Software Design

for

NifflerTM: A Command line Turing Machine

Author:	Devan Farrell
Semester:	Spring 2017
Course:	CPT_S 322 Software Engineering

May 1st 2017

Completed Design

List of Figures

Revision History

Revision Date	Description	Rational
1/14/2017	Design as of initial release	Initial release
5/1/2017	updates	Text edits and diagram design

1. Introduction

1.1 Purpose and Use

The purpose of this document is for use in the development and maintenance of the NifflerTM source code.

1.2 Intended Audience

The primary user of this document is the developers and maintainers of the software. NifflerTM was designed for academic purposes, it will not be publicly available but will be hosted on GitHub for use in a programming portfolio.

1.3 Document Layout

The A**rchitecture** section of the document displays the relationships in the object-oriented software design and diagrams of each class in the design. NifflerTM makes use of the three-tier design pattern and includes frames to help illustrate this.

The D**ata Dictionary** section of the document describes each class in detail and includes details on their purpose, lifetime, associations, attributes, method operations, and pseudocode of complex methods.

The **User Interface** section of the document shows examples of the input and output displayed to the user while operating each command and running the application.

The **Files** section of the document shows examples of proper formatting of the definition and input files as well as erroneous versions of them.

2. Architecture

Figure 2-1 Three-Tier Relationship Diagram

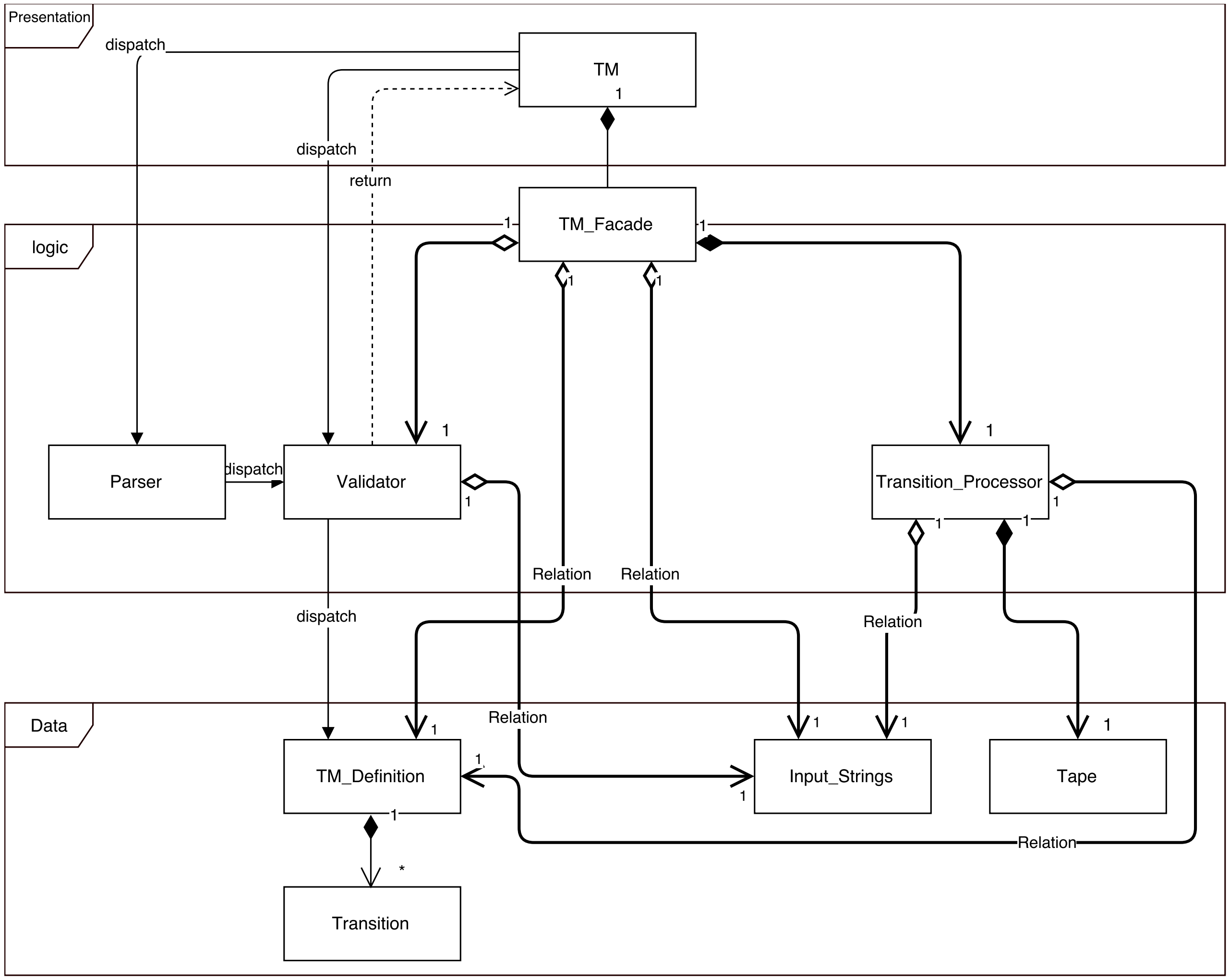
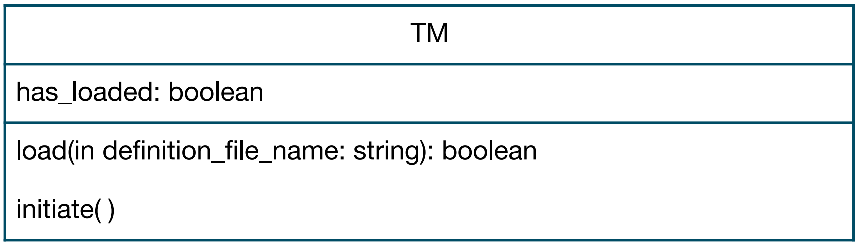
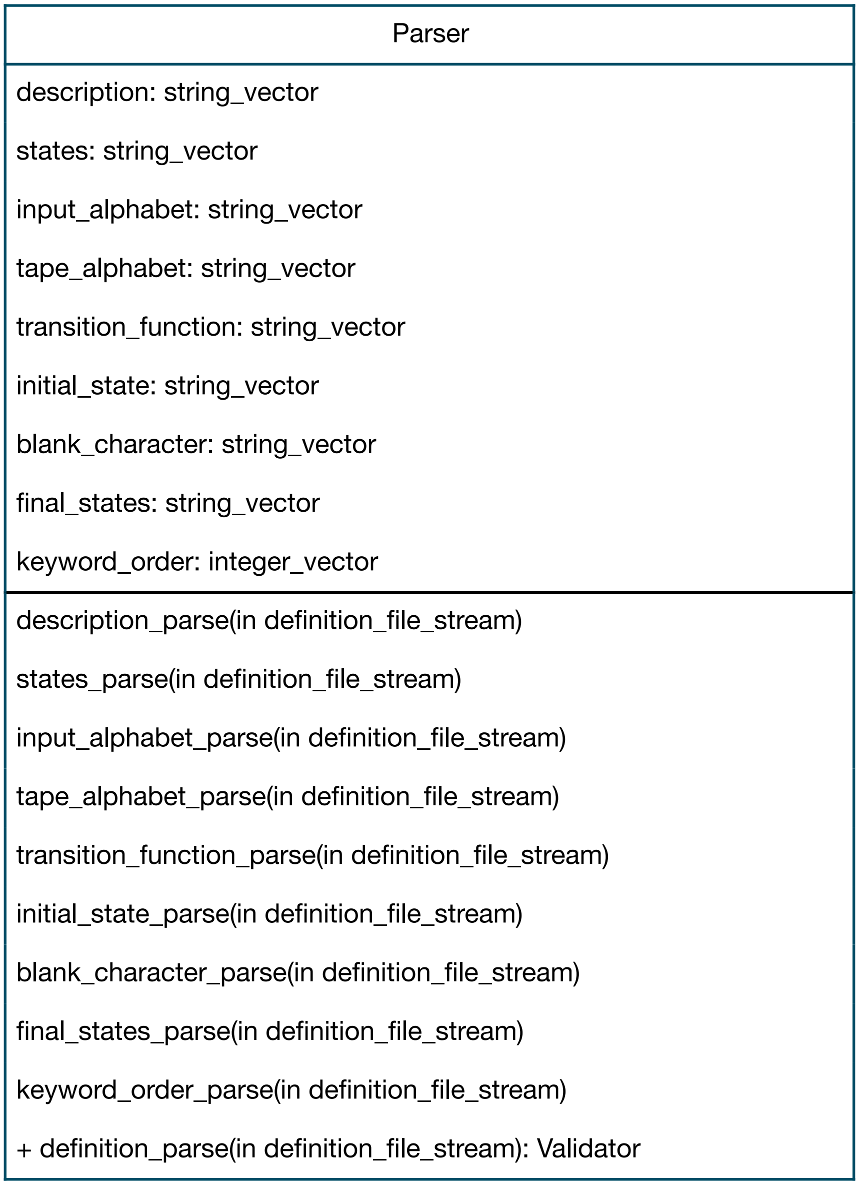


