

Flowbots Documentation














Table of Contents

1. Api	1
1.1. class API	1
1.1.1. Private Instance Methods	1
2. Apierror	2
2.1. class Flowbots::APIError	2
3. Accumulatefilteredsegmentstask	3
3.1. class AccumulateFilteredSegmentsTask	3
3.1.1. Public Instance Methods	3
3.1.2. Private Instance Methods	4
4. Actions	6
4.1. module Sublayer::Actions	6
5. Agent	7
5.1. class Agent	7
5.1.1. Public Instance Methods	7
6. Agenterror	9
6.1. class Flowbots::AgentError	9
7. Basetask	10
7.1. class Flowbots::BaseTask	10
7.1.1. Attributes	10
7.1.2. Public Class Methods	10
7.1.3. Public Instance Methods	10
8. Batchcompletiontask	12
8.1. class BatchCompletionTask	12
8.1.1. Public Instance Methods	12
8.1.2. Private Instance Methods	12
9. Batchprocessor	14
9.1. class Flowbots::BatchProcessor	14
9.1.1. Attributes	14
9.1.2. Public Class Methods	14
9.1.3. Public Instance Methods	14
9.1.4. Private Instance Methods	15
10. Box	17
10.1. module UI::Box	17
10.1.1. Public Instance Methods	17
11. Cli	20
11.1. class Flowbots::CLI	20
11.1.1. Public Class Methods	20
11.1.2. Public Instance Methods	20

12. Cartridge	23
12.1. class Cartridge	23
12.1.1. Public Instance Methods	23
13. Compressiontask	25
13.1. class CompressionTask	25
13.1.1. Public Instance Methods	25
14. Compressiontestassessmenttask	26
14.1. class CompressionTestAssessmentTask	26
14.1.1. Public Instance Methods	26
15. Compressiontestevaltask	27
15.1. class CompressionTestEvalTask	27
15.1.1. Public Instance Methods	27
16. Compressiontesttask	28
16.1. class CompressionTestTask	28
16.1.1. Public Instance Methods	28
17. Configurationerror	29
17.1. class Flowbots::ConfigurationError	29
18. Converter	30
18.1. class TTY::Markdown::Converter	30
18.1.1. Public Instance Methods	30
19. Cursor	32
19.1. class Cursor	32
19.1.1. Public Class Methods	32
20. Displayresultstask	33
20.1. class DisplayResultsTask	33
20.1.1. Public Instance Methods	33
20.1.2. Private Instance Methods	33
21. Document0	36
21.1. module MarkdownYaml::Document0	36
21.1.1. Public Instance Methods	36
22. Exceptionagent	37
22.1. class Flowbots::ExceptionAgent	37
22.1.1. Public Class Methods	37
22.1.2. Public Instance Methods	37
22.1.3. Private Instance Methods	38
23. Exceptionhandler	42
23.1. class Flowbots::ExceptionHandler	42
23.1.1. Public Class Methods	42
24. Filediscovery	44
24.1. module Flowbots::FileDiscovery	44
24.1.1. Constants	44

24.1.2. Public Class Methods	44
25. Fileloader	46
25.1. class Flowbots::FileLoader	46
25.1.1. Attributes	46
25.1.2. Public Class Methods	46
25.1.3. Private Instance Methods	46
26. Fileloadertask	50
26.1. class FileLoaderTask	50
26.1.1. Public Instance Methods	50
26.1.2. Private Instance Methods	50
27. Filenotfounderror	52
27.1. class Flowbots::FileNotFoundError	52
28. Fileobject	53
28.1. class FileObject	53
28.1.1. Public Class Methods	53
28.1.2. Public Instance Methods	54
28.1.3. Protected Instance Methods	57
29. Fileobjecterror	58
29.1. class Flowbots::FileObjectError	58
30. Filtersegmenttask	59
30.1. class FilterSegmentsTask	59
30.1.1. Public Instance Methods	59
30.1.2. Private Instance Methods	59
31. Finalreporttask	62
31.1. class FinalReportTask	62
31.1.1. Public Instance Methods	62
32. Flowboterror	64
32.1. class Flowbots::FlowbotError	64
32.1.1. Attributes	64
32.1.2. Public Class Methods	64
33. Flowbots	65
33.1. module Flowbots	65
33.1.1. Constants	65
33.1.2. Public Class Methods	65
33.1.3. Private Class Methods	65
34. Flowiseapiclient	67
34.1. class FlowiseApiClient	67
34.1.1. Public Class Methods	67
34.1.2. Public Instance Methods	67
34.1.3. Private Instance Methods	69
35. Grammarprocessor	70

35.1. class Flowbots::GrammarProcessor	70
35.1.1. Public Class Methods	70
35.1.2. Public Instance Methods	70
35.1.3. Private Instance Methods	71
36. Inputretrieval	73
36.1. module InputRetrieval	73
36.1.1. Public Instance Methods	73
37. Jongleur	75
37.1. module Jongleur	75
38. License	76
39. Lemma	77
39.1. class Lemma	77
40. Llmanalysistask	78
40.1. class LlmAnalysisTask	78
40.1.1. Public Instance Methods	78
40.1.2. Private Instance Methods	79
41. Loadfileobjecttask	82
41.1. class LoadFileObjectTask	82
41.1.1. Public Instance Methods	82
41.1.2. Private Instance Methods	83
42. Loadtextfilestask	85
42.1. class LoadTextFilesTask	85
42.1.1. Public Instance Methods	85
42.1.2. Private Instance Methods	86
43. Logging	88
43.1. module Logging	88
43.1.1. Constants	88
43.1.2. Public Class Methods	88
43.1.3. Public Instance Methods	89
44. Markdown	91
44.1. module TTY::Markdown	91
45. Markdownyaml	92
45.1. module MarkdownYaml	92
45.1.1. Public Instance Methods	92
46. Markdownyamlparser	98
46.1. class MarkdownYamlParser	98
47. Microagenttask	99
47.1. class MicroAgentTask	99
47.1.1. Public Class Methods	99
47.1.2. Public Instance Methods	99
47.1.3. Private Instance Methods	99

48. Monadicerror	101
48.1. class MonadicError	101
49. Nlpprocessor	102
49.1. class Flowbots::NLPPProcessor	102
49.1.1. Public Class Methods	102
49.1.2. Public Instance Methods	102
49.1.3. Private Instance Methods	103
50. Nlpanalysistask	105
50.1. class NlpAnalysisTask	105
50.1.1. Public Instance Methods	105
50.1.2. Private Instance Methods	105
51. Object	108
51.1. class Object	108
51.1.1. Constants	108
52. Preprocessfileobjecttask	109
52.1. class PreprocessFileObjectTask	109
52.1.1. Public Instance Methods	109
52.1.2. Private Instance Methods	110
53. Preprocesstextfiletask	115
53.1. class PreprocessTextFileTask	115
53.1.1. Public Instance Methods	115
53.1.2. Private Instance Methods	116
54. Promptx	118
54.1. class TTY::PromptX	118
54.1.1. Attributes	118
54.1.2. Public Class Methods	118
54.1.3. Public Instance Methods	118
55. Readme md	120
55.1. Flowbots 	120
55.1.1. Features 	120
55.1.2. System Architecture 	120
55.1.3. Project Structure 	120
55.1.4. Key Components 	120
55.2. Detailed Operation 	121
55.2.1. Key Interactions 	122
55.3. Ruby Gems Used in Flowbots 	122
55.3.1. Workflow and Task Management 	122
55.3.2. Data Persistence 	122
55.3.3. Parallel Processing 	122
55.3.4. Development and Debugging 	122
55.3.5. Natural Language Processing 	123

55.3.6. Command-Line Interface	123
55.3.7. Parsing and Data Handling	123
55.3.8. Terminal Output Formatting	123
55.3.9. Topic Modeling	123
56. Redisconnection	124
56.1. class RedisConnection	124
56.1.1. Constants	124
56.1.2. Attributes	124
56.1.3. Public Class Methods	124
57. Rediskeys	125
57.1. module RedisKeys	125
57.1.1. Constants	125
57.1.2. Public Class Methods	125
58. Runrubyteststask	127
58.1. class RunRubyTestsTask	127
58.1.1. Public Instance Methods	127
59. Runtestcommandaction	128
59.1. class Sublayer::Actions::RunTestCommandAction	128
59.1.1. Public Class Methods	128
59.1.2. Public Instance Methods	128
60. Scrollablebox	129
60.1. module UI::ScrollableBox	129
60.1.1. Private Class Methods	129
60.1.2. Public Instance Methods	131
61. Segment	133
61.1. class Segment	133
61.1.1. Public Instance Methods	133
62. Speechototextaction	135
62.1. class Sublayer::Actions::SpeechToTextAction	135
62.1.1. Public Class Methods	135
62.1.2. Public Instance Methods	135
63. Sublayer	137
63.1. module Sublayer	137
64. Tty	138
64.1. module TTY	138
65. Task	139
65.1. class Task	139
65.1.1. Public Class Methods	139
65.1.2. Public Instance Methods	140
66. Tasknotfounderror	143
66.1. class Flowbots::TaskNotFoundError	143

67. Textprocessingworkflow	144
67.1. class Flowbots::TextProcessingWorkflow	144
67.1.1. Attributes	144
67.1.2. Public Class Methods	144
67.1.3. Public Instance Methods	144
67.1.4. Private Instance Methods	145
68. Textprocessor	148
68.1. class Flowbots::TextProcessor	148
68.1.1. Public Class Methods	148
68.1.2. Public Instance Methods	148
69. Textsegmentprocessor	150
69.1. class Flowbots::TextSegmentProcessor	150
69.1.1. Constants	150
69.1.2. Attributes	150
69.1.3. Public Class Methods	150
69.1.4. Public Instance Methods	150
69.1.5. Private Instance Methods	151
70. Textsegmenttask	153
70.1. class TextSegmentTask	153
70.1.1. Public Instance Methods	153
70.1.2. Private Instance Methods	153
71. Texttaggerprocessor	155
71.1. class Flowbots::TextTaggerProcessor	155
71.1.1. Public Class Methods	155
71.1.2. Public Instance Methods	155
71.1.3. Private Instance Methods	158
72. Texttaggertask	159
72.1. class TextTaggerTask	159
72.1.1. Public Instance Methods	159
72.1.2. Private Instance Methods	160
73. Texttospeechaction	162
73.1. class Sublayer::Actions::TextToSpeechAction	162
73.1.1. Public Class Methods	162
73.1.2. Public Instance Methods	162
74. Texttokenizeprocessor	163
74.1. class Flowbots::TextTokenizeProcessor	163
74.1.1. Constants	163
74.1.2. Attributes	163
74.1.3. Public Class Methods	163
74.1.4. Public Instance Methods	163
74.1.5. Private Instance Methods	164

75. Texttokenizetask	166
75.1. class TextTokenizeTask	166
75.1.1. Public Instance Methods	166
76. Tokenizesegmentstask	167
76.1. class TokenizeSegmentsTask	167
76.1.1. Public Instance Methods	167
76.1.2. Private Instance Methods	167
77. Topic	169
77.1. class Topic	169
78. Topicmodelprocessor	170
78.1. class Flowbots::TopicModelProcessor	170
78.1.1. Attributes	170
78.1.2. Public Class Methods	170
78.1.3. Public Instance Methods	171
78.1.4. Private Instance Methods	173
79. Topicmodeltrainerworkflow	176
79.1. class Flowbots::TopicModelTrainerWorkflow	176
79.1.1. Attributes	176
79.1.2. Public Class Methods	176
79.1.3. Public Instance Methods	176
79.1.4. Private Instance Methods	177
80. Topicmodeltrainerworkflowtest	179
80.1. class Flowbots::TopicModelTrainerWorkflowtest	179
80.1.1. Constants	179
80.1.2. Attributes	179
80.1.3. Public Class Methods	179
80.1.4. Public Instance Methods	179
80.1.5. Private Instance Methods	180
81. Topicmodelingtask	183
81.1. class TopicModelingTask	183
81.1.1. Public Class Methods	183
81.1.2. Public Instance Methods	183
81.1.3. Private Instance Methods	183
82. Traintopicmodeltask	186
82.1. class TrainTopicModelTask	186
82.1.1. Public Instance Methods	186
83. Ui	188
83.1. module UI	188
83.1.1. Constants	188
83.1.2. Public Class Methods	188
83.1.3. Public Instance Methods	192

84. Unifiedfileprocessingpipeline	196
84.1. class Flowbots::UnifiedFileProcessingPipeline	196
84.1.1. Attributes	196
84.1.2. Public Class Methods	196
84.1.3. Public Instance Methods	196
84.1.4. Private Instance Methods	197
85. Word	200
85.1. class Word	200
86. Workertask	201
86.1. class Jongleur::WorkerTask	201
87. Workflowagent	202
87.1. class WorkflowAgent	202
87.2. Key Features	202
87.3. Relation to Workflow	202
87.4. Example Usage	202
87.5. Integration	202
87.5.1. Constants	202
87.5.2. Attributes	203
87.5.3. Public Class Methods	203
87.5.4. Public Instance Methods	203
87.5.5. Private Instance Methods	204
88. Workflowerror	206
88.1. class Flowbots::WorkflowError	206
89. Workflowinitializertask	207
89.1. class WorkflowInitializerTask	207
89.1.1. Constants	207
89.1.2. Public Instance Methods	207
89.1.3. Private Instance Methods	207
90. Workfloworchestrator	210
90.1. class WorkflowOrchestrator	210
90.1.1. Constants	210
90.1.2. Public Class Methods	210
90.1.3. Public Instance Methods	210
91. Workflows	214
91.1. class Flowbots::Workflows	214
91.1.1. Public Class Methods	214
91.1.2. Public Instance Methods	215
91.1.3. Private Instance Methods	216
92. Writefileaction	218
92.1. class Sublayer::Actions::WriteFileAction	218
92.1.1. Public Class Methods	218

92.1.2. Public Instance Methods	218
93. Yamlfrontmatter0	219
93.1. module MarkdownYaml::YamlFrontMatter0	219
94. Yamlfrontmatter1	220
94.1. module MarkdownYaml::YamlFrontMatter1	220
94.1.1. Public Instance Methods	220

Chapter 1. Api

1.1. class API

This class defines the **API** for the **Flowbots** application.

It provides endpoints for accessing and manipulating data stored in the Ohm models.

1.1.1. Private Instance Methods

splat_sort(splat_vals) [click to toggle source](#)

Sorts splat values from the request URL and organizes them into a hash.

@param splat_vals [Array] The splat values from the request URL.

@return [Hash] A hash containing sorted splat values.

```
# File lib/api.rb, line 182
def splat_sort(splat_vals)
  @output = {}
  @output[:object] = @object
  @output[:object_bucket] = @object_bucket
  @output[:collection] = @collection
  @output[:splat_vals] = splat_vals

  if @object_bucket == "document"
    @output[:segments] = splat_vals[splat_vals.index("segments") + 1..-1] if
@splat_vals.include?("segments")
    @output[:words] = splat_vals[splat_vals.index("words") + 1..-1] if
@splat_vals.include?("words")
  end

  @output
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 2. Apierror

2.1. class Flowbots::APIError

Error raised when there is a problem with an [API](#) call.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 3. AccumulateFilteredSegmentsTask

3.1. class AccumulateFilteredSegmentsTask

Task to accumulate filtered segments from all processed files.

3.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

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]

```
# File lib/tasks/accumulate_filtered_segments_task.rb, line 14
def execute
  logger.info "Starting AccumulateFilteredSegmentsTask"

  # Retrieve the current filtered segments from Redis
  current_filtered_segments = JSON.parse(RedisKeys.get("current_filtered_segments") ||
"[]")

  # Retrieve the accumulated filtered segments from Redis
  all_filtered_segments = JSON.parse(RedisKeys.get("all_filtered_segments") || "[]")

  # Clean the current filtered segments
  cleaned_segments = clean_segments(current_filtered_segments)

  # Append the cleaned segments to the accumulated filtered segments
  all_filtered_segments += cleaned_segments

  # Store the updated accumulated filtered segments in Redis
  RedisKeys.set("all_filtered_segments", all_filtered_segments.to_json)

  # Log a message indicating the number of accumulated segments
  logger.info "Accumulated #{cleaned_segments.length} cleaned segments. Total
segments: #{all_filtered_segments.length}"

  # Update the FileObject with the accumulated segments
  update_file_object(cleaned_segments)

  logger.info "Completed AccumulateFilteredSegmentsTask"
end
```

3.1.2. Private Instance Methods

`clean_segments(segments)` [click to toggle source](#)

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.

```
# File lib/tasks/accumulate_filtered_segments_task.rb, line 55
def clean_segments(segments)
  segments.reject do |segment|
    segment.empty? ||
    segment == %w[layout page] ||
    segment == ["true"] ||
    segment.first == "date" ||
    segment.include?("tags") ||
    segment.include?("title") ||
    segment.include?("toc")
  end.map do |segment|
    segment.reject { |word| word.to_s.match?(/^\\d+$/) } # Remove purely numeric words
  end.reject(&:empty?)
end
```

`retrieve_input()` [click to toggle source](#)

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

@return [FileObject] The current `FileObject`.

```
# File lib/tasks/accumulate_filtered_segments_task.rb, line 46
def retrieve_input
  retrieve_file_object
end
```

`update_file_object(cleaned_segments)` [click to toggle source](#)

Updates the `FileObject` with the given cleaned segments.

@param cleaned_segments [Array<Array<String>>] The cleaned segments to add to the `FileObject`.

@return [void]

```
# File lib/tasks/accumulate_filtered_segments_task.rb, line 74
def update_file_object(cleaned_segments)
  file_object = retrieve_input
  return if file_object.nil?
```

```
file_object.add_segments(cleaned_segments.map { |segment| segment.join(" ") })
logger.info "Updated FileObject #{file_object.id} with #{cleaned_segments.length}
new segments"
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 4. Actions

4.1. module Sublayer::Actions

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 5. Agent

5.1. class Agent

This file defines the Ohm models used in the **Flowbots** application.

5.1.1. Public Instance Methods

reflect(content) click to toggle source

Record a reflection for the agent.

@param content [String] The content of the reflection.

@return [Reflection] The created reflection.

```
# File lib/components/OhmModels.rb, line 44
def reflect(content)
  Reflection.create(agent: self, content: content, timestamp: Time.now)
end
```

retrieve_memories(type, query) click to toggle source

Retrieve memories based on type and query.

@param type [Symbol] The type of memory to retrieve (:episodic, :semantic, :reflection). @param query [String] The query to search for.

@return [Object] The retrieved memory object, or nil if no match is found.

```
# File lib/components/OhmModels.rb, line 54
def retrieve_memories(type, query)
  case type
  when :episodic
    episodic_memories.find(content: query).first
  when :semantic
    semantic_memories.find(content: query).first
  when :reflection
    reflections.find(content: query).first
  end
end
```

store_episodic_memory(content) click to toggle source

Store an episodic memory for the agent.

@param content [String] The content of the episodic memory.

@return [EpisodicMemory] The created episodic memory.

```
# File lib/components/OhmModels.rb, line 26
def store_episodic_memory(content)
  EpisodicMemory.create(agent: self, content: content, timestamp: Time.now)
end
```

update_semantic_memory(content) click to toggle source

Update the agent's semantic memory.

@param content [String] The content of the semantic memory.

@return [SemanticMemory] The created semantic memory.

```
# File lib/components/OhmModels.rb, line 35
def update_semantic_memory(content)
  SemanticMemory.create(agent: self, content: content, timestamp: Time.now)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 6. Agenterror

6.1. class Flowbots::AgentError

Error raised when there is a problem with an agent.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 7. Basetask

7.1. class Flowbots::BaseTask

The base class for all tasks in the **Flowbots** workflow.

Provides a common framework for task execution, status tracking, and error handling.

7.1.1. Attributes

ohm_task[R]

The OhmTask object associated with this task.

@return [OhmTask] The OhmTask object.

7.1.2. Public Class Methods

new(workflow, sourcefile) click to toggle source

Initializes a new **BaseTask** instance.

@param workflow [Workflow] The workflow object associated with this task. @param sourcefile [Sourcefile] The source file object associated with this task.

```
# File lib/tasks/base_task.rb, line 19
def initialize(workflow, sourcefile)
  @ohm_task = OhmTask.create(workflow, sourcefile)
end
```

7.1.3. Public Instance Methods

execute() click to toggle source

Executes the task.

Updates the task status to “running”, performs the task logic, updates the task status based on the result, and handles any errors.

@return [void]

```
# File lib/tasks/base_task.rb, line 29
def execute
  @ohm_task.update_status("running")
  begin
    result = perform
    @ohm_task.update_status("completed", result)
  rescue StandardError => e
```

```
@ohm_task.update_status("failed", e.message)
  raise
end
end
```

perform() click to toggle source

Performs the task logic.

This method must be implemented in subclasses to define the specific actions performed by the task.

@return [Object] The result of the task execution.

```
# File lib/tasks/base_task.rb, line 46
def perform
  raise NotImplementedError, "#{self.class.name}#perform must be implemented in
  subclass"
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 8. Batchcompletiontask

8.1. class BatchCompletionTask

This task handles the completion of a batch or single file workflow.

8.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the batch completion task.

Retrieves the current workflow from Redis and determines whether it's a batch workflow or a single file workflow. Calls the appropriate completion method based on the workflow type.

@return [void]

```
# File lib/tasks/batch_completion_task.rb, line 13
def execute
  workflow_id = Ohm.redis.get("current_workflow_id")
  workflow = Workflow[workflow_id]

  if workflow.is_batch_workflow
    complete_batch(workflow)
  else
    complete_single_file_workflow(workflow)
  end
end
```

8.1.2. Private Instance Methods

`complete_batch(workflow)` [click to toggle source](#)

Completes a batch in a batch workflow.

Updates the status of the current batch to “completed” and increments the current batch number if there are more batches to process. If all batches are completed, updates the workflow status to “completed” and sets the end time.

@param workflow [Workflow] The workflow object.

@return [void]

```
# File lib/tasks/batch_completion_task.rb, line 35
def complete_batch(workflow)
  current_batch = workflow.batches.find(number: workflow.current_batch_number).first
  current_batch.update(status: "completed")
end
```

```
    if workflow.current_batch_number < workflow.batches.count
      workflow.update(current_batch_number: workflow.current_batch_number + 1)
      logger.info "Completed Batch #{current_batch.number}, moving to next batch"
    else
      workflow.update(status: "completed", end_time: Time.now.to_s)
      logger.info "All batches completed. Workflow finished."
    end
  end
end
```

`complete_single_file_workflow(workflow)` [click to toggle source](#)

Completes a single file workflow.

Updates the workflow status to “completed” and sets the end time.

@param workflow [Workflow] The workflow object.

@return [void]

```
# File lib/tasks/batch_completion_task.rb, line 55
def complete_single_file_workflow(workflow)
  workflow.update(status: "completed", end_time: Time.now.to_s)
  logger.info "Single file workflow completed. File:
#{workflow.sourcefiles.first.path}"
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 9. Batchprocessor

9.1. 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.

9.1.1. 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.

9.1.2. Public Class Methods

`new(input_folder_path, batch_size=10, file_types=nil)` [click to toggle source](#)

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'].

```
# File lib/components/BatchProcessor.rb, line 23
def initialize(input_folder_path, batch_size=10, file_types=nil)
  @input_folder_path = input_folder_path || prompt_for_folder
  @batch_size = batch_size
  @file_types = file_types || FileDiscovery::FILE_TYPES[:text]
end
```

9.1.3. Public Instance Methods

`process_files(&block)` [click to toggle source](#)

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]

```
# File lib/components/BatchProcessor.rb, line 36
def process_files(&block)
  all_file_paths = discover_files
  total_files = all_file_paths.count
  num_batches = (total_files.to_f / batch_size).ceil

  num_batches.times do |i|
    batch_start = i * batch_size
    batch_files = all_file_paths[batch_start, batch_size]

    UI.say(:ok, "Processing batch #{i + 1} of #{num_batches}")
    logger.info "Processing batch #{i + 1} of #{num_batches}"

    process_batch(batch_files, &block)
  end
end
```

9.1.4. Private Instance Methods

discover_files() [click to toggle source](#)

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

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

```
# File lib/components/BatchProcessor.rb, line 70
def discover_files
  Dir.glob(File.join(input_folder_path, "**{,/*/**}/*#{file_types_pattern}")).sort
end
```

file_types_pattern() [click to toggle source](#)

Constructs a regular expression pattern from the `file_types` array.

@return [String] The regular expression pattern.

