

```
side_by_side_boxes(text1, text2, title1: "Box 1", title2: "Box 2")
```

Creates and displays two scrollable boxes side-by-side for comparison.

@param text1 [String] The text to display in the first box. @param text2 [String] The text to display in the second box. @param title1 [String] The title for the first box (default: "Box 1"). @param title2 [String] The title for the second box (default: "Box 2").

```
@return [void]
```

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class AccumulateFilteredSegmentsTask

**Task** to accumulate filtered segments from all processed files.

---

## Public Instance Methods

### execute()

Executes the task to accumulate and clean filtered segments.

Retrieves filtered segments from Redis, cleans them, accumulates them, updates the **FileObject** with the cleaned segments, and logs the progress.

@return [void]

---

## Private Instance Methods

### clean\_segments(segments)

Cleans the given segments by removing unwanted segments and words.

@param segments [Array<Array<String>>] The segments to clean.

@return [Array<Array<String>>] The cleaned segments.

### retrieve\_input()

Retrieves the input for the task, which is the current **FileObject**.

@return [FileObject] The current **FileObject**.

### update\_file\_object(cleaned\_segments)

Updates the **FileObject** with the given cleaned segments.

@param cleaned\_segments [Array<Array<String>>] The cleaned segments to add to the **FileObject**.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class API

... (Ohm model definitions from OhmModels.rb)  
...

---

## Private Instance Methods

### splat\_sort(splat\_vals)

... (Add routes for other object buckets and their attributes) ...

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

Home
Pages Classes Methods
Parent
Jongleur::WorkerTask
Methods
#execute

# class CompressionTask

This task compresses a prompt using a `WorkflowAgent`.

---

## Public Instance Methods

### `execute()`

Executes the task.

@return [void] @raises [StandardError] If an error occurs during the task execution.

Validate  
Generated by `RDoc 6.4.0`.  
Based on `Darkfish` by `Michael Granger`.

Home
Pages Classes Methods
Parent
Jongleur:WorkerTask
Methods
#execute

# class CompressionTestAssessmentTask

This task assesses a compression test evaluation using a `WorkflowAgent`.

---

## Public Instance Methods

### `execute()`

Executes the task.

@return [void] @raises [StandardError] If an error occurs during the task execution.

Validate  
Generated by `RDoc 6.4.0`.  
Based on `Darkfish` by `Michael Granger`.

# class CompressionTestEvalTask

This task evaluates a compression test design using a `WorkflowAgent`.

---

## Public Instance Methods

### `execute()`

Executes the task.

@return [void] @raises [StandardError] If an error occurs during the task execution.

Validate

Generated by `RDoc` 6.4.0.

Based on `Darkfish` by `Michael Granger`.



# class CompressionTestTask

This task designs a test for a compressed prompt using a `WorkflowAgent`.

---

## Public Instance Methods

### `execute()`

Executes the task.

@return [void] @raises [StandardError] If an error occurs during the task execution.

Validate

Generated by `RDoc` 6.4.0.

Based on `Darkfish` by `Michael Granger`.

Home
Pages Classes Methods
Parent
Object
Methods
::pos

# class Cursor

## Public Class Methods

pos()

Validate  
Generated by RDoc 6.4.0.  
Based on Darkfish by Michael Granger.

```
#display_results  
#execute  
#format_analysis  
#format_file_info  
#retrieve_input
```

# class DisplayResultsTask

This task displays the results of the text processing workflow.

---

## Public Instance Methods

### `execute()`

Executes the task to display the results of the text processing workflow.

Retrieves the processed Textfile object and its LLM analysis results. Formats and displays the file information and analysis results in a user-friendly format.

@return [void]

---

## Private Instance Methods

### `display_results(textfile, analysis_result)`

Displays the results of the text processing workflow.

@param textfile [Textfile] The processed Textfile object. @param analysis\_result [String, Hash] The LLM analysis results.

@return [void]

### `format_analysis(analysis_result)`

Formats the analysis results for display.

@param analysis\_result [String, Hash] The LLM analysis results.

@return [String] The formatted analysis results.

### `format_file_info(textfile)`

Formats the file information for display.

@param textfile [Textfile] The processed Textfile object.

@return [String] The formatted file information.

### `retrieve_input()`

Retrieves the input for the task, which is the current `FileObject`.

@return [FileObject] The current `FileObject`.

Validate

Generated by `RDoc` 6.4.0.

Based on `Darkfish` by Michael Granger.

```
def new
  #format_exception
  #process_exception
  #store_exception_report
```

# class ExceptionAgent

## Public Class Methods

```
new(agent_role, cartridge_file)
```

## Public Instance Methods

```
process_exception(exception)
```

## Private Instance Methods

```
format_exception(exception)
```

```
store_exception_report(report)
```

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
#execute
#retrieve_input
#store_FileObject_id
```

# class FileLoaderTask

This task loads a text file and stores its ID in Redis.

---

## Public Instance Methods

### execute()

Executes the task to load a `FileObject` and store its ID in Redis.

Retrieves the input file path, processes the file using `Flowbots::FileLoader`, stores the `FileObject` ID in Redis, and logs the progress.

@return [void] @raises [FlowbotError] If the `FileObject` is not found or its ID is nil.

---

## Private Instance Methods

### retrieve\_input()

Retrieves the input file path from Redis.

@return [String] The input file path.

### store\_FileObject\_id(id)

Stores the `FileObject` ID in Redis.

@param id [Integer] The ID of the `FileObject`.

@return [void]

Validate

Generated by `RDoc 6.4.0`.

Based on `Darkfish` by `Michael Granger`.

Home
Pages Classes Methods
Parent
Ohm::Model
Included Modules
Ohm::DataTypes Ohm::Callbacks
Methods
<code>::current_batch</code> <code>::find_or_create_by_path</code> <code>::latest</code> <code>#add_lemma</code> <code>#add_lemmas</code> <code>#add_segment</code> <code>#add_segments</code> <code>#add_topics</code> <code>#after_delete</code> <code>#after_save</code> <code>#retrieve_segment_texts</code> <code>#retrieve_segments</code> <code>#retrieve_word_texts</code> <code>#retrieve_words</code>

# class FileObject

## Public Class Methods

`current_batch()`

`find_or_create_by_path(file_path, attributes={})`

`latest(limit=nil)`

## Public Instance Methods

`add_lemma(lemma_data)`

`add_lemmas(lemmas_data)`

`add_segment(text)`

`add_segments(new_segments)`

`add_topics(new_topics)`

`retrieve_segment_texts()`

`retrieve_segments()`

`retrieve_word_texts()`

`retrieve_words()`

---

## Protected Instance Methods

`after_delete()`

`after_save()`

Validate

Generated by **RDoc** 6.4.0.

Based on **Darkfish** by **Michael Granger**.



Home
Pages Classes Methods
Parent
Jongleur::WorkerTask
Included Modules
InputRetrieval
Methods
#display_filtered_segments #execute #filter_segment_words #filter_segments #retrieve_input

# class FilterSegmentsTask

lib/tasks/filter\_segments\_task.rb

## Public Instance Methods

execute()

## Private Instance Methods

display\_filtered\_segments(filtered\_segments)

filter\_segment\_words(segment)

filter\_segments(file\_object)

retrieve\_input()

# class FinalReportTask

This task generates a final report using a `WorkflowAgent`.

---

## Public Instance Methods

### `execute()`

Executes the task.

@return [void] @raises [StandardError] If an error occurs during the task execution.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::initialize  
::load_components  
::setup_redis  
::shutdown  
::stop_running_workflows
```

# module Flowbots

Module for `Flowbots` application.

---

## Constants

### BATCH

Constant indicating whether the application is running in batch mode.

### IN\_CONTAINER

Constant indicating whether the application is running in a container.

---

## Public Class Methods

### `initialize()`

Initializes the `Flowbots` application.  
@return [void]

### `shutdown()`

Shuts down the `Flowbots` application.  
@return [void]

---

## Private Class Methods

### `load_components()`

Loads the necessary components for the application.  
@return [void]

## **setup\_redis()**

Sets up the Redis connection for Ohm.

@return [void] @raise [Ohm::Error] If there is an error connecting to Redis.

## **stop\_running\_workflows()**

Stops any running workflows.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::new  
#handle_response  
#predict  
#upsert_document
```

# class FlowiseApiClient

This class provides an interface for interacting with the Flowise [API](#).

---

## Public Class Methods

### `new(base_url)`

Initializes a new `FlowiseApiClient` instance.

@param base\_url [String] The base URL of the Flowise [API](#).

@return [void]

---

## Public Instance Methods

### `predict(chatflow_id, options={})`

Sends a prediction request to the Flowise [API](#).

@param chatflow\_id [String] The ID of the chatflow to use for prediction. @param options [Hash] A hash of options for the prediction request. - :question [String] The question to ask the chatflow. - :history [Array] A list of previous questions and answers. - :overrideConfig [Hash] A hash of configuration overrides for the chatflow. - :socketIOClientId [String] The socket.io client ID. - :file\_path [String] The path to a file to upload for prediction. - :worker\_name [String] The name of the worker to use for prediction. - :worker\_prompt [String] The prompt to use for the worker. - :prompt\_values [Hash] A hash of prompt values to use for the worker.

@return [Hash] The response from the Flowise [API](#).

```
upsert_document(chatflow_id, file_path,  
local_ai_config={})
```

Sends a document upsert request to the Flowise [API](#).

```
@param chatflow_id [String] The ID of the  
chatflow to use for document upsert. @param  
file_path [String] The path to the file to  
upload. @param local_ai_config [Hash] A hash  
of configuration options for the local AI. -  
:api_key [String] The API key for the local  
AI. - :base_path [String] The base path for  
the local AI. - :model_name [String] The  
name of the model to use for the local AI.  
@return [Hash] The response from the Flowise  
API.
```

---

## Private Instance Methods

```
handle_response(response)
```

Handles the response from the Flowise [API](#).

```
@param response [Faraday::Response] The  
response from the Flowise API.
```

```
@return [Hash] The parsed response body.  
@raise [RuntimeError] If the response status  
is not 200.
```

Validate

Generated by [RDoc](#) 6.4.0.

Based on [Darkfish](#) by [Michael Granger](#).

Home
Pages Classes Methods
Pages
LICENSE README
Class and Module Index
API AccumulateFilteredSegmentsTask CompressionTask CompressionTestAssessmentTask CompressionTestEvalTask CompressionTestTask Cursor DisplayResultsTask ExceptionAgent FileLoaderTask FileObject FilterSegmentsTask FinalReportTask Flowbots Flowbots::APIError Flowbots::AgentError Flowbots::BatchProcessor Flowbots::CLI Flowbots::ConfigurationError Flowbots::ExceptionAgent Flowbots::ExceptionHandler Flowbots::FileDiscovery Flowbots::FileLoader Flowbots::FileNotFoundError Flowbots::FlowbotError Flowbots::GrammarProcessor Flowbots::NLPPProcessor Flowbots::Task Flowbots::TaskNotFoundError Flowbots::TextProcessingWorkflow Flowbots::TextProcessor Flowbots::TextSegmentProcessor Flowbots::TextTaggerProcessor Flowbots::TextTokenizeProcessor Flowbots::TopicModelProcessor Flowbots::TopicModelTrainerWorkflow Flowbots::TopicModelTrainerWorkflowtest Flowbots::UnifiedFileProcessingPipeline Flowbots::WorkflowError Flowbots::Workflows FlowwiseApiClient InputRetrieval Jongleur Jongleur::WorkerTask Lemma LlmAnalysisTask LoadFileObjectTask LoadTextFilesTask Logging MarkdownYaml MarkdownYaml::Document0 MarkdownYaml::YamlFrontMatter0 MarkdownYaml::YamlFrontMatter1 MarkdownYamlParser MicroAgentTask MonadicError NlpAnalysisTask Object PreprocessFileObjectTask RedisConnection RedisKeys RunRubyTestsTask Segment

# Flowbots

Flowbots is an advanced text processing and analysis system that combines the power of nano-bots, workflow orchestration, and natural language processing to provide a flexible and powerful tool for document analysis and topic modeling.

## Features

- Text processing workflows for individual files and batch processing
- Advanced NLP capabilities including tokenization, part-of-speech tagging, and named entity recognition
- Topic modeling with dynamic model training and inference
- Flexible workflow system using Jongleur for task orchestration
- Redis-based data persistence using Ohm models
- Custom nano-bot cartridges for specialized AI-powered tasks
- Robust error handling and logging system
- User-friendly CLI interface

## System Architecture

### Class Diagram