Figure 2-2 TM Class Diagram



TM
has_loaded: boolean
load(in definition_file_name: string): boolean initiate( )

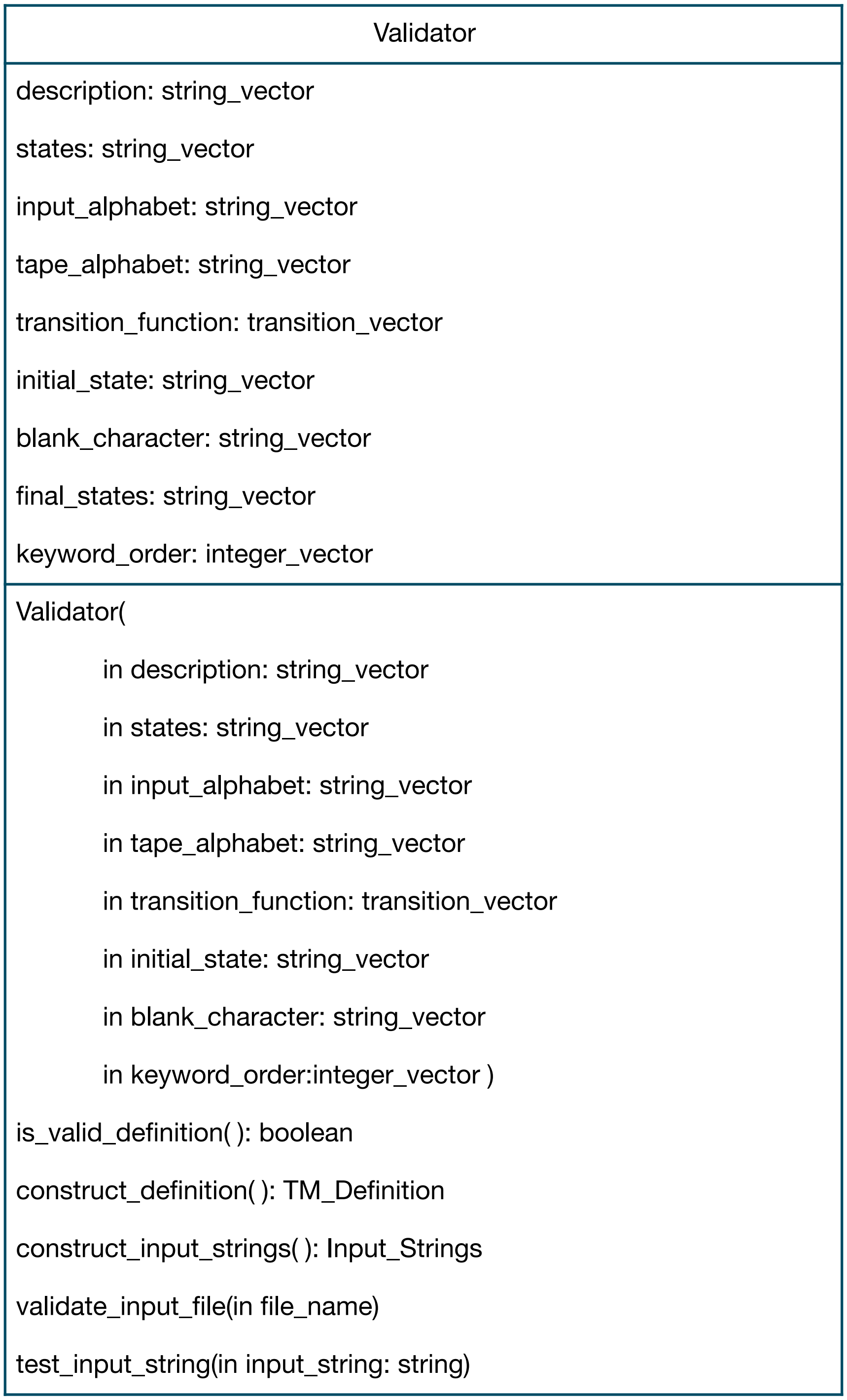
Figure 2-3 Parser Class Diagram



Parser
description: string_vector states: string_vector input_alphabet: string_vector tape_alphabet: string_vector transition_function: string_vector initial_state: string_vector blank_character: string_vector final_states: string_vector keyword_order: integer_vector