```
# File lib/components/BatchProcessor.rb, line 77
def file_types_pattern
  ".#{file_types.join(',')}"
end
```

process_batch(batch_files) \{ |file_path| ... } [click to toggle source](#)

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]

```
# File lib/components/BatchProcessor.rb, line 86
def process_batch(batch_files)
  batch_files.each do |file_path|
    yield(file_path) if block_given? && file_path && File.exist?(file_path)
  rescue StandardError => e
    logger.error "Error processing file #{file_path}: #{e.message}"
    UI.say(:error, "Error processing file #{file_path}: #{e.message}")
  end
end
```

prompt_for_folder() click to toggle source

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.

```
# File lib/components/BatchProcessor.rb, line 58
def prompt_for_folder
  get_folder_path = `gum file --directory`.chomp.strip
  folder_path = File.join(get_folder_path)

  raise FlowbotError.new("Folder not found", "FOLDERNOTFOUND") unless
    File.directory?(folder_path)

  folder_path
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 10. Box

10.1. module UI::Box

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

10.1.1. Public Instance Methods

`comparison_box(text1, text2, title1: "Text 1", title2: "Text 2")` [click to toggle source](#)

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.

```
# File lib/ui/box.rb, line 21
def comparison_box(text1, text2, title1: "Text 1", title2: "Text 2")
  # Calculate the screen width and text width.
  screen_width = TTY::Screen.width - 2
  text_width = [text1.length, text2.length, 40].max + 4
  # Determine the width of the boxes based on screen width, text width, and title
  width.
  width = [text_width, screen_width, TITLE_WIDTH].min

  # Create the first box with the specified title and padding.
  box1 = TTY::Box.frame(width:, title: { top_left: title1 }, padding: 1) { text1 }
  # Create the second box with the specified title and padding.
  box2 = TTY::Box.frame(width:, title: { top_left: title2 }, padding: 1) { text2 }
  # Concatenate the two boxes and return the result.
  box1 + box2
end
```

`eval_result_box(result, title: "Evaluation Result")` [click to toggle source](#)

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.

```
# File lib/ui/box.rb, line 42
def eval_result_box(result, title: "Evaluation Result")
  # Create a success box with the specified result, title, width, height, and padding.
  TTY::Box.success(
```

```

    result,
    title: { top_left: title },
    width: 80,
    height: (TTY::Screen.height / 1.25).round,
    padding: 1
  )
end

```

`exception_box(message)` [click to toggle source](#)

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.

```

# File lib/ui/box.rb, line 58
def exception_box(message)
  # Create an error box with the specified message, title, width, and padding.
  TTY::Box.error(
    message,
    title: { top_left: "Error" },
    width: [TTY::Screen.width - 2, TITLE_WIDTH].min,
    padding: 1
  )
end

```

`info_box(message, title: "Info")` [click to toggle source](#)

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.

```

# File lib/ui/box.rb, line 74
def info_box(message, title: "Info")
  # Create an info box with the specified message, title, width, and padding.
  TTY::Box.info(message, title: { top_left: title }, width: [TTY::Screen.width - 2,
50].min, padding: 1)
end

```

`multi_column_box(data, titles)` [click to toggle source](#)

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.

```
# File lib/ui/box.rb, line 85
def multi_column_box(data, titles)
  # Calculate the maximum width for each column based on the data.
  max_widths = data.transpose.map { |col| col.map(&:to_s).map(&:length).max }
  # Calculate the total width of the box based on column widths and separators.
  total_width = max_widths.sum + (3 * (max_widths.size - 1)) + 4

  # Format each row of data with appropriate padding.
  rows = data.map do |row|
    row.zip(max_widths).map { |cell, width| cell.to_s.ljust(width) }.join(" | ")
  end

  # Format the header row with appropriate padding.
  header = titles.zip(max_widths).map { |title, width| title.to_s.ljust(width) }.join(" | ")
  # Create a separator line for the header.
  separator = max_widths.map { |w| "-" * w }.join("-+-")

  # Combine the header, separator, and data rows into a single string.
  content = [
    header,
    separator,
    rows.join("\n")
  ].join("\n")

  # Create a framed box with the formatted content and appropriate width.
  TTY::Box.frame(content, width: [total_width, TTY::Screen.width - 2].min, padding: 1)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 11. Cli

11.1. class Flowbots::CLI

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

11.1.1. Public Class Methods

`exit_on_failure?()` [click to toggle source](#)

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.

```
# File lib/cli.rb, line 13
def self.exit_on_failure?
  true
end
```

11.1.2. Public Instance Methods

`process_text(file)` [click to toggle source](#)

Processes a text file using the text processing workflow.

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

@return [void]

```
# File lib/cli.rb, line 112
def process_text(file)
  pastel = Pastel.new

  # Check if the specified file exists.
  unless File.exist?(file)
    say pastel.red("File not found: #{file}")
    exit
  end

  # Display a message indicating the start of text processing.
  say pastel.green("Processing file: #{file}")

  # Process the text file and handle potential errors.
  begin
    workflow = TextProcessingWorkflow.new(file)
    workflow.run
    say pastel.green("Text processing completed successfully")
  rescue StandardError => e
```

```
ExceptionHandler.handle_exception(self.class.name, e)
end
end
```

`train_topic_model(folder)` [click to toggle source](#)

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]

```
# File lib/cli.rb, line 80
def train_topic_model(folder)
  pastel = Pastel.new

  # Check if the specified folder exists.
  unless Dir.exist?(folder)
    say pastel.red("Folder not found: #{folder}")
    exit
  end

  # Display a message indicating the start of topic model training.
  say pastel.green("Training topic model using files in: #{folder}")

  # Train the topic model and handle potential errors.
  begin
    workflow = TopicModelTrainerWorkflow.new(folder)
    workflow.run
    say pastel.green("Topic model training completed successfully")
  rescue StandardError => e
    ExceptionHandler.handle_exception(self.class.name, e)
  end
end
```

`version()` [click to toggle source](#)

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

@return [void]

```
# File lib/cli.rb, line 30
def version
  say "flowbots/#{VERSION} #{RUBY_DESCRIPTION}"
end
```

`workflows()` [click to toggle source](#)

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

@return [void]

```
# File lib/cli.rb, line 42
def workflows
  pastel = Pastel.new

  # Create a new Workflows instance.
  workflows = Workflows.new

  # List and select a workflow from the available options.
  selected_workflow = workflows.list_and_select

  # Exit if no workflow is selected.
  return unless selected_workflow

  # Run the selected workflow and handle potential errors.
  begin
    workflows.run(selected_workflow)
    say pastel.green("Workflow completed successfully")
  rescue Interrupt
    say pastel.yellow("Workflow interrupted by user")
  rescue FileNotFoundError => e
    say pastel.red(e.message)
  rescue StandardError => e
    ErrorHandler.handle_exception(self.class.name, e)
  ensure
    # Shut down Flowbots after the workflow execution.
    Flowbots.shutdown
  end
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 12. Cartridge

12.1. class Cartridge

Defines the `Cartridge` model.

12.1.1. Public Instance Methods

`evaluate(input)` [click to toggle source](#)

Evaluate the cartridge content (implementation depends on the cartridge format).

@param input [String] The input to `evaluate`.

@return [String] The result of the evaluation.

```
# File lib/components/OhmModels.rb, line 202
def evaluate(input)
  # Access settings from the loaded YAML hash
  provider = content['provider']
  settings = content['provider']['settings']

  # Example: If the cartridge is for Cohere
  if provider['id'] == 'cohere'
    # Use the Cohere API to generate text based on settings
    response = Cohere.generate(
      model: settings['model'],
      prompt: input,
      temperature: settings['temperature'],
      k: settings['k'],
      p: settings['p'],
      # ... other settings
    )
    return response.text
  end

  # ... (Implement evaluation logic for other providers)
end
```

`load_content()` [click to toggle source](#)

Load the cartridge content from the file.

@return [Hash] The loaded cartridge content as a hash.

```
# File lib/components/OhmModels.rb, line 193
def load_content
  YAML.safe_load(File.read(path)) # Use YAML.safe_load for secure parsing
```

```
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 13. Compressiontask

13.1. class CompressionTask

This task compresses a prompt using a [WorkflowAgent](#).

13.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the task.

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

```
# File lib/example.rb, line 106
def execute
  # Create a new WorkflowAgent instance for prompt compression.
  agent = WorkflowAgent.new("compression", "promptCompressor.yml")
  # Load the agent's state from Redis.
  agent.load_state

  # Prompt the user to enter a prompt to compress.
  puts "Enter prompt to compress:"

  # Get the prompt from the user using yad.
  sourcePrompt = `yad --entry`

  # Process the prompt using the agent.
  result = agent.process("#{sourcePrompt}")

  # Save the agent's state to Redis.
  agent.save_state

  # Store the compressed prompt in Redis.
  @@redis.set("initial_compression", result.to_json)
rescue StandardError => e
  # Log an error message if an exception occurs.
  logger.error "Error in CompressionTask: #{e.message}"
  # Re-raise the exception.
  raise
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 14. Compression test assessment task

14.1. class CompressionTestAssessmentTask

This task assesses a compression test evaluation using a [WorkflowAgent](#).

14.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the task.

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

```
# File lib/example.rb, line 231
def execute
  # Create a new WorkflowAgent instance for test design assessment.
  agent = WorkflowAgent.new("testdesigneval", "promptCompressorAssessment.yml")
  # Load the agent's state from Redis.
  agent.load_state

  # Retrieve the test evaluation from Redis.
  test_eval = JSON.parse(@@redis.get("testeval"))
  # Process the test evaluation using the agent to assess it.
  testassesment = agent.process("#{test_eval}")

  # Save the agent's state to Redis.
  agent.save_state
  # Store the test assessment in Redis.
  @@redis.set("assessment", testassesment.to_json)
rescue StandardError => e
  # Log an error message if an exception occurs.
  logger.error "Error in CompressionTestAssessmentTask: #{e.message}"
  # Re-raise the exception.
  raise
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 15. CompressionTestEvalTask

15.1. class CompressionTestEvalTask

This task evaluates a compression test design using a [WorkflowAgent](#).

15.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the task.

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

```
# File lib/example.rb, line 202
def execute
  # Create a new WorkflowAgent instance for test design evaluation.
  agent = WorkflowAgent.new("testdesigneval", "promptCompressorTestEval.yml")
  # Load the agent's state from Redis.
  agent.load_state

  # Retrieve the test design from Redis.
  test_design = JSON.parse(@@redis.get("testdesign"))
  # Process the test design using the agent to evaluate it.
  testeval = agent.process("#{test_design}")

  # Save the agent's state to Redis.
  agent.save_state
  # Store the test evaluation in Redis.
  @@redis.set("testeval", testeval.to_json)
rescue StandardError => e
  # Log an error message if an exception occurs.
  logger.error "Error in CompressionTestEvalTask: #{e.message}"
  # Re-raise the exception.
  raise
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 16. Compressiontesttask

16.1. class CompressionTestTask

This task designs a test for a compressed prompt using a [WorkflowAgent](#).

16.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the task.

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

```
# File lib/example.rb, line 140
def execute
  # Create a new WorkflowAgent instance for test design.
  agent = WorkflowAgent.new("testdesign", "promptCompressorTest.yml")
  # Load the agent's state from Redis.
  agent.load_state

  # Retrieve the compressed prompt from Redis.
  initial_compression = JSON.parse(@@redis.get("initial_compression"))
  # Process the compressed prompt using the agent to design a test.
  test_design = agent.process("#{initial_compression}")

  # Save the agent's state to Redis.
  agent.save_state

  # Save the Ruby test code to a file.
  File.write("compressed_prompt_test.rb", test_design)

  # Store the test design in Redis.
  @@redis.set("testdesign", test_design.to_json)
rescue StandardError => e
  # Log an error message if an exception occurs.
  logger.error "Error in CompressionTestTask: #{e.message}"
  # Re-raise the exception.
  raise
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 17. Configurationerror

17.1. class Flowbots::ConfigurationError

Error raised when there is a problem with the configuration.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 18. Converter

18.1. class TTY::Markdown::Converter

A converter class for converting Kramdown::Document trees to terminal output.

18.1.1. Public Instance Methods

`convert_p(ell, opts)` [click to toggle source](#)

Converts a paragraph element to terminal output.

@param ell [Kramdown::Element] The paragraph element to convert. @param opts [Hash] Options for the conversion.

@return [Array<String>] The converted paragraph text.

```
# File lib/helper.rb, line 87
def convert_p(ell, opts)
  indent = SPACE * @current_indent
  result = []

  # Add indentation unless the parent element is a blockquote or list item.
  result << indent unless %i[blockquote li].include?(opts[:parent].type)

  # Set the indentation level for the current element.
  opts[:indent] = @current_indent
  # Reset indentation to 0 if the parent element is a blockquote.
  opts[:indent] = 0 if opts[:parent].type == :blockquote

  # Convert the inner content of the paragraph element.
  content = inner(ell, opts)

  # Define a regular expression to match symbols that should not be followed by a
  newline.
  symbols = %q{[-!$%^&*()_+|~='{}\\[\]:";'<>?,.\./]}
  # Remove newlines that are not preceded or followed by symbols.
  # result << content.join.gsub(/(?<#{symbols})\n(?!#{symbols})/m) { " " }.gsub(/ +/)
  { " " }
  # Remove newlines that are not preceded or followed by symbols.
  result << content.join.gsub(/(?<#{symbols})\n(?!#{symbols})/m) { "" }
  # Add a newline if the last element does not end with a newline.
  result << NEWLINE unless result.last.to_s.end_with?(NEWLINE)
  result
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 19. Cursor

19.1. class Cursor

Provides helper methods for cursor positioning and terminal interaction.

19.1.1. Public Class Methods

`pos()` [click to toggle source](#)

Returns the current cursor position.

@return [Hash] A hash containing the row and column of the cursor.

```
# File lib/helper.rb, line 18
def pos
  res = +""
  $stdin.raw do |stdin|
    $stdout << "\e[6n"
    $stdout.flush
    while (c = stdin.getc) != "R"
      res << c if c
    end
  end
  m = res.match(/(?<row>\d+);(?<column>\d+)/)
  { row: Integer(m[:row]), column: Integer(m[:column]) }
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 20. Displayresultstask

20.1. class DisplayResultsTask

This task displays the results of the text processing workflow.

20.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

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]

```
# File lib/tasks/display_results_task.rb, line 14
def execute
  logger.info "Starting DisplayResultsTask"

  textfile = retrieve_input
  analysis_result = textfile.llm_analysis

  display_results(textfile, analysis_result)

  logger.info "DisplayResultsTask completed"
end
```

20.1.2. Private Instance Methods

`display_results(textfile, analysis_result)` [click to toggle source](#)

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]

```
# File lib/tasks/display_results_task.rb, line 40
def display_results(textfile, analysis_result)
  file_info = format_file_info(textfile)
  analysis = format_analysis(analysis_result)

  puts UI::ScrollableBox.side_by_side_boxes(
    file_info,
    analysis,
```

```

    title1: "File Information",
    title2: "LLM Analysis Result"
  )
end

```

`format_analysis(analysis_result)` [click to toggle source](#)

Formats the analysis results for display.

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

@return [String] The formatted analysis results.

```

# File lib/tasks/display_results_task.rb, line 75
def format_analysis(analysis_result)
  analysis_result.is_a?(String) ? analysis_result :
  JSON.pretty_generate(analysis_result)
end

```

`format_file_info(textfile)` [click to toggle source](#)

Formats the file information for display.

@param textfile [Textfile] The processed Textfile object.

@return [String] The formatted file information.

```

# File lib/tasks/display_results_task.rb, line 57
def format_file_info(textfile)
  <<~INFO
    Filename: #{textfile.name}
    Topics: #{textfile.topics.to_a.map(&:name).join(', ')}

    Content Preview:
    #{textfile.content}

    Total Segments: #{textfile.segments.count}
    Total Words: #{textfile.lemmas.count}
  INFO
end

```

`retrieve_input()` [click to toggle source](#)

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

@return [FileObject] The current `FileObject`.

```

# File lib/tasks/display_results_task.rb, line 30

```

```
def retrieve_input
  retrieve_file_object
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 21. Document0

21.1. module MarkdownYaml::Document0

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

21.1.1. Public Instance Methods

`markdown_content()` [click to toggle source](#)

The Markdown content section of the document.

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

```
# File lib/grammars/markdown_yaml.rb, line 26
def markdown_content
  elements[1]
end
```

`yaml_front_matter()` [click to toggle source](#)

The YAML front matter section of the document.

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

```
# File lib/grammars/markdown_yaml.rb, line 19
def yaml_front_matter
  elements[0]
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 22. Exceptionagent

22.1. class Flowbots::ExceptionAgent

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

22.1.1. Public Class Methods

`new()` [click to toggle source](#)

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

@return [void]

Calls superclass method **WorkflowAgent::new**

```
# File lib/components/ExceptionAgent.rb, line 13
def initialize
  logger.debug "Initialized ExceptionAgent"
  begin
    super("exception_handler", File.join(CARTRIDGE_DIR, "@b08x", "cartridges",
"exception_handler.yml"))
    @file_structure = load_file_structure
  rescue StandardError => e
    logger.error "#{e.message}"
  ensure
    UI.say(:error, "#{e}")
  end
end
end
```

22.1.2. Public Instance Methods

`process_exception(classname, exception)` [click to toggle source](#)

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.

```
# File lib/components/ExceptionAgent.rb, line 31
def process_exception(classname, exception)
  exception_details = {
    classname:,
    message: exception.message,
    backtrace: exception.backtrace&.join("\n")
  }
}
```



```

relevant_files = extract_relevant_files(exception)
exception_details[:relevant_files] = relevant_files

prompt = generate_exception_prompt(exception_details)

begin
  response = process(prompt)
  report = format_exception_report(response, exception_details)
  write_markdown_report(report, exception_details)
  report
rescue StandardError => e
  logger.error("Exception in ExceptionAgent: #{e.message}")
  fallback_report = fallback_exception_report(exception_details)
  write_markdown_report(fallback_report, exception_details)
  fallback_report
end
end

```

22.1.3. Private Instance Methods

`extract_relevant_files(exception)` [click to toggle source](#)

Extracts relevant files from the exception backtrace.

@param exception [Exception] The exception object.

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

```

# File lib/components/ExceptionAgent.rb, line 71
def extract_relevant_files(exception)
  relevant_files = {}
  exception.backtrace.each do |trace_line|
    file_path = trace_line.split(":").first
    file_name = File.basename(file_path)
    file_info = @file_structure["files"].find { |f| f["filename"] == file_name }
    relevant_files[file_name] = file_info["content"] if file_info &&
!relevant_files.key?(file_name)
  end
  relevant_files
end

```

`fallback_exception_report(exception_details)` [click to toggle source](#)

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.

```

# File lib/components/ExceptionAgent.rb, line 136
def fallback_exception_report(exception_details)
  <<~REPORT
    # □ FlowBot Exception Report (Fallback) □

    Oops! We encountered an exception, and we're having trouble generating a
    detailed report right now. Here's what we know:

    ## Exception Details

    - **Class:** #{exception_details[:classname]}
    - **Message:** #{exception_details[:message]}

    ### Backtrace

    ```
 #{exception_details[:backtrace]}
    ```

    We're working on resolving this issue. In the meantime, you might want to:

    1. Check your internet connection
    2. Verify that all required services are running
    3. Try your request again in a few minutes

    If the problem persists, please contact the development team with the above
    information.

    We apologize for any inconvenience!
  REPORT
end

```

`format_exception_report(agent_response, exception_details)` [click to toggle source](#)

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.

```

# File lib/components/ExceptionAgent.rb, line 110
def format_exception_report(agent_response, exception_details)
  <<~REPORT
    # □ FlowBot Exception Report □

    #{agent_response}

    ## Exception Details

```

```

- **Class:** #{exception_details[:classname]}
- **Message:** #{exception_details[:message]}

### Backtrace

```
#{exception_details[:backtrace]}
```

If you need more information, please check the logs or contact the development
team.
REPORT
end

```

`generate_exception_prompt(exception_details)` [click to toggle source](#)

Generates a prompt for the exception handler agent.

@param exception_details [Hash] A hash containing exception details.

@return [String] The prompt for the agent.

```

# File lib/components/ExceptionAgent.rb, line 87
def generate_exception_prompt(exception_details)
  <<~PROMPT
    Oh my stars!! Something terrible has happened!!

    Class: #{exception_details[:classname]}
    Message: #{exception_details[:message]}

    Backtrace:
    #{exception_details[:backtrace]}

    Relevant Files:
    #{exception_details[:relevant_files].map { |name, content|
    "#{name}:\n#{content}\n" }.join("\n")}

    Could you whip up a report that's easy for our team to understand but still
    includes all the important technical details? Thanks a bunch!
  PROMPT
end

```

`load_file_structure()` [click to toggle source](#)

Loads the file structure from the flowbots.json file.

@return [Hash] The file structure.

```
# File lib/components/ExceptionAgent.rb, line 61
def load_file_structure
  file_path = File.expand_path("../../flowbots.json", __dir__)
  JSON.parse(File.read(file_path))
end
```

`write_markdown_report(report, exception_details)` [click to toggle source](#)

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]

```
# File lib/components/ExceptionAgent.rb, line 171
def write_markdown_report(report, exception_details)
  timestamp = Time.now.strftime("%Y%m%d_%H%M%S")
  filename = "exception_report_#{timestamp}.md"
  dir_path = File.expand_path("../../exception_reports", __dir__)
  FileUtils.mkdir_p(dir_path)
  file_path = File.join(dir_path, filename)

  File.write(file_path, report)
  logger.info("Exception report written to: #{file_path}")
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 23. Exceptionhandler

23.1. class Flowbots::ExceptionHandler

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

23.1.1. Public Class Methods

`handle_exception(classname=nil, exception)` [click to toggle source](#)

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.

```
# File lib/components/ExceptionHandler.rb, line 20
def handle_exception(classname=nil, exception)
  exception_handler = ExceptionAgent.new

  begin
    report = exception_handler.process_exception(classname, exception)
  rescue APIError => e
    UI.info "API exception in ExceptionAgent: #{e.message}"
    logger.error("API exception in ExceptionAgent: #{e.message}")
    report = exception_handler.fallback_exception_report(classname, exception)
  end

  log_exception(exception)
  notify_exception(report)

  report
end
```

`log_exception(exception)` [click to toggle source](#)

Logs an exception to the application's logger.

@param exception [Exception] The exception object.

@return [void]

```
# File lib/components/ExceptionHandler.rb, line 42
def log_exception(exception)
  logger.error("FlowBot exception: #{exception.message}")
  logger.error(exception.backtrace.join("\n")) if exception.backtrace
end
```

```
end
```

`notify_exception(report)` click to toggle source

Notifies relevant parties about an exception.

@param report [String] The formatted exception report.