```
classDiagram
    class CLI {
        +version()
        +workflows()
        +train_topic_model(folder)
        +process_text(file)
    }

    class Workflows {
```



Sublayer  
 Sublayer::Actions  
 Sublayer::Actions::RunTestCommandAction  
 Sublayer::Actions::SpeechToTextAction  
 Sublayer::Actions::TextToSpeechAction  
 Sublayer::Actions::WriteFileAction  
 TTY  
 TTY::Markdown  
 TTY::Markdown::Converter  
 TTY::PromptX  
 Task  
 TextSegmentTask  
 TextTaggerTask  
 TextTokenizeTask  
 TokenizeSegmentsTask  
 Topic  
 TopicModelingTask  
 TrainTopicModelTask  
 UI  
 UI::Box  
 UI::ScrollableBox  
 Word  
 WorkflowAgent  
 WorkflowOrchestrator

```

    -prompt: TTY::Prompt
    +list_and_select()
    +run(workflow_name)
    -get_workflows()
    -display_workflows(workflows)
    -select_workflow(workflows)
    -extract_workflow_description(file)
  }

  class WorkflowOrchestrator {
    -agents: Map
    +add_agent(role, cartridge_file)
    +define_workflow(workflow_definition)
    +run_workflow()
  }

  class WorkflowAgent {
    -role: String
    -state: Map
    -bot: NanoBot
    +process(input)
    +save_state()
    +load_state()
  }

  class Task {
    <<abstract>>
    +execute()
  }

  class TextProcessingWorkflow {
    -input_file_path: String
    -orchestrator: WorkflowOrchestrator
    +run()
  }

  class TopicModelTrainerWorkflow {
    -input_folder_path: String
    -orchestrator: WorkflowOrchestrator
    +run()
  }

  class TextProcessor {
    <<abstract>>
    +process(text)
  }

  class NLPProcessor {
    -nlp_model: Object
    +process(segment, options)
  }

  class TopicModelProcessor {
    -model_path: String
    -model: Object
    -model_params: Map
    +load_or_create_model()
    +train_model(documents, iterations)
    +infer_topics(document)
  }

```

```

class FileLoader {
    -file_data: Textfile
    +initialize(file_path)
}

class Textfile {
    +path: String
    +name: String
    +content: String
    +preprocessed_content: String
    +metadata: Map
    +topics: Set~Topic~
    +segments: List~Segment~
    +lemmas: List~Lemma~
}

class Segment {
    +text: String
    +tokens: List
    +tagged: Map
    +words: List~Word~
}

class Word {
    +word: String
    +pos: String
    +tag: String
    +dep: String
    +ner: String
}

class Topic {
    +name: String
    +description: String
    +vector: List
}

CLI --> Workflows : uses
Workflows --> TextProcessingWorkflow : runs
Workflows --> TopicModelTrainerWorkflow : runs
TextProcessingWorkflow --> WorkflowOrchestrator
TopicModelTrainerWorkflow --> WorkflowOrchestrator
WorkflowOrchestrator --> WorkflowAgent : manages
WorkflowOrchestrator --> Task : executes
Task <|-- FileLoaderTask
Task <|-- PreprocessTextFileTask
Task <|-- TextSegmentTask
Task <|-- TokenizeSegmentsTask
Task <|-- NlpAnalysisTask
Task <|-- TopicModelingTask
Task <|-- LlmAnalysisTask
Task <|-- DisplayResultsTask
TextProcessor <|-- NLPProcessor
TextProcessor <|-- TopicModelProcessor
NlpAnalysisTask --> NLPProcessor : uses
TopicModelingTask --> TopicModelProcessor : uses
FileLoaderTask --> FileLoader : uses
Textfile "1" *-- "many" Segment
Segment "1" *-- "many" Word

```

```
Textfile "1" *-- "many" Topic
Textfile "1" *-- "many" Lemma
```

## Flowbots Project Overview

**Flowbots** is an advanced text processing and analysis system that combines the power of nano-bots, workflow orchestration, and natural language processing to provide a flexible and powerful tool for document analysis and topic modeling.

### Key Features

1. Text processing workflows for individual files and batch processing
2. Advanced NLP capabilities including tokenization, part-of-speech tagging, and named entity recognition
3. **Topic** modeling with dynamic model training and inference
4. Flexible workflow system using **Jongleur** for task orchestration
5. Redis-based data persistence using Ohm models
6. Custom nano-bot cartridges for specialized AI-powered tasks
7. Robust error handling and logging system
8. User-friendly CLI interface

### Project Structure

The **Flowbots** project is organized into several key directories:

- `/lib`: Main application code

- /components: Core system components
- /processors: Text and NLP processors
- /tasks: Individual workflow tasks
- /workflows: Workflow definitions
- /ohm: Ohm model definitions
- /utils: Utility functions and classes
- /nano-bots/cartridges: Nano-bot cartridge definitions
- /test: Test files and test helpers
- /log: Log files

## Key Components

1. CLI: The main entry point for user interaction, allowing users to select and run workflows.
2. WorkflowOrchestrator: Manages the execution of workflows and their constituent tasks.
3. Task Processors: Specialized classes for text processing, NLP analysis, and topic modeling.
4. Ohm Models: Data persistence layer for storing document information and workflow states.
5. NanoBot Integration: Utilizes nano-bot cartridges for specialized AI-powered tasks.
6. Logging System: Comprehensive logging for debugging and monitoring.

## Workflow Execution

1. User selects a workflow through the CLI.
2. The selected workflow is initialized

and configured.

3. The `WorkflowOrchestrator` sets up the task graph based on the workflow definition.
4. Tasks are executed in the defined order, with results passed between tasks as needed.
5. Results are stored in Redis and Ohm models for persistence.
6. The workflow completes, and final results are displayed or stored as appropriate.

This project demonstrates a sophisticated approach to text analysis and processing, combining multiple technologies and techniques to create a powerful and flexible system.

## Workflows

`Flowbots` uses a flexible workflow system to orchestrate various text processing and analysis tasks. The two main workflows defined in the project are:

1. `TextProcessingWorkflow`
2. `TopicModelTrainerWorkflow`

## TextProcessingWorkflow

This workflow is designed to process a single text file through a series of tasks.

```

stateDiagram-v2
    [*] --> Initialized
    Initialized --> PromptingForFile: No file path
    PromptingForFile --> FileSelected: User select
    Initialized --> FileSelected: File path provid
    FileSelected --> ProcessingFile: Start process
    ProcessingFile --> TextTagging: File processe
    TextTagging --> TopicModeling: Tagging complet
    TopicModeling --> LlmAnalysis: Modeling complet
    LlmAnalysis --> DisplayingResults: Analysis co
    DisplayingResults --> [*]: Workflow complete

    state ProcessingFile {
        [*] --> LoadingFile
        LoadingFile --> PreprocessingFile
        PreprocessingFile --> SegmentingText
        SegmentingText --> TokenizingText
        TokenizingText --> [*]
    }

    DisplayingResults --> ErrorState: Error occurs
    ProcessingFile --> ErrorState: Error occurs
    TextTagging --> ErrorState: Error occurs
    TopicModeling --> ErrorState: Error occurs
    LlmAnalysis --> ErrorState: Error occurs
    ErrorState --> [*]: Log error and exit

```

## Key Steps:

1. **File Loading:** Loads the input file into the system.
2. **Preprocessing:** Extracts metadata and preprocesses the text content.
3. **Text Segmentation:** Splits the text into manageable segments.
4. **Tokenization:** Breaks down segments into individual tokens.
5. **NLP Analysis:** Performs part-of-speech tagging, dependency parsing, and named entity recognition.
6. **Topic Modeling:** Infers topics from the processed text.
7. **LLM Analysis:** Uses a language model to generate insights about the text.
8. **Result Display:** Presents the analysis results to the user.

# TopicModelTrainerWorkflow

This workflow is designed to process multiple files in batches and train a topic model.

## Key Steps:

1. **Batch Processing:** Processes files in batches of a defined size.
2. **File Loading:** Loads each file in the batch.
3. **Preprocessing:** Extracts metadata and preprocesses each file's content.
4. **Text Segmentation:** Splits each file's content into segments.
5. **Tokenization:** Breaks down segments into tokens.
6. **NLP Analysis:** Performs NLP tasks on the tokenized segments.
7. **Filtering:** Filters segments based on predefined criteria.
8. **Accumulation:** Accumulates filtered segments across all processed files.
9. **Topic Model Training:** Trains a topic model using the accumulated segments.

## Workflow Execution

Both workflows use the `WorkflowOrchestrator` class to manage task execution. The orchestrator:

1. Initializes the workflow and its tasks.
2. Sets up the task graph based on the workflow definition.
3. Executes tasks in the defined order.
4. Manages data flow between tasks using Redis for temporary storage.

5. Handles errors and exceptions during workflow execution.

## Workflow Flexibility

The workflow system is designed to be flexible and extensible:

- New workflows can be easily added by creating new workflow classes.
- Existing workflows can be modified by adding, removing, or reordering tasks.
- Tasks are modular and can be reused across different workflows.

This flexibility allows **Flowbots** to adapt to various text processing and analysis needs.



```
sequenceDiagram
    participant User
    participant TextProcessingWorkflow
    participant UnifiedFileProcessingPipeline
    participant WorkflowOrchestrator
    participant TextTaggerTask
    participant TopicModelingTask
    participant LlmAnalysisTask
    participant DisplayResultsTask
    participant Logger
    participant UI

    User->>TextProcessingWorkflow: Initialize with
    alt No file path provided
        TextProcessingWorkflow->>User: Prompt for
        User->>TextProcessingWorkflow: Provide file path
    end
    TextProcessingWorkflow->>UnifiedFileProcessingPipeline: Initialize
    TextProcessingWorkflow->>Logger: Log workflow
    TextProcessingWorkflow->>UI: Display workflow
    TextProcessingWorkflow->>UnifiedFileProcessingPipeline: Process file
    UnifiedFileProcessingPipeline-->>TextProcessingWorkflow: Return results
    TextProcessingWorkflow->>WorkflowOrchestrator: Execute tasks
    TextProcessingWorkflow->>WorkflowOrchestrator: Execute tasks
    WorkflowOrchestrator->>TextTaggerTask: Execute task
    TextTaggerTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>TopicModelingTask: Execute task
    TopicModelingTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>LlmAnalysisTask: Execute task
    LlmAnalysisTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>DisplayResultsTask: Execute task
    DisplayResultsTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator-->>TextProcessingWorkflow: Return results
    TextProcessingWorkflow->>Logger: Log workflow
    TextProcessingWorkflow->>UI: Display completion
    TextProcessingWorkflow-->>User: Workflow finished
```

```
sequenceDiagram
    actor User
    participant CLI
    participant TextProcessingWorkflow
    participant WorkflowOrchestrator
    participant FileLoaderTask
    participant PreprocessTextFileTask
    participant TextSegmentTask
    participant TokenizeSegmentsTask
    participant NlpAnalysisTask
    participant TopicModelingTask
    participant LlmAnalysisTask
    participant DisplayResultsTask
    participant Redis
    participant Textfile

    User->>CLI: process_text(file)
    activate CLI
    CLI->>TextProcessingWorkflow: new(input_file_path)
    activate TextProcessingWorkflow
```