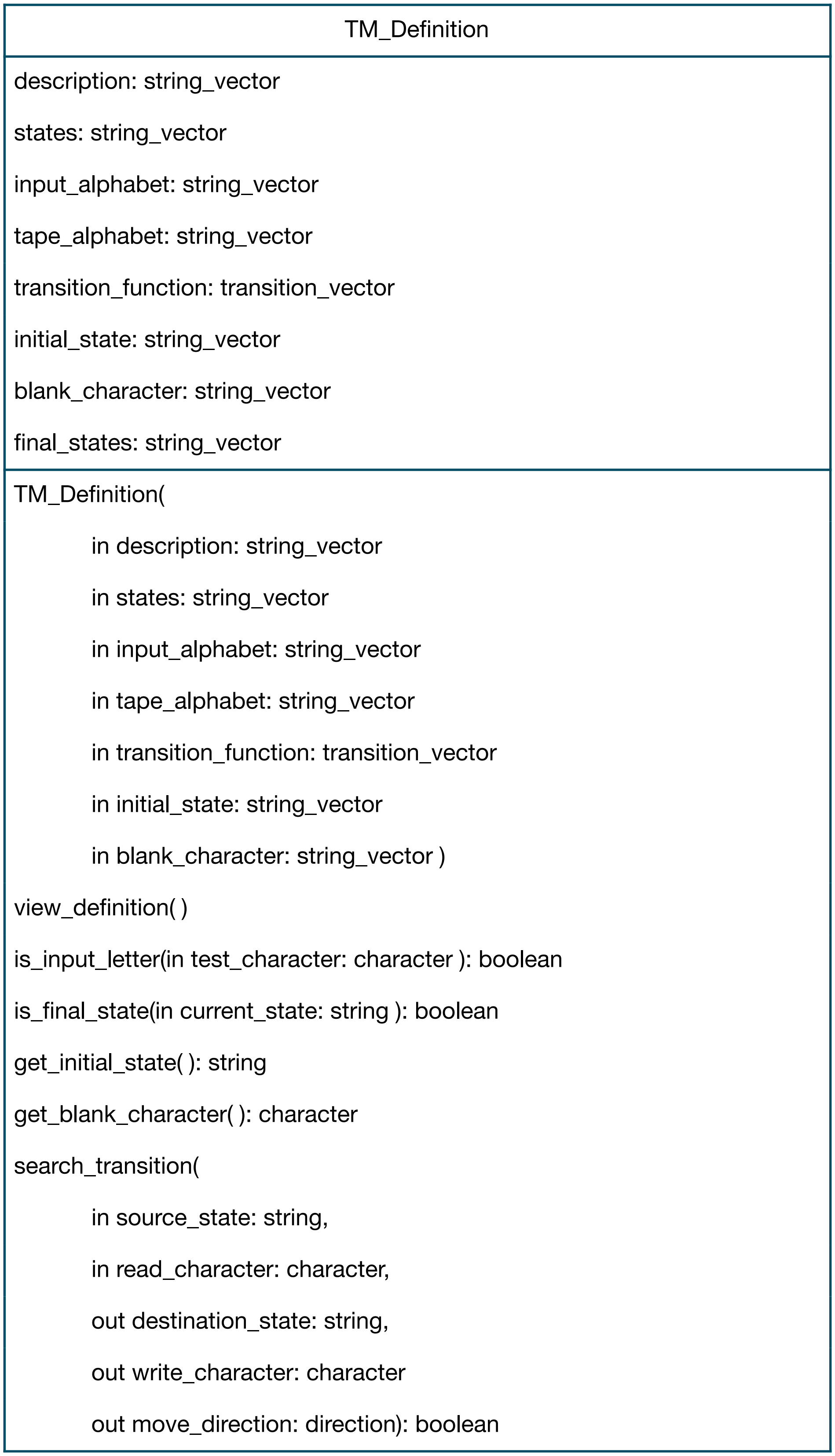
description_parse(in definition_file_stream)
states_parse(in definition_file_stream)
input_alphabet_parse(in definition_file_stream)
tape_alphabet_parse(in definition_file_stream)
transition_function_parse(in definition_file_stream)
initial_state_parse(in definition_file_stream)
blank_character_parse(in definition_file_stream)
final_states_parse(in definition_file_stream)
keyword_order_parse(in definition_file_stream)
+ definition_parse(in definition_file_stream): Validator