@return [void]

```
# File lib/components/ExceptionHandler.rb, line 52
def notify_exception(report)
  # Implement notification logic here
  # This could be sending an email, posting to a Slack channel, etc.
  UI.exception report
  logger.warn "Exception Notification:"
  logger.warn report
  logger.info "A detailed Markdown report has been generated in the exception_reports
directory."
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 24. FileDiscovery

24.1. module Flowbots::FileDiscovery

This module provides file discovery utilities for **Flowbots**.

24.1.1. Constants

FILE_TYPES

A constant hash defining file extensions grouped by their types.

24.1.2. Public Class Methods

discover_files(directory) [click to toggle source](#)

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 lib/components/FileDiscovery.rb, line 20
def self.discover_files(directory)
  files = Hash.new { |h, k| h[k] = [] }

  Dir.glob(File.join(directory, "**", "*")).each do |file|
    next unless File.file?(file)

    extension = File.extname(file).downcase
    FILE_TYPES.each do |type, extensions|
      if extensions.include?(extension)
        files[type] << file
        break
      end
    end
  end

  files
end
```

file_count(files) [click to toggle source](#)

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.

```
# File lib/components/FileDiscovery.rb, line 42
def self.file_count(files)
```

```
files.transform_values(&:count)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 25. Fileloader

25.1. class Flowbots::FileLoader

This class handles loading and processing text files.

25.1.1. Attributes

file_data[RW]

The `FileObject` object representing the loaded file.

25.1.2. Public Class Methods

new(file_path) [click to toggle source](#)

Initializes a new `FileLoader` instance.

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

@return [void]

```
# File lib/components/FileLoader.rb, line 20
def initialize(file_path)
  file_type = classify_file(file_path)

  extracted_text = extract_text(file_type, file_path)
  @file_data = store_file_data(file_path, extracted_text)
end
```

25.1.3. Private Instance Methods

classify_file(file_path) [click to toggle source](#)

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.

```
# File lib/components/FileLoader.rb, line 34
def classify_file(file_path)
  begin
    mime = MimeMagic.by_magic(file_path)
  rescue StandardError => e
    logger.debug "Unable to determine mime by magic alone, attempting by_path"
    mime = MimeMagic.by_path(file_path)
  end
```

```

case mime&.type
when /^text/
  :text
when %r{^application/pdf}
  :pdf
when /^image/
  :image
when /^video/
  :video
when /^audio/
  :audio
when /^json/
  :json
else
  :unknown
end
end
end

```

[extract_text\(file_type, file_path\)](#) click to toggle source

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.

```

# File lib/components/FileLoader.rb, line 79
def extract_text(file_type, file_path)
  case file_type
  # when :json
  #   extract_text_json(file_path)
  when :text
    File.read(file_path)
  when :unknown
    File.read(file_path)
    # extract_text_json(file_path)
  when :pdf
    parse_pdf(file_path)
  else
    raise NotImplementedError, "Unsupported file type: #{@file_type}"
  end
end
end

```

[extract_text_json\(file_path\)](#) click to toggle source

```

# File lib/components/FileLoader.rb, line 95
def extract_text_json(file_path)
  json_data = JSON.parse(File.read(file_path))

```

```

    text = json_data["results"]["channels"][0]["alternatives"][0]["transcript"]
    puts json_data
    return text
  end
rescue JSON::ParserError => e
  puts "Error parsing JSON: #{e.message}"
  nil
rescue StandardError => e
  puts "An error occurred: #{e.message}"
  nil
end

```

`parse_pdf(file_path)` [click to toggle source](#)

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.

```

# File lib/components/FileLoader.rb, line 65
def parse_pdf(file_path)
  text = ""
  PDF::Reader.new(file_path).pages.each do |page|
    text << page.text
  end
  text
end

```

`store_file_data(file_path, extracted_text)` [click to toggle source](#)

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.

```

# File lib/components/FileLoader.rb, line 115
def store_file_data(file_path, extracted_text)
  file = FileObject.find_or_create_by_path(
    file_path, attributes = { content: extracted_text }
  )

  return file

  logger.info "Stored file data for: #{file_path}"
  UI.say(:ok, "Stored file data for: #{file_path}")
end

```

```
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 26. Fileloadertask

26.1. class FileLoaderTask

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

26.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

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.

```
# File lib/tasks/file_loader_task.rb, line 13
def execute
  logger.info "Starting FileLoaderTask"

  input_file_path = retrieve_input

  file_processor = Flowbots::FileLoader.new(input_file_path)
  text_file = file_processor.file_data

  if text_file.nil? || text_file.id.nil?
    logger.error "Failed to load FileObject"
    raise FlowbotError.new("FileObject not found", "FILENOTFOUND")
  end

  store_FileObject_id(text_file.id)

  logger.info "Loaded FileObject with ID: #{text_file.id}"
  UI.say(:ok, "Loaded FileObject with ID: #{text_file.id}")
end
```

26.1.2. Private Instance Methods

`retrieve_input()` [click to toggle source](#)

Retrieves the input file path from Redis.

`@return [String]` The input file path.

```
# File lib/tasks/file_loader_task.rb, line 37
def retrieve_input
  retrieve_file_path
```

```
end
```

store_FileObject_id(id) click to toggle source

Stores the `FileObject` ID in Redis.

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

@return [void]

```
# File lib/tasks/file_loader_task.rb, line 46
def store_FileObject_id(id)
  RedisKeys.set(RedisKeys::CURRENT_FileObject_ID, id)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 27. FileNotFoundError

27.1. class Flowbots::FileNotFoundError

Custom error class for workflow file not found.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 28. Fileobject

28.1. class FileObject

Defines the `FileObject` model.

28.1.1. Public Class Methods

`current_batch()` [click to toggle source](#)

Retrieves all `FileObjects` from the current batch.

@return [Array<FileObject>] An array of `FileObjects` from the current batch.

```
# File lib/components/OhmModels.rb, line 395
def self.current_batch
  batch_id = redis.call("GET", "current_batch_id")
  find(batch: batch_id)
end
```

`find_or_create_by_path(file_path, attributes = {})` [click to toggle source](#)

Finds or creates a new `FileObject` instance based on the file path.

@param file_path [String] The path to the file. @param attributes [Hash] A hash of attributes for the new `FileObject`.

@return [FileObject] The found or created `FileObject` instance.

```
# File lib/components/OhmModels.rb, line 261
def self.find_or_create_by_path(file_path, attributes = {})
  raise ArgumentError, "file_path must be a String, got #{file_path.class}" unless
    file_path.is_a?(String)

  existing_file = find(path: file_path).first
  return existing_file if existing_file

  file_name = File.basename(file_path)
  extension = File.extname(file_path)

  create(
    attributes.merge(
      name: file_name,
      path: file_path,
      extension:,
      content: File.read(file_path)
    )
  )
end
```



```

rescue Errno::ENOENT
  raise FileNotFound, "File not found: #{file_path}"
rescue StandardError => e
  raise FileObjectError, "Error creating FileObject: #{e.message}"
end

```

latest(limit = nil) [click to toggle source](#)

Retrieves the **latest FileObject** from Redis.

@param limit [Integer] The maximum number of FileObjects to retrieve.

@return [FileObject, Array<FileObject>] The **latest FileObject** or an array of FileObjects.

```

# File lib/components/OhmModels.rb, line 381
def self.latest(limit = nil)
  if limit.nil?
    ids = redis.call("ZREVRANGE", key[:latest], 0, 0)
    result = fetch(ids)
    result.empty? ? nil : result.first
  else
    ids = redis.call("ZREVRANGE", key[:latest], 0, limit - 1)
    fetch(ids)
  end
end

```

28.1.2. Public Instance Methods

add_lemma(lemma_data) [click to toggle source](#)

Adds a new lemma to the **FileObject**.

@param lemma_data [Hash] A hash containing lemma data.

@return [Lemma] The created lemma.

```

# File lib/components/OhmModels.rb, line 312
def add_lemma(lemma_data)
  lemma = Lemma.create(lemma_data.merge(file_object: self))
  lemmas.push(lemma)
  save
  lemma
end

```

add_lemmas(lemmas_data) [click to toggle source](#)

Adds multiple lemmas to the **FileObject**.

@param lemmas_data [Array<Hash>] An array of lemma data hashes.

@return [Array<Lemma>] An array of created lemmas.

```
# File lib/components/OhmModels.rb, line 324
def add_lemmas(lemmas_data)
  new_lemmas = lemmas_data.map do |lemma_data|
    lemma = Lemma.create(lemma_data.merge(file_object: self))
    lemmas.push(lemma)
  end
  lemma
end
save
new_lemmas
end
```

`add_segment(text)` [click to toggle source](#)

Adds a new segment to the `FileObject`.

@param text [String] The text of the new segment.

@return [Segment] The created segment.

```
# File lib/components/OhmModels.rb, line 289
def add_segment(text)
  segment = Segment.create(text:, file_object: self)
  segments.push(segment)
  save
  segment
end
```

`add_segments(new_segments)` [click to toggle source](#)

Adds multiple segments to the `FileObject`.

@param new_segments [Array<String>] An array of segment texts.

@return [void]

```
# File lib/components/OhmModels.rb, line 301
def add_segments(new_segments)
  new_segments.each do |segment_text|
    add_segment(segment_text)
  end
end
```

`add_topics(new_topics)` [click to toggle source](#)

Adds multiple topics to the `FileObject`.

@param new_topics [Array<String>] An array of topic names.

@return [void]

```
# File lib/components/OhmModels.rb, line 339
def add_topics(new_topics)
  new_topics.each do |word|
    topics.add(Topic.create(name: word))
  rescue StandardError => e
    logger.warn e.message.to_s
  end
  save
end
```

retrieve_segment_texts() [click to toggle source](#)

Retrieves the text of all segments associated with the **FileObject**.

@return [Array<String>] An array of segment texts.

```
# File lib/components/OhmModels.rb, line 358
def retrieve_segment_texts
  retrieve_segments.map(&:text)
end
```

retrieve_segments() [click to toggle source](#)

Retrieves all segments associated with the **FileObject**.

@return [Array<Segment>] An array of segments.

```
# File lib/components/OhmModels.rb, line 351
def retrieve_segments
  segments.to_a
end
```

retrieve_word_texts() [click to toggle source](#)

Retrieves the text of all words associated with the **FileObject**.

@return [Array<String>] An array of word texts.

```
# File lib/components/OhmModels.rb, line 372
def retrieve_word_texts
  retrieve_words.map(&:word)
end
```

[retrieve_words\(\)](#) click to toggle source

Retrieves all words associated with the **FileObject**.

@return [Array<Word>] An array of words.

```
# File lib/components/OhmModels.rb, line 365
def retrieve_words
  segments.to_a.flat_map { |segment| segment.words.to_a }
end
```

28.1.3. Protected Instance Methods

[after_delete\(\)](#) click to toggle source

Removes the **FileObject**'s ID from the Redis set for **latest** FileObjects.

@return [void]

```
# File lib/components/OhmModels.rb, line 412
def after_delete
  redis.call("ZREM", model.key[:latest], id)
end
```

[after_save\(\)](#) click to toggle source

Adds the **FileObject**'s ID to the Redis set for **latest** FileObjects.

@return [void]

```
# File lib/components/OhmModels.rb, line 405
def after_save
  redis.call("ZADD", model.key[:latest], Time.now.to_f, id)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 29. Fileobjecterror

29.1. class Flowbots::FileObjectError

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 30. Filtersegmentstask

30.1. class FilterSegmentsTask

lib/tasks/filter_segments_task.rb

30.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the segment filtering task.

Retrieves the file object, filters segments based on word tags, stores the filtered segments in Redis, logs relevant information, and displays the filtered segments to the user.

@return [void]

```
# File lib/tasks/filter_segments_task.rb, line 16
def execute
  file_object = retrieve_input

  logger.info "Processing FilterSegmentsTask for file: #{file_object.name}"

  filtered_segments = filter_segments(file_object)
  Jongleur::WorkerTask.class_variable_get(:@@redis).set("current_filtered_segments",
  filtered_segments.to_json)

  logger.info "Filtered #{filtered_segments.length} segments"

  display_filtered_segments(filtered_segments)
end
```

30.1.2. Private Instance Methods

`display_filtered_segments(filtered_segments)` [click to toggle source](#)

Displays the filtered segments to the user.

@param filtered_segments [Array<Array<String>>] An array of filtered segments. @return [void]

```
# File lib/tasks/filter_segments_task.rb, line 86
def display_filtered_segments(filtered_segments)
  UI.say(:ok, "#{filtered_segments}")
end
```

`filter_segment_words(segment)` [click to toggle source](#)

Filters words from a segment based on relevant parts of speech (POS) tags.

@param segment [Segment] The segment to filter words from. @return [Array<String>, nil] An array of filtered words or nil if the segment has invalid tagged data.

```
# File lib/tasks/filter_segments_task.rb, line 54
def filter_segment_words(segment)

  logger.debug "Processing segment: #{segment.id}"

  tagged = segment.tagged

  if tagged.nil? || !tagged.is_a?(Hash) || !tagged["pos"]
    logger.warn "Segment #{segment.id} has invalid tagged data: #{tagged.inspect}"
    return nil
  end

  relevant_pos = %w[PROPN NOUN]
  relevant_dep = %w[nsubj compound nsubjpass]
  relevant_ner = %w[PERSON ORGANIZATION]

  words = tagged["pos"].keys

  filtered_words = words.select do |word|
    relevant_pos.include?(tagged["pos"][word]) ||
    relevant_dep.include?(tagged["dep"][word]) ||
    relevant_ner.include?(tagged["ner"][word])
  end

  logger.debug "Filtered #{filtered_words.length} words from segment #{segment.id}"

  filtered_words
end
```

`filter_segments(file_object)` [click to toggle source](#)

Filters segments based on their word tags.

@param file_object [FileObject] The file object containing the segments. @return [Array<Array<String>>] An array of filtered segments, where each segment is an array of words.

```
# File lib/tasks/filter_segments_task.rb, line 42
def filter_segments(file_object)
  segments = file_object.segments

  segments.map do |segment|
    filter_segment_words(segment)
  end.compact
end
```

`retrieve_input()` click to toggle source

Retrieves the input file object.

@return [FileObject] The retrieved file object.

```
# File lib/tasks/filter_segments_task.rb, line 34
def retrieve_input
  retrieve_file_object
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 31. Finalreporttask

31.1. class FinalReportTask

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

31.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the task.

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

```
# File lib/example.rb, line 260
def execute
  # Create a new WorkflowAgent instance for final report generation.
  agent = WorkflowAgent.new("finalreport", "finalReportGenerator.yml")
  # Load the agent's state from Redis.
  agent.load_state

  # Retrieve the initial compression, test design, test results, test evaluation, and
  # assessment from Redis.
  initial_compression = JSON.parse(@@redis.get("initial_compression") || "{}")
  test_design = JSON.parse(@@redis.get("testdesign") || "{}")
  test_results = @@redis.get("test_results") || ""
  test_eval = JSON.parse(@@redis.get("testeval") || "{}")
  assessment = JSON.parse(@@redis.get("assessment") || "{}")

  # Create a hash containing the report input data.
  report_input = {
    initial_compression:,
    test_design:,
    test_results:,
    test_eval:,
    assessment:
  }

  # Process the report input using the agent to generate the final report.
  final_report = agent.process(report_input.to_json)

  # Save the agent's state to Redis.
  agent.save_state

  # Save the final report as a Markdown file.
  File.write("final_report.md", final_report)

  # Print a message indicating that the final report has been generated.
  puts "Final report has been generated and saved as 'final_report.md'"
end
```

```
rescue StandardError => e
  # Log an error message if an exception occurs.
  logger.error "Error in FinalReportTask: #{e.message}"
  logger.error e.backtrace.join("\n")
  # Re-raise the exception.
  raise
end
```

Validate

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 32. Flowboterror

32.1. class Flowbots::FlowbotError

Base class for all **Flowbots** errors.

32.1.1. Attributes

details[R]

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

error_code[R]

@return [String] The error code.

32.1.2. Public Class Methods

new(message, error_code, details={}) [click to toggle source](#)

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

```
# File lib/flowbots/errors.rb, line 17
def initialize(message, error_code, details={})
  super(message)
  @error_code = error_code
  @details = details
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 33. Flowbots

33.1. module Flowbots

Module for **Flowbots** application.

This module defines the base class for all tasks in the **Flowbots** workflow.

33.1.1. Constants

BATCH

Constant indicating whether the application is running in batch mode.

IN_CONTAINER

Constant indicating whether the application is running in a container.

33.1.2. Public Class Methods

`initialize()` [click to toggle source](#)

Initializes the **Flowbots** application.

@return [void]

```
# File lib/flowbots.rb, line 105
def initialize
  setup_redis
  load_components
end
```

`shutdown()` [click to toggle source](#)

Shuts down the **Flowbots** application.

@return [void]

```
# File lib/flowbots.rb, line 113
def shutdown
  Ohm.redis.quit
  stop_running_workflows
  logger.info "Flowbots shut down gracefully"
end
```

33.1.3. Private Class Methods

`load_components()` [click to toggle source](#)

Loads the necessary components for the application.

@return [void]

```
# File lib/flowbots.rb, line 134
def load_components
  Workflows.load_workflows
  Task.load_tasks
end
```

setup_redis() click to toggle source

Sets up the Redis connection for Ohm.

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

```
# File lib/flowbots.rb, line 125
def setup_redis
  Ohm.redis = Redic.new("redis://localhost:6378/0")
rescue Ohm::Error => e
  ExceptionHandler.handle_exception(e)
end
```

stop_running_workflows() click to toggle source

Stops any running workflows.

@return [void]

```
# File lib/flowbots.rb, line 142
def stop_running_workflows
  logger.info "All workflows stopped"
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 34. FlowiseApiClient

34.1. class FlowiseApiClient

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

34.1.1. Public Class Methods

`new(base_url)` click to toggle source

Initializes a new `FlowiseApiClient` instance.

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

@return [void]

```
# File lib/integrations/flowise.rb, line 15
def initialize(base_url)
  @base_url = base_url
  @conn = Faraday.new(url: @base_url) do |faraday|
    faraday.request :multipart
    faraday.request :json
    faraday.adapter Faraday.default_adapter
  end
end
```

34.1.2. Public Instance Methods

`predict(chatflow_id, options={})` click to toggle source

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](#).

```
# File lib/integrations/flowise.rb, line 38
def predict(chatflow_id, options={})
  endpoint = "/api/v1/prediction/#{chatflow_id}"

  if options[:file_path]
    # File upload case
```

```

    payload = {
      files: Faraday::Multipart::FilePart.new(
        options[:file_path],
        "application/octet-stream",
        File.basename(options[:file_path])
      )
    }
    payload[:workerName] = options[:worker_name] if options[:worker_name]
    payload[:workerPrompt] = options[:worker_prompt] if options[:worker_prompt]
    payload[:promptValues] = options[:prompt_values].to_json if
options[:prompt_values]

    response = @conn.post(endpoint) do |req|
      req.headers["Content-Type"] = "multipart/form-data"
      req.body = payload
    end
  else
    # Simple JSON query case
    payload = options.slice(:question, :history, :overrideConfig, :socketIOClientId)

    response = @conn.post(endpoint) do |req|
      req.headers["Content-Type"] = "application/json"
      req.body = payload.to_json
    end
  end

  handle_response(response)
end

```

upsert_document(chatflow_id, file_path, local_ai_config={}) click to toggle source

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](#).

```

# File lib/integrations/flowise.rb, line 81
def upsert_document(chatflow_id, file_path, local_ai_config={})
  endpoint = "/api/v1/vector/upsert/#{chatflow_id}"

  payload = {
    files: Faraday::Multipart::FilePart.new(
      file_path,
      "application/octet-stream",
      File.basename(file_path)
    ),

```

```

    localAIApiKey: local_ai_config[:api_key],
    basePath: local_ai_config[:base_path],
    modelName: local_ai_config[:model_name]
  }

  response = @conn.post(endpoint) do |req|
    req.headers["Content-Type"] = "multipart/form-data"
    req.body = payload
  end

  handle_response(response)
end

```

34.1.3. Private Instance Methods

handle_response(response) [click to toggle source](#)

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.

```

# File lib/integrations/flowise.rb, line 111
def handle_response(response)
  case response.status
  when 200
    JSON.parse(response.body)
  else
    raise "Flowise API Error: #{response.status} - #{response.body}"
  end
end

```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 35. Grammarprocessor

35.1. class Flowbots::GrammarProcessor

This class handles parsing text using a specified grammar.

35.1.1. Public Class Methods

`new(grammar_name)` [click to toggle source](#)

Initializes a new `GrammarProcessor` instance.

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

@return [void]

```
# File lib/processors/GrammarProcessor.rb, line 12
def initialize(grammar_name)
  @grammar_name = grammar_name
  load_grammar
end
```

35.1.2. Public Instance Methods

`parse(text)` [click to toggle source](#)

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.

```
# File lib/processors/GrammarProcessor.rb, line 23
def parse(text)
  logger.debug "Parsing text: #{text[0..100]}..." # Log first 100 characters of the
text
  result = @parser.parse(text)
  if result
    yaml_front_matter = extract_yaml_front_matter(result)
    markdown_content = extract_markdown_content(result)
    logger.debug "Parsing successful."
    logger.debug "YAML Front Matter: #{yaml_front_matter.inspect}"
    logger.debug "Markdown Content: #{markdown_content[0..100].inspect}..." # Log
first 100 characters
    {
      yaml_front_matter:,
      markdown_content:
```

```

    }
  else
    logger.error "Parsing failed. Parser errors: #{@parser.failure_reason}"
    logger.error "Failure index: #{@parser.index}"
    logger.error "Failure line and column:
#{@parser.failure_line}:#{@parser.failure_column}"
    logger.error "Context: #{text[@parser.index, 50]}" # Log 50 characters of context
    around failure point
    nil
  end
end
end

```

35.1.3. Private Instance Methods

`extract_markdown_content(parse_result)` [click to toggle source](#)

Extracts the Markdown content from the `parse` result.

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

@return [String] The extracted Markdown content.

```

# File lib/processors/GrammarProcessor.rb, line 81
def extract_markdown_content(parse_result)
  markdown_content = parse_result.elements[1].text_value
  markdown_content.strip
end

```

`extract_yaml_front_matter(parse_result)` [click to toggle source](#)

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.

```

# File lib/processors/GrammarProcessor.rb, line 70
def extract_yaml_front_matter(parse_result)
  yaml_front_matter = parse_result.elements[0].text_value
  yaml_content = yaml_front_matter.gsub(/\A---\n/, "").gsub(/\n---\n\nz/, "")
  yaml_content.strip
end

```

`load_grammar()` [click to toggle source](#)

Loads the grammar file and creates a parser instance.

@return [void]

```
# File lib/processors/GrammarProcessor.rb, line 50
def load_grammar
  grammar_file = File.join(GRAMMAR_DIR, "#{@grammar_name}.treetop")
  logger.debug "Loading grammar file: #{grammar_file}"
  Treetop.load(grammar_file)
  logger.debug "Grammar file loaded successfully"

  parser_class_name = "#{@grammar_name.split('_').map(&:capitalize).join}Parser"
  @parser = Object.const_get(parser_class_name).new
  logger.debug "Parser created: #{@parser.class}"
rescue StandardError => e
  logger.error "Error loading grammar: #{e.message}"
  logger.error e.backtrace.join("\n")
  raise
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 36. Inputretrieval

36.1. module InputRetrieval

Module for retrieving input data.

36.1.1. Public Instance Methods

`retrieve_file_object()` [click to toggle source](#)

Retrieves the `FileObject` from Redis.

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

```
# File lib/components/InputRetrieval.rb, line 32
def retrieve_file_object
  file_object_id = RedisKeys.get(RedisKeys::CURRENT_FILE_OBJECT_ID)
  if file_object_id.nil?
    logger.error "No FileObject ID found in Redis"
    return nil
  end

  file_object = FileObject[file_object_id]
  if file_object.nil?
    logger.error "No FileObject found for ID: #{file_object_id}"
    return nil
  end

  file_object
end
```

`retrieve_file_path()` [click to toggle source](#)

Retrieves the file path from Redis.