```

TextProcessingWorkflow->>WorkflowOrchestrator: activate
TextProcessingWorkflow->>WorkflowOrchestrator: activate WorkflowOrchestrator
WorkflowOrchestrator->>FileLoaderTask: execute
activate FileLoaderTask
FileLoaderTask->>Redis: set("current_textfile")
FileLoaderTask->>Textfile: create
deactivate FileLoaderTask
WorkflowOrchestrator->>PreprocessTextFileTask: activate
activate PreprocessTextFileTask
PreprocessTextFileTask->>Textfile: update(preprocessed_text)
deactivate PreprocessTextFileTask
WorkflowOrchestrator->>TextSegmentTask: execute
activate TextSegmentTask
TextSegmentTask->>Textfile: add_segments()
deactivate TextSegmentTask
WorkflowOrchestrator->>TokenizeSegmentsTask: activate
activate TokenizeSegmentsTask
TokenizeSegmentsTask->>Textfile: update_segments()
deactivate TokenizeSegmentsTask
WorkflowOrchestrator->>NlpAnalysisTask: execute
activate NlpAnalysisTask
NlpAnalysisTask->>Textfile: update_segments_with_nlp()
deactivate NlpAnalysisTask
WorkflowOrchestrator->>TopicModelingTask: execute
activate TopicModelingTask
TopicModelingTask->>Textfile: add_topics()
deactivate TopicModelingTask
WorkflowOrchestrator->>LlmAnalysisTask: execute
activate LlmAnalysisTask
LlmAnalysisTask->>Textfile: update(analysis)
deactivate LlmAnalysisTask
WorkflowOrchestrator->>DisplayResultsTask: execute
activate DisplayResultsTask
DisplayResultsTask->>Textfile: retrieve_data()
DisplayResultsTask-->>User: display results
deactivate DisplayResultsTask
deactivate WorkflowOrchestrator
TextProcessingWorkflow-->>CLI: workflow complete
deactivate TextProcessingWorkflow
CLI-->>User: display completion message
deactivate CLI

```

## Task Processors

**Flowbots** uses a variety of task processors to handle different aspects of text processing and analysis. These processors are modular and can be combined in workflows to create complex text processing pipelines.

## Key Task Processors

1. FileLoaderTask
2. Loads input files into the system.
3. Stores file content in Ohm models for further processing.
4. PreprocessTextFileTask
5. Extracts metadata from file content (e.g., YAML front matter in Markdown files).
6. Preprocesses the main content for further analysis.
7. TextSegmentTask
8. Splits preprocessed text into manageable segments.
9. Uses the TextSegmentProcessor for actual segmentation logic.
10. TokenizeSegmentsTask
11. Breaks down text segments into individual tokens.
12. Uses the TextTokenizeProcessor for tokenization.
13. NlpAnalysisTask
14. Performs various NLP tasks on tokenized segments.
15. Includes part-of-speech tagging, dependency parsing, and named entity recognition.
16. Uses the NLPProcessor which wraps the Spacy library for NLP operations.
17. FilterSegmentsTask
18. Filters processed segments based on predefined criteria.
19. Removes irrelevant or low-quality segments to improve analysis quality.
20. TopicModelingTask
21. Infers topics from processed text

- segments.
22. Uses the `TopicModelProcessor` which implements topic modeling algorithms.
  23. `LlmAnalysisTask`
  24. Utilizes a language model (via `NanoBot`) to generate insights about the text.
  25. Provides high-level analysis and summarization of the processed content.
  26. `DisplayResultsTask`
  27. Formats and displays the results of the text processing and analysis pipeline.

## Task Processor Architecture

Each task processor:

1. Inherits from `Jongleur::WorkerTask` or `Flowbots::BaseTask`.
2. Implements an `execute` method that performs the core task logic.
3. Uses Redis for temporary data storage and passing data between tasks.
4. Interacts with Ohm models for persistent data storage.
5. Includes error handling and logging for robust execution.

## Extensibility

The task processor system is designed to be easily extensible:

- New task processors can be added by creating new classes inheriting from `Jongleur::WorkerTask` or `Flowbots::BaseTask`.

- Existing task processors can be modified or extended to support new functionality.
- **Task** processors can be combined in different ways within workflows to create custom text processing pipelines.

This modular design allows **Flowbots** to adapt to various text processing and analysis requirements.

## **Flowbots** Detailed Operation

### 1. Workflow Initialization

When a user selects a workflow through the CLI, the system initializes the chosen workflow (e.g., `TextProcessingWorkflow` or `TopicModelTrainerWorkflow`). The **WorkflowOrchestrator** sets up the task graph based on the workflow definition.

### 2. Task Execution

The **WorkflowOrchestrator** executes tasks in the defined order. Each task follows a similar pattern:

1. Retrieve necessary data from Redis or Ohm models.
2. Process the data using specialized processors (e.g., `NLPProcessor`, `TopicModelProcessor`).
3. Store the results back in Redis (for temporary storage) or Ohm models (for persistence).

### 3. Data Flow

- Redis is used for storing temporary data and passing information between tasks. This includes file IDs, current batch information, and intermediate processing results.
- Ohm models, backed by Redis, are used for persistent storage of document information, segments, tokens, and analysis results.

### 4. NLP and **Topic** Modeling

- The `NlpAnalysisTask` uses the ruby-spacy gem to perform tasks like tokenization, part-of-speech tagging, and named entity recognition.
- The `TopicModelingTask` uses the tomoto gem to implement topic modeling algorithms.

### 5. LLM Integration

The `LlmAnalysisTask` integrates with external language models through the NanoBot system. This allows for high-level analysis and insights generation based on the processed text data.

### 6. Error Handling and **Logging**

Each task and the `WorkflowOrchestrator` include error handling mechanisms. Errors are caught, logged, and in some cases, trigger the `ExceptionAgent` for detailed error analysis.

### 7. Batch Processing

For the `TopicModelTrainerWorkflow`, files are processed in batches. The `WorkflowOrchestrator` manages the batch state, ensuring all files in a batch are processed before moving to the next batch.

## 8. Result Presentation

The `DisplayResultsTask` formats the analysis results and presents them to the user through the CLI. This may include summaries, topic distributions, and insights generated by the LLM.

### Key Interactions

1. `CLI <-> WorkflowOrchestrator`: The CLI initiates workflow execution and receives final results.
2. `WorkflowOrchestrator <-> Tasks`: The orchestrator manages task execution order and handles task results.
3. `Tasks <-> Redis`: Tasks use Redis for short-term storage and inter-task communication.
4. `Tasks <-> Ohm Models`: Tasks interact with Ohm models for persistent storage of document data and analysis results.
5. `NLP and Topic Modeling Tasks <-> External Libraries`: These tasks utilize external Ruby gems for specialized processing.
6. `LlmAnalysisTask <-> NanoBot`: This task interacts with the NanoBot system to leverage external language models.

This architecture allows `Flowbots` to process text data through a series of specialized tasks, each building upon the

results of previous tasks, to provide comprehensive text analysis and insights.

## Ruby Gems Used in **Flowbots**

**Flowbots** leverages a variety of Ruby gems to provide its functionality. Here's a comprehensive list of the gems used in the project, along with their purposes:

1. `jongleur`
2. Purpose: Workflow orchestration and task management
3. Usage: Core component for defining and executing task workflows
4. `ohm`
5. Purpose: **Object**-hash mapping for Redis
6. Usage: Data persistence layer for storing document information and workflow states
7. `redis`
8. Purpose: In-memory data structure store
9. Usage: Temporary data storage and passing data between tasks
10. `json`
11. Purpose: JSON parsing and generation
12. Usage: Handling JSON data throughout the application
13. `parallel`
14. Purpose: Parallel processing
15. Usage: Potential use for parallel execution of tasks (not prominently used in the current implementation)
16. `pry` and `pry-stack_explorer`
17. Purpose: Enhanced REPL and debugging



- tools
- 18. Usage: Development and debugging
- 19. ruby-spacy
- 20. Purpose: Ruby bindings for the Spacy NLP library
- 21. Usage: Natural Language Processing tasks
- 22. thor
- 23. Purpose: Building command-line interfaces
- 24. Usage: Creating the CLI for **Flowbots**
- 25. treetop
- 26. Purpose: parsing expression grammar (PEG) parser generator
- 27. Usage: Custom grammar parsing, particularly for Markdown with YAML front matter
- 28. yaml
  - Purpose: YAML parsing and generation
  - Usage: Handling YAML data, particularly in configuration files and document front matter
- 29. faraday and faraday/multipart
  - Purpose: HTTP client library
  - Usage: Making HTTP requests, potentially for integrations with external services
- 30. logging
  - Purpose: Flexible logging
  - Usage: Comprehensive logging system throughout the application
- 31. tty-box, tty-cursor, tty-prompt, tty-

screen, tty-spinner, tty-table

- Purpose: Various terminal output formatting and interaction tools
- Usage: Creating rich command-line interfaces and displaying formatted output

### 32. pastel

- Purpose: Terminal output styling
- Usage: Adding colors and styles to terminal output

### 33. highline

- Purpose: High-level command-line interface building
- Usage: Additional CLI features and user input handling

### 34. cli-ui

- Purpose: CLI user interface components
- Usage: Enhancing the command-line interface with advanced **UI** elements

### 35. kramdown

- Purpose: Markdown parsing and conversion
- Usage: Handling Markdown content in documents

### 36. lingua

- Purpose: Natural language detection and processing
- Usage: Additional NLP capabilities

### 37. pragmatic\_segmenter

- Purpose: Text segmentation
- Usage: Splitting text into meaningful segments

#### 38. `pragmatic_tokenizer`

- Purpose: Text tokenization
- Usage: Breaking text into individual tokens

#### 39. `tomoto`

- Purpose: **Topic** modeling
- Usage: Implementing topic modeling algorithms

#### 40. `minitest` and `minitest/rg`

- Purpose: Testing framework
- Usage: Writing and running tests for the application

These gems provide a robust foundation for **Flowbots**, covering areas such as data persistence, natural language processing, command-line interfaces, HTTP communications, and more. The combination of these tools allows **Flowbots** to offer a comprehensive text processing and analysis system with a user-friendly interface.

Validate

Generated by RDoc 6.4.0.

Based on **Darkfish** by Michael Granger.

```
#retrieve_file_object  
#retrieve_file_path  
#retrieve_input
```

# module InputRetrieval

Module for retrieving input data.

---

## Public Instance Methods

### retrieve\_file\_object()

Retrieves the `FileObject` from Redis.

@return [FileObject, nil] The retrieved `FileObject` or nil if no `FileObject` is found.

### retrieve\_file\_path()

Retrieves the file path from Redis.

@return [String] The retrieved file path.  
@raise [ArgumentError] If the file path is not found in Redis.

### retrieve\_input()

Retrieves the input data for a task.

This method first attempts to retrieve a `FileObject` from Redis. If a `FileObject` is not found, it will attempt to retrieve a file path from Redis.

@return [FileObject, String, nil] The retrieved `FileObject`, file path, or nil if no input is found.

Validate

Generated by `RDoc 6.4.0`.

Based on `Darkfish` by `Michael Granger`.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

Home
Pages Classes Methods
Parent
Ohm::Model
Included Modules
Ohm::DataTypes Ohm::Callbacks

# class Lemma

Validate  
Generated by RDoc 6.4.0.  
Based on Darkfish by Michael Granger.

<a href="#">Home</a> <a href="#">Pages</a> <a href="#">Classes</a> <a href="#">Methods</a>	<h1>The MIT License (MIT)</h1>
	Copyright © 2024 Robert Pannick
<a href="#">Pages</a>	Permission is hereby granted, free of
<a href="#">LICENSE</a> <a href="#">README</a>	charge, to any person obtaining a copy of
	<p>this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:</p> <p>The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.</p> <p>THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.</p> <p><a href="#">Validate</a> Generated by <a href="#">RDoc</a> 6.4.0. Based on <a href="#">Darkfish</a> by <a href="#">Michael Granger</a>.</p>

Home
Pages Classes Methods
Parent
Jongleur::WorkerTask
Included Modules
InputRetrieval
Methods
<div>#execute</div> <div>#format_nlp_result</div> <div>#generate_analysis_prompt</div> <div>#retrieve_file_metadata</div> <div>#retrieve_input</div> <div>#retrieve_nlp_result</div> <div>#store_analysis_result</div> <div>#write_markdown</div> <div>#write_markdown_report</div>

# class LlmAnalysisTask

This task performs LLM analysis on a text file using a pre-trained model.

This task performs LLM analysis on a text file using a pre-trained model.

## Public Instance Methods

### execute()

Executes the task.

@return [void]



## Private Instance Methods

### `format_nlp_result(nlp_result)`

Formats the NLP results for display in the prompt.

@param nlp\_result [Array] The NLP results for the segments of the Textfile.

@return [String] The formatted NLP results.

### `generate_analysis_prompt(textfile, content, metadata, nlp_result)`



Please structure your response in a clear, concise

PROMPT end

### `retrieve_file_metadata()`

Retrieves the file metadata from Redis.

@return [Hash] The file metadata.

### `retrieve_input()`

### `retrieve_nlp_result(textfile)`

Retrieves the NLP results for the segments of the Textfile.

@param textfile [Textfile] The Textfile object.

@return [Array] An array of NLP results for each segment.

**store\_analysis\_result(textfile, result)**

Stores the analysis result in the Textfile.

@param textfile [Textfile] The Textfile object. @param result [String] The analysis result from the LLM agent.

@return [void]

**write\_markdown(textfile, analysis\_result)**

Writes the LLM analysis result to a Markdown file.

@param textfile [Textfile] The Textfile object. @param analysis\_result [String] The LLM analysis result.

@return [void]

**write\_markdown\_report(result)**

Writes the exception report to a markdown file.

@param report [String] The exception report. @param exception\_details [Hash] A hash containing exception details.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
#execute
#retrieve_file_path
#store_file_object_id
```

# class LoadFileObjectTask

Task to load a `FileObject` based on a file path stored in Redis.