Figure 2-4 Validator Class Diagram



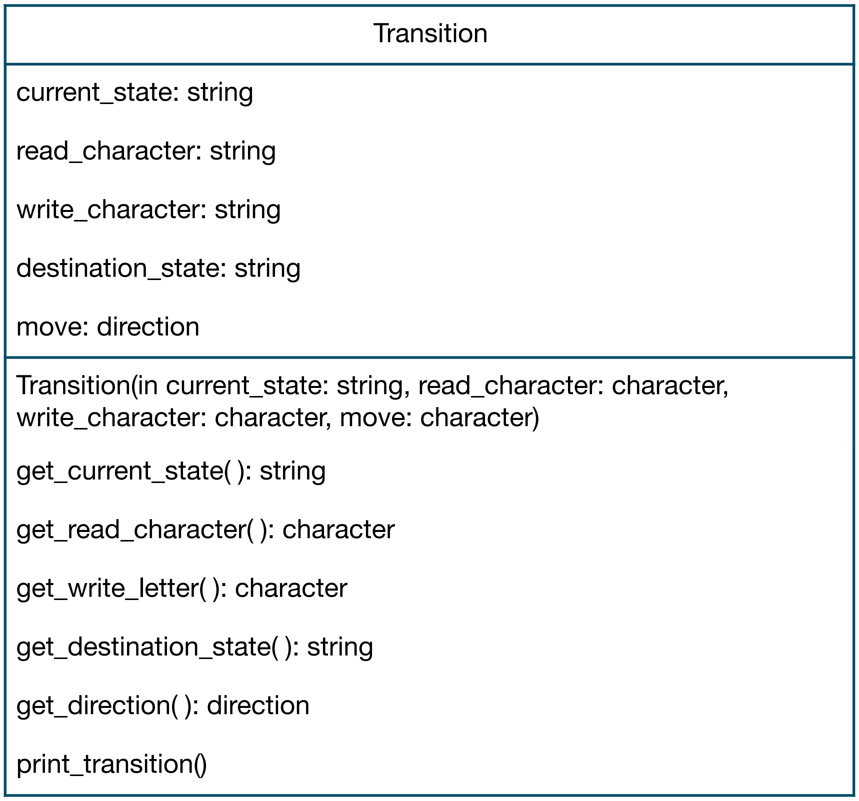
```
Validator
---
description: string_vector
states: string_vector
input_alphabet: string_vector
tape_alphabet: string_vector
transition_function: transition_vector
initial_state: string_vector
blank_character: string_vector
final_states: string_vector
keyword_order: integer_vector
---
Validator(
        in description: string_vector
        in states: string_vector
        in input_alphabet: string_vector
        in tape_alphabet: string_vector
        in transition_function: transition_vector
        in initial_state: string_vector
        in blank_character: string_vector
        in keyword_order:integer_vector )
is_valid_definition( ): boolean
construct_definition( ): TM_Definition
construct_input_strings( ): Input_Strings
validate_input_file(in file_name)
test_input_string(in input_string: string)
```

Figure 2-5 TM_Definition Class Diagram



```
TM_Definition
-------------------------------------------------
description: string_vector
states: string_vector
input_alphabet: string_vector
tape_alphabet: string_vector
transition_function: transition_vector
initial_state: string_vector
blank_character: string_vector
final_states: string_vector
-------------------------------------------------
TM_Definition(
        in description: string_vector
        in states: string_vector
        in input_alphabet: string_vector
        in tape_alphabet: string_vector
        in transition_function: transition_vector
        in initial_state: string_vector
        in blank_character: string_vector )
view_definition( )
is_input_letter(in test_character: character ): boolean
is_final_state(in current_state: string ): boolean
get_initial_state( ): string
get_blank_character( ): character
search_transition(
        in source_state: string,
        in read_character: character,
        out destination_state: string,
        out write_character: character
        out move_direction: direction): boolean
```

Figure 2-6 Transition Class Diagram



Transition

current_state: string

read_character: string

write_character: string

destination_state: string

move: direction

Transition(in current_state: string, read_character: character, write_character: character, move: character)