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

```
# File lib/components/InputRetrieval.rb, line 20
def retrieve_file_path
  file_path = RedisKeys.get(RedisKeys::CURRENT_FILE_PATH)
  if file_path.nil? || file_path.empty?
    logger.error "File path not found in Redis"
    raise ArgumentError, "File path not found"
  end

  file_path
end
```

[retrieve_input\(\)](#) click to toggle source

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.

```
# File lib/components/InputRetrieval.rb, line 12
def retrieve_input
  retrieve_file_object
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 37. Jongleur

37.1. module Jongleur

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 38. License

The MIT License (MIT)

Copyright © 2024 Robert Pannick

Permission is hereby granted, free of charge, to any person obtaining a copy of 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:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

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.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 39. Lemma

39.1. class Lemma

Defines the `Lemma` model.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 40. Llmanalysistask

40.1. class LlmAnalysisTask

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

40.1.1. Public Instance Methods

execute() [click to toggle source](#)

Executes the task.

@return [void]

```
# File lib/tasks/llm_analysis_task.rb, line 10
def execute
  UI.footer

  UI.info "Starting LLMAnalysisTask"

  begin
    agent = WorkflowAgent.new(
      "ironically_literal",
      File.join(CARTRIDGE_DIR, "@b08x", "cartridges", "assistants/antisteve.yml")
    )

    logger.debug "Created WorkflowAgent instance"

    agent.load_state
    logger.debug "Loaded agent state"

    textfile = retrieve_input
    content = textfile.preprocessed_content
    metadata = textfile.metadata || {}
    nlp_result = retrieve_nlp_result(textfile)

    prompt = generate_analysis_prompt(textfile, content, metadata, nlp_result)

    analysis_result = agent.process(prompt)

    logger.debug "Agent processing completed"

    agent.save_state
    logger.debug "Saved agent state"

    store_analysis_result(textfile, analysis_result)

    write_markdown_report(analysis_result)
```

```

    logger.debug "Stored analysis result"

    logger.info "LLMAalysisTask completed"
  rescue StandardError => e
    logger.error "Error in LLMAalysisTask: #{e.message}"
    logger.error e.backtrace.join("\n")
    raise
  end
end
end

```

40.1.2. Private Instance Methods

`format_nlp_result(nlp_result)` [click to toggle source](#)

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.

```

# File lib/tasks/llm_analysis_task.rb, line 143
def format_nlp_result(nlp_result)
  nlp_result.first(10).map do |segment|
    <<~SEGMENT
      Segment: "#{segment[:text]}"
      Parts of Speech: #{segment[:tagged][:pos].to_a.first(5).map { |word, pos|
        "#{word}({pos})" }.join(', ')}
      Named Entities: #{segment[:tagged][:ner].to_a.first(5).map { |word, ner|
        "#{word}({ner})" }.join(', ')}
      SEGMENT
    end.join("\n")
  end
end

```

`generate_analysis_prompt(textfile, content, metadata, nlp_result)` [click to toggle source](#)

```
Please structure your response in a clear, concise manner. Thank you!
```

PROMPT end

```

# File lib/tasks/llm_analysis_task.rb, line 126
def generate_analysis_prompt(textfile, content, metadata, nlp_result)
  <<~PROMPT
    Greetings, Here's what we have:

    Document Name: #{textfile.name}

    Content:
    #{content}
  end
end

```

```
PROMPT
end
```

`retrieve_file_metadata()` [click to toggle source](#)

Retrieves the file metadata from Redis.

@return [Hash] The file metadata.

```
# File lib/tasks/llm_analysis_task.rb, line 78
def retrieve_file_metadata
  JSON.parse(Jongleur::WorkerTask.class_variable_get(:@@redis).get("file_metadata") ||
"{}")
end
```

`retrieve_input()` [click to toggle source](#)

```
# File lib/tasks/llm_analysis_task.rb, line 56
def retrieve_input
  retrieve_file_object
end
```

`retrieve_nlp_result(textfile)` [click to toggle source](#)

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.

```
# File lib/tasks/llm_analysis_task.rb, line 87
def retrieve_nlp_result(textfile)
  textfile.retrieve_segments.map do |segment|
    {
      text: segment.text,
      tagged: segment.tagged
    }
  end
end
```

`store_analysis_result(textfile, result)` [click to toggle source](#)

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]

```
# File lib/tasks/llm_analysis_task.rb, line 159
def store_analysis_result(textfile, result)
  textfile.update(llm_analysis: result)
  # Jongleur::WorkerTask.class_variable_get(:@@redis).set("analysis_result",
  result.to_json)
end
```

write_markdown_report(result) [click to toggle source](#)

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]

```
# File lib/tasks/llm_analysis_task.rb, line 170
def write_markdown_report(result)
  timestamp = Time.now.strftime("%Y%m%d_%H%M%S")
  filename = "llm_analysis_task_#{timestamp}.md"
  dir_path = File.expand_path("../..//llm_analysis", __dir__)
  FileUtils.mkdir_p(dir_path)
  file_path = File.join(dir_path, filename)

  File.write(file_path, result)
  logger.info("Exception report written to: #{file_path}")
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 41. Loadfileobjecttask

41.1. class LoadFileObjectTask

This task loads a `FileObject` from the file system and stores its ID in Redis.

41.1.1. Public Instance Methods

`execute()` click to toggle source

Executes the task to load a `FileObject`.

This method 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.

The loading process involves:

1. **Retrieving the file path:** The `retrieve_file_path` method is called to retrieve the file path from Redis.
2. **Finding or creating a FileObject:** The `FileObject.find_or_create_by_path` method is called with the retrieved file path to find an existing `FileObject` or create a new one if it doesn't exist. The result is stored in the `file_object` variable.
3. **Handling errors:** If the `file_object` is nil, meaning the `FileObject` could not be found or created, an error message is logged and a `RuntimeError` is raised.
4. **Storing the FileObject ID:** If the `FileObject` was successfully found or created, the `store_file_object_id` method is called to store the `FileObject` ID in Redis.
5. **Logging information:** The `logger` object is used to log relevant information about the loading process, including success messages and error messages.

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

```
# File lib/tasks/load_file_object_task.rb, line 32
def execute
  logger.info "Starting LoadFileObjectTask"

  begin
    # Retrieve the file path from Redis.
    file_path = retrieve_file_path
    # Find or create a FileObject based on the retrieved file path.
    file_object = FileObject.find_or_create_by_path(file_path)

    # Raise an error if the FileObject is not found or created.
    if file_object.nil?
      error_message = "Failed to create or find FileObject for: #{file_path}"
      logger.error error_message
      raise(error_message)
    end
  end
end
```

```

        return
    end

    # Store the FileObject ID in Redis.
    store_file_object_id(file_object.id)
    # Log messages indicating the successful loading of the FileObject.
    logger.info "Loaded FileObject with ID: #{file_object.id}"
    UI.say(:ok, "Loaded FileObject with ID: #{file_object.id}")
    complete("Successfully loaded FileObject for: #{file_path}")
  rescue ArgumentError => e
    # Handle ArgumentError, typically raised for invalid input.
    error_message = "Invalid input: #{e.message}"
    logger.error error_message
    raise(error_message)
  rescue StandardError => e
    # Handle any other StandardError during file loading.
    error_message = "Error loading file: #{e.message}"
    logger.error error_message
    logger.error e.backtrace.join("\n")
    UI.say(:error, "Failed to load file")
    raise(error_message)
  end
end
end

```

41.1.2. Private Instance Methods

`retrieve_file_path()` [click to toggle source](#)

Retrieves the file path from Redis.

This method retrieves the file path from Redis using the `RedisKeys` class. It gets the value of the `RedisKeys::CURRENT_FILE_PATH` key, which represents the current file path being processed.

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

```

# File lib/tasks/load_file_object_task.rb, line 79
def retrieve_file_path
  RedisKeys.get(RedisKeys::CURRENT_FILE_PATH)
end

```

`store_file_object_id(id)` [click to toggle source](#)

Stores the `FileObject` ID in Redis.

This method stores the `FileObject` ID in Redis using the `RedisKeys` class. It sets the value of the `RedisKeys::CURRENT_FILE_OBJECT_ID` key to the provided `id`. This key is used to track the currently loaded `FileObject`.

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

```
# File lib/tasks/load_file_object_task.rb, line 91
def store_file_object_id(id)
  RedisKeys.set(RedisKeys::CURRENT_FILE_OBJECT_ID, id)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 42. Loadtextfilestask

42.1. class LoadTextFilesTask

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

42.1.1. Public Instance Methods

execute() click to toggle source

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

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

The loading process involves:

1. **Retrieving the file path:** The `retrieve_input` method is called to retrieve the file path from Redis.
2. **Loading the file:** A new `Flowbots::FileLoader` instance is created with the retrieved file path. The `file_data` method of the `FileLoader` is called to load the file data. The loaded data is stored in the `textfile` variable.
3. **Storing the Textfile ID:** If the file was loaded successfully and the `textfile` variable contains a valid `Textfile` object, the `store_textfile_id` method is called to store the `Textfile` ID in Redis.
4. **Logging information:** The `logger` object is used to log relevant information about the loading process, including success messages, error messages, and debug information.

@return [void]

```
# File lib/tasks/load_text_files_task.rb, line 30
def execute
  # Log the start of the task.
  logger.info "Starting LoadTextFilesTask"

  # Retrieve the file path from Redis.
  file_path = retrieve_input

  begin
    # Create a new FileLoader instance with the retrieved file path.
    file_loader = Flowbots::FileLoader.new(file_path)
    # Load the file data using the FileLoader.
    textfile = file_loader.file_data

    # Check if the file was loaded successfully and is a Textfile object.
    if textfile && textfile.is_a?(Textfile)
      # Store the Textfile ID in Redis.
      store_textfile_id(textfile.id)
    end
  end
end
```



```

    # Log the successful file loading.
    logger.debug "Loaded file: #{file_path}"
  else
    # Log an error if the Textfile object creation failed.
    logger.error "Failed to create Textfile object for: #{file_path}"
    # Return nil to indicate failure.
    nil
  end
rescue StandardError => e
  # Log an error message if any exception occurs during file loading.
  logger.error "Error loading file #{file_path}: #{e.message}"
  # Display an error message to the user.
  UI.say(:error, "Failed to load file: #{file_path}")
  # Return nil to indicate failure.
  nil
end
end

```

42.1.2. Private Instance Methods

`retrieve_input()` [click to toggle source](#)

Retrieves the input file path from Redis.

This method retrieves the input file path from Redis using the `retrieve_file_path` method from the `InputRetrieval` module.

@return [String] The input file path.

```

# File lib/tasks/load_text_files_task.rb, line 73
def retrieve_input
  retrieve_file_path
end

```

`store_textfile_id(id)` [click to toggle source](#)

Stores the Textfile ID in Redis.

This method stores the `Textfile` ID in Redis using the `RedisKeys` class. It sets the value of the `RedisKeys::CURRENT_TEXTFILE_ID` key to the provided `id`.

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

@return [void]

```

# File lib/tasks/load_text_files_task.rb, line 85
def store_textfile_id(id)
  RedisKeys.set(RedisKeys::CURRENT_TEXTFILE_ID, id)
end

```

end

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 43. Logging

43.1. module Logging

43.1.1. 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.

43.1.2. Public Class Methods

`configure_logger_for(_classname, _methodname)` [click to toggle source](#)

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.*

```
# File lib/logging.rb, line 69
def configure_logger_for(_classname, _methodname)
  # Get the current date in YYYY-MM-DD format.
  current_date = Time.now.strftime("%Y-%m-%d")

  # Construct the log file path.
  log_file = File.join(LOG_DIR, "flowbots-#{current_date}.log")

  # Create a new logger object.
  logger = Logger.new(log_file, LOG_MAX_FILES, LOG_MAX_SIZE)

  # Set the log level.
  logger.level = log_level

  # Return the configured logger object.
  logger
end
```

`log_level()` [click to toggle source](#)

Returns the default log level.

@return [Integer] The log level.

```
# File lib/logging.rb, line 48
def log_level
  Logger::INFO
end
```

logger_for(classname, methodname) [click to toggle source](#)

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.

```
# File lib/logging.rb, line 58
def logger_for(classname, methodname)
  # Get the logger for the specified class, or create a new one if it doesn't exist.
  @loggers[classname] ||= configure_logger_for(classname, methodname)
end
```

43.1.3. Public Instance Methods

logger() [click to toggle source](#)

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

@return [Logger] The **logger** object.

```
# File lib/logging.rb, line 27
def logger
  # Get the name of the current class.
  classname = self.class.name

  # Get the name of the current method.
  methodname = caller(1..1).first[/'([^']*)'/, 1]

  # Get the logger for the current class, or create a new one if it doesn't exist.
  @logger ||= Logging.logger_for(classname, methodname)

  # Set the progame of the logger to include the class and method name.
  @logger.progame = "#{classname}##{methodname}"

  # Return the logger object.
  @logger
```

```
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 44. Markdown

44.1. module TTY::Markdown

Provides functionality for converting **Markdown** to terminal-friendly output.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 45. MarkdownYaml

45.1. module MarkdownYaml

Autogenerated from a Treetop grammar. Edits may be lost.

45.1.1. Public Instance Methods

`_nt_document()` [click to toggle source](#)

Parses the document node.

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

```
# File lib/grammars/markdown_yaml.rb, line 34
def _nt_document
  start_index = index
  if node_cache[:document].has_key?(index)
    cached = node_cache[:document][index]
    if cached
      node_cache[:document][index] = cached = SyntaxNode.new(input, index...(index +
1)) if cached == true
      @index = cached.interval.end
    end
    return cached
  end

  i0 = index
  s0 = []
  r1 = _nt_yaml_front_matter
  s0 << r1
  if r1
    r2 = _nt_markdown_content
    s0 << r2
  end
  if s0.last
    r0 = instantiate_node(SyntaxNode, input, i0...index, s0)
    r0.extend(Document0)
  else
    @index = i0
    r0 = nil
  end

  node_cache[:document][start_index] = r0

  r0
end
```

[_nt_markdown_content\(\) click to toggle source](#)

Parses the Markdown content node.

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

```
# File lib/grammars/markdown_yaml.rb, line 193
def _nt_markdown_content
  start_index = index
  if node_cache[:markdown_content].has_key?(index)
    cached = node_cache[:markdown_content][index]
    if cached
      node_cache[:markdown_content][index] = cached = SyntaxNode.new(input,
index...(index + 1)) if cached == true
      @index = cached.interval.end
    end
    return cached
  end

  s0 = []
  i0 = index
  loop do
    if index < input_length
      r1 = true
      @index += 1
    else
      terminal_parse_failure("any character")
      r1 = nil
    end
    break unless r1

    s0 << r1
  end
  r0 = instantiate_node(SyntaxNode, input, i0...index, s0)

  node_cache[:markdown_content][start_index] = r0

  r0
end
```

[_nt_newline\(\) click to toggle source](#)

Parses the newline node.

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

```
# File lib/grammars/markdown_yaml.rb, line 228
def _nt_newline
  start_index = index
  if node_cache[:newline].has_key?(index)
```



```

    cached = node_cache[:newline][index]
    if cached
      node_cache[:newline][index] = cached = SyntaxNode.new(input, index...(index +
1)) if cached == true
      @index = cached.interval.end
    end
    return cached
  end

  s0 = []
  i0 = index
  loop do
    if has_terminal?(@regexps[gr = '\A[\n\r]'] ||= Regexp.new(gr), :regexp, index)
      r1 = true
      @index += 1
    else
      terminal_parse_failure('[\n\r]')
      r1 = nil
    end
    break unless r1

    s0 << r1
  end
  if s0.empty?
    @index = i0
    r0 = nil
  else
    r0 = instantiate_node(SyntaxNode, input, i0...index, s0)
  end

  node_cache[:newline][start_index] = r0

  r0
end

```

`_nt_yaml_front_matter()` [click to toggle source](#)

Parses the YAML front matter node.

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

```

# File lib/grammars/markdown_yaml.rb, line 90
def _nt_yaml_front_matter
  start_index = index
  if node_cache[:yaml_front_matter].has_key?(index)
    cached = node_cache[:yaml_front_matter][index]
    if cached
      node_cache[:yaml_front_matter][index] = cached = SyntaxNode.new(input,
index...(index + 1)) if cached == true
      @index = cached.interval.end
    end
  end
end

```

```

    end
    return cached
end

i0 = index
s0 = []
if (match_len = has_terminal?("---", false, index))
    r1 = instantiate_node(SyntaxNode, input, index...(index + match_len))
    @index += match_len
else
    terminal_parse_failure("'---'")
    r1 = nil
end
s0 << r1
if r1
    r2 = _nt_newline
    s0 << r2
    if r2
        s3 = []
        i3 = index
        loop do
            i4 = index
            s4 = []
            i5 = index
            if (match_len = has_terminal?("---", false, index))
                r6 = instantiate_node(SyntaxNode, input, index...(index + match_len))
                @index += match_len
            else
                terminal_parse_failure("'---'")
                r6 = nil
            end
            if r6
                @index = i5
                r5 = nil
                terminal_parse_failure("'---'", true)
            else
                @terminal_failures.pop
                @index = i5
                r5 = instantiate_node(SyntaxNode, input, index...index)
            end
            s4 << r5
            if r5
                if index < input_length
                    r7 = true
                    @index += 1
                else
                    terminal_parse_failure("any character")
                    r7 = nil
                end
                s4 << r7
            end
        end
    end
end

```

```

    if s4.last
      r4 = instantiate_node(SyntaxNode, input, i4...index, s4)
      r4.extend(YamlFrontMatter0)
    else
      @index = i4
      r4 = nil
    end
    break unless r4

    s3 << r4
  end
  r3 = instantiate_node(SyntaxNode, input, i3...index, s3)
  s0 << r3
  if r3
    if (match_len = has_terminal?("---", false, index))
      r8 = instantiate_node(SyntaxNode, input, index...(index + match_len))
      @index += match_len
    else
      terminal_parse_failure("'---'")
      r8 = nil
    end
    s0 << r8
    if r8
      r9 = _nt_newline
      s0 << r9
    end
  end
end
end
if s0.last
  r0 = instantiate_node(SyntaxNode, input, i0...index, s0)
  r0.extend(YamlFrontMatter1)
else
  @index = i0
  r0 = nil
end

node_cache[:yaml_front_matter][start_index] = r0

r0
end

```

`root()` click to toggle source

The `root` node of the grammar.

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

```

# File lib/grammars/markdown_yaml.rb, line 9
def root

```

```
@root ||= :document  
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 46. Markdownyamllparser

46.1. class MarkdownYamlParser

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

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 47. Microagenttask

47.1. class MicroAgentTask

47.1.1. Public Class Methods

`new(agent_role, cartridge_file)` [click to toggle source](#)

```
# File lib/general_task_agent.rb, line 40
def initialize(agent_role, cartridge_file)
  @agent = WorkflowAgent.new(agent_role, cartridge_file)
end
```

47.1.2. Public Instance Methods

`execute()` [click to toggle source](#)

```
# File lib/general_task_agent.rb, line 44
def execute
  # Perform LLM analysis using the agent
  input = get_input_for_analysis
  result = @agent.process(input)
  store_result(result)
end
```

47.1.3. Private Instance Methods

`get_input_for_analysis()` [click to toggle source](#)

```
# File lib/general_task_agent.rb, line 53
def get_input_for_analysis
  # Get or generate input
end
```

`store_result(result)` [click to toggle source](#)

```
# File lib/general_task_agent.rb, line 57
def store_result(result)
  # Store the response
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 48. Monadiceror

48.1. class MonadError

Defines a custom error class for monadic errors.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 49. Nlpprocessor

49.1. class Flowbots::NLPProcessor

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

49.1.1. Public Class Methods

`new()` [click to toggle source](#)

Initializes a new `NLPProcessor` instance.

@return [void]

```
# File lib/processors/NLPProcessor.rb, line 10
def initialize
  # Loads the NLP model.
  load_model
end
```

49.1.2. Public Instance Methods

`process(segment, options={})` [click to toggle source](#)

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.

```
# File lib/processors/NLPProcessor.rb, line 21
def process(segment, options={})
  # Logs a message indicating the start of NLP processing for the current segment.
  logger.debug "Processing segment: #{segment.inspect}"
  logger.debug "Options: #{options.inspect}"

  # Creates a Spacy::Doc object from the segment's tokens.
  doc = create_doc(segment)

  # Initializes an empty array to store the processed tokens.
  result = []

  # Iterates through each token in the Spacy::Doc object and extracts relevant
  information.
  doc.each do |token|
    token_data = {
      word: token.text,
```

```

      pos: token.pos_,
      tag: token.tag_,
      dep: token.dep_,
      ner: token.ent_type_
    }
    result << token_data
  end

  # Logs the processed result for debugging purposes.
  logger.debug "Processed result: #{result.inspect}"

  # Returns the array of processed tokens.
  result
end

```

49.1.3. Private Instance Methods

`create_doc(segment)` [click to toggle source](#)

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

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

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

```

# File lib/processors/NLPProcessor.rb, line 84
def create_doc(segment)
  # Logs a message indicating the start of Spacy::Doc creation.
  logger.debug "Creating doc for segment: #{segment.text.length} characters"

  # Creates a Spacy::Doc object from the segment's tokens.
  doc = @nlp.read(segment.tokens.join(" "))

  # Logs a message indicating successful Spacy::Doc creation.
  logger.debug "Doc created successfully"

  # Returns the Spacy::Doc object.
  doc
end

```

`load_model()` [click to toggle source](#)

Loads the NLP model from the specified environment variable.

@return [void]

```

# File lib/processors/NLPProcessor.rb, line 56
def load_model
  # Retrieves the NLP model name from the environment variable.

```

```
nlp_model = ENV.fetch("SPACY_MODEL", nil)

# Logs a message indicating the start of NLP model loading.
logger.debug "Loading NLP model: #{nlp_model}"

# Attempts to load the NLP model using the Spacy::Language class.
begin
  @nlp = Spacy::Language.new(nlp_model)

  # Logs a message indicating successful NLP model loading.
  logger.debug "NLP model loaded successfully"
  UI.say(:ok, "NLP model loaded successfully")
rescue StandardError => e
  # Logs an error message if NLP model loading fails.
  logger.error "Error loading NLP model: #{e.message}"
  logger.error e.backtrace.join("\n")
  UI.say(:error, "Error loading NLP model: #{e.message}")
  raise
end
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 50. Nlpanalysistask

50.1. class NlpAnalysisTask

50.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the task.

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

@return [void]

```
# File lib/tasks/nlp_analysis_task.rb, line 14
def execute
  # Log the start of the task.
  logger.info "Starting NlpAnalysisTask"

  # Retrieve the FileObject from Redis.
  textfile = retrieve_input
  # Get an instance of the NLPPProcessor.
  nlp_processor = Flowbots::NLPPProcessor.instance

  # Initialize a hash to store lemma counts.
  lemma_counts = Hash.new(0)

  # Iterate through each segment of the FileObject.
  textfile.retrieve_segments.each do |segment|
    # Process the segment using the NLPPProcessor.
    processed_tokens = nlp_processor.process(segment, pos: true, dep: true, ner: true,
tag: true, lemma: true)
    # Update the segment with the processed NLP data.
    update_segment_with_nlp_data(segment, processed_tokens, lemma_counts)
  end

  # Add the accumulated lemmas to the FileObject.
  add_lemmas_to_textfile(textfile, lemma_counts)

  # Log the completion of the task.
  logger.info "NlpAnalysisTask completed"
end
```

50.1.2. Private Instance Methods

`add_lemmas_to_textfile(textfile, lemma_counts)` [click to toggle source](#)

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`]

```
# File lib/tasks/nlp_analysis_task.rb, line 108
def add_lemmas_to_textfile(textfile, lemma_counts)
  # Map the lemma counts hash to an array of lemma data.
  lemmas_data = lemma_counts.map do |(lemma, pos), count|
    { lemma:, pos:, count: }
  end
  # Add the lemmas to the FileObject.
  textfile.add_lemmas(lemmas_data)
end
```

`add_words_to_segment(segment, processed_tokens)` [click to toggle source](#)

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 `NLPPProcessor`.

@return [`void`]

```
# File lib/tasks/nlp_analysis_task.rb, line 91
def add_words_to_segment(segment, processed_tokens)
  # Map the processed tokens to an array of word data.
  words = processed_tokens.map do |token|
    { word: token[:word], pos: token[:pos], tag: token[:tag], dep: token[:dep], ner:
token[:ner] }
  end
  # Add the words to the segment.
  segment.add_words(words)
end
```

`retrieve_input()` [click to toggle source](#)

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

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

```
# File lib/tasks/nlp_analysis_task.rb, line 46
def retrieve_input
```

```
retrieve_file_object
end
```

`update_segment_with_nlp_data(segment, processed_tokens, lemma_counts)` [click to toggle source](#)

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]

```
# File lib/tasks/nlp_analysis_task.rb, line 60
def update_segment_with_nlp_data(segment, processed_tokens, lemma_counts)
  # Initialize a hash to store tagged data.
  tagged = { pos: {}, dep: {}, ner: {}, tag: {} }
  # Iterate through the processed tokens.
  processed_tokens.each do |token|
    # Extract relevant information from the token.
    word = token[:word]
    tagged[:pos][word] = token[:pos]
    tagged[:dep][word] = token[:dep]
    tagged[:ner][word] = token[:ner]
    tagged[:tag][word] = token[:tag]

    # Increment the lemma count for the current lemma and POS tag.
    lemma_key = [token[:lemma], token[:pos]]
    lemma_counts[lemma_key] += 1
  end

  # Update the segment with the tagged data.
  segment.update(tagged:)
  # Add the processed words to the segment.
  add_words_to_segment(segment, processed_tokens)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 51. Object

51.1. class Object

51.1.1. 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.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 52. Preprocessfileobjecttask

52.1. class PreprocessFileObjectTask

Task to preprocess a `FileObject`.