---

## Public Instance Methods

### `execute()`

Executes the task to load a `FileObject`.

Retrieves the file path from Redis, finds or creates a `FileObject` associated with the path, stores the `FileObject` ID in Redis, and logs the progress.

@return [void] @raises [RuntimeError] If the `FileObject` cannot be created or found, or if there is an error during file loading.

## Private Instance Methods

### `retrieve_file_path()`

Retrieves the file path from Redis.

@return [String] The file path retrieved from Redis.

### `store_file_object_id(id)`

Stores the `FileObject` ID in Redis.

@param id [Integer] The ID of the `FileObject` to store. @return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class LoadTextFilesTask

**Task** to load text files and store their IDs in Redis.

---

## Public Instance Methods

### **execute()**

Executes the task to load a text file using the `Flowbots::FileLoader`.

Retrieves the file path from Redis, loads the file using `Flowbots::FileLoader`, stores the Textfile ID in Redis if successful, and logs the progress or errors.

@return [void]

## Private Instance Methods

### `retrieve_input()`

Retrieves the input file path from Redis.

@return [String] The input file path.

### `store_textfile_id(id)`

Stores the Textfile ID in Redis.

@param id [Integer] The ID of the Textfile.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::configure_logger_for  
::log_level  
::logger_for  
#logger
```

# module Logging

---

## Constants

### LOG\_DIR

The directory where log files will be stored.

### LOG\_LEVEL

The default log level.

### LOG\_MAX\_FILES

The maximum number of log files to keep.

### LOG\_MAX\_SIZE

The maximum size of a log file in bytes.

---

## Public Class Methods

### `configure_logger_for(_classname, _methodname)`

Configures a `logger` for the specified class and method.

@param *classname* [String] The name of the class. @param *methodname* [String] The name of the method.

@return [Logger] The configured `logger` object.

### `log_level()`

Returns the default log level.

@return [Integer] The log level.

### `logger_for(classname, methodname)`

Returns the `logger` for the specified class and method.

@param classname [String] The name of the class. @param methodname [String] The name of the method.

@return [Logger] The `logger` object.

---

## Public Instance Methods

### `logger()`

Returns the `logger` for the current class and method.

@return [Logger] The `logger` object.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.



Included Modules

Treetop::Runtime

Methods

#\_nt\_document  
#\_nt\_markdown\_content  
#\_nt\_newline  
#\_nt\_yaml\_front\_matter  
#root

# module MarkdownYaml

Autogenerated from a Treetop grammar. Edits may be lost.

---

## Public Instance Methods

### \_nt\_document()

Parses the document node.

@return [Treetop::Runtime::SyntaxNode] The parsed document node.

### \_nt\_markdown\_content()

Parses the Markdown content node.

@return [Treetop::Runtime::SyntaxNode] The parsed Markdown content node.

**\_nt\_newline()**

Parses the newline node.

@return [Treetop::Runtime::SyntaxNode] The  
parsed newline node.

**\_nt\_yaml\_front\_matter()**

Parses the YAML front matter node.

@return [Treetop::Runtime::SyntaxNode] The  
parsed YAML front matter node.

**root()**

The **root** node of the grammar.

@return [Treetop::Runtime::SyntaxNode] The  
**root** node of the grammar.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class MarkdownYamlParser

The `MarkdownYamlParser` class is responsible for parsing the Markdown YAML grammar.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
::new  
#execute  
#get_input_for_analysis  
#store_result
```

# class MicroAgentTask

---

## Public Class Methods

```
new(agent_role, cartridge_file)
```

---

## Public Instance Methods

```
execute()
```

---

## Private Instance Methods

```
get_input_for_analysis()
```

```
store_result(result)
```

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class MonadicError

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
#add_lemmas_to_textfile  
#add_words_to_segment  
#execute  
#retrieve_input  
#update_segment_with_nlp_data
```

# class NlpAnalysisTask

This task performs natural language processing (NLP) analysis on the segments of a text file.

---

## Public Instance Methods

### execute()

Executes the task.

Retrieves the `FileObject` from Redis, processes each segment using the `NLPProcessor`, updates the segments with NLP data, adds lemmas to the `FileObject`, and logs the progress.

@return [void]

---

## Private Instance Methods

### add\_lemmas\_to\_textfile(textfile, lemma\_counts)

Adds lemmas to a `FileObject`.

Converts the lemma counts hash to an array of lemma data and adds it to the `FileObject`'s lemmas.

@param textfile [`FileObject`] The `FileObject` to add lemmas to. @param lemma\_counts [`Hash`] A hash containing lemma counts.

@return [void]

**add\_words\_to\_segment(segment,  
processed\_tokens)**

Adds processed words to a segment.

Extracts word information from the processed tokens and adds it to the segment's words.

@param segment [Segment] The segment to add words to. @param processed\_tokens [Array<Hash>] An array of processed tokens from the NLPProcessor.

@return [void]

**retrieve\_input()**

Retrieves the input for the task, which is the current **FileObject**.

@return [FileObject] The current **FileObject**.

**update\_segment\_with\_nlp\_data(segment,  
processed\_tokens, lemma\_counts)**

Updates a segment with NLP data.

Extracts relevant NLP information from the processed tokens and updates the segment with tagged data, words, and lemma counts.

@param segment [Segment] The segment to update. @param processed\_tokens [Array<Hash>] An array of processed tokens from the NLPProcessor. @param lemma\_counts [Hash] A hash to store lemma counts.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.



```
#format_output
#get_object_attributes
#get_object_bucket
#get_object_by_name
#get_object_collections
#get_object_indexed_attributes
#get_object_references
#get_objects
#get_objects_by_collection
#get_objects_by_query
#get_objects_by_reference
#get_objects_by_regex
#main
```

# class Object

---

## Constants

### CARTRIDGE\_DIR

Define the directory for cartridges

### GRAMMAR\_DIR

Constant for the directory containing grammars.

### PASTEL

### TASK\_DIR

Constant for the directory containing tasks.

### TOPIC\_MODEL\_PATH

### WORKFLOW\_DIR

Constant for the directory containing workflows.

---

## Public Instance Methods

**format\_output(objects)**

**get\_object\_attributes(object\_bucket)**

**get\_object\_bucket(object\_bucket)**

**get\_object\_by\_name(object\_bucket, object\_name)**

```
get_object_collections(object_bucket)
```

```
get_object_indexed_attributes(object_bucket)
```

```
get_object_references(object_bucket)
```

```
get_objects(object_bucket, object_name,  
query=nil)
```

```
get_objects_by_collection(object_bucket,  
collection_name)
```

```
get_objects_by_query(object_bucket,  
query)
```

```
get_objects_by_reference(object_bucket,  
reference_name, reference_value)
```

```
get_objects_by_regex(object_bucket,  
regex)
```

```
main()
```

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

```
#determine_preprocessing_method
#execute
#extract_metadata
#extract_pdf_metadata
#extract_text_from_pdf
#preprocess_file
#preprocess_json
#preprocess_markdown_yaml
#preprocess_pdf
#preprocess_plain_text
#store_preprocessed_data
```

# class PreprocessFileObjectTask

**Task** to preprocess a **FileObject**.

---

## Public Instance Methods

### **execute()**

Executes the task to preprocess a **FileObject**.

Retrieves the **FileObject** from Redis, determines the appropriate preprocessing method based on the file extension, preprocesses the file content and metadata, stores the preprocessed data in the **FileObject**, and logs the progress or errors.

@return [void]

## Private Instance Methods

### `determine_preprocessing_method(file_object)`

Determines the preprocessing method based on the file extension.

@param file\_object [FileObject] The `FileObject` to determine the preprocessing method for.

@return [Symbol] The symbol representing the preprocessing method.

### `extract_metadata(yaml_front_matter)`

Extracts metadata from YAML front matter.

@param yaml\_front\_matter [String] The YAML front matter string.

@return [Hash] The extracted metadata.

### `extract_pdf_metadata(pdf_path)`

Extracts metadata from a PDF file.

@param pdf\_path [String] The path to the PDF file.

@return [Hash] The extracted metadata.

### `extract_text_from_pdf(pdf_path)`

Extracts text content from a PDF file.

@param pdf\_path [String] The path to the PDF file.

@return [String] The extracted text content.

### **preprocess\_file(file\_object)**

Preprocesses the file based on its extension.

@param file\_object [FileObject] The **FileObject** to preprocess.

@return [Array(String, Hash)] An array containing the preprocessed content and metadata.

### **preprocess\_json(file\_object)**

Preprocesses JSON files.

@param file\_object [FileObject] The **FileObject** to preprocess.

@return [Array(String, Hash)] An array containing the preprocessed content and metadata.

### **preprocess\_markdown\_yaml(file\_object)**

Preprocesses Markdown files with YAML front matter.

@param file\_object [FileObject] The **FileObject** to preprocess.

@return [Array(String, Hash)] An array containing the preprocessed content and metadata.

### **preprocess\_pdf(file\_object)**

Preprocesses PDF files.

@param file\_object [FileObject] The **FileObject** to preprocess.

@return [Array(String, Hash)] An array containing the preprocessed content and metadata.

### **preprocess\_plain\_text(file\_object)**

Preprocesses plain text files.

@param file\_object [FileObject] The **FileObject** to preprocess.

@return [Array(String, Hash)] An array containing the preprocessed content and metadata.

### **store\_preprocessed\_data(content, metadata)**

Stores the preprocessed content and metadata in the **FileObject**.

@param content [String] The preprocessed content. @param metadata [Hash] The extracted metadata.

@return [void]

Validate

Generated by **RDoc** 6.4.0.

Based on **Darkfish** by **Michael Granger**.

Home
Pages Classes Methods
Table of Contents
Flowbots
Features
System Architecture
Class Diagram
Flowbots Project Overview
Key Features
Project Structure
Key Components
Workflow Execution
Workflows
TextProcessingWorkflow
Key Steps:
TopicModelTrainerWorkflow
Key Steps:
Workflow Execution
Workflow Flexibility
Task Processors
Key Task Processors
Task Processor Architecture
Extensibility
Flowbots Detailed Operation
1. Workflow Initialization
2. Task Execution
3. Data Flow
4. NLP and Topic Modeling
5. LLM Integration
6. Error Handling and Logging
7. Batch Processing
8. Result Presentation
Key Interactions
Ruby Gems Used in Flowbots
Pages
LICENSE
README

# Flowbots

Flowbots is an advanced text processing and analysis system that combines the power of nano-bots, workflow orchestration, and natural language processing to provide a flexible and powerful tool for document analysis and topic modeling.

## Features

- Text processing workflows for individual files and batch processing
- Advanced NLP capabilities including tokenization, part-of-speech tagging, and named entity recognition
- Topic modeling with dynamic model training and inference
- Flexible workflow system using Jongleur for task orchestration
- Redis-based data persistence using Ohm models
- Custom nano-bot cartridges for specialized AI-powered tasks
- Robust error handling and logging system
- User-friendly CLI interface

## System Architecture

### Class Diagram

```
classDiagram
class CLI {
+version()
+workflows()
+train_topic_model(folder)
+process_text(file)
}

class Workflows {
```

```

        -prompt: TTY::Prompt
        +list_and_select()
        +run(workflow_name)
        -get_workflows()
        -display_workflows(workflows)
        -select_workflow(workflows)
        -extract_workflow_description(file)
    }

    class WorkflowOrchestrator {
        -agents: Map
        +add_agent(role, cartridge_file)
        +define_workflow(workflow_definition)
        +run_workflow()
    }

    class WorkflowAgent {
        -role: String
        -state: Map
        -bot: NanoBot
        +process(input)
        +save_state()
        +load_state()
    }

    class Task {
        <<abstract>>
        +execute()
    }

    class TextProcessingWorkflow {
        -input_file_path: String
        -orchestrator: WorkflowOrchestrator
        +run()
    }

    class TopicModelTrainerWorkflow {
        -input_folder_path: String
        -orchestrator: WorkflowOrchestrator
        +run()
    }

    class TextProcessor {
        <<abstract>>
        +process(text)
    }

    class NLPProcessor {
        -nlp_model: Object
        +process(segment, options)
    }

    class TopicModelProcessor {
        -model_path: String
        -model: Object
        -model_params: Map
        +load_or_create_model()
        +train_model(documents, iterations)
        +infer_topics(document)
    }

```