get_current_state( ): string

get_read_character( ): character

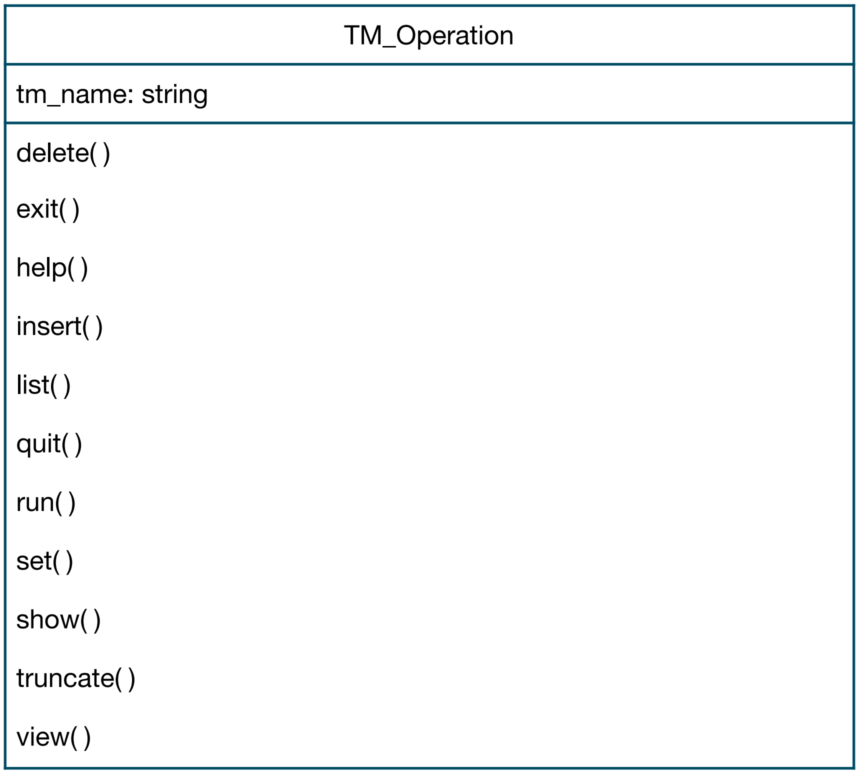
get_write_letter( ): character

get_destination_state( ): string

get_direction( ): direction

print_transition()

Figure 2-7 TM_Facade Class Diagram



TM_Operation

tm_name: string

delete( )

exit( )

help( )

insert( )

list( )

quit( )

run( )

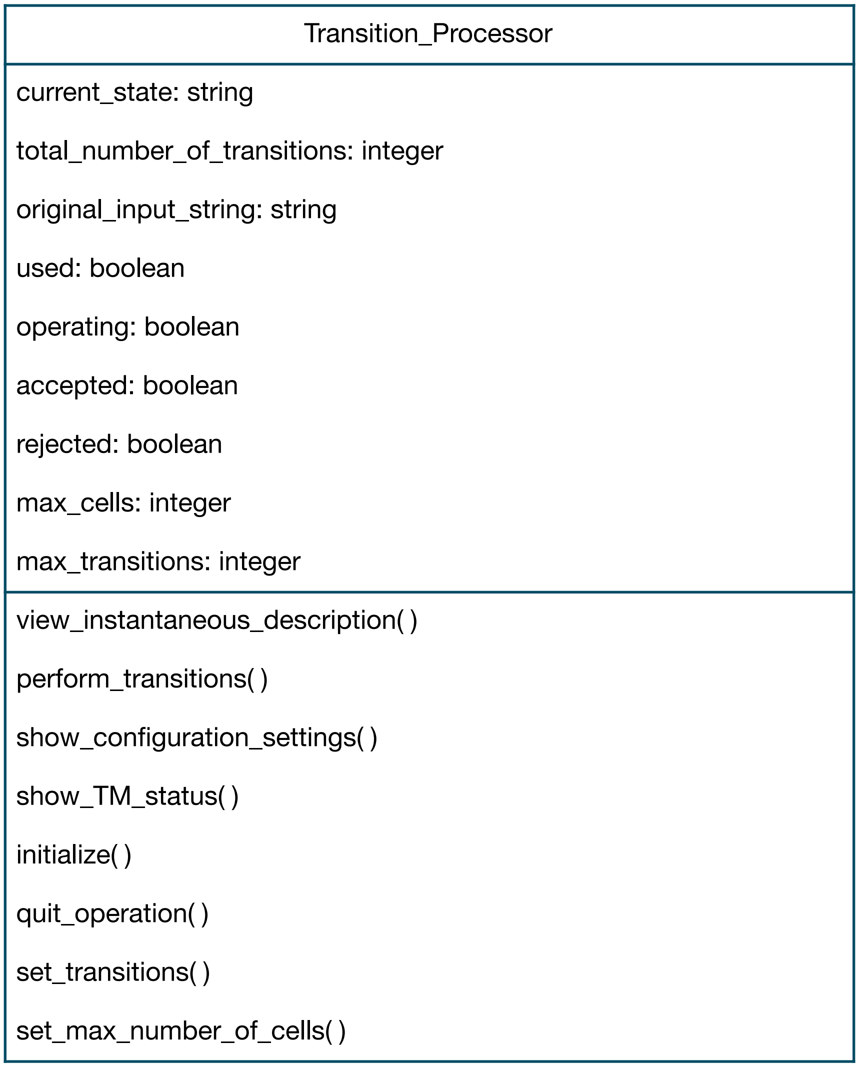
set( )

show( )

truncate( )

view( )

Figure 2-8 Transition_Processor Class Diagram



Transition_Processor

current_state: string

total_number_of_transitions: integer

original_input_string: string

used: boolean

operating: boolean

accepted: boolean

rejected: boolean

max_cells: integer

max_transitions: integer

view_instantaneous_description( )

perform_transitions( )

show_configuration_settings( )

show_TM_status( )

initialize( )

quit_operation( )

set_transitions( )

set_max_number_of_cells( )