52.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

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]

```
# File lib/tasks/preprocess_file_object_task.rb, line 16
def execute
  # Log the start of the task.
  logger.info "Starting PreprocessFileObjectTask"

  # Retrieve the FileObject from Redis.
  @file_object = retrieve_file_object

  # Check if the FileObject was retrieved successfully.
  if @file_object.nil?
    # Log an error message if the FileObject retrieval failed.
    error_message = "Failed to retrieve FileObject"
    logger.error error_message
    # Raise an error to stop the task execution.
    raise(error_message)
    return
  end

  # Begin error handling block.
  begin
    # Preprocess the file and retrieve the preprocessed content and metadata.
    preprocessed_content, metadata = preprocess_file(@file_object)
    # Store the preprocessed content and metadata in the FileObject.
    store_preprocessed_data(preprocessed_content, metadata)
    # Log a success message with the file name.
    logger.info "Successfully preprocessed file: #{@file_object.name}"
    # Mark the task as complete with a success message.
    complete("Successfully preprocessed file: #{@file_object.name}")
  rescue StandardError => e
    # Log an error message if any exception occurs during preprocessing.
    logger.error "Error in preprocessing file: #{e.message}"
  end
end
```



```

    # Log the backtrace of the exception.
    logger.error e.backtrace.join("\n")
    # Display an exception message to the user.
    Flowbots::UI.exception("#{e.message}")
    # Raise the exception to stop the task execution.
    raise(e.message)
  end

  # Log the completion of the task.
  logger.info "PreprocessFileObjectTask completed"
end

```

52.1.2. Private Instance Methods

`determine_preprocessing_method(file_object)` [click to toggle source](#)

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.

```

# File lib/tasks/preprocess_file_object_task.rb, line 88
def determine_preprocessing_method(file_object)
  # Get the file extension in lowercase.
  case File.extname(file_object.path).downcase
  when ".md", ".markdown"
    # Markdown files with YAML front matter.
    :markdown_yaml
  when ".pdf"
    # PDF files.
    :pdf
  when ".json"
    # JSON files.
    :json
  else
    # Plain text files.
    :plain_text
  end
end

```

`extract_metadata(yaml_front_matter)` [click to toggle source](#)

Extracts metadata from YAML front matter.

@param yaml_front_matter [String] The YAML front matter string.

@return [Hash] The extracted metadata.

```
# File lib/tasks/preprocess_file_object_task.rb, line 179
def extract_metadata(yaml_front_matter)
  # Return an empty hash if the YAML front matter is empty.
  return {} if yaml_front_matter.to_s.empty?

  # Parse the YAML front matter.
  YAML.safe_load(yaml_front_matter)
rescue StandardError => e
  # Log an error message if parsing fails.
  logger.error "Error parsing YAML front matter: #{e.message}"
  # Return an empty hash in case of an error.
  {}
end
```

`extract_pdf_metadata(pdf_path)` [click to toggle source](#)

Extracts metadata from a PDF file.

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

@return [Hash] The extracted metadata.

```
# File lib/tasks/preprocess_file_object_task.rb, line 211
def extract_pdf_metadata(pdf_path)
  # Require the 'pdf-reader' gem.
  require "pdf-reader"
  # Create a new PDF::Reader instance.
  reader = PDF::Reader.new(pdf_path)
  # Return the PDF info hash.
  reader.info
end
```

`extract_text_from_pdf(pdf_path)` [click to toggle source](#)

Extracts text content from a PDF file.

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

@return [String] The extracted text content.

```
# File lib/tasks/preprocess_file_object_task.rb, line 197
def extract_text_from_pdf(pdf_path)
  # Require the 'pdf-reader' gem.
  require "pdf-reader"
  # Create a new PDF::Reader instance.
  reader = PDF::Reader.new(pdf_path)
  # Extract text from each page and join them with newlines.
  reader.pages.map(&:text).join("\n")
end
```

```
end
```

`preprocess_file(file_object)` [click to toggle source](#)

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.

```
# File lib/tasks/preprocess_file_object_task.rb, line 65
def preprocess_file(file_object)
  # Determine the preprocessing method based on the file extension.
  case determine_preprocessing_method(file_object)
  when :markdown_yaml
    # Preprocess Markdown files with YAML front matter.
    preprocess_markdown_yaml(file_object)
  when :pdf
    # Preprocess PDF files.
    preprocess_pdf(file_object)
  when :json
    # Preprocess JSON files.
    preprocess_json(file_object)
  else
    # Preprocess plain text files.
    preprocess_plain_text(file_object)
  end
end
```

`preprocess_json(file_object)` [click to toggle source](#)

Preprocesses JSON files.

@param file_object [FileObject] The **FileObject** to preprocess.

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

```
# File lib/tasks/preprocess_file_object_task.rb, line 153
def preprocess_json(file_object)
  # Parse the JSON content.
  json_data = JSON.parse(file_object.content)
  # Extract the text content or use the entire JSON data as a string.
  content = json_data["text"] || json_data.to_s
  # Extract metadata by removing the "text" key from the JSON data.
  metadata = json_data.except("text")
  # Return the extracted content and metadata.
  [content, metadata]
end
```

`preprocess_markdown_yaml(file_object)` [click to toggle source](#)

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.

```
# File lib/tasks/preprocess_file_object_task.rb, line 111
def preprocess_markdown_yaml(file_object)
  # Create a new GrammarProcessor instance for Markdown with YAML front matter.
  grammar_processor = Flowbots::GrammarProcessor.new("markdown_yaml")
  # Parse the file content using the GrammarProcessor.
  parse_result = grammar_processor.parse(file_object.content)

  # Check if the parsing was successful.
  if parse_result
    # Extract the Markdown content and metadata from the parse result.
    content = parse_result[:markdown_content]
    metadata = extract_metadata(parse_result[:yaml_front_matter])
  else
    # Log a warning message if parsing failed.
    logger.warn "Failed to parse Markdown with YAML front matter"
    # Use the original content and an empty metadata hash.
    content = file_object.content
    metadata = {}
  end

  # Return the preprocessed content and metadata.
  [content, metadata]
end
```

`preprocess_pdf(file_object)` [click to toggle source](#)

Preprocesses PDF files.

@param file_object [FileObject] The **FileObject** to preprocess.

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

```
# File lib/tasks/preprocess_file_object_task.rb, line 139
def preprocess_pdf(file_object)
  # Extract text content from the PDF file.
  content = extract_text_from_pdf(file_object.path)
  # Extract metadata from the PDF file.
  metadata = extract_pdf_metadata(file_object.path)
  # Return the extracted content and metadata.
  [content, metadata]
end
```

`preprocess_plain_text(file_object)` [click to toggle source](#)

Preprocesses plain text files.

@param file_object [FileObject] The **FileObject** to preprocess.

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

```
# File lib/tasks/preprocess_file_object_task.rb, line 169
def preprocess_plain_text(file_object)
  # Return the original content and an empty metadata hash.
  [file_object.content, {}]
end
```

`store_preprocessed_data(content, metadata)` [click to toggle source](#)

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

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

@return [void]

```
# File lib/tasks/preprocess_file_object_task.rb, line 226
def store_preprocessed_data(content, metadata)
  # Update the FileObject with the preprocessed content and metadata.
  @file_object.update(preprocessed_content: content, metadata:)
  # Log the stored preprocessed content (first 100 characters).
  logger.debug "Stored preprocessed content (first 100 chars): #{content[0..100]}"
  # Log the stored metadata.
  logger.debug "Stored metadata: #{metadata.inspect}"
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 53. Preprocesstextfiletask

53.1. class PreprocessTextFileTask

This task preprocesses a text file, extracting metadata and cleaning the content.

53.1.1. Public Instance Methods

execute() click to toggle source

Executes the text file preprocessing task.

This method retrieves the text file object, parses it using a custom grammar, extracts metadata, updates the text file with preprocessed content and metadata, and logs relevant information.

The preprocessing process involves:

1. **Retrieving the text file object:** The `retrieve_input` method is called to retrieve the text file object from the appropriate source (e.g., Redis).
2. **Parsing the text file:** The `Flowbots::GrammarProcessor` is used to parse the text file content using a custom grammar (e.g., "markdown_yaml"). The parsing result is stored in the `parse_result` variable.
3. **Extracting metadata:** If the parsing is successful, the `extract_metadata` method is called to extract metadata from the YAML front matter of the parsed text file. The extracted metadata is stored in the `metadata` variable.
4. **Updating the text file:** The `update` method is called on the text file object to store the preprocessed content (extracted from the parsing result) and the extracted metadata.
5. **Logging information:** The `logger` object is used to log relevant information about the preprocessing process, including success messages, error messages, and debug information.

@return [String] A success message if the preprocessing is successful, otherwise an error message.

```
# File lib/tasks/preprocess_text_file_task.rb, line 32
def execute
  logger.info "Starting PreprocessTextFileTask"

  @textfile = retrieve_input

  begin
    grammar_processor = Flowbots::GrammarProcessor.new("markdown_yaml")
    parse_result = grammar_processor.parse(@textfile.content)
    logger.debug "Parse result: #{parse_result.inspect}"

    if parse_result
      sourcefile.update(
        preprocessed_content: parse_result[:markdown_content],
        metadata: extract_metadata(parse_result[:yaml_front_matter])
      )
    end
  end
end
```

```

    )
    "Successfully preprocessed file: #{sourcefile.path}"
  else
    logger.error "Failed to parse the document with custom grammar"
    @textfile.update(preprocessed_content: "", metadata: {})
  end
rescue StandardError => e
  logger.error "Error in grammar processing: #{e.message}"
  logger.error e.backtrace.join("\n")
  UI.exception("#{e.message}")
  @textfile.update(preprocessed_content: "", metadata: {})
end
end
end

```

53.1.2. Private Instance Methods

`extract_metadata(yaml_front_matter)` [click to toggle source](#)

Extracts metadata from the YAML front matter of the parsed text file.

This method extracts metadata from the YAML front matter of the parsed text file. It uses the `YAML.safe_load` method to parse the YAML front matter and returns a hash containing the extracted metadata. If the YAML front matter is empty or an error occurs during parsing, an empty hash is returned.

@param `yaml_front_matter` [String] The YAML front matter extracted from the text file.

@return [Hash] A hash containing the extracted metadata.

```

# File lib/tasks/preprocess_text_file_task.rb, line 81
def extract_metadata(yaml_front_matter)
  return {} if yaml_front_matter.empty?

  YAML.safe_load(yaml_front_matter)
rescue StandardError => e
  logger.error "Error parsing YAML front matter: #{e.message}"
  {}
end

```

`retrieve_input()` [click to toggle source](#)

Retrieves the input text file object.

This method retrieves the text file object from the appropriate source (e.g., Redis).

@return [Textfile] The retrieved text file object.

```

# File lib/tasks/preprocess_text_file_task.rb, line 67
def retrieve_input

```

```
retrieve_textfile
end
```

`store_preprocessed_data(content, metadata)` [click to toggle source](#)

Stores the preprocessed content and metadata in the text file object.

This method updates the text file object with the preprocessed content and extracted metadata. It uses the `update` method on the text file object to store the preprocessed content and metadata. It also logs debug information about the stored content and metadata.

@param content [String] The preprocessed content of the text file. @param metadata [Hash] The extracted metadata.

@return [void]

```
# File lib/tasks/preprocess_text_file_task.rb, line 101
def store_preprocessed_data(content, metadata)
  @textfile.update(preprocessed_content: content, metadata:)
  logger.debug "Stored preprocessed content (first 100 chars): #{content[0..100]}"
  logger.debug "Stored metadata: #{metadata.inspect}"
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 54. Promptx

54.1. class TTY::PromptX

A custom prompt class with enhanced features.

54.1.1. Attributes

prefix[R]

The **prefix** used for the prompt.

@return [String] The prompt **prefix**.

54.1.2. Public Class Methods

new(active_color:, prefix:, history: true) [click to toggle source](#)

Initializes a new **PromptX** instance.

@param active_color [Symbol] The color for the active prompt. @param **prefix** [String] The **prefix** for the prompt. @param history [Boolean] Whether to enable history for the prompt.

Calls superclass method

```
# File lib/helper.rb, line 47
def initialize(active_color:, prefix:, history: true)
  # Define an interrupt handler to handle Ctrl+C.
  @interrupt = lambda do
    print TTY::Cursor.clear_screen_down
    print "\e[2J\e[f"
    res = TTY::Prompt.new.yes?("Quit the app?")
    exit if res
  end

  # Initialize the superclass with the provided options.
  super(active_color: active_color, prefix: prefix, interrupt: @interrupt)
  @history = history
  @prefix = prefix
end
```

54.1.3. Public Instance Methods

readline(text = "") [click to toggle source](#)

Reads a line of input from the user.

@param text [String] The text to display before the input field.

@return [String] The input string entered by the user.

```
# File lib/helper.rb, line 67
def readline(text = "")
  puts @prefix
  begin
    Readline.readline(text, @history)
  rescue Interrupt
    @interrupt.call
  end
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 55. Readme md

55.1. Flowbots ¶

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

55.1.1. Features ¶

- Text processing workflows for individual files and batch processing
- Advanced NLP methods 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

55.1.2. System Architecture ¶

55.1.3. 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

55.1.4. 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.

55.2. Detailed Operation ¶ □

[[1-workflow-initialization-]] ===== 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-]] ===== 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-]] ===== 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-]] ===== 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-]] ===== 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-]] ===== 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-]] ===== 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-]] ==== 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.

55.2.1. Key Interactions ¶

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

55.3. Ruby Gems Used in `Flowbots` ¶

55.3.1. Workflow and Task Management ¶

- `jongleur`: Core component for defining and executing task workflows, providing workflow orchestration and task management capabilities.

55.3.2. Data Persistence ¶

- `ohm`: `Object`-hash mapping for Redis, used as the data persistence layer for storing document information and workflow states.

55.3.3. Parallel Processing ¶

- `parallel`: Enables parallel processing, with potential use for parallel execution of tasks (not prominently used in the current implementation).

55.3.4. Development and Debugging ¶

- `pry` and `pry-stack_explorer`: Enhanced REPL and debugging tools for development and debugging purposes.

55.3.5. Natural Language Processing ¶

- [ruby-spacy](#): Ruby bindings for the Spacy NLP library, used for Natural Language Processing tasks.
- [lingua](#): Provides additional natural language detection and processing capabilities.
- [pragmatic_segmenter](#): Used for text segmentation, splitting text into meaningful segments.
- [pragmatic_tokenizer](#): Handles text tokenization, breaking text into individual tokens.

55.3.6. Command-Line Interface ¶

- [thor](#): Used for building command-line interfaces, specifically for creating the CLI for [Flowbots](#).

55.3.7. Parsing and Data Handling ¶

- [treetop](#): A parsing expression grammar (PEG) parser generator, used for custom grammar parsing, particularly for Markdown with YAML front matter.
- [yaml](#): Handles YAML parsing and generation, particularly for configuration files and document front matter.

55.3.8. Terminal Output Formatting ¶

- [tty-box](#), [tty-cursor](#), [tty-prompt](#), [tty-screen](#), [tty-spinner](#), [tty-table](#): Various terminal output formatting and interaction tools for creating rich command-line interfaces and displaying formatted output.

55.3.9. Topic Modeling ¶

- [tomoto](#): Used for implementing topic modeling algorithms.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 56. Redisconnection

56.1. class RedisConnection

Class to manage Redis connection.

56.1.1. Constants

REDIS_CONFIG

Redis configuration.

56.1.2. Attributes

redis[R]

Returns the Redis connection.

@return [Redis] The Redis connection.

56.1.3. Public Class Methods

new() [click to toggle source](#)

Initializes a **new RedisConnection** instance.

@return [void]

```
# File lib/flowbots.rb, line 72
def initialize
  @redis = Redis.new(REDIS_CONFIG)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 57. Rediskeys

57.1. module RedisKeys

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

57.1.1. 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.

57.1.2. Public Class Methods

get(key) click to toggle source

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.

```
# File lib/components/RedisKeys.rb, line 21
def self.get(key)
  Jongleur::WorkerTask.class_variable_get(:@@redis).get(key)
end
```

set(key, value) click to toggle source

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**.

```
# File lib/components/RedisKeys.rb, line 30
def self.set(key, value)
  Jongleur::WorkerTask.class_variable_get(:@@redis).set(key, value)
```



```
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 58. Runrubyteststask

58.1. class RunRubyTestsTask

This task runs Ruby tests from a file.

58.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the task.

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

```
# File lib/example.rb, line 173
def execute
  # Retrieve the test code from Redis.
  test_code = JSON.parse(@@redis.get("testdesign"))

  # Save the test code to a temporary file.
  File.write("temp_test.rb", test_code)

  # Run the Ruby tests using the `ruby` command.
  result = `ruby temp_test.rb`

  # Store the test results in Redis.
  @@redis.set("test_results", result)

  # Clean up the temporary file.
  File.delete("temp_test.rb")
rescue StandardError => e
  # Log an error message if an exception occurs.
  logger.error "Error in RunRubyTestsTask: #{e.message}"
  # Re-raise the exception.
  raise
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 59. Runttestcommandaction

59.1. class Sublayer::Actions::RunTestCommandAction

59.1.1. Public Class Methods

new(test_command:) click to toggle source

```
# File lib/utils/command.rb, line 4
def initialize(test_command:)
  @test_command = test_command
end
```

59.1.2. Public Instance Methods

call() click to toggle source

```
# File lib/utils/command.rb, line 8
def call
  stdout, stderr, status = Open3.capture3(@test_command)
  [stdout, stderr, status]
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 60. Scrollablebox

60.1. module UI::ScrollableBox

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

60.1.1. Private Class Methods

`create_scrollable_box(text, width, height, title)` [click to toggle source](#)

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.

```
# File lib/ui/scrollable_box.rb, line 46
def self.create_scrollable_box(text, width, height, title)
  # Split the text into lines.
  lines = text.split("\n")
  # Calculate the total number of pages based on the text length and box height.
  total_pages = (lines.length.to_f / (height - 2)).ceil
  # Return a hash containing the box data.
  {
    title:,
    lines:,
    width:,
    height:,
    total_pages:,
    current_page: 1
  }
end
```

`display_boxes(box1, box2, box_height)` [click to toggle source](#)

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]

```
# File lib/ui/scrollable_box.rb, line 69
def self.display_boxes(box1, box2, box_height)
  # Loop until the user quits.
  loop do
    # Clear the screen.
```

```

system("clear") || system("cls")
# Print the boxes and navigation info.
print_boxes(box1, box2, box_height)
print_navigation_info(box1, box2)

# Get user input.
input = STDIN.getch
# Handle user input for navigation and quitting.
case input.downcase
when "q"
  break
when "a"
  box1[:current_page] = [1, box1[:current_page] - 1].max
when "d"
  box1[:current_page] = [box1[:total_pages], box1[:current_page] + 1].min
when "j"
  box2[:current_page] = [1, box2[:current_page] - 1].max
when "l"
  box2[:current_page] = [box2[:total_pages], box2[:current_page] + 1].min
end
end
end

```

`print_boxes(box1, box2, box_height)` [click to toggle source](#)

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]

```

# File lib/ui/scrollable_box.rb, line 103
def self.print_boxes(box1, box2, box_height)
  # Calculate the starting line for each box based on the current page.
  start_line1 = (box1[:current_page] - 1) * (box_height - 2)
  start_line2 = (box2[:current_page] - 1) * (box_height - 2)

  # Extract the content for each box based on the starting line and box height.
  box1_content = box1[:lines][start_line1, box_height - 2].join("\n")
  box2_content = box2[:lines][start_line2, box_height - 2].join("\n")

  # Create framed boxes for each box with the extracted content.
  box1_frame = TTY::Box.frame(width: box1[:width], height: box_height, title: {
    top_left: box1[:title] }) do
    box1_content
  end
  box2_frame = TTY::Box.frame(width: box2[:width], height: box_height, title: {
    top_left: box2[:title] }) do
    box2_content
  end
end

```

```

end

# Print the boxes side-by-side.
puts box1_frame.split("\n").zip(box2_frame.split("\n")).map { |a, b| "#{a}  #{b}"
}.join("\n")
end

```

`print_navigation_info(box1, box2)` [click to toggle source](#)

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]

```

# File lib/ui/scrollable_box.rb, line 130
def self.print_navigation_info(box1, box2)
  # Print the current page and total pages for each box, along with navigation
  instructions.
  puts "Box 1: Page #{box1[:current_page]}/#{box1[:total_pages]} (A/D to navigate)"
  puts "Box 2: Page #{box2[:current_page]}/#{box2[:total_pages]} (J/L to navigate)"
  puts "Press Q to exit"
end

```

60.1.2. Public Instance Methods

`side_by_side_boxes(text1, text2, title1: "Box 1", title2: "Box 2")` [click to toggle source](#)

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]

```

# File lib/ui/scrollable_box.rb, line 21
def side_by_side_boxes(text1, text2, title1: "Box 1", title2: "Box 2")
  # Calculate screen width, screen height, box width, and box height.
  screen_width = TTY::Screen.width / 1.25
  screen_height = TTY::Screen.height
  box_width = (screen_width / 2.5) - 2
  box_height = screen_height - 4 # Leave some space for prompts

  # Create scrollable boxes for the given texts.
  box1 = create_scrollable_box(text1, box_width, box_height, title1)
  box2 = create_scrollable_box(text2, box_width, box_height, title2)

  # Display the boxes and handle user navigation.

```

```
display_boxes(box1, box2, box_height)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 61. Segment

61.1. class Segment

Defines the `Segment` model.

61.1.1. Public Instance Methods

`add_word(word_data)` [click to toggle source](#)

Adds a new word to the `Segment`.

@param word_data [Hash] A hash containing word data.

@return [Word] The created word.

```
# File lib/components/OhmModels.rb, line 438
def add_word(word_data)
  word = Word.create(word_data.merge(segment: self))
  words.push(word)
  save
  word
end
```

`add_words(new_words)` [click to toggle source](#)

Adds multiple words to the `Segment`.

@param new_words [Array<Hash>] An array of word data hashes.

@return [void]

```
# File lib/components/OhmModels.rb, line 450
def add_words(new_words)
  new_words.each do |word_data|
    word = Word.create(word_data.merge(segment: self))
    words.push(word)
  end
  save
end
```

`retrieve_word_texts()` [click to toggle source](#)

Retrieves the text of all words associated with the `Segment`.