```

class FileLoader {
    -file_data: Textfile
    +initialize(file_path)
}

class Textfile {
    +path: String
    +name: String
    +content: String
    +preprocessed_content: String
    +metadata: Map
    +topics: Set~Topic~
    +segments: List~Segment~
    +lemmas: List~Lemma~
}

class Segment {
    +text: String
    +tokens: List
    +tagged: Map
    +words: List~Word~
}

class Word {
    +word: String
    +pos: String
    +tag: String
    +dep: String
    +ner: String
}

class Topic {
    +name: String
    +description: String
    +vector: List
}

CLI --> Workflows : uses
Workflows --> TextProcessingWorkflow : runs
Workflows --> TopicModelTrainerWorkflow : runs
TextProcessingWorkflow --> WorkflowOrchestrator
TopicModelTrainerWorkflow --> WorkflowOrchestrator
WorkflowOrchestrator --> WorkflowAgent : manages
WorkflowOrchestrator --> Task : executes
Task <|-- FileLoaderTask
Task <|-- PreprocessTextFileTask
Task <|-- TextSegmentTask
Task <|-- TokenizeSegmentsTask
Task <|-- NlpAnalysisTask
Task <|-- TopicModelingTask
Task <|-- LlmAnalysisTask
Task <|-- DisplayResultsTask
TextProcessor <|-- NLPProcessor
TextProcessor <|-- TopicModelProcessor
NlpAnalysisTask --> NLPProcessor : uses
TopicModelingTask --> TopicModelProcessor : uses
FileLoaderTask --> FileLoader : uses
Textfile "1" *-- "many" Segment
Segment "1" *-- "many" Word

```

```
Textfile "1" *-- "many" Topic
Textfile "1" *-- "many" Lemma
```

## Flowbots Project Overview

**Flowbots** is an advanced text processing and analysis system that combines the power of nano-bots, workflow orchestration, and natural language processing to provide a flexible and powerful tool for document analysis and topic modeling.

### Key Features

1. Text processing workflows for individual files and batch processing
2. Advanced NLP capabilities including tokenization, part-of-speech tagging, and named entity recognition
3. **Topic** modeling with dynamic model training and inference
4. Flexible workflow system using **Jongleur** for task orchestration
5. Redis-based data persistence using Ohm models
6. Custom nano-bot cartridges for specialized AI-powered tasks
7. Robust error handling and logging system
8. User-friendly CLI interface

### Project Structure

The **Flowbots** project is organized into several key directories:

- `/lib`: Main application code

- `/components`: Core system components
- `/processors`: Text and NLP processors
- `/tasks`: Individual workflow tasks
- `/workflows`: Workflow definitions
- `/ohm`: Ohm model definitions
- `/utils`: Utility functions and classes
- `/nano-bots/cartridges`: Nano-bot cartridge definitions
- `/test`: Test files and test helpers
- `/log`: Log files

## Key Components

1. **CLI**: The main entry point for user interaction, allowing users to select and run workflows.
2. **WorkflowOrchestrator**: Manages the execution of workflows and their constituent tasks.
3. **Task Processors**: Specialized classes for text processing, NLP analysis, and topic modeling.
4. **Ohm Models**: Data persistence layer for storing document information and workflow states.
5. **NanoBot Integration**: Utilizes nano-bot cartridges for specialized AI-powered tasks.
6. **Logging System**: Comprehensive logging for debugging and monitoring.

## Workflow Execution

1. User selects a workflow through the CLI.
2. The selected workflow is initialized

and configured.

3. The `WorkflowOrchestrator` sets up the task graph based on the workflow definition.
4. Tasks are executed in the defined order, with results passed between tasks as needed.
5. Results are stored in Redis and Ohm models for persistence.
6. The workflow completes, and final results are displayed or stored as appropriate.

This project demonstrates a sophisticated approach to text analysis and processing, combining multiple technologies and techniques to create a powerful and flexible system.

## Workflows

`Flowbots` uses a flexible workflow system to orchestrate various text processing and analysis tasks. The two main workflows defined in the project are:

1. `TextProcessingWorkflow`
2. `TopicModelTrainerWorkflow`

## TextProcessingWorkflow

This workflow is designed to process a single text file through a series of tasks.

```

stateDiagram-v2
    [*] --> Initialized
    Initialized --> PromptingForFile: No file path
    PromptingForFile --> FileSelected: User select
    Initialized --> FileSelected: File path provided
    FileSelected --> ProcessingFile: Start process
    ProcessingFile --> TextTagging: File processed
    TextTagging --> TopicModeling: Tagging complete
    TopicModeling --> LlmAnalysis: Modeling complete
    LlmAnalysis --> DisplayingResults: Analysis complete
    DisplayingResults --> [*]: Workflow complete

    state ProcessingFile {
        [*] --> LoadingFile
        LoadingFile --> PreprocessingFile
        PreprocessingFile --> SegmentingText
        SegmentingText --> TokenizingText
        TokenizingText --> [*]
    }

    DisplayingResults --> ErrorState: Error occurs
    ProcessingFile --> ErrorState: Error occurs
    TextTagging --> ErrorState: Error occurs
    TopicModeling --> ErrorState: Error occurs
    LlmAnalysis --> ErrorState: Error occurs
    ErrorState --> [*]: Log error and exit

```

## Key Steps:

1. **File Loading:** Loads the input file into the system.
2. **Preprocessing:** Extracts metadata and preprocesses the text content.
3. **Text Segmentation:** Splits the text into manageable segments.
4. **Tokenization:** Breaks down segments into individual tokens.
5. **NLP Analysis:** Performs part-of-speech tagging, dependency parsing, and named entity recognition.
6. **Topic Modeling:** Infers topics from the processed text.
7. **LLM Analysis:** Uses a language model to generate insights about the text.
8. **Result Display:** Presents the analysis results to the user.

# TopicModelTrainerWorkflow

This workflow is designed to process multiple files in batches and train a topic model.

## Key Steps:

1. **Batch Processing:** Processes files in batches of a defined size.
2. **File Loading:** Loads each file in the batch.
3. **Preprocessing:** Extracts metadata and preprocesses each file's content.
4. **Text Segmentation:** Splits each file's content into segments.
5. **Tokenization:** Breaks down segments into tokens.
6. **NLP Analysis:** Performs NLP tasks on the tokenized segments.
7. **Filtering:** Filters segments based on predefined criteria.
8. **Accumulation:** Accumulates filtered segments across all processed files.
9. **Topic Model Training:** Trains a topic model using the accumulated segments.

## Workflow Execution

Both workflows use the `WorkflowOrchestrator` class to manage task execution. The orchestrator:

1. Initializes the workflow and its tasks.
2. Sets up the task graph based on the workflow definition.
3. Executes tasks in the defined order.
4. Manages data flow between tasks using Redis for temporary storage.

5. Handles errors and exceptions during workflow execution.

## Workflow Flexibility

The workflow system is designed to be flexible and extensible:

- New workflows can be easily added by creating new workflow classes.
- Existing workflows can be modified by adding, removing, or reordering tasks.
- Tasks are modular and can be reused across different workflows.

This flexibility allows **Flowbots** to adapt to various text processing and analysis needs.

```
sequenceDiagram
    participant User
    participant TextProcessingWorkflow
    participant UnifiedFileProcessingPipeline
    participant WorkflowOrchestrator
    participant TextTaggerTask
    participant TopicModelingTask
    participant LlmAnalysisTask
    participant DisplayResultsTask
    participant Logger
    participant UI

    User->>TextProcessingWorkflow: Initialize with
    alt No file path provided
        TextProcessingWorkflow->>User: Prompt for
        User->>TextProcessingWorkflow: Provide file path
    end
    TextProcessingWorkflow->>UnifiedFileProcessingPipeline: Initialize
    TextProcessingWorkflow->>Logger: Log workflow
    TextProcessingWorkflow->>UI: Display workflow
    TextProcessingWorkflow->>UnifiedFileProcessingPipeline: Process file
    UnifiedFileProcessingPipeline-->>TextProcessingWorkflow: Return results
    TextProcessingWorkflow->>WorkflowOrchestrator: Execute tasks
    TextProcessingWorkflow->>WorkflowOrchestrator: Execute tasks
    WorkflowOrchestrator->>TextTaggerTask: Execute task
    TextTaggerTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>TopicModelingTask: Execute task
    TopicModelingTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>LlmAnalysisTask: Execute task
    LlmAnalysisTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator->>DisplayResultsTask: Execute task
    DisplayResultsTask-->>WorkflowOrchestrator: Task completed
    WorkflowOrchestrator-->>TextProcessingWorkflow: Return results
    TextProcessingWorkflow->>Logger: Log workflow
    TextProcessingWorkflow->>UI: Display completion
    TextProcessingWorkflow-->>User: Workflow finished
```

```
sequenceDiagram
    actor User
    participant CLI
    participant TextProcessingWorkflow
    participant WorkflowOrchestrator
    participant FileLoaderTask
    participant PreprocessTextFileTask
    participant TextSegmentTask
    participant TokenizeSegmentsTask
    participant NlpAnalysisTask
    participant TopicModelingTask
    participant LlmAnalysisTask
    participant DisplayResultsTask
    participant Redis
    participant Textfile

    User->>CLI: process_text(file)
    activate CLI
    CLI->>TextProcessingWorkflow: new(input_file_path)
    activate TextProcessingWorkflow
```