Figure 2-9 Tape Class Diagram

Tape
cells: string = “ “
current_cell: integer = 0
initialize(in input_string:string)
update(in write_character:character, in move_direction:direction)
read_character( ):character
is_first_cell( ): boolean
is_last_cell( ): boolean
append_blank(in blank_character: character)
left(in maximum_number_of_cells:integer):string
right(in maximum_number_of_cells:integer):string

Figure 2-10 Input_Strings Class Diagram

Input_Strings
input_strings: string_vector
append_string(in input_string: string)
delete_string(in index: integer): boolean
list_strings( )
is_duplicate(in input_string: string): boolean
is_string(in index: integer): boolean
get_string(in index: integer): string

3. Data Dictionary

3.1 TM

3.1.1 Description

The TM class is the presentation layer of the software. It is instantiated once and the lifetime of the TM object is the duration of the application. With it, the user is able to load the Turing machine definition file and input string file and initiate operation on the TM.

3.1.2 Associations

During the course of the load method it instantiates and is able to send messages to a Parser object and the Validator object. The rest of the application lifetime the TM class is composed of the class TM_Operation.

3.1.3 Attributes

has_loaded: boolean

has_loaded by default is false and is made true when the load function is called so that another TM can't be loaded from the same instance of the TM class.

3.1.4 Methods

load(in definition_file_name: string): boolean

The **load** method attempts to open the definition file and sends the opened file stream to the parser class. It then sends a message to the validator class to validate the file. If the file is valid it will attempt to open the input string file and send its contents to the validator class. It will also construct the rest of the classes and associate them. This method can only be called once per instance of the class

initiate( )

The **initiate** method begins operation of the TM by initiating the command prompt.

3.2 Parser

3.2.1 Description

The Parser class is a member of the logic layer of the application. It is instantiated only once and only temporarily. Its lifetime is the duration of the TM load method. I parses the TM definition file and sends the parsed definition to the Validator object.

3.2.2 Associations

The Parser is instantiated and sent the file to parse by the TM object. Parser instantiates the Validator class and fills it with all the attributes within the Parser object.

3.2.3 Attributes

description: string_vector

The attribute **description** is used to store the description of the TM at the top of the definition file, maintaining all original formatting.

states: string_vector

The attribute **states** is used to store the states of the TM from the definition file.

input_alphabet

The attribute **input_alphabet** is used to store the input alphabet of the TM from the definition file.

tape_alphabet: string_vector

The attribute **tape_alphabet** is used to store the tape alphabet of the TM from the definition file.

transition_function: string_vector

The attribute **transition_function** is used to store every group of characters in the TM the definition file for later processing.

initial_state: string_vector

The attribute initial**_state** is used to store the initial state of the TM from the definition file.

blank_character: string_vector

The attribute **blank_character** is used to store the blank character of the TM from the definition file.

final_states: string_vector

The attribute **final_states** is used to store the final states of the TM from the definition file.

keyword_order: integer vector

The attribute **keyword_order** is used to store the order the keywords appear in the definition file.

3.2.4 Methods

definition_parse(in definition_file_stream): Validator

The **definition_parse** method calls all other methods of class and then instantiates Validator with all of the parser's now loaded attributes

description_parse(in defintion_file_stream)

The **description_parse** method parses the description of the TM definition file

states_parse(in definition_file_stream)

The **states_parse** method parses the states of the TM definition file

input_alphabet_parse(in definition_file_stream)

The **input_alphabet_parse** method parses the input alphabet of the TM definition file

tape_alphabet_parse(in definition_file_stream)

The **tape_alphabet_parse** method parses the tape alphabet of the TM definition file

transition_function_parse(in definition_file_stream)

The **transition_function_parse** method parses each word in the transition_function section of the input file and saves them for later processing in validation.

initial_state_parse(in definition_file_stream)

The **initial_state_parse** method parses the initial state of the TM definition file. Uses a string_vector for better error checking in the validator.

blank_character_parse(in definition_file_stream)

The **blank_character_parse** method parses the blank character of the TM definition file. Uses a string_vector for better error checking in the validator.

final_states_parse(in definition_file_stream)

The **final_states_parse** method parses the final states of the TM definition file.

keyword_order_parse(in definition_file_stream)

The **keyword_order_parse** method parses the entire definition file to make sure every section is present, is in the correct order, and there are no duplicates. It appends an integer representing the order a given keyword should be placed each time a keyword is found.

3.3 Validator

3.3.1 Description

The Validator class is a member of the logic layer of the application. It is instantiated only once and its lifetime is the duration of the application. The purpose of the object is to validate the TM definition.