@return [Array<String>] An array of word texts.


```
# File lib/components/OhmModels.rb, line 468
def retrieve_word_texts
  retrieve_words.map(&:word)
end
```

retrieve_words() click to toggle source

Retrieves all words associated with the [Segment](#).

@return [Array<Word>] An array of words.

```
# File lib/components/OhmModels.rb, line 461
def retrieve_words
  words.to_a
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 62. Speecho totextaction

62.1. class Sublayer::Actions::SpeechToTextAction

62.1.1. Public Class Methods

new(audio_data) click to toggle source

```
# File lib/utils/transcribe.rb, line 4
def initialize(audio_data)
  @audio_data = audio_data
end
```

62.1.2. Public Instance Methods

call() click to toggle source

```
# File lib/utils/transcribe.rb, line 8
def call
  tempfile = Tempfile.new(["audio", ".webm"], encoding: "ascii-8bit")
  tempfile.write(@audio_data.read)
  tempfile.rewind

  text = HTTParty.post(
    "https://api.openai.com/v1/audio/transcriptions",
    headers: {
      "Authorization" => "Bearer #{ENV.fetch('OPENAI_API_KEY', nil)}",
      "Content-Type" => "multipart/form-data"
    },
    body: {
      file: tempfile,
      model: "whisper-1"
    }
  )

  tempfile.close
  tempfile.unlink

  text["text"]
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 63. Sublayer

63.1. module Sublayer

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

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 64. Tty

64.1. module TTY

Extends the TTY::Prompt class with custom functionality.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 65. Task

65.1. class Task

Defines the **Task** model.

65.1.1. Public Class Methods

`completed()` [click to toggle source](#)

Finds all **completed** tasks.

@return [Array<Task>] An array of **completed** tasks.

```
# File lib/components/OhmModels.rb, line 158
def self.completed
  find(status: "completed")
end
```

`create_with_timestamp(attributes = {})` [click to toggle source](#)

Creates a new **Task** instance with a timestamp.

@param attributes [Hash] A hash of attributes for the new **Task**.

@return [Task] The created **Task** instance.

```
# File lib/components/OhmModels.rb, line 95
def self.create_with_timestamp(attributes = {})
  new_task = create(attributes.merge(start_time: Time.now.to_s))
  new_task.save
  new_task
end
```

`failed()` [click to toggle source](#)

Finds all **failed** tasks.

@return [Array<Task>] An array of **failed** tasks.

```
# File lib/components/OhmModels.rb, line 165
def self.failed
  find(status: "failed")
end
```

`in_progress()` [click to toggle source](#)

Finds all tasks in progress.

@return [Array<Task>] An array of tasks in progress.

```
# File lib/components/OhmModels.rb, line 172
def self.in_progress
  find(status: "in_progress")
end
```

[pending\(\)](#) click to toggle source

Finds all **pending** tasks.

@return [Array<Task>] An array of **pending** tasks.

```
# File lib/components/OhmModels.rb, line 151
def self.pending
  find(status: "pending")
end
```

65.1.2. Public Instance Methods

[complete\(result = nil\)](#) click to toggle source

Completes the task and updates its status and result.

@param result [Object] The result of the task execution.

@return [void]

```
# File lib/components/OhmModels.rb, line 106
def complete(result = nil)
  update(status: "completed", result:, end_time: Time.now.to_s)
end
```

[duration\(\)](#) click to toggle source

Returns the **duration** of the task in seconds.

@return [Integer] The **duration** of the task in seconds, or nil if start_time or end_time is nil.

```
# File lib/components/OhmModels.rb, line 122
def duration
  return nil if start_time.nil? || end_time.nil?

  start = Time.parse(start_time)
  finish = Time.parse(end_time)
```

```
(finish - start).to_i  
end
```

[execute\(\)](#) click to toggle source

Executes the task.

This method must be implemented in subclasses to define the specific actions performed by the task.

@return [void]

```
# File lib/components/OhmModels.rb, line 136  
def execute  
  raise NotImplementedError, "#{self.class.name}#execute must be implemented in  
  subclass"  
end
```

[fail\(error_message\)](#) click to toggle source

Fails the task and updates its status and result with an error message.

@param error_message [String] The error message.

@return [void]

```
# File lib/components/OhmModels.rb, line 115  
def fail(error_message)  
  update(status: "failed", result: error_message, end_time: Time.now.to_s)  
end
```

[retrieve_input\(\)](#) click to toggle source

Retrieves the input file object from Redis.

@return [FileObject] The retrieved file object.

```
# File lib/components/OhmModels.rb, line 143  
def retrieve_input  
  file_object_id = RedisKeys.get(RedisKeys::CURRENT_FILE_OBJECT_ID)  
  FileObject[file_object_id]  
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 66. Tasknotfounderror

66.1. class Flowbots::TaskNotFoundError

Custom error class for task not found.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 67. Textprocessingworkflow

67.1. 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.

67.1.1. Attributes

pipeline[R]

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

67.1.2. Public Class Methods

new(input_file_path=nil, batch_mode=false) [click to toggle source](#)

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]

```
# File lib/workflows/text_processing_workflow.rb, line 18
def initialize(input_file_path=nil, batch_mode=false)
  @input_file_path = input_file_path || prompt_for_file
  @batch_mode = batch_mode
  @pipeline = UnifiedFileProcessingPipeline.new(
    @input_file_path,
    batch_size: batch_mode ? nil : 1,
    file_types: %w[txt md markdown]
  )
end
```

67.1.3. Public Instance Methods

run() [click to toggle source](#)

Runs the text processing workflow.

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

@return [void]

```
# File lib/workflows/text_processing_workflow.rb, line 33
def run
```

```

UI.say(:ok, "Setting Up Text Processing Workflow")
logger.info "Setting Up Text Processing Workflow"

begin
  @pipeline.process
  if @batch_mode
    process_batch
  else
    process_single_file
  end
  UI.say(:ok, "Text Processing Workflow completed")
  logger.info "Text Processing Workflow completed"
rescue StandardError => e
  UI.say(:error, "Error in Text Processing Workflow: #{e.message}")
  logger.error "Error in Text Processing Workflow: #{e.message}"
  logger.error e.backtrace.join("\n")
end
end

```

67.1.4. Private Instance Methods

`create_or_fetch_file_object(file_path)` [click to toggle source](#)

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**.

```

# File lib/workflows/text_processing_workflow.rb, line 108
def create_or_fetch_file_object(file_path)
  FileObject.find_or_create_by_path(file_path)
rescue ArgumentError => e
  logger.error "Invalid file path: #{e.message}"
  raise FlowbotError.new("Invalid file path", "INVALID_FILE_PATH", details: e.message)
rescue StandardError => e
  logger.error "Error creating or fetching FileObject: #{e.message}"
  raise FlowbotError.new("Error processing file", "FILE_PROCESSING_ERROR", details:
e.message)
end

```

`fetch_unprocessed_file_ids()` [click to toggle source](#)

Fetches the IDs of unprocessed files.

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

```

# File lib/workflows/text_processing_workflow.rb, line 121
def fetch_unprocessed_file_ids

```

```
# Assuming we're using ActiveRecord and there's a FileObject model
# This query fetches all file IDs where llm_analysis is nil or an empty string
ids = []
FileObject.all.each { |f| ids << f.id.to_i if f.llm_analysis.nil? }
ids
end
```

`perform_additional_tasks(_file_id)` [click to toggle source](#)

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 (unused).

@return [void]

```
# File lib/workflows/text_processing_workflow.rb, line 136
def perform_additional_tasks(_file_id)
  additional_tasks = {
    TextTaggerTask: [:TopicModelingTask],
    TopicModelingTask: [:LlmAnalysisTask],
    LlmAnalysisTask: []
  }

  @pipeline.orchestrator.define_workflow(additional_tasks)
  @pipeline.orchestrator.run_workflow
end
```

`process_batch()` [click to toggle source](#)

Processes files in batch mode.

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

@return [void]

```
# File lib/workflows/text_processing_workflow.rb, line 73
def process_batch
  file_ids = fetch_unprocessed_file_ids
  file_ids.each do |file_id|
    perform_additional_tasks(file_id)
  end
end
```

`process_single_file()` [click to toggle source](#)

Processes a single file.

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

@return [void]

```
# File lib/workflows/text_processing_workflow.rb, line 85
def process_single_file
  file_object = create_or_fetch_file_object(@input_file_path)
  perform_additional_tasks(file_object.id)
rescue ArgumentError => e
  logger.error "Invalid input for file object: #{e.message}"
  UI.say(:error, "Invalid input for file object: #{e.message}")
rescue FileNotFoundError => e
  logger.error e.message
  UI.say(:error, e.message)
rescue FileObjectError => e
  logger.error "Error processing file: #{e.message}"
  UI.say(:error, "Error processing file: #{e.message}")
rescue StandardError => e
  logger.error "Unexpected error in text processing: #{e.message}"
  logger.error e.backtrace.join("\n")
  UI.say(:error, "Unexpected error in text processing. Check logs for details.")
end
```

`prompt_for_file()` click to toggle source

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.

```
# File lib/workflows/text_processing_workflow.rb, line 59
def prompt_for_file
  get_file_path = `gum file`.chomp.strip
  file_path = File.join(get_file_path)

  raise FlowbotError.new("File not found", "FILENOTFOUND") unless
File.exist?(file_path)

  file_path
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 68. Textprocessor

68.1. class Flowbots::TextProcessor

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

68.1.1. Public Class Methods

`new()` [click to toggle source](#)

Initializes a new **TextProcessor** instance.

@return [void]

```
# File lib/processors/TextProcessor.rb, line 13
def initialize
  # Logs a message indicating the start of initialization for the current class.
  logger.info "Initializing #{self.class.name}"

  # Displays a success message to the user indicating the initialization of the
  current class.
  UI.say(:ok, "Initializing #{self.class.name}")

  # Logs a message indicating the completion of initialization for the current class.
  logger.info "#{self.class.name} initialization completed"

  # Displays a success message to the user indicating the completion of initialization
  for the current class.
  UI.say(:ok, "#{self.class.name} initialization completed")
end
```

68.1.2. Public Instance Methods

`process(text)` [click to toggle source](#)

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.

```
# File lib/processors/TextProcessor.rb, line 35
def process(text)
  raise NotImplementedError, "#{self.class.name}#process must be implemented in
  subclass"
```

end

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 69. Textsegmentprocessor

69.1. class Flowbots::TextSegmentProcessor

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

69.1.1. Constants

DEFAULT_OPTIONS

Default `options` for the segmenter.

69.1.2. Attributes

options[RW]

The `options` for the segmenter.

text[RW]

The `text` to be segmented.

69.1.3. Public Class Methods

new() click to toggle source

Initializes a new `TextSegmentProcessor` instance.

@return [void]

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

```
# File lib/processors/TextSegmentProcessor.rb, line 22
def initialize
  super
end
```

69.1.4. Public Instance Methods

process(text, opts={}) click to toggle source

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.

```
# File lib/processors/TextSegmentProcessor.rb, line 32
```

```

def process(text, opts={})
  @text = text
  @options = DEFAULT_OPTIONS.merge(opts)
  logger.debug "TextSegmenter initialized with options: #{@options}"

  logger.info "Starting text segmentation"
  logger.debug "Input text type: #{@text.class}"

  if @text.instance_of?(Array)
    segment_array
  else
    segment_string(@text)
  end
end
end

```

69.1.5. Private Instance Methods

`segment_array()` [click to toggle source](#)

Segments an array of **text**.

@return [Array] An array of segments.

```

# File lib/processors/TextSegmentProcessor.rb, line 52
def segment_array
  logger.debug "Segmenting array of strings"
  @text.flat_map do |txt|
    segment_string(txt)
  end
end
end

```

`segment_string(txt)` [click to toggle source](#)

Segments a single string.

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

@return [Array] An array of segments.

```

# File lib/processors/TextSegmentProcessor.rb, line 64
def segment_string(txt)
  logger.debug "Segmenting text of length: #{txt.length}"
  ps = PragmaticSegmenter::Segmenter.new(text: txt, **@options)
  segments = ps.segment
  logger.debug "Segmentation result: #{segments.length} segments"
  segments
end
end

```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 70. Textsegmenttask

70.1. class TextSegmentTask

This task segments the text content of a Textfile into smaller units.

70.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

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]

```
# File lib/tasks/text_segment_task.rb, line 16
def execute
  logger.info "Starting TextSegmentTask"

  textfile = retrieve_input
  preprocessed_content = textfile.preprocessed_content

  text_segmenter = Flowbots::TextSegmentProcessor.instance
  segments = text_segmenter.process(preprocessed_content, { clean: true })

  store_segments(textfile, segments)

  logger.info "TextSegmentTask completed"
end
```

70.1.2. Private Instance Methods

`retrieve_input()` [click to toggle source](#)

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

@return [FileObject] The current `FileObject`.

```
# File lib/tasks/text_segment_task.rb, line 35
def retrieve_input
  retrieve_file_object
end
```

`store_segments(textfile, segments)` [click to toggle source](#)

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]

```
# File lib/tasks/text_segment_task.rb, line 45
def store_segments(textfile, segments)
  logger.info "Storing #{segments.length} segments for file: #{textfile.name}"
  textfile.add_segments(segments)
  logger.debug "Segments stored successfully"
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 71. Texttaggerprocessor

71.1. class Flowbots::TextTaggerProcessor

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

71.1.1. Public Class Methods

`new()` [click to toggle source](#)

Initializes a new `TextTaggerProcessor` instance.

@return [void]

```
# File lib/processors/TextTaggerProcessor.rb, line 15
def initialize
  # Initializes a logger object.
  logger = Logger.new(STDOUT)

  # Loads the EngTagger library.
  load_engtagger
end
```

71.1.2. Public Instance Methods

`analyze_transitivity(text)` [click to toggle source](#)

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.

```
# File lib/processors/TextTaggerProcessor.rb, line 110
def analyze_transitivity(text)
  # Splits the text into sentences and removes empty sentences.
  sentences = text.split(/[.!?]+/).map(&:strip).reject(&:empty?)

  # Iterates through each sentence and analyzes its transitivity.
  sentences.map do |sent|
    sent_tagged = @tgr.add_tags(sent)
    if sent_tagged.nil?
      # Logs a warning message if the sentence cannot be tagged.
      logger.warn "Failed to tag sentence for transitivity analysis: #{sent}"
      { sentence: sent, process: nil, actor: nil, goal: nil }
    else
      # Extracts the process, actor, and goal from the tagged sentence.
      process = sent_tagged.scan(%r{<vb[dgnpz]?>([^\<]+)</vb[dgnpz]?>}).flatten.first
```

```

        actor = sent_tagged.scan(%r{<nn[ps]?>([^\<]+)</nn[ps]?>}).flatten.first
        goal = sent_tagged.scan(%r{<nn[ps]?>([^\<]+)</nn[ps]?>}).flatten[1]
        { sentence: sent, process:, actor:, goal: }
      end
    end
  end
end

```

`extract_main_topics(text, limit=5)` [click to toggle source](#)

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.

```

# File lib/processors/TextTaggerProcessor.rb, line 64
def extract_main_topics(text, limit=5)
  # Gets the noun phrases from the text using EngTagger.
  noun_phrases = @tgr.get_noun_phrases(text)

  # If noun phrases are found, sorts them by count in descending order and returns the
  top `limit` phrases.
  return if noun_phrases.nil?

  sorted_phrases = noun_phrases.sort_by { |_, count| -count }
  sorted_phrases.first(limit).map { |phrase, _| phrase }
end

```

`identify_speech_acts(text)` [click to toggle source](#)

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.

```

# File lib/processors/TextTaggerProcessor.rb, line 80
def identify_speech_acts(text)
  # Splits the text into sentences and removes empty sentences.
  sentences = text.split(/[.!?]+/).map(&:strip).reject(&:empty?)

  # Iterates through each sentence and classifies its speech act.
  sentences.map do |sent|
    sent_tagged = @tgr.add_tags(sent)
    if sent_tagged.nil?
      # Logs a warning message if the sentence cannot be tagged.
      logger.warn "Failed to tag sentence: #{sent}"
      "unknown"
    end
  end
end

```

```

    elsif sent_tagged.include?("<uh>")
      "interjection"
    elsif sent.end_with?("?")
      "question"
    elsif sent_tagged.start_with?("<md>", "<vb>")
      "request"
    elsif sent.end_with?("!")
      "exclamation"
    else
      "statement"
    end
  end
end
end

```

`process(text, options={})` [click to toggle source](#)

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.

```

# File lib/processors/TextTaggerProcessor.rb, line 29
def process(text, options={})
  # Logs a message indicating the start of text processing.
  logger.debug "Processing text with TextTaggerProcessor"
  logger.debug "Options: #{options.inspect}"

  # Creates a hash to store the tagged results.
  result = {}

  # Performs various tagging operations based on the provided options.
  result[:tagged_text] = @tgr.add_tags(text) if options[:tagged_text]
  result[:readable_text] = @tgr.get_readable(text) if options[:readable_text]
  result[:words] = @tgr.get_words(text) if options[:words]
  result[:nouns] = @tgr.get_nouns(text) if options[:nouns]
  result[:proper_nouns] = @tgr.get_proper_nouns(result[:tagged_text]) if
options[:proper_nouns]
  result[:past_tense_verbs] = @tgr.get_past_tense_verbs(result[:tagged_text]) if
options[:past_tense_verbs]
  result[:present_tense_verbs] = @tgr.get_present_verbs(result[:tagged_text]) if
options[:present_tense_verbs]
  result[:base_present_verbs] = @tgr.get_base_present_verbs(result[:tagged_text]) if
options[:base_present_verbs]
  result[:gerund_verbs] = @tgr.get_gerund_verbs(result[:tagged_text]) if
options[:gerund_verbs]
  result[:adjectives] = @tgr.get_adjectives(result[:tagged_text]) if
options[:adjectives]
  result[:noun_phrases] = @tgr.get_noun_phrases(result[:tagged_text]) if

```



```

options[:noun_phrases]
  result[:pronoun_list] = @tgr.get_pronoun_list(result[:tagged_text]) if
options[:pronoun_list]

  # Logs the processed results for debugging purposes.
  logger.debug "Processed result: #{result.inspect}"

  # Returns the hash of tagged results.
  result
end

```

71.1.3. Private Instance Methods

`load_engtagger()` [click to toggle source](#)

Loads the EngTagger library.

@return [void]

```

# File lib/processors/TextTaggerProcessor.rb, line 136
def load_engtagger
  # Logs a message indicating the start of EngTagger loading.
  logger.debug "Loading EngTagger"

  # Initializes a new EngTagger instance.
  @tgr = EngTagger.new

  # Logs a message indicating successful EngTagger loading.
  logger.debug "EngTagger loaded successfully"
  UI.say(:ok, "EngTagger loaded successfully")
rescue StandardError => e
  # Logs an error message if EngTagger loading fails.
  logger.error "Error loading EngTagger: #{e.message}"
  logger.error e.backtrace.join("\n")
  UI.say(:error, "Error loading EngTagger: #{e.message}")
  raise
end

```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 72. Texttaggertask

72.1. class TextTaggerTask

This class performs text tagging on a given text.

72.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

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.

```
# File lib/tasks/text_tagger_task.rb, line 18
def execute
  logger.info "Starting TextTaggerTask"

  # Retrieve the FileObject from Redis.
  text_tagger = Flowbots::TextTaggerProcessor.instance
  file_object = retrieve_input

  # Raise an error if the FileObject retrieval fails.
  if file_object.nil?
    logger.error "Failed to retrieve FileObject"
    raise("Failed to retrieve FileObject")
    return
  end

  # Get the preprocessed content from the FileObject.
  preprocessed_content = file_object.preprocessed_content

  # Raise an error if the preprocessed content is empty or nil.
  if preprocessed_content.nil? || preprocessed_content.empty?
    logger.error "Preprocessed content is empty or nil for FileObject:
#{file_object.id}"
    raise("Preprocessed content is empty or nil")
    return
  end

  # Perform text tagging using the TextTaggerProcessor.
  result = text_tagger.process(
    preprocessed_content,
    tagged_text: true,
```

```

    nouns: true,
    noun_phrases: true,
    adjectives: true,
    past_tense_verbs: true,
    present_tense_verbs: true
  )

  # Extract main topics from the preprocessed content.
  begin
    main_topics = text_tagger.extract_main_topics(preprocessed_content)
  rescue StandardError => e
    logger.warn "Error extracting main topics: #{e.message}"
    main_topics = []
  end

  # Identify speech acts in the preprocessed content.
  begin
    speech_acts = text_tagger.identify_speech_acts(preprocessed_content)
  rescue StandardError => e
    logger.warn "Error identifying speech acts: #{e.message}"
    speech_acts = []
  end

  # Analyze transitivity in the preprocessed content.
  begin
    transitivity = text_tagger.analyze_transitivity(preprocessed_content)
  rescue StandardError => e
    logger.warn "Error analyzing transitivity: #{e.message}"
    transitivity = []
  end

  # Store the tagging results in the FileObject.
  store_result(file_object, result, main_topics, speech_acts, transitivity)

  # Log the completion of the task.
  logger.info("Text tagging completed for FileObject: #{file_object.id}")
end

```

72.1.2. Private Instance Methods

`retrieve_input()` [click to toggle source](#)

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

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

```

# File lib/tasks/text_tagger_task.rb, line 89
def retrieve_input
  retrieve_file_object

```

```
end
```

`store_result(file_object, result, main_topics, speech_acts, transitivity)` [click to toggle source](#)

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]

```
# File lib/tasks/text_tagger_task.rb, line 102
def store_result(file_object, result, main_topics, speech_acts, transitivity)
  file_object.update(
    tagged: result,
    main_topics:,
    speech_acts:,
    transitivity:
  )
  logger.debug "Stored tagging result for FileObject: #{file_object.id}"
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 73. Texttospeechaction

73.1. class Sublayer::Actions::TextToSpeechAction

73.1.1. Public Class Methods

new(text) [click to toggle source](#)

```
# File lib/utils/tts.rb, line 4
def initialize(text)
  @text = text
end
```

73.1.2. Public Instance Methods

call() [click to toggle source](#)

```
# File lib/utils/tts.rb, line 8
def call
  speech = HTTParty.post(
    "https://api.openai.com/v1/audio/speech",
    headers: {
      "Authorization" => "Bearer #{ENV["OPENAI_API_KEY"]}",
      "Content-Type" => "application/json",
    },
    body: {
      "model": "tts-1",
      "input": @text,
      "voice": "nova",
      "response_format": "wav"
    }.to_json
  )

  speech
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 74. Texttokenizeprocessor

74.1. class Flowbots::TextTokenizeProcessor

This class provides functionality for tokenizing `text`.

74.1.1. Constants

DEFAULT_OPTIONS

Default `options` for the tokenizer.

74.1.2. Attributes

options[RW]

The `options` for the tokenizer.

text[RW]

The `text` to be tokenized.

74.1.3. Public Class Methods

new() click to toggle source

Initializes a new `TextTokenizeProcessor` instance.

@return [void]

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

```
# File lib/processors/TextTokenizeProcessor.rb, line 38
def initialize
  super
end
```

74.1.4. Public Instance Methods

process(text, opts={}) click to toggle source

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.

```
# File lib/processors/TextTokenizeProcessor.rb, line 48
```

```

def process(text, opts={})
  @text = text
  @options = DEFAULT_OPTIONS.merge(opts)
  logger.debug "TextTokenize initialized with options: #{@options}"

  logger.info "Starting text tokenizer"
  logger.debug "Input text type: #{@text.class}"

  if @text.instance_of?(Array)
    tokenize_array
  else
    tokenize_string(@text)
  end
end
end

```

74.1.5. Private Instance Methods

`tokenize_array()` [click to toggle source](#)

Tokenizes an array of strings.

@return [Array] An array of tokens.