```

TextProcessingWorkflow->>WorkflowOrchestrator: activate
TextProcessingWorkflow->>WorkflowOrchestrator: activate WorkflowOrchestrator
WorkflowOrchestrator->>FileLoaderTask: execute
activate FileLoaderTask
FileLoaderTask->>Redis: set("current_textfile")
FileLoaderTask->>Textfile: create
deactivate FileLoaderTask
WorkflowOrchestrator->>PreprocessTextFileTask: activate
activate PreprocessTextFileTask
PreprocessTextFileTask->>Textfile: update(preprocessed_text)
deactivate PreprocessTextFileTask
WorkflowOrchestrator->>TextSegmentTask: execute
activate TextSegmentTask
TextSegmentTask->>Textfile: add_segments()
deactivate TextSegmentTask
WorkflowOrchestrator->>TokenizeSegmentsTask: activate
activate TokenizeSegmentsTask
TokenizeSegmentsTask->>Textfile: update_segments()
deactivate TokenizeSegmentsTask
WorkflowOrchestrator->>NlpAnalysisTask: execute
activate NlpAnalysisTask
NlpAnalysisTask->>Textfile: update_segments_with_nlp()
deactivate NlpAnalysisTask
WorkflowOrchestrator->>TopicModelingTask: execute
activate TopicModelingTask
TopicModelingTask->>Textfile: add_topics()
deactivate TopicModelingTask
WorkflowOrchestrator->>LlmAnalysisTask: execute
activate LlmAnalysisTask
LlmAnalysisTask->>Textfile: update(analysis)
deactivate LlmAnalysisTask
WorkflowOrchestrator->>DisplayResultsTask: execute
activate DisplayResultsTask
DisplayResultsTask->>Textfile: retrieve_data()
DisplayResultsTask-->>User: display results
deactivate DisplayResultsTask
deactivate WorkflowOrchestrator
TextProcessingWorkflow-->>CLI: workflow complete
deactivate TextProcessingWorkflow
CLI-->>User: display completion message
deactivate CLI

```

## Task Processors

**Flowbots** uses a variety of task processors to handle different aspects of text processing and analysis. These processors are modular and can be combined in workflows to create complex text processing pipelines.

## Key Task Processors

1. FileLoaderTask
2. Loads input files into the system.
3. Stores file content in Ohm models for further processing.
4. PreprocessTextFileTask
5. Extracts metadata from file content (e.g., YAML front matter in Markdown files).
6. Preprocesses the main content for further analysis.
7. TextSegmentTask
8. Splits preprocessed text into manageable segments.
9. Uses the TextSegmentProcessor for actual segmentation logic.
10. TokenizeSegmentsTask
11. Breaks down text segments into individual tokens.
12. Uses the TextTokenizeProcessor for tokenization.
13. NlpAnalysisTask
14. Performs various NLP tasks on tokenized segments.
15. Includes part-of-speech tagging, dependency parsing, and named entity recognition.
16. Uses the NLPProcessor which wraps the Spacy library for NLP operations.
17. FilterSegmentsTask
18. Filters processed segments based on predefined criteria.
19. Removes irrelevant or low-quality segments to improve analysis quality.
20. TopicModelingTask
21. Infers topics from processed text

- segments.
22. Uses the TopicModelProcessor which implements topic modeling algorithms.
  23. LlmAnalysisTask
  24. Utilizes a language model (via NanoBot) to generate insights about the text.
  25. Provides high-level analysis and summarization of the processed content.
  26. DisplayResultsTask
  27. Formats and displays the results of the text processing and analysis pipeline.

## Task Processor Architecture

Each task processor:

1. Inherits from `Jongleur::WorkerTask` or `Flowbots::BaseTask`.
2. Implements an `execute` method that performs the core task logic.
3. Uses Redis for temporary data storage and passing data between tasks.
4. Interacts with Ohm models for persistent data storage.
5. Includes error handling and logging for robust execution.

## Extensibility

The task processor system is designed to be easily extensible:

- New task processors can be added by creating new classes inheriting from `Jongleur::WorkerTask` or `Flowbots::BaseTask`.

- Existing task processors can be modified or extended to support new functionality.
- **Task** processors can be combined in different ways within workflows to create custom text processing pipelines.

This modular design allows **Flowbots** to adapt to various text processing and analysis requirements.

## **Flowbots** Detailed Operation

### **1. Workflow Initialization**

When a user selects a workflow through the CLI, the system initializes the chosen workflow (e.g., `TextProcessingWorkflow` or `TopicModelTrainerWorkflow`). The **WorkflowOrchestrator** sets up the task graph based on the workflow definition.

### **2. Task Execution**

The **WorkflowOrchestrator** executes tasks in the defined order. Each task follows a similar pattern:

1. Retrieve necessary data from Redis or Ohm models.
2. Process the data using specialized processors (e.g., `NLPProcessor`, `TopicModelProcessor`).
3. Store the results back in Redis (for temporary storage) or Ohm models (for persistence).

### 3. Data Flow

- Redis is used for storing temporary data and passing information between tasks. This includes file IDs, current batch information, and intermediate processing results.
- Ohm models, backed by Redis, are used for persistent storage of document information, segments, tokens, and analysis results.

### 4. NLP and **Topic** Modeling

- The `NlpAnalysisTask` uses the ruby-spacy gem to perform tasks like tokenization, part-of-speech tagging, and named entity recognition.
- The `TopicModelingTask` uses the tomoto gem to implement topic modeling algorithms.

### 5. LLM Integration

The `LlmAnalysisTask` integrates with external language models through the NanoBot system. This allows for high-level analysis and insights generation based on the processed text data.

### 6. Error Handling and **Logging**

Each task and the `WorkflowOrchestrator` include error handling mechanisms. Errors are caught, logged, and in some cases, trigger the `ExceptionAgent` for detailed error analysis.

### 7. Batch Processing

For the `TopicModelTrainerWorkflow`, files are processed in batches. The `WorkflowOrchestrator` manages the batch state, ensuring all files in a batch are processed before moving to the next batch.

## 8. Result Presentation

The `DisplayResultsTask` formats the analysis results and presents them to the user through the CLI. This may include summaries, topic distributions, and insights generated by the LLM.

## Key Interactions

1. `CLI <-> WorkflowOrchestrator`: The CLI initiates workflow execution and receives final results.
2. `WorkflowOrchestrator <-> Tasks`: The orchestrator manages task execution order and handles task results.
3. `Tasks <-> Redis`: Tasks use Redis for short-term storage and inter-task communication.
4. `Tasks <-> Ohm Models`: Tasks interact with Ohm models for persistent storage of document data and analysis results.
5. `NLP and Topic Modeling Tasks <-> External Libraries`: These tasks utilize external Ruby gems for specialized processing.
6. `LlmAnalysisTask <-> NanoBot`: This task interacts with the NanoBot system to leverage external language models.

This architecture allows `Flowbots` to process text data through a series of specialized tasks, each building upon the

results of previous tasks, to provide comprehensive text analysis and insights.

## Ruby Gems Used in **Flowbots**

**Flowbots** leverages a variety of Ruby gems to provide its functionality. Here's a comprehensive list of the gems used in the project, along with their purposes:

1. `jongleur`
2. Purpose: Workflow orchestration and task management
3. Usage: Core component for defining and executing task workflows
4. `ohm`
5. Purpose: **Object**-hash mapping for Redis
6. Usage: Data persistence layer for storing document information and workflow states
7. `redis`
8. Purpose: In-memory data structure store
9. Usage: Temporary data storage and passing data between tasks
10. `json`
11. Purpose: JSON parsing and generation
12. Usage: Handling JSON data throughout the application
13. `parallel`
14. Purpose: Parallel processing
15. Usage: Potential use for parallel execution of tasks (not prominently used in the current implementation)
16. `pry` and `pry-stack_explorer`
17. Purpose: Enhanced REPL and debugging

tools

18. Usage: Development and debugging
19. `ruby-spacy`
20. Purpose: Ruby bindings for the Spacy NLP library
21. Usage: Natural Language Processing tasks
22. `thor`
23. Purpose: Building command-line interfaces
24. Usage: Creating the CLI for `Flowbots`
25. `treetop`
26. Purpose: parsing expression grammar (PEG) parser generator
27. Usage: Custom grammar parsing, particularly for Markdown with YAML front matter
28. `yaml`
  - Purpose: YAML parsing and generation
  - Usage: Handling YAML data, particularly in configuration files and document front matter
29. `faraday` and `faraday/multipart`
  - Purpose: HTTP client library
  - Usage: Making HTTP requests, potentially for integrations with external services
30. `logging`
  - Purpose: Flexible logging
  - Usage: Comprehensive logging system throughout the application
31. `tty-box`, `tty-cursor`, `tty-prompt`, `tty-`



screen, tty-spinner, tty-table

- Purpose: Various terminal output formatting and interaction tools
- Usage: Creating rich command-line interfaces and displaying formatted output

### 32. pastel

- Purpose: Terminal output styling
- Usage: Adding colors and styles to terminal output

### 33. highline

- Purpose: High-level command-line interface building
- Usage: Additional CLI features and user input handling

### 34. cli-ui

- Purpose: CLI user interface components
- Usage: Enhancing the command-line interface with advanced **UI** elements

### 35. kramdown

- Purpose: Markdown parsing and conversion
- Usage: Handling Markdown content in documents

### 36. lingua

- Purpose: Natural language detection and processing
- Usage: Additional NLP capabilities

### 37. pragmatic\_segmenter

- Purpose: Text segmentation
- Usage: Splitting text into meaningful segments

#### 38. `pragmatic_tokenizer`

- Purpose: Text tokenization
- Usage: Breaking text into individual tokens

#### 39. `tomoto`

- Purpose: **Topic** modeling
- Usage: Implementing topic modeling algorithms

#### 40. `minitest` and `minitest/rg`

- Purpose: Testing framework
- Usage: Writing and running tests for the application

These gems provide a robust foundation for **Flowbots**, covering areas such as data persistence, natural language processing, command-line interfaces, HTTP communications, and more. The combination of these tools allows **Flowbots** to offer a comprehensive text processing and analysis system with a user-friendly interface.

Validate

Generated by RDoc 6.4.0.

Based on **Darkfish** by Michael Granger.

# class RedisConnection

Class to manage Redis connection.

---

## Constants

### REDIS\_CONFIG

Redis configuration.

---

## Attributes

### redis [R]

Returns the Redis connection.

@return [Redis] The Redis connection.

---

## Public Class Methods

### new()

Initializes a new RedisConnection instance.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# module RedisKeys

Module for managing Redis keys used in the **Flowbots** application.

---

## Constants

### ALL\_FILTERED\_SEGMENTS

Redis key for storing all filtered segments.

### CURRENT\_BATCH\_ID

Redis key for storing the ID of the current batch.

### CURRENT\_FILE\_OBJECT\_ID

Redis key for storing the ID of the current **FileObject**.

### CURRENT\_FILE\_PATH

Redis key for storing the path of the current file.

### CURRENT\_FILTERED\_SEGMENTS

Redis key for storing the currently filtered segments.

---

## Public Class Methods

### get(key)

Retrieves the value associated with the given key from Redis.

@param key [String] The Redis key. @return [String, nil] The value associated with the key, or nil if the key does not exist.

## `set(key, value)`

Sets the value associated with the given key in Redis.

@param key [String] The Redis key. @param value [String] The value to `set`. @return [String] The value that was `set`.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class RunRubyTestsTask

This task runs Ruby tests from a file.

---

## Public Instance Methods

### execute()

Executes the task.

@return [void] @raises [StandardError] If an error occurs during the task execution.

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

Home
Pages Classes Methods
Parent
Ohm::Model
Included Modules
Ohm::DataTypes Ohm::Callbacks
Methods
#add_word #add_words #retrieve_word_texts #retrieve_words

# class Segment

## Public Instance Methods

add\_word(word\_data)

add\_words(new\_words)

retrieve\_word\_texts()

retrieve\_words()

Validate  
Generated by RDoc 6.4.0.  
Based on Darkfish by Michael Granger.

`blueprints.sublayer.com/blueprints/70562717-70c5-4406-a792-358d169f9f0b`

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.



# Table of Contents - flowbots v0.1

## Pages

LICENSE

README

Flowbots

Features

System Architecture

Class Diagram

Flowbots Project Overview

Key Features

Project Structure

Key Components

Workflow Execution

Workflows

TextProcessingWorkflow

Key Steps:

TopicModelTrainerWorkflow

Key Steps:

Workflow Execution

Workflow Flexibility

Task Processors

Key Task Processors

Task Processor Architecture

Extensibility

Flowbots Detailed Operation

1. Workflow Initialization

2. Task Execution

3. Data Flow

4. NLP and Topic Modeling

5. LLM Integration

6. Error Handling and Logging

7. Batch Processing

8. Result Presentation

Key Interactions

Ruby Gems Used in Flowbots

# Classes and Modules

API  
AccumulateFilteredSegmentsTask  
CompressionTask  
CompressionTestAssessmentTask  
CompressionTestEvalTask  
CompressionTestTask  
Cursor  
DisplayResultsTask  
ExceptionAgent  
FileLoaderTask  
FileObject  
FilterSegmentsTask  
FinalReportTask  
Flowbots  
Flowbots::APIError  
Flowbots::AgentError  
Flowbots::BatchProcessor  
Flowbots::CLI  
Flowbots::ConfigurationError  
Flowbots::ExceptionAgent  
Flowbots::ExceptionHandler  
Flowbots::FileDiscovery  
Flowbots::FileLoader  
Flowbots::FileNotFoundError  
Flowbots::FlowbotError  
Flowbots::GrammarProcessor  
Flowbots::NLPPProcessor  
Flowbots::Task  
Flowbots::TaskNotFoundError  
Flowbots::TextProcessingWorkflow  
Flowbots::TextProcessor  
Flowbots::TextSegmentProcessor  
Flowbots::TextTaggerProcessor  
Flowbots::TextTokenizeProcessor  
Flowbots::TopicModelProcessor  
Flowbots::TopicModelTrainerWorkflow  
Flowbots::TopicModelTrainerWorkflowtest  
Flowbots::UnifiedFileProcessingPipeline  
Flowbots::WorkflowError

Flowbots::Workflows  
FlowiseApiClient  
InputRetrieval  
Jongleur  
Jongleur::WorkerTask  
Lemma  
LlmAnalysisTask  
LoadFileObjectTask  
LoadTextFilesTask  
Logging  
MarkdownYaml  
MarkdownYaml::Document0  
MarkdownYaml::YamlFrontMatter0  
MarkdownYaml::YamlFrontMatter1  
MarkdownYamlParser  
MicroAgentTask  
MonadicError  
NlpAnalysisTask  
Object  
PreprocessFileObjectTask  
RedisConnection  
RedisKeys  
RunRubyTestsTask  
Segment  
Sublayer  
Sublayer::Actions  
Sublayer::Actions::RunTestCommandAction  
Sublayer::Actions::SpeechToTextAction  
Sublayer::Actions::TextToSpeechAction  
Sublayer::Actions::WriteFileAction  
TTY  
TTY::Markdown  
TTY::Markdown::Converter  
TTY::PromptX  
Task  
TextSegmentTask  
TextTaggerTask  
TextTokenizeTask  
TokenizeSegmentsTask  
Topic

```
TopicModelingTask
TrainTopicModelTask
UI
UI::Box
UI::ScrollableBox
Word
WorkflowAgent
WorkflowOrchestrator
```

## Methods

```
::completed — Task
::configure_logger_for — Logging
::create_scrollable_box — UI::ScrollableBox
::create_with_timestamp — Task
::current_batch — FileObject
::discover_files — Flowbots::FileDiscovery
::display_boxes — UI::ScrollableBox
::exit_on_failure? — Flowbots::CLI
::failed — Task
::file_count — Flowbots::FileDiscovery
::find_or_create_by_path — FileObject
::get — RedisKeys
::handle_exception — Flowbots::ExceptionHandler
::in_progress — Task
::initialize — Flowbots
::latest — FileObject
::load_components — Flowbots
::load_tasks — Flowbots::Task
::load_workflows — Flowbots::Workflows
::log_exception — Flowbots::ExceptionHandler
::log_level — Logging
::logger_for — Logging
::new — Flowbots::BatchProcessor
::new — Flowbots::ExceptionAgent
::new — Flowbots::FileLoader
::new — WorkflowAgent
::new — WorkflowOrchestrator
::new — RedisConnection
::new — Flowbots::FlowbotError
```

```
::new — MicroAgentTask
::new — TopicModelingTask
::new — ExceptionAgent
::new — TTY::PromptX
::new — FlowiseApiClient
::new — Flowbots::UnifiedFileProcessingPipeline
::new — Flowbots::GrammarProcessor
::new — Flowbots::NLPPProcessor
::new — Flowbots::TextProcessor
::new — Flowbots::TextSegmentProcessor
::new — Flowbots::TextTaggerProcessor
::new — Flowbots::TextTokenizeProcessor
::new — Flowbots::TopicModelProcessor
::new — Flowbots::Task
::new — Sublayer::Actions::RunTestCommandAction
::new — Sublayer::Actions::SpeechToTextAction
::new — Sublayer::Actions::TextToSpeechAction
::new — Sublayer::Actions::WriteFileAction
::new — Flowbots::Workflows
::new — Flowbots::TextProcessingWorkflow
::new — Flowbots::TopicModelTrainerWorkflow
::new — Flowbots::TopicModelTrainerWorkflowtest
::notify_exception — Flowbots::ExceptionHandler
::pending — Task
::pos — Cursor
::print_boxes — UI::ScrollableBox
::print_navigation_info — UI::ScrollableBox
::set — RedisKeys
::setup_redis — Flowbots
::shutdown — Flowbots
::stop_running_workflows — Flowbots
#_nt_document — MarkdownYaml
#_nt_markdown_content — MarkdownYaml
#_nt_newline — MarkdownYaml
#_nt_yaml_front_matter — MarkdownYaml
#add_agent — WorkflowOrchestrator
#add_lemma — FileObject
#add_lemmas — FileObject
#add_lemmas_to_textfile — NlpAnalysisTask
#add_segment — FileObject
```

```
#add_segments — FileObject
#add_topics — FileObject
#add_word — Segment
#add_words — Segment
#add_words_to_segment — NlpAnalysisTask
#after_delete — FileObject
#after_save — FileObject
#analyze_transitivity — Flowbots::TextTaggerProcessor
#call — Sublayer::Actions::RunTestCommandAction
#call — Sublayer::Actions::SpeechToTextAction
#call — Sublayer::Actions::TextToSpeechAction
#call — Sublayer::Actions::WriteFileAction
#classify_file — Flowbots::FileLoader
#clean_segments — AccumulateFilteredSegmentsTask
#clean_segments_for_modeling —
Flowbots::TopicModelTrainerWorkflow
#clean_segments_for_modeling —
Flowbots::TopicModelTrainerWorkflowtest
#cleanup — WorkflowOrchestrator
#comparison_box — UI::Box
#complete — Task
#convert_p — TTY::Markdown::Converter
#create_doc — Flowbots::NLPProcessor
#create_new_model — Flowbots::TopicModelProcessor
#create_or_fetch_file_object — Flowbots::TextProcessingWorkflow
#define_workflow — WorkflowOrchestrator
#determine_preprocessing_method — PreprocessFileObjectTask
#discover_files — Flowbots::BatchProcessor
#display_filtered_segments — FilterSegmentsTask
#display_results — DisplayResultsTask
#display_workflows — Flowbots::Workflows
#duration — Task
#ensure_model_exists — Flowbots::TopicModelProcessor
#eval_result_box — UI::Box
#exception_box — UI::Box
#execute — Task
#execute — CompressionTask
#execute — CompressionTestTask
#execute — RunRubyTestsTask
#execute — CompressionTestEvalTask
#execute — CompressionTestAssessmentTask
```

```
#execute — FinalReportTask
#execute — MicroAgentTask
#execute — TopicModelingTask
#execute — Flowbots::Task
#execute — AccumulateFilteredSegmentsTask
#execute — DisplayResultsTask
#execute — FileLoaderTask
#execute — FilterSegmentsTask
#execute — LlmAnalysisTask
#execute — LoadFileObjectTask
#execute — LoadTextFilesTask
#execute — NlpAnalysisTask
#execute — PreprocessFileObjectTask
#execute — TextSegmentTask
#execute — TextTaggerTask
#execute — TextTokenizeTask
#execute — TokenizeSegmentsTask
#execute — TrainTopicModelTask
#extract_main_topics — Flowbots::TextTaggerProcessor
#extract_markdown_content — Flowbots::GrammarProcessor
#extract_metadata — PreprocessFileObjectTask
#extract_pdf_metadata — PreprocessFileObjectTask
#extract_relevant_files — Flowbots::ExceptionAgent
#extract_text — Flowbots::FileLoader
#extract_text_from_pdf — PreprocessFileObjectTask
#extract_text_json — Flowbots::FileLoader
#extract_workflow_description — Flowbots::Workflows
#extract_yaml_front_matter — Flowbots::GrammarProcessor
#fail — Task
#fallback_exception_report — Flowbots::ExceptionAgent
#fetch_unprocessed_file_ids — Flowbots::TextProcessingWorkflow
#file_types_pattern — Flowbots::BatchProcessor
#filter_segment_words — TopicModelingTask
#filter_segment_words — FilterSegmentsTask
#filter_segments — FilterSegmentsTask
#flush_redis_cache — Flowbots::UnifiedFileProcessingPipeline
#flush_redis_cache — Flowbots::TopicModelTrainerWorkflowtest
#footer — UI
#format_analysis — DisplayResultsTask
#format_exception — ExceptionAgent
```

```
#format_exception_report — Flowbots::ExceptionAgent
#format_file_info — DisplayResultsTask
#format_nlp_result — LlmAnalysisTask
#format_output — Object
#generate_analysis_prompt — LlmAnalysisTask
#generate_exception_prompt — Flowbots::ExceptionAgent
#get_documents — TopicModelingTask
#get_input_for_analysis — MicroAgentTask
#get_object_attributes — Object
#get_object_bucket — Object
#get_object_by_name — Object
#get_object_collections — Object
#get_object_indexed_attributes — Object
#get_object_references — Object
#get_objects — Object
#get_objects_by_collection — Object
#get_objects_by_query — Object
#get_objects_by_reference — Object
#get_objects_by_regex — Object
#get_workflows — Flowbots::Workflows
#handle_response — FlowiseApiClient
#header — UI
#identify_speech_acts — Flowbots::TextTaggerProcessor
#infer_topics — Flowbots::TopicModelProcessor
#info — UI
#info_box — UI::Box
#list_and_select — Flowbots::Workflows
#load_engtagger — Flowbots::TextTaggerProcessor
#load_existing_model — Flowbots::TopicModelProcessor
#load_file_structure — Flowbots::ExceptionAgent
#load_grammar — Flowbots::GrammarProcessor
#load_model — Flowbots::NLPPProcessor
#load_or_create_model — Flowbots::TopicModelProcessor
#load_state — WorkflowAgent
#logger — Logging
#main — Object
#main_menu — UI
#markdown_content — MarkdownYaml::Document0
#multi_column_box — UI::Box
#newline1 — MarkdownYaml::YamlFrontMatter1
```



```
#newline2 — MarkdownYaml::YamlFrontMatter1
#parse — Flowbots::GrammarProcessor
#parse_pdf — Flowbots::FileLoader
#perform_additional_tasks — Flowbots::TextProcessingWorkflow
#predict — FlowiseApiClient
#preprocess_file — PreprocessFileObjectTask
#preprocess_json — PreprocessFileObjectTask
#preprocess_markdown_yaml — PreprocessFileObjectTask
#preprocess_pdf — PreprocessFileObjectTask
#preprocess_plain_text — PreprocessFileObjectTask
#process — WorkflowAgent
#process — Flowbots::UnifiedFileProcessingPipeline
#process — Flowbots::NLPProcessor
#process — Flowbots::TextProcessor
#process — Flowbots::TextSegmentProcessor
#process — Flowbots::TextTaggerProcessor
#process — Flowbots::TextTokenizeProcessor
#process_batch — Flowbots::BatchProcessor
#process_batch — Flowbots::UnifiedFileProcessingPipeline
#process_batch — Flowbots::TextProcessingWorkflow
#process_batch — Flowbots::TopicModelTrainerWorkflowtest
#process_exception — Flowbots::ExceptionAgent
#process_exception — ExceptionAgent
#process_file — Flowbots::UnifiedFileProcessingPipeline
#process_files — Flowbots::BatchProcessor
#process_files — Flowbots::TopicModelTrainerWorkflowtest
#process_single_file — Flowbots::UnifiedFileProcessingPipeline
#process_single_file — Flowbots::TextProcessingWorkflow
#process_text — Flowbots::CLI
#prompt — UI
#prompt_for_file — Flowbots::TextProcessingWorkflow
#prompt_for_folder — Flowbots::BatchProcessor
#prompt_for_folder — Flowbots::TopicModelTrainerWorkflow
#prompt_for_folder — Flowbots::TopicModelTrainerWorkflowtest
#readline — TTY::PromptX
#retrieve_file_metadata — LlmAnalysisTask
#retrieve_file_object — InputRetrieval
#retrieve_file_path — LoadFileObjectTask
#retrieve_file_path — InputRetrieval
#retrieve_filtered_words — TopicModelingTask
```

```
#retrieve_input — Task
#retrieve_input — TopicModelingTask
#retrieve_input — AccumulateFilteredSegmentsTask
#retrieve_input — DisplayResultsTask
#retrieve_input — FileLoaderTask
#retrieve_input — FilterSegmentsTask
#retrieve_input — LlmAnalysisTask
#retrieve_input — LoadTextFilesTask
#retrieve_input — NlpAnalysisTask
#retrieve_input — TextSegmentTask
#retrieve_input — TextTaggerTask
#retrieve_input — TokenizeSegmentsTask
#retrieve_input — InputRetrieval
#retrieve_nlp_result — LlmAnalysisTask
#retrieve_segment_texts — FileObject
#retrieve_segments — FileObject
#retrieve_word_texts — FileObject
#retrieve_word_texts — Segment
#retrieve_words — FileObject
#retrieve_words — Segment
#root — MarkdownYaml
#run — Flowbots::Workflows
#run — Flowbots::TextProcessingWorkflow
#run — Flowbots::TopicModelTrainerWorkflow
#run — Flowbots::TopicModelTrainerWorkflowtest
#run_workflow — WorkflowOrchestrator
#save_model — Flowbots::TopicModelProcessor
#save_state — WorkflowAgent
#say — UI
#segment_array — Flowbots::TextSegmentProcessor
#segment_string — Flowbots::TextSegmentProcessor
#select_workflow — Flowbots::Workflows
#setup_workflow — Flowbots::UnifiedFileProcessingPipeline
#setup_workflow — Flowbots::TopicModelTrainerWorkflowtest
#side_by_side_boxes — UI::ScrollableBox
#spinner — UI
#splat_sort — API
#store_FileObject_id — FileLoaderTask
#store_analysis_result — LlmAnalysisTask
#store_exception_report — ExceptionAgent
```