```

# File lib/processors/TextTokenizeProcessor.rb, line 68
def tokenize_array
  logger.debug "Segmenting array of strings"
  @text.flat_map do |str|
    tokenize_string(str)
  end
end
end

```

`tokenize_string(str)` [click to toggle source](#)

Tokenizes a single string.

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

@return [Array] An array of tokens.

```

# File lib/processors/TextTokenizeProcessor.rb, line 80
def tokenize_string(str)
  logger.debug "Tokenizing string: #{str}"
  tokenizer = PragmaticTokenizer::Tokenizer.new(**@options)
  result = tokenizer.tokenize(str)
  logger.debug "Tokenization result: #{result}"
  result
end
end

```

Validate

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 75. Texttokenizetask

75.1. class TextTokenizeTask

This task tokenizes the segments of a text file.

75.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the task.

@return [void]

```
# File lib/tasks/text_tokenize_task.rb, line 9
def execute
  # Retrieve the segments from the latest FileObject.
  segments = FileObject.latest.retrieve_segments

  # Get an instance of the TextTokenizeProcessor.
  text_tokenizer = Flowbots::TextTokenizeProcessor.instance

  # Iterate through each segment and tokenize its text.
  segments.each do |segment|
    # Tokenize the segment's text using the TextTokenizeProcessor.
    tokens = text_tokenizer.process(segment.text, { clean: true })

    # Update the segment with the tokenized data.
    segment.update(tokens:)

    # Save the updated segment.
    segment.save
  end
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 76. Tokenizesegmentstask

76.1. class TokenizeSegmentsTask

This task tokenizes the segments of a text file.

76.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the task to tokenize the segments of a `FileObject`.

Retrieves the `FileObject` from Redis, tokenizes each segment using the `TextTokenizerProcessor`, updates the segments with the tokenized data, and logs the progress.

@return [void]

```
# File lib/tasks/tokenize_segments_task.rb, line 16
def execute
  # Log the start of the task.
  logger.info "Starting TokenizeSegmentsTask"

  # Retrieve the FileObject from Redis.
  textfile = retrieve_input
  # Get an instance of the TextTokenizerProcessor.
  text_tokenizer = Flowbots::TextTokenizerProcessor.instance

  # Iterate through each segment of the FileObject and tokenize its text.
  textfile.retrieve_segments.each do |segment|
    # Tokenize the segment's text using the TextTokenizerProcessor.
    tokens = text_tokenizer.process(segment.text, { clean: true })
    # Update the segment with the tokenized data.
    segment.update(tokens:)
  end

  # Log the completion of the task.
  logger.info "TokenizeSegmentsTask completed"
end
```

76.1.2. Private Instance Methods

`retrieve_input()` [click to toggle source](#)

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

@return [FileObject] The current `FileObject`.

```
# File lib/tasks/tokenize_segments_task.rb, line 42
```

```
def retrieve_input
  retrieve_file_object
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 77. Topic

77.1. class Topic

Defines the `Topic` model.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 78. Topicmodelprocessor

78.1. class Flowbots::TopicModelProcessor

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

78.1.1. 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.

78.1.2. Public Class Methods

`new()` click to toggle source

Initializes a new `TopicModelProcessor` instance.

@return [void]

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

```
# File lib/processors/TopicModelProcessor.rb, line 23
def initialize
  super
  @model_params = {
    k: 20, # Number of topics
    tw: :one, # Topic weight
    min_cf: 3, # Minimum count for a word to be included in the vocabulary
    min_df: 2, # Minimum document frequency for a word to be included in the
vocabulary
    rm_top: 4, # Number of top words to remove from the vocabulary
    alpha: 0.1, # Dirichlet prior parameter for the topic distribution
    eta: 0.01, # Dirichlet prior parameter for the word distribution
    seed: 42 # Random seed for reproducibility
  }
  @model_path = TOPIC_MODEL_PATH
  @model = nil
end
```

78.1.3. Public Instance Methods

`infer_topics(document)` [click to toggle source](#)

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.

```
# File lib/processors/TopicModelProcessor.rb, line 106
def infer_topics(document)
  ensure_model_exists

  doc = @model.make_doc(document.split)
  topic_dist, = @model.infer(doc)

  return {} if topic_dist.nil?

  most_probable_topic = topic_dist.each_with_index.max_by { |prob, _| prob }[1]
  top_words = @model.topic_words(most_probable_topic, top_n: 10)

  {
    most_probable_topic:,
    topic_distribution: topic_dist.to_a,
    top_words:
  }
end
```

`load_or_create_model()` [click to toggle source](#)

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

@return [void]

```
# File lib/processors/TopicModelProcessor.rb, line 42
def load_or_create_model
  if File.exist?(@model_path)
    load_existing_model
  else
    create_new_model
  end
end
```

`train_model(documents, iterations=100)` [click to toggle source](#)

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]

```
# File lib/processors/TopicModelProcessor.rb, line 56
def train_model(documents, iterations=100)
  ensure_model_exists

  logger.info "Training topic model"
  UI.say(:ok, "Training topic model")

  documents.each do |doc|
    words = doc.split if doc.instance_of?(String)
    words = doc if doc.instance_of?(Array)
    @model.add_doc(words) if words.any?
  end

  @model.burn_in = iterations
  @model.train(0)

  ui.info "Removed Top Words" do
    ui.puts "#{@model.removed_top_words}"
    ui.puts "Training..."
    ui.framed do
      100.times do |i|
        @model.train(10)
        ui.puts "Iteration: #{i * 10}\tLog-likelihood: #{@model.ll_per_word}"
      end
    end
    ui.framed do
      ui.puts @model.summary
    end
  end

  ui.space

  if @model.num_words == 0
    raise FlowbotError.new("No valid words found in the provided documents",
"EMPTY_VOCABULARY")
  end

  ui.info "Vocab" do
    ui.puts "Documents added to model. Vocab size: #{@model.used_vocabs.length}, Total
words: #{@model.num_words}"
  end

  logger.info "Model training completed"
  UI.say(:ok, "Model training completed")
end
```

```
save_model
end
```

78.1.4. Private Instance Methods

`create_new_model()` [click to toggle source](#)

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

@return [void]

```
# File lib/processors/TopicModelProcessor.rb, line 151
def create_new_model
  logger.info "Creating new model"
  begin
    @model = Tomoto::LDA.new(**@model_params)
    logger.debug "New Model created"
  rescue StandardError => e
    logger.error "Failed to create new model: #{e.message}"
    raise FlowbotError.new("Failed to create topic model", "MODEL_CREATE_ERROR",
details: e.message)
  end
end
```

`ensure_model_exists()` [click to toggle source](#)

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

@return [void]

```
# File lib/processors/TopicModelProcessor.rb, line 129
def ensure_model_exists
  load_or_create_model if @model.nil?
end
```

`load_existing_model()` [click to toggle source](#)

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

@return [void]

```
# File lib/processors/TopicModelProcessor.rb, line 136
def load_existing_model
  UI.info "Loading existing model from #{@model_path}"
  begin
    @model = Tomoto::LDA.load(@model_path)
    logger.debug "Model loading completed"
    UI.say(:ok, "Topic model loading completed")
  end
end
```



```

rescue StandardError => e
  logger.error "Failed to load existing model: #{e.message}"
  raise FlowbotError.new("Failed to load topic model", "MODEL_LOAD_ERROR", details:
e.message)
end
end

```

save_model() [click to toggle source](#)

Saves the topic **model** to the specified path.

@return [void]

```

# File lib/processors/TopicModelProcessor.rb, line 165
def save_model
  logger.info "Attempting to save model to #{TOPIC_MODEL_PATH}"
  UI.say(:ok, "Attempting to save topic model")

  begin
    # Check if the directory exists, create it if it doesn't
    dir = File.dirname(TOPIC_MODEL_PATH)
    FileUtils.mkdir_p(dir) unless File.directory?(dir)

    # Check if we have write permissions
    unless File.writable?(dir)
      raise FlowbotError.new(
        "No write permission for directory: #{dir}",
        "PERMISSION_ERROR"
      )
    end

    # Check available disk space (example threshold: 100MB)
    available_space = `df -k #{dir} | awk '{print $4}' | tail -n 1`.to_i * 1024
    if available_space < 100_000_000 # 100MB in bytes
      raise FlowbotError.new(
        "Insufficient disk space. Only #{available_space / 1_000_000}MB available.",
        "DISK_SPACE_ERROR"
      )
    end

    # Attempt to save the model
    @model.save(TOPIC_MODEL_PATH)

    logger.info "Model successfully saved to #{TOPIC_MODEL_PATH}"
    UI.say(:ok, "Topic model saved successfully")
  rescue Tomoto::Error => e
    logger.error "Tomoto gem error while saving model: #{e.message}"
    raise FlowbotError.new(
      "Failed to save topic model due to Tomoto gem error: #{e.message}",
      "TOMOTO_SAVE_ERROR"
    )
  end
end

```

```
)  
  rescue FlowbotError => e  
    logger.error e.message  
    raise e  
  rescue StandardError => e  
    logger.error "Unexpected error while saving model: #{e.message}"  
    raise FlowbotError.new("Unexpected error while saving topic model: #{e.message}",  
"SAVE_ERROR")  
  end  
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 79. TopicModelTrainerWorkflow

79.1. 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.

79.1.1. Attributes

pipeline[R]

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

79.1.2. Public Class Methods

new(input_folder_path=nil) [click to toggle source](#)

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]

```
# File lib/workflows/topic_model_trainer_workflow.rb, line 18
def initialize(input_folder_path=nil)
  @input_folder_path = input_folder_path || prompt_for_folder
  @pipeline = UnifiedFileProcessingPipeline.new(@input_folder_path, batch_size: 10,
file_types: %w[md markdown])
end
```

79.1.3. Public Instance Methods

run() [click to toggle source](#)

Runs the topic model trainer workflow.

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

@return [void]

```
# File lib/workflows/topic_model_trainer_workflow.rb, line 28
def run
  UI.say(:ok, "Setting Up Topic Model Trainer Workflow")
  logger.info "Setting Up Topic Model Trainer Workflow"
```

```

begin
  @pipeline.process
  train_topic_model
  UI.say(:ok, "Topic Model Trainer Workflow completed")
  logger.info "Topic Model Trainer Workflow completed"
rescue StandardError => e
  UI.say(:error, "Error in Topic Model Trainer Workflow: #{e.message}")
  logger.error "Error in Topic Model Trainer Workflow: #{e.message}"
  logger.error e.backtrace.join("\n")
end
end

```

79.1.4. Private Instance Methods

`clean_segments_for_modeling(segments)` [click to toggle source](#)

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.

```

# File lib/workflows/topic_model_trainer_workflow.rb, line 95
def clean_segments_for_modeling(segments)
  segments.reject do |segment|
    segment.include?("tags") || segment.include?("title") || segment.include?("toc")
  end.map do |segment|
    segment.reject do |word|
      word.to_s.length < 3 || # Remove very short words
      word.to_s.match?(/\^d+$/) || # Remove purely numeric words
      word.to_s.match?(/\^[[:punct:]]+$/) # Remove punctuation-only words
    end
  end.reject(&:empty?)
end

```

`prompt_for_folder()` [click to toggle source](#)

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.

```

# File lib/workflows/topic_model_trainer_workflow.rb, line 50
def prompt_for_folder
  get_folder_path = `gum file --directory`.chomp.strip
  folder_path = File.join(get_folder_path)

  raise FlowbotError.new("Folder not found", "FOLDERNOTFOUND") unless
    File.directory?(folder_path)

```

```
    folder_path
  end
```

`train_topic_model()` [click to toggle source](#)

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]

```
# File lib/workflows/topic_model_trainer_workflow.rb, line 65
def train_topic_model
  all_filtered_segments =
  JSON.parse(Jongleur::WorkerTask.class_variable_get(:@@redis).get("all_filtered_segments") || "[]")

  if all_filtered_segments.empty?
    logger.warn "No filtered segments available for topic modeling"
    UI.say(:warn, "No filtered segments available for topic modeling")
    return
  end

  cleaned_segments = clean_segments_for_modeling(all_filtered_segments)

  if cleaned_segments.empty?
    logger.warn "No cleaned segments available for topic modeling after filtering"
    UI.say(:warn, "No cleaned segments available for topic modeling after filtering")
    return
  end

  logger.info "Cleaned segments for topic modeling. Original count:
#{all_filtered_segments.length}, Cleaned count: #{cleaned_segments.length}"

  topic_processor = Flowbots::TopicModelProcessor.instance
  topic_processor.train_model(cleaned_segments)
  logger.info "Topic model training completed for all files"
  UI.say(:ok, "Topic model training completed for all files")
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 80. Topicmodeltrainerworkflowtest

80.1. class Flowbots::TopicModelTrainerWorkflowtest

80.1.1. Constants

BATCH_SIZE

80.1.2. Attributes

input_folder_path[R]

orchestrator[RW]

80.1.3. Public Class Methods

new(input_folder_path=nil) [click to toggle source](#)

```
# File lib/workflows/topic_model_trainer_workflowtest.rb, line 11
def initialize(input_folder_path=nil)
  @input_folder_path = input_folder_path || prompt_for_folder
  @orchestrator = WorkflowOrchestrator.new
end
```

80.1.4. Public Instance Methods

run() [click to toggle source](#)

```
# File lib/workflows/topic_model_trainer_workflowtest.rb, line 16
def run
  UI.say(:ok, "Setting Up Topic Model Trainer Workflow")
  logger.info "Setting Up Topic Model Trainer Workflow"

  begin
    setup_workflow
    flush_redis_cache
    process_files
    UI.say(:ok, "Topic Model Trainer Workflow completed")
    logger.info "Topic Model Trainer Workflow completed"
  rescue StandardError => e
    UI.say(:error, "Error in Topic Model Trainer Workflow: #{e.message}")
    logger.error "Error in Topic Model Trainer Workflow: #{e.message}"
    logger.error e.backtrace.join("\n")
  end
end
```

80.1.5. Private Instance Methods

`clean_segments_for_modeling(segments)` [click to toggle source](#)

```
# File lib/workflows/topic_model_trainer_workflowtest.rb, line 112
def clean_segments_for_modeling(segments)
  segments.reject do |segment|
    segment.include?("tags") || segment.include?("title") || segment.include?("toc")
  end.map do |segment|
    segment.reject do |word|
      word.to_s.length < 3 || # Remove very short words
      word.to_s.match?(/\^d+$/) || # Remove purely numeric words
      word.to_s.match?(/\^[[:punct:]]+$/) # Remove punctuation-only words
    end
  end.reject(&:empty?)
end
```

`flush_redis_cache()` [click to toggle source](#)

```
# File lib/workflows/topic_model_trainer_workflowtest.rb, line 54
def flush_redis_cache
  redis = Jongleur::WorkerTask.class_variable_get(:@@redis)
  redis.flushdb
  logger.info "Redis cache flushed"
end
```

`process_batch(batch_files)` [click to toggle source](#)

```
# File lib/workflows/topic_model_trainer_workflowtest.rb, line 78
def process_batch(batch_files)
  batch_files.each do |file_path|
    Jongleur::WorkerTask.class_variable_get(:@@redis).set("current_file_path",
file_path)
    @orchestrator.run_workflow
  end
end
```

`process_files()` [click to toggle source](#)

```
# File lib/workflows/topic_model_trainer_workflowtest.rb, line 60
def process_files
  all_file_paths = Dir.glob(File.join(@input_folder_path,
"**{,/*/**}/*.{md,markdown}")).sort
  total_files = all_file_paths.count
  num_batches = (total_files.to_f / BATCH_SIZE).ceil

  num_batches.times do |i|
```

```

    batch_start = i * BATCH_SIZE
    batch_files = all_file_paths[batch_start, BATCH_SIZE]

    UI.say(:ok, "Processing batch #{i + 1} of #{num_batches}")
    logger.info "Processing batch #{i + 1} of #{num_batches}"

    process_batch(batch_files)
  end

  train_topic_model
end

```

`prompt_for_folder()` [click to toggle source](#)

```

# File lib/workflows/topic_model_trainer_workflowtest.rb, line 35
def prompt_for_folder
  get_folder_path = `gum file --directory`.chomp.strip
  folder_path = File.join(get_folder_path)

  raise FlowbotError.new("Folder not found", "FOLDERNOTFOUND") unless
File.directory?(folder_path)

  folder_path
end

```

`setup_workflow()` [click to toggle source](#)

```

# File lib/workflows/topic_model_trainer_workflowtest.rb, line 44
def setup_workflow
  workflow_graph = {
    LoadFileObjectsTask: [:PreprocessFileObjectTask],
    PreprocessFileObjectTask: []
  }

  @orchestrator.define_workflow(workflow_graph)
  logger.debug "Workflow setup completed"
end

```

`train_topic_model()` [click to toggle source](#)

```

# File lib/workflows/topic_model_trainer_workflowtest.rb, line 85
def train_topic_model
  all_filtered_segments =
JSON.parse(Jongleur::WorkerTask.class_variable_get(:@@redis).get("all_filtered_segments") || "[]")

  if all_filtered_segments.empty?

```



```

    logger.warn "No filtered segments available for topic modeling"
    UI.say(:warn, "No filtered segments available for topic modeling")
    return
  end

  cleaned_segments = clean_segments_for_modeling(all_filtered_segments)

  if cleaned_segments.empty?
    logger.warn "No cleaned segments available for topic modeling after filtering"
    UI.say(:warn, "No cleaned segments available for topic modeling after filtering")
    return
  end

  logger.info "Cleaned segments for topic modeling. Original count:
#{all_filtered_segments.length}, Cleaned count: #{cleaned_segments.length}"

  topic_processor = Flowbots::TopicModelProcessor.instance
  topic_processor.train_model(cleaned_segments)
  logger.info "Topic model training completed for all files"
  UI.say(:ok, "Topic model training completed for all files")
end

```

Validate

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 81. Topicmodelingtask

81.1. class TopicModelingTask

This task performs topic modeling on a text file using a pre-trained model.

81.1.1. Public Class Methods

`new(model_params)` [click to toggle source](#)

```
# File lib/general_task_agent.rb, line 63
def initialize(model_params)
  @model = TopicModel.new(model_params)
end
```

81.1.2. Public Instance Methods

`execute()` [click to toggle source](#)

```
# File lib/general_task_agent.rb, line 67
def execute
  # Perform topic modeling
  documents = get_documents
  topics = @model.extract_topics(documents)
  store_topics(topics)
end
```

81.1.3. Private Instance Methods

`filter_segment_words(segment)` [click to toggle source](#)

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.

```
# File lib/tasks/topic_modeling_task.rb, line 70
def filter_segment_words(segment)
  # Define the relevant POS tags for filtering.
  relevant_tags = %w[NN JJ RB]
  # Retrieve the POS tagged tokens from the segment.
  tokens = segment.tagged&.&("pos") || []
  # Select tokens with relevant POS tags and extract their words.
  tokens.select { |token| relevant_tags.include?(token["tag"]) }
    .map { |token| token["word"] }
```

```
end
```

`get_documents()` [click to toggle source](#)

```
# File lib/general_task_agent.rb, line 76
def get_documents
  # Retrieve documents for topic modeling
end
```

`retrieve_filtered_words(textfile)` [click to toggle source](#)

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.

```
# File lib/tasks/topic_modeling_task.rb, line 58
def retrieve_filtered_words(textfile)
  # Retrieve the segments from the FileObject.
  segments = textfile.retrieve_segments
  # Filter words from each segment and flatten the resulting array.
  segments.flat_map { |segment| filter_segment_words(segment) }
end
```

`retrieve_input()` [click to toggle source](#)

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

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

```
# File lib/tasks/topic_modeling_task.rb, line 48
def retrieve_input
  retrieve_file_object
end
```

`store_topic_result(textfile, result)` [click to toggle source](#)

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`]

```

# File lib/tasks/topic_modeling_task.rb, line 89
def store_topic_result(textfile, result)
  # Extract unique words from the topic results.
  unique_words = result.each_with_object(Set.new) do |hash, words|
    # Add top words from the topic.
    hash[:top_words].keys.each do |key|
      words << key if key.is_a?(String) && key.match?(/^[a-zA-Z]+$/)
    end
    # Add sorted words from the topic.
    hash[:sorted_words].each do |word|
      words << word[0] if word[0].is_a?(String) && word[0].match?(/^[a-zA-Z]+$/)
    end
  end

  # Add the unique words as topics to the FileObject.
  textfile.add_topics(unique_words.to_a)
  # Log the number of topics stored.
  logger.info "Stored #{result.length} topics for Textfile #{textfile.id}"
end

```

store_topics(topics) [click to toggle source](#)

```

# File lib/general_task_agent.rb, line 80
def store_topics(topics)
  # Store the extracted topics
end

```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 82. TrainTopicModelTask

82.1. class TrainTopicModelTask

This task trains a topic model using filtered segments from multiple batches.

82.1.1. Public Instance Methods

`execute()` [click to toggle source](#)

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]

```
# File lib/tasks/train_topic_model_task.rb, line 13
def execute
  # Retrieve the current batch ID from Redis.
  batch_id = Jongleur::WorkerTask.class_variable_get(:@@redis).get("current_batch_id")
  logger.info "Processing TrainTopicModelTask for batch #{batch_id}"

  # Retrieve filtered segments from Redis.
  filtered_segments =
JSON.parse(Jongleur::WorkerTask.class_variable_get(:@@redis).get("filtered_segments")
|| "[]")

  # Accumulate filtered segments across batches.
  all_filtered_segments =
JSON.parse(Jongleur::WorkerTask.class_variable_get(:@@redis).get("all_filtered_segment
s") || "[]")
  all_filtered_segments += filtered_segments
  Jongleur::WorkerTask.class_variable_get(:@@redis).set("all_filtered_segments",
all_filtered_segments.to_json)

  # Only train the model on the last batch.
  if batch_id.end_with?(
    "#{Dir.glob(
      File.join(
        @input_folder,
        '*.txt'
      )
    ).count.to_f / Flowbots::TopicModelTrainerWorkflow::BATCH_SIZE).ceil}"
  )
  # Create an instance of the TopicModelProcessor.
  topic_processor = Flowbots::TopicModelProcessor.instance
```

```
# Train the topic model using the accumulated filtered segments.
topic_processor.train_model(all_filtered_segments)

# Log and display a success message.
logger.info "Topic model training completed for all batches"
UI.say(:ok, "Topic model training completed for all batches")
else
  # Log and display a message indicating that filtered segments have been
  accumulated.
  logger.info "Accumulated filtered segments for batch #{batch_id}"
  UI.say(:ok, "Accumulated filtered segments for batch #{batch_id}")
end
end
```

Validate

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 83. Ui

83.1. module UI

Provides user interface functionality for the **Flowbots** application.

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**.

83.1.1. Constants

PASTEL

An instance of the Pastel gem for colorizing text.

TITLE_WIDTH

The width of the title box.

83.1.2. Public Class Methods

`custom_workflow()` [click to toggle source](#)

Creates a custom workflow.

Prompts the user for a workflow name and tasks, and displays the created workflow.

@return [void]

```
# File lib/ui.rb, line 94
def self.custom_workflow
  say(:info, "Custom Workflow Creator")
  workflow_name = prompt.ask("Enter a name for your custom workflow:")
  tasks = []
  loop do
    task = prompt.ask("Enter a task for your workflow (or 'done' to finish):")
    break if task.downcase == "done"

    tasks << task
  end
  say(:ok, "Custom workflow '#{workflow_name}' created with #{tasks.size} tasks.")
  tasks.each_with_index do |task, index|
    say(:info, "Task #{index + 1}: #{task}")
  end
end
```

`edit_api_settings()` [click to toggle source](#)

Edits **API** settings.

Displays a placeholder message indicating the **API** settings editor.

@return [void]

```
# File lib/ui.rb, line 166
def self.edit_api_settings
  # Implement API settings editor
  say(:info, "Editing API Settings")
  # Add more specific setting options here
end
```

edit_general_settings() click to toggle source

Edits general settings.

Displays a placeholder message indicating the general settings editor.

@return [void]