```
#store_file_data - Flowbots::FileLoader
#store_file_object_id - LoadFileObjectTask
#store_preprocessed_data - PreprocessFileObjectTask
#store_result - MicroAgentTask
#store_result - TextTaggerTask
#store_segments - TextSegmentTask
#store_textfile_id - LoadTextFilesTask
#store_topic_result - TopicModelingTask
#store_topics - TopicModelingTask
#tokenize_array - Flowbots::TextTokenizeProcessor
#tokenize_string - Flowbots::TextTokenizeProcessor
#train_model - Flowbots::TopicModelProcessor
#train_topic_model - Flowbots::CLI
#train_topic_model - Flowbots::TopicModelTrainerWorkflow
#train_topic_model - Flowbots::TopicModelTrainerWorkflowtest
#update_file_object - AccumulateFilteredSegmentsTask
#update_segment_with_nlp_data - NlpAnalysisTask
#update_state - WorkflowAgent
#upsert_document - FlowiseApiClient
#version - Flowbots::CLI
#workflows - Flowbots::CLI
#write_markdown - LlmAnalysisTask
#write_markdown_report - Flowbots::ExceptionAgent
#write_markdown_report - LlmAnalysisTask
#yaml_front_matter - MarkdownYaml::Document0
```

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

Home
Pages Classes Methods
Parent
Ohm::Model
Included Modules
Ohm::DataTypes Ohm::Callbacks InputRetrieval
Methods
::completed ::create_with_timestamp ::failed ::in_progress ::pending #complete #duration #execute #fail #retrieve_input

# class Task

## Public Class Methods

- completed()
- create\_with\_timestamp(attributes={})
- failed()
- in\_progress()
- pending()

## Public Instance Methods

- complete(result=nil)
- duration()
- execute()
- fail(error\_message)
- retrieve\_input()

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class TextSegmentTask

This task segments the text content of a Textfile into smaller units.

---

## Public Instance Methods

### execute()

Executes the task to segment the text content of a `FileObject`.

Retrieves the `FileObject` from Redis, extracts its preprocessed content, segments the content using the `TextSegmentProcessor`, stores the segments in the `FileObject`, and logs the progress.

@return [void]

---

## Private Instance Methods

### retrieve\_input()

Retrieves the input for the task, which is the current `FileObject`.

@return [FileObject] The current `FileObject`.

### store\_segments(textfile, segments)

Stores the given segments in the given `FileObject`.

@param textfile [FileObject] The `FileObject` to store the segments in. @param segments [Array<String>] The segments to store.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class TextTaggerTask

This class performs text tagging on a given text.

---

## Public Instance Methods

### execute()

Executes the text tagging task.

Retrieves the `FileObject` from Redis, extracts its preprocessed content, performs text tagging using the `TextTaggerProcessor`, extracts main topics, identifies speech acts, analyzes transitivity, stores the results in the `FileObject`, and logs the progress.

@return [void] @raises [RuntimeError] If the `FileObject` retrieval fails or the preprocessed content is empty or nil.

---

## Private Instance Methods

### retrieve\_input()

Retrieves the input for the task, which is the current `FileObject`.

@return [FileObject] The current `FileObject`.

### store\_result(file\_object, result, main\_topics, speech\_acts, transitivity)

Stores the tagging results in the `FileObject`.

@param file\_object [FileObject] The `FileObject` to store the results in. @param result [Hash] The text tagging results. @param main\_topics [Array<String>] The extracted main topics. @param speech\_acts



```
[Array<String>] The identified speech acts.  
@param transitivity [Array<String>] The  
analyzed transitivity.  
@return [void]
```

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class TextTokenizeTask

This task tokenizes the segments of a text file.

---

## Public Instance Methods

### **execute()**

Executes the task.

@return [void]

### Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class TokenizeSegmentsTask

This task tokenizes the segments of a text file.

---

## Public Instance Methods

### execute()

Executes the task to tokenize the segments of a `FileObject`.

Retrieves the `FileObject` from Redis, tokenizes each segment using the `TextTokenizeProcessor`, updates the segments with the tokenized data, and logs the progress.

@return [void]

---

## Private Instance Methods

### retrieve\_input()

Retrieves the input for the task, which is the current `FileObject`.

@return [FileObject] The current `FileObject`.

Validate

Generated by `RDoc 6.4.0`.

Based on `Darkfish` by `Michael Granger`.

Home
Pages Classes Methods
Parent
Ohm::Model
Included Modules
Ohm::DataTypes Ohm::Callbacks

# class Topic

Validate  
Generated by RDoc 6.4.0.  
Based on Darkfish by Michael Granger.

```
::new  
#execute  
#filter_segment_words  
#get_documents  
#retrieve_filtered_words  
#retrieve_input  
#store_topic_result  
#store_topics
```

# class TopicModelingTask

This task performs topic modeling on a text file using a pre-trained model.

## Public Class Methods

**new(model\_params)**

## Public Instance Methods

**execute()**

## Private Instance Methods

**filter\_segment\_words(segment)**

Filters words from a segment based on their POS tags.

@param segment [Segment] The segment to filter words from.

@return [Array<String>] An array of filtered words.

**get\_documents()**

**retrieve\_filtered\_words(textfile)**

Retrieves filtered words from the segments of the given `FileObject`.

@param textfile [FileObject] The `FileObject` to retrieve filtered words from.

@return [Array<Array<String>>] An array of arrays, where each inner array contains filtered words from a segment.

### `retrieve_input()`

Retrieves the input for the task, which is the current `FileObject`.

@return [FileObject] The current `FileObject`.

### `store_topic_result(textfile, result)`

Stores the topic modeling results in the given `FileObject`.

Extracts unique words from the topic results, adds them as topics to the `FileObject`, and logs the progress.

@param textfile [FileObject] The `FileObject` to store the topic results in. @param result [Array<Hash>] An array of topic modeling results.

@return [void]

### `store_topics(topics)`

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

# class TrainTopicModelTask

This task trains a topic model using filtered segments from multiple batches.

---

## Public Instance Methods

### `execute()`

Executes the task to train a topic model using accumulated filtered segments.

Retrieves the current batch ID and filtered segments from Redis. Accumulates filtered segments across batches and trains the topic model only on the last batch. Logs and displays progress messages.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.



Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.

#footer  
#header  
#info  
#main\_menu  
#prompt  
#say  
#spinner

# module UI

This module provides user interface (**UI**) elements and functions for the **Flowbots** application.

This module provides methods for creating and displaying boxes in the **UI**.

This module provides methods for creating and displaying scrollable boxes in the **UI**.

---

## Constants

### PASTEL

An instance of the `Pastel` gem for colorizing text.

### TITLE\_WIDTH

The width of the title box.

---

## Public Instance Methods

### footer()

Displays the **Flowbots footer** in a framed box.

@return [void]

### header()

Displays the **Flowbots header** in a framed box.

@return [void]

### **info(text)**

Displays an information message in a framed box.

@param text [String] The text to display in the **info** box.

@return [void]

### **main\_menu()**

Displays the main menu and prompts the user for a choice.

@return [Symbol] The value of the selected choice.

### **prompt()**

Returns the TTY::Prompt instance used for user interaction.

@return [TTY::Prompt] The TTY::Prompt instance.

### **say(type, statement)**

Displays a message to the user with the specified type and logs it.

@param type [Symbol] The type of message (:ok, :warn, :error, or nil). @param statement [String] The message to display.

@return [void]

## `spinner(text)`

Creates and returns a `TTY::Spinner` instance with the specified text.

@param text [String] The text to display next to the `spinner`.

@return [TTY::Spinner] The `TTY::Spinner` instance.

Validate

Generated by `RDoc` 6.4.0.

Based on `Darkfish` by Michael Granger.

Home
Pages Classes Methods
Parent
Ohm::Model
Included Modules
Ohm::DataTypes Ohm::Callbacks

# class Word

Validate  
Generated by RDoc 6.4.0.  
Based on Darkfish by Michael Granger.

```
::new  
#load_state  
#process  
#save_state  
#update_state
```

# class WorkflowAgent

This class represents an agent in a workflow. Class representing an individual agent within a workflow in the **Flowbots** system.

The **WorkflowAgent** is a key component in the text processing pipeline, responsible for performing specific tasks as defined by its **role** and cartridge configuration. It's designed to be flexible, maintainable, and integrated seamlessly with the **WorkflowOrchestrator**.

## == Key Features

- Role-based processing: Each agent has a specific **role** in the workflow.
- Cartridge-based configuration: Agent behavior is defined by a cartridge file.
- State management: Agents can save and load **state** from Redis.
- Real-time feedback: Provides visual feedback during processing.
- **Logging**: Includes logging for monitoring and debugging.

## == Relation to Workflow

**WorkflowAgent** instances are typically: 1. Created and managed by the **WorkflowOrchestrator**. 2. Arranged in a sequence to form the text processing pipeline. 3. Called upon by the orchestrator to **process** input and produce output. 4. Capable of maintaining **state** across multiple workflow steps or runs.

## == Example Usage

```
ruby agent =  
WorkflowAgent.new("preprocessor",  
"path/to/cartridge.yml") result =  
agent.process(input_text) agent.save_state
```

== Integration

The `WorkflowOrchestrator` would typically:

- \* Initialize multiple `WorkflowAgents` for different stages of processing.
- \* Call the `#process` method of each agent in sequence.
- \* Manage the flow of data between agents in the workflow.

This class represents a workflow agent that can `process` input using a `NanoBot`.

---

## Constants

### `CARTRIDGE_DIR`

The base directory for cartridges.

---

## Attributes

`role` [R]

`state` [R]

---

## Public Class Methods

### `new(role, cartridge_file)`

Initializes a new `WorkflowAgent` instance.

@param `role` [String] The `role` of the agent. @param `cartridge_file` [String] The path to the cartridge file.

@return [void]

## Public Instance Methods

### `load_state()`

Loads the agent's `state` from Redis.

@return [void]

### `process(input)`

Processes the given input using the agent's cartridge.

@param input [String] The input to `process`.

@return [String] The agent's response.

### `save_state()`

Saves the agent's `state` to Redis.

@return [void]

---

## Private Instance Methods

### `update_state(response)`

Updates the agent's `state` with the latest response.

@param response [String] The agent's response.

@return [void]

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.



```
::new  
#add_agent  
#cleanup  
#define_workflow  
#run_workflow
```

# class WorkflowOrchestrator

Orchestrates the execution of workflows in the **Flowbots** application.

The **WorkflowOrchestrator** is responsible for managing the lifecycle of a workflow, from defining its structure to executing its tasks and handling their results. It acts as a central coordinator, ensuring that agents are properly initialized, tasks are executed in the correct order, and errors are handled gracefully.

This class orchestrates a workflow of tasks using **Jongleur**.

---

## Constants

### CARTRIDGE\_BASE\_DIR

The base directory for cartridges.

---

## Public Class Methods

### new()

Initializes a new **WorkflowOrchestrator** instance.

@return [void]

---

## Public Instance Methods

**add\_agent(role, cartridge\_file, author: "@b08x")**

Adds an agent to the orchestrator.

@param role [String] The role of the agent in the workflow.  
@param cartridge\_file [String] The name of the cartridge file

defining the agent's behavior. @param author [String] The author of the cartridge (default: "@b08x").

@return [void] @raise [RuntimeError] If the specified cartridge file is not found.

### **cleanup()**

Performs **cleanup** operations for the workflow.

This method is called after the workflow has finished or has been interrupted. It ensures that any resources held by the workflow are released and that the system is in a clean state for the next workflow execution.

@return [void]

### **define\_workflow(workflow\_definition)**

Defines the workflow structure using a task graph.

The workflow definition is a hash that outlines the tasks to be executed and their dependencies. It is used by the **Jongleur** library to create a task graph that represents the workflow.

@param workflow\_definition [Hash] The workflow definition in a format understood by **Jongleur**.

@return [void]

### **run\_workflow()**

Runs the defined workflow.

This method initiates the workflow execution, managing the lifecycle of tasks and handling any errors that occur during the process. It uses the **Jongleur** library to execute the tasks in the defined order and provides hooks for monitoring the

```
progress and handling events.
```

```
@return [void]
```

Validate

Generated by RDoc 6.4.0.

Based on Darkfish by Michael Granger.