```
# File lib/ui.rb, line 144
def self.edit_general_settings
  # Implement general settings editor
  say(:info, "Editing General Settings")
  # Add more specific setting options here
end
```

edit_workflow_settings() click to toggle source

Edits workflow settings.

Displays a placeholder message indicating the workflow settings editor.

@return [void]

```
# File lib/ui.rb, line 155
def self.edit_workflow_settings
  # Implement workflow settings editor
  say(:info, "Editing Workflow Settings")
  # Add more specific setting options here
end
```

open_settings() click to toggle source

Opens the settings menu.

Displays a menu of settings categories and handles user selection.

@return [void]

```
# File lib/ui.rb, line 115
def self.open_settings
  settings_choices = [
    { name: "General Settings", value: :general },
    { name: "Workflow Settings", value: :workflow },
    { name: "API Settings", value: :api },
    { name: "Back to Main Menu", value: :back }
  ]

  loop do
    choice = prompt.select("Select a settings category:", settings_choices)

    case choice
    when :general
      edit_general_settings
    when :workflow
      edit_workflow_settings
    when :api
      edit_api_settings
    when :back
      break
    end
  end
end
```

run_workflow(workflow_name) [click to toggle source](#)

Runs a workflow with a given name.

Displays a **spinner** indicating workflow execution and simulates completion.

@param workflow_name [String] The name of the workflow to run.

@return [void]

```
# File lib/ui.rb, line 79
def self.run_workflow(workflow_name)
  spinner = TTY::Spinner.new("[:spinner] Running #{workflow_name} Workflow...",
    format: :dots)
  spinner.auto_spin

  # Simulate workflow execution
  sleep(3)

  spinner.success("(#{workflow_name} Workflow completed successfully!)")
end
```

`start()` click to toggle source

Starts the main **UI** loop.

Displays the main menu and handles user interactions.

@return [void]

```
# File lib/ui.rb, line 28
def self.start
  loop do
    header
    case main_menu
    when :workflow
      start_workflow
    when :settings
      open_settings
    when :logs
      view_logs
    when :exit
      say(:ok, "Thank you for using Flowbots. Goodbye!")
      break
    end
  end
end
```

`start_workflow()` click to toggle source

Starts the workflow selection process.

Displays a menu of available workflows and handles user selection.

@return [void]

```
# File lib/ui.rb, line 50
def self.start_workflow
  workflow_choices = [
    { name: "Text Processing Workflow", value: :text_processing },
    { name: "Topic Model Trainer Workflow", value: :topic_model },
    { name: "Custom Workflow", value: :custom },
    { name: "Back to Main Menu", value: :back }
  ]

  choice = prompt.select("Select a workflow to start:", workflow_choices)

  case choice
  when :text_processing
    run_workflow("Text Processing")
  when :topic_model
    run_workflow("Topic Model Trainer")
  end
end
```

```

when :custom
  custom_workflow
when :back
  nil
end
end

```

`view_logs()` click to toggle source

Views the application logs.

Attempts to read the log file and display it in a scrollable box. Handles potential errors during file reading.

@return [void]

```

# File lib/ui.rb, line 178
def self.view_logs
  log_file = File.join(LOG_DIR, "flowbots-#{current_date}.log")
  begin
    log_content = File.read(log_file)
    ScrollableBox.side_by_side_boxes(
      log_content,
      "Log Analysis Goes Here",
      title1: "Log Viewer",
      title2: "Log Analysis"
    )
  rescue Errno::ENOENT
    say(:error, "Log file not found: #{log_file}")
  rescue StandardError => e
    say(:error, "Error reading log file: #{e.message}")
  end
end

```

83.1.3. Public Instance Methods

`footer()` click to toggle source

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

@return [void]

```

# File lib/ui/base.rb, line 75
def footer
  # Create a framed box with the "Alright" title in the bottom right corner.
  box = TTY::Box.frame(width: TITLE_WIDTH, title: { bottom_right: " Alright " })
  # Print the box.
  puts box
end

```

```
end
```

`header()` click to toggle source

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

@return [void]

```
# File lib/ui/base.rb, line 65
def header
  # Create a framed box with the Flowbots title and magenta foreground color.
  box = TTY::Box.frame(width: TITLE_WIDTH, title: { top_left: " Flowbots " }, style: {
    fg: :magenta })
  # Print the box.
  puts box
end
```

`info(text)` click to toggle source

Displays an information message in a framed box.

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

@return [void]

```
# File lib/ui/base.rb, line 51
def info(text)
  # Display the header.
  header
  # Create a framed box with the specified text and title.
  box = TTY::Box.frame(width: TITLE_WIDTH, title: { top_left: " Information " },
    padding: 1) do
    text
  end
  # Print the box.
  puts box
end
```

`main_menu()` click to toggle source

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

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

```
# File lib/ui/base.rb, line 85
def main_menu
  # Define the choices for the main menu.
  choices = [
```

```

    { name: "Start Workflow", value: :workflow },
    { name: "Configure Settings", value: :settings },
    { name: "View Logs", value: :logs },
    { name: "Exit", value: :exit }
  ]

  # Display the main menu prompt and return the selected choice.
  prompt.select("Welcome to Flowbots. What would you like to do?", choices)
end

```

prompt() click to toggle source

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

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

```

# File lib/ui/base.rb, line 16
def prompt
  @prompt ||= TTY::Prompt.new(enable_color: true, active_color: :cyan)
end

```

say(type, statement) click to toggle source

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]

```

# File lib/ui/base.rb, line 26
def say(type, statement)
  # Default to :ok type if nil.
  type = :ok if type.nil?

  # Display the message based on the type and log it.
  case type
  when :ok
    prompt.ok(statement)
    logger.info statement
  when :warn
    prompt.warn(statement)
    logger.warn statement
  when :error
    prompt.error(statement)
    logger.fatal statement
  else
    puts PASTEL.say(statement)
  end
end

```

```
end
```

spinner(text) click to toggle source

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.

```
# File lib/ui/base.rb, line 103
def spinner(text)
  TTY::Spinner.new("[:spinner] #{text}", format: :dots)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 84. Unifiedfileprocessingpipeline

84.1. 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.

84.1.1. Attributes

`batch_processor[R]`

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

`orchestrator[R]`

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

84.1.2. Public Class Methods

`new(input_path, batch_size: 10, file_types: %w[md markdown txt pdf json])` [click to toggle source](#)

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]

```
# File lib/pipelines/unified_file_processing.rb, line 20
def initialize(input_path, batch_size: 10, file_types: %w[md markdown txt pdf json])
  @input_path = input_path
  @batch_size = batch_size
  @file_types = file_types
  @orchestrator = WorkflowOrchestrator.new
  @batch_processor = BatchProcessor.new(@input_path, @batch_size, @file_types)
end
```

84.1.3. Public Instance Methods

`process()` [click to toggle source](#)

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

@return [void]

```
# File lib/pipelines/unified_file_processing.rb, line 31
def process
```

```

setup_workflow
# flush_redis_cache

if File.directory?(@input_path)
  process_batch
else
  process_single_file
end
end
end

```

84.1.4. Private Instance Methods

`flush_redis_cache()` [click to toggle source](#)

Flushes the Redis cache.

@return [void]

```

# File lib/pipelines/unified_file_processing.rb, line 65
def flush_redis_cache
  redis = Jongleur::WorkerTask.class_variable_get(:@@redis)
  redis.flushdb
  logger.info "Redis cache flushed"
end

```

`process_batch()` [click to toggle source](#)

Processes a batch of files using the batch processor.

@return [void]

```

# File lib/pipelines/unified_file_processing.rb, line 74
def process_batch
  @batch_processor.process_files do |file_path|
    process_file(file_path)
  end
end
end

```

`process_file(file_path)` [click to toggle source](#)

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]

```

# File lib/pipelines/unified_file_processing.rb, line 92
def process_file(file_path)

```



```

RedisKeys.set(RedisKeys::CURRENT_FILE_PATH, file_path)

logger.info "Processing file: #{file_path}"

result = @orchestrator.run_workflow

if result.nil?
  logger.warn "Workflow returned nil for file: #{file_path}"
else
  logger.info "Successfully processed file: #{file_path}"
end

rescue StandardError => e
  logger.error "Error processing file #{file_path}: #{e.message}"
  logger.error e.backtrace.join("\n")
  UI.say(:error, "Error processing file #{file_path}: #{e.message}")
ensure
  # Clear the current file path and file object ID from Redis after processing
  # RedisKeys.set(RedisKeys::CURRENT_FILE_PATH, nil)
  # RedisKeys.set(RedisKeys::CURRENT_FILE_OBJECT_ID, nil)
  puts "HEEEEEEEEEEEY!"
end

```

`process_single_file()` [click to toggle source](#)

Processes a single file.

@return [void]

```

# File lib/pipelines/unified_file_processing.rb, line 83
def process_single_file
  process_file(@input_path)
end

```

`setup_workflow()` [click to toggle source](#)

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

@return [void]

```

# File lib/pipelines/unified_file_processing.rb, line 47
def setup_workflow
  workflow_graph = {
    LoadFileObjectTask: [:PreprocessFileObjectTask],
    PreprocessFileObjectTask: [:TextSegmentTask],
    TextSegmentTask: [:TokenizeSegmentsTask],
    TokenizeSegmentsTask: [:NlpAnalysisTask],
    NlpAnalysisTask: [:FilterSegmentsTask],
    FilterSegmentsTask: [:AccumulateFilteredSegmentsTask],
    AccumulateFilteredSegmentsTask: []
  }
end

```

```
}  
  
@orchestrator.define_workflow(workflow_graph)  
  logger.debug "Workflow setup completed"  
end
```

Validate

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 85. Word

85.1. class Word

Defines the **Word** model.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 86. Workertask

86.1. 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](#) 6.6.3.1.

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

Chapter 87. Workflowagent

87.1. 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**.

87.2. 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.

87.3. 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.

87.4. Example Usage

```
+ruby agent = WorkflowAgent.new("preprocessor", "path/to/cartridge.yml") result =  
agent.process(input_text) agent.save_state +
```

87.5. 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.

87.5.1. Constants

CARTRIDGE_DIR

The base directory for cartridges.

87.5.2. Attributes

role[R]

state[R]

87.5.3. Public Class Methods

new(role, cartridge_file) [click to toggle source](#)

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]

```
# File lib/components/WorkflowAgent.rb, line 52
def initialize(role, cartridge_file)
  @role = role
  @state = {}
  @bot = NanoBot.new(
    cartridge: cartridge_file
  )
  UI.info "Initialized WorkflowAgent with role: #{role}, cartridge: #{cartridge_file}"
end
```

87.5.4. Public Instance Methods

load_state() [click to toggle source](#)

Loads the agent's `state` from Redis.

@return [void]

```
# File lib/components/WorkflowAgent.rb, line 99
def load_state
  state_json = Jongleur::WorkerTask.class_variable_get(:@@redis).hget(
    Process.pid.to_s,
    "agent:#{@role}"
  )
  @state = JSON.parse(state_json) if state_json
end
```

process(input) [click to toggle source](#)

Processes the given input using the agent's cartridge.

@param `input` [String] The input to `process`.

@return [String] The agent's response.

```
# File lib/components/WorkflowAgent.rb, line 66
def process(input)
  logger.debug "Processing input for #{@role}: #{input}"

  pastel = Pastel.new
  puts UI::Box.eval_result_box(
    @bot.eval(input) do |content, fragment, finished, meta|
      @response = content unless content.nil?
      pastel.blue(fragment) unless fragment.nil?
      sleep 0.025
    end
  )

  sleep(2)

  # UI.info(@response)
  update_state(@response)
  @response
end
```

save_state() [click to toggle source](#)

Saves the agent's **state** to Redis.

@return [void]

```
# File lib/components/WorkflowAgent.rb, line 88
def save_state
  Jongleur::WorkerTask.class_variable_get(:@@redis).hset(
    Process.pid.to_s,
    "agent:#{@role}",
    @state.to_json
  )
end
```

87.5.5. Private Instance Methods

update_state(response) [click to toggle source](#)

Updates the agent's **state** with the latest response.

@param response [String] The agent's response.

@return [void]

```
# File lib/components/WorkflowAgent.rb, line 114
```

```
def update_state(response)
  @state[:last_response] = response
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 88. Workflowerror

88.1. class Flowbots::WorkflowError

Error raised when there is a problem with a workflow.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 89. Workflowinitializertask

89.1. class WorkflowInitializerTask

This task initializes a workflow based on the specified workflow type.

89.1.1. Constants

BATCH_SIZE

The batch size for processing files in a batch workflow.

89.1.2. Public Instance Methods

`execute()` [click to toggle source](#)

Executes the workflow initialization task.

Determines the workflow type from Redis and initializes either a batch workflow or a single file workflow accordingly.

@return [void]

```
# File lib/tasks/workflow_initializer_task.rb, line 15
def execute
  workflow_type = Ohm.redis.get("workflow_type")

  if workflow_type == "topic_model_trainer"
    initialize_batch_workflow
  else
    initialize_single_file_workflow
  end
end
```

89.1.3. Private Instance Methods

`create_batches(workflow, file_paths)` [click to toggle source](#)

Creates batches of source files for a batch workflow.

Iterates through the file paths in batches and creates a new batch for each slice of files. Each source file is associated with its corresponding batch and workflow.

@param workflow [Workflow] The workflow to associate the batches with. @param file_paths [Array<String>] The list of file paths to create batches from.

@return [void]

```
# File lib/tasks/workflow_initializer_task.rb, line 85
```

```

def create_batches(workflow, file_paths)
  file_paths.each_slice(BATCH_SIZE).with_index(1) do |batch_files, batch_number|
    batch = Batch.create(number: batch_number, status: "pending", workflow: workflow)
    batch_files.each do |file_path|
      sourcefile = Sourcefile.find_or_create_by_path(file_path, workflow: workflow,
batch: batch)
      batch.sourcefiles.add(sourcefile)
    end
    workflow.batches.add(batch)
  end
end
end

```

[initialize_batch_workflow\(\)](#) click to toggle source

Initializes a batch workflow for topic model training.

Retrieves the input folder path from Redis, creates a new workflow, and creates batches of source files based on the specified batch size.

@return [void]

```

# File lib/tasks/workflow_initializer_task.rb, line 33
def initialize_batch_workflow
  input_folder = Ohm.redis.get("input_folder_path")
  all_file_paths = Dir.glob(File.join(input_folder,
  "**{,/*/**}/*.md,markdown}")).sort

  workflow = Workflow.create(
    name: "TopicModelTrainerWorkflow",
    status: "started",
    start_time: Time.now.to_s,
    current_batch_number: 1,
    is_batch_workflow: true
  )

  create_batches(workflow, all_file_paths)

  Ohm.redis.set("current_workflow_id", workflow.id)
  logger.info "Initialized Batch Workflow with ID: #{workflow.id}, Total Batches:
  #{workflow.batches.count}"
end

```

[initialize_single_file_workflow\(\)](#) click to toggle source

Initializes a single file workflow for text processing.

Retrieves the input file path from Redis, creates a new workflow, and associates the input file with the workflow.

@return [void]

```
# File lib/tasks/workflow_initializer_task.rb, line 57
def initialize_single_file_workflow
  input_file_path = Ohm.redis.get("input_file_path")

  workflow = Workflow.create(
    name: "TextProcessingWorkflow",
    status: "started",
    start_time: Time.now.to_s,
    is_batch_workflow: false
  )

  sourcefile = Sourcefile.find_or_create_by_path(input_file_path, workflow: workflow)
  workflow.sourcefiles.add(sourcefile)

  Ohm.redis.set("current_workflow_id", workflow.id)
  Ohm.redis.set("current_file_id", sourcefile.id)
  logger.info "Initialized Single File Workflow with ID: #{workflow.id}, File:
  #{sourcefile.path}"
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 90. Workfloworchestrator

90.1. 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**.

90.1.1. Constants

CARTRIDGE_BASE_DIR

The base directory for cartridges.

90.1.2. Public Class Methods

new() click to toggle source

Initializes a new **WorkflowOrchestrator** instance.

@return [void]

```
# File lib/components/WorkflowOrchestrator.rb, line 19
def initialize
  # A hash to store the registered agents, keyed by their roles.
  @agents = {}
end
```

90.1.3. Public Instance Methods

add_agent(role, cartridge_file, author: "@b08x") click to toggle source

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.

```
# File lib/components/WorkflowOrchestrator.rb, line 32
def add_agent(role, cartridge_file, author: "@b08x")
  logger.debug "Adding agent: #{role}"
  cartridge_path = File.join(CARTRIDGE_BASE_DIR, author, "cartridges", cartridge_file)
```

```

    raise "Cartridge file not found: \"#{cartridge_path}\"" unless
File.exist?(cartridge_path)

    @agents[role] = WorkflowAgent.new(role, cartridge_path)
    logger.debug "Agent added: #{role}"
end

```

`cleanup()` [click to toggle source](#)

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]

```

# File lib/components/WorkflowOrchestrator.rb, line 129
def cleanup
  # Perform any necessary cleanup for the workflow
  Jongleur::API.trap_quit_signals
  # Add any other cleanup operations here
end

```

`define_workflow(workflow_definition)` [click to toggle source](#)

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]

```

# File lib/components/WorkflowOrchestrator.rb, line 51
def define_workflow(workflow_definition)
  logger.debug "Defining workflow"
  logger.debug "Workflow definition: #{workflow_definition}"
  Jongleur::API.add_task_graph(workflow_definition)
  logger.debug "Workflow defined"
end

```

`run_workflow()` [click to toggle source](#)

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]

```
# File lib/components/WorkflowOrchestrator.rb, line 66
def run_workflow
  logger.info "Starting workflow execution"
  @running = true

  begin
    logger.debug "Printing graph to /tmp"
    Jongleur::API.print_graph("/tmp")

    UI.info "Starting Jongleur::API.run"
    Jongleur::API.run do |on|
      # Event handler for task start.
      on.start do |task|
        ui.framed do
          ui.puts "Starting task: #{task}"
        end
      end

      # Event handler for task finish.
      on.finish do |task|
        ui.framed do
          ui.puts "Finished task: #{task}"
        end
        ui.space
      end

      # Event handler for task errors.
      on.error do |task, error|
        logger.error "Error in task #{task}: #{error.message}"
        logger.error error.backtrace.join("\n")
      end

      # Event handler for workflow completion.
      on.completed do |task_matrix|
        puts "Workflow completed"
        logger.info "Task matrix: #{task_matrix}"
        @running = false
        begin
          return "next" if BATCH
        rescue StandardError
          logger.warn "Not a batch job"
          return "next"
        end
      end
    end
  end
end
```

```
rescue Interrupt
  logger.info "Workflow interrupted"
  UI.say(:warn, "Workflow interrupted. Cleaning up...")
  cleanup
rescue StandardError => e
  logger.error "Error during workflow execution: #{e.message}"
  logger.error e.backtrace.join("\n")
  cleanup
  raise
end
end
```

Validate

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 91. Workflows

91.1. class Flowbots::Workflows

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

91.1.1. Public Class Methods

`load_workflows()` [click to toggle source](#)

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]

```
# File lib/workflows.rb, line 50
def self.load_workflows
  workflows_to_load = {}
  user_workflow_dir = if IN_CONTAINER
                        "/data/app/lib/workflows"
                      else
                        File.expand_path("../workflows/custom", __dir__)
                      end

  Dir["#{File.join(WORKFLOW_DIR, '**')} + File::SEPARATOR}*.rb"].sort.each do |file|
    workflows_to_load[File.basename(file)] = file
  end

  if Dir.exist?(user_workflow_dir)
    Dir["#{File.join(user_workflow_dir, '**')} + File::SEPARATOR}*.rb"].sort.each do |file|
      workflows_to_load[File.basename(file)] = file
    end
  end

  workflows_to_load.each_value do |file|
    begin
      require file
    rescue StandardError => e
      UI.error "Unable to load workflow #{e.message}"
      logger.fatal "Unable to load workflow #{e.message}"
    end
    UI.info "Loaded workflow file: #{file}"
    logger.debug "Loaded workflow file: #{file}"
  end
end
```

`new()` [click to toggle source](#)

Initializes a new **Workflows** instance.

@return [void]

```
# File lib/workflows.rb, line 10
def initialize
  @prompt = TTY::Prompt.new
end
```

91.1.2. Public Instance Methods

`list_and_select()` [click to toggle source](#)

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.

```
# File lib/workflows.rb, line 17
def list_and_select
  workflows = get_workflows
  display_workflows(workflows)
  select_workflow(workflows)
end
```

`run(workflow_name)` [click to toggle source](#)

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.

```
# File lib/workflows.rb, line 29
def run(workflow_name)
  workflow_file = File.join(WORKFLOW_DIR, "#{workflow_name}.rb")

  unless File.exist?(workflow_file)
    logger.error "Workflow file not found: #{workflow_file}"
    raise FileNotFoundError, "Workflow file not found: #{workflow_file}"
  end

  UI.info "Running workflow: #{workflow_name}"

  workflow_class = workflow_name.split("_").map(&:capitalize).join
  workflow = Flowbots.const_get(workflow_class).new

  logger.debug workflow
  workflow.run
```

end

91.1.3. Private Instance Methods

`display_workflows(workflows)` [click to toggle source](#)

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]

```
# File lib/workflows.rb, line 99
def display_workflows(workflows)
  if workflows.empty?
    logger.warn "No workflows found in #{WORKFLOW_DIR}"
    return
  end

  table = TTY::Table.new(
    header: [UI::PASTEL.cyan("Workflow"), UI::PASTEL.cyan("Description")],
    rows: workflows
  )

  puts table.render(:unicode, padding: [0, 1])
end
```

`extract_workflow_description(file)` [click to toggle source](#)

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.

```
# File lib/workflows.rb, line 130
def extract_workflow_description(file)
  first_line = File.open(file, &:readline).strip
  first_line.start_with?("# Description:") ? first_line[14..-1] : "No description
available"
end
```

`get_workflows()` [click to toggle source](#)

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.

```
# File lib/workflows.rb, line 85
def get_workflows
  Dir.glob(File.join(WORKFLOW_DIR, "*.rb")).map do |file|
    name = File.basename(file, ".rb")
    description = extract_workflow_description(file)
    [name, description]
  end
end
```

select_workflow(workflows) click to toggle source

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.

```
# File lib/workflows.rb, line 119
def select_workflow(workflows)
  workflow_names = workflows.map { |w| w[0] }
  @prompt.select("Choose a workflow to run:", workflow_names)
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 92. Writefileaction

92.1. class Sublayer::Actions::WriteFileAction

92.1.1. Public Class Methods

`new(file_contents:, file_path:)` [click to toggle source](#)

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

```
# File lib/utils/writefile.rb, line 8
def initialize(file_contents:, file_path:)
  @file_contents = file_contents
  @file_path = file_path
end
```

92.1.2. Public Instance Methods

`call()` [click to toggle source](#)

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

```
# File lib/utils/writefile.rb, line 15
def call
  File.open(@file_path, 'wb') do |file|
    file.write(@file_contents)
  end
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 93. Yamlfrontmatter0

93.1. module MarkdownYaml::YamlFrontMatter0

The YamlFrontMatter node represents the YAML front matter section.

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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

Chapter 94. Yamlfrontmatter1

94.1. module MarkdownYaml::YamlFrontMatter1

The YamlFrontMatter node represents the YAML front matter section.

94.1.1. Public Instance Methods

newline1() [click to toggle source](#)

The first newline character after the “—” delimiter.

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

```
# File lib/grammars/markdown_yaml.rb, line 75
def newline1
  elements[1]
end
```

newline2() [click to toggle source](#)

The second newline character after the “—” delimiter.

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

```
# File lib/grammars/markdown_yaml.rb, line 82
def newline2
  elements[4]
end
```

[Validate](#)

Generated by [RDoc](#) 6.6.3.1.

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