3.3.2 Associations

The object is instantiated from the Parser object. For the duration of the TM load method, Validator object is able to receive messages from the TM object, send a message to TM_Definition, and send messages to Input_Strings. After the load method, it is an aggregation of TM_Operation object and aggregates the Input_Strings object.

3.3.3 Attributes

description: string_vector

The attribute **description** is used to store the description of the TM at the top of the definition file, maintaining all original formatting.

states: string_vector

The attribute **states** is used to validate the states of the TM.

input_alphabet

The attribute **input_alphabet** is used to validate the input alphabet of the TM.

tape_alphabet: string_vector

The attribute **tape_alphabet** is used to validate the tape alphabet of the TM.

transition_function_strings: strings_vector

The attribute **transition_function_strings** is used to validate the transitions parsed in TM definition file and construct validated Transition objects.

transition_function: Transition_vector

The attribute **transition_function** is used to store validated transitions.

initial_state: string_vector

The attribute **initial_state** is used to validate the initial state of the TM.

blank_character: string_vector

The attribute **blank_character** is used to validate the blank character of the TM.

final_states: string_vector

The attribute **final_states** is used to validate the final states of the TM.

keyword_order: integer vector

The attribute **keyword_order** is used to validate the order, existence, and duplicity that the keywords appear in the definition file.

3.3.4 Methods

```
Validator(   in description: string_vector,
             in states: string_vector,
             in input_alphabet: string_vector,
             in tape_alphabet: string_vector,
             in transition_function: string_vector,
             in initial_state: string_vector,
             in blank_character: string_vector,
             in keyword_order: integer_vector )
```

The **Validator** constructor is used to initialize all attributes of the object. There is no other way to set the variables outside of the constructor.

is_valid_definition( ): boolean

The **is_valid_definition** method tests the definition for possible flaws that would render it erroneous.Tests other than transition tests include: states contain all legal characters, no duplication of a state, no illegal characters in input alphabet, no duplication of letters in input alphabet, no illegal characters in tape alphabet, no duplication of letters in tape alphabet, all characters in input alphabet are in tape alphabet, there is exactly one initial state, the initial state is in states, there is exactly one blank character, blank character not in the input alphabet, blank character is in tape alphabet, no duplicates in final states, all final states are in states, keywords are in proper order and are not duplicated. Transition tests include: the current state of each transition is in states, the read letter of each is in the tape alphabet, the write letter of each is in the tape alphabet, the destination state of each is in states, the move direction is a valid direction, no two transitions contain the read letter and current state (no nondeterministic definition), and no current states are final states (cannot transition out of a final state)

construct_definition( ): TM_Definition

The **construct_definition** method instantiates the TM_Definition class with all the attributes of the Validator objects after they have been validated. It then returns the TM_Definition object to be associated with the appropriate objects.

construct_input_strings( ): Input_Strings

The **construct_input_strings** method instantiates the Input_Strings class and returns the Input_Strings object to be associated with the appropriate objects.

validate_input_file(in file_name)

The **validate_input_file** method attempts to open the input string file then calls the test_input_string method on each line to input the legal ones into the Input_Strings object.

test_input_string(in input_string: string)

The **test_input_string** procedure validates that every character in the input sting is one of the input alphabet letters.

3.4 TM_Definition

3.4.1 Description

The TM_Definition class is a member of the data layer of the application. It is instantiated only once and its lifetime is duration of the application once it has been instantiated by the Validator object. The purpose of the object is to store and encapsulate the TM definition.

3.4.2 Associations

The object is instantiated from the Validator object but after instantiation it no longer can receive any messages from the Validator object. For the duration of the application the object is an aggregates the TM_Operation object and the Transition_Processor object

3.4.3 Attributes

description: string_vector

The attribute **description** is used to store the description of the TM at the top of the definition file, maintaining all original formatting.

states: string_vector

The attribute **states** is used to store the states of the TM from the definition file.

input_alphabet

The attribute **input_alphabet** is used to store the input alphabet of the TM from the definition file.

tape_alphabet: string_vector

The attribute **tape_alphabet** is used to store the tape alphabet of the TM from the definition file.

transition_function: Transition_vector

The attribute **transition_function** is used to store the transition function objects of the TM from the definition file.

initial_state: string_vector

The attribute initial**_state** is used to store the initial state of the TM from the definition file.

blank_character: string_vector

The attribute **blank_character** is used to store the blank character of the TM from the definition file.

final_states: string_vector

The attribute **final_states** is used to store the final states of the TM from the definition file.

3.4.4 Methods

TM_Definition(in description: string_vector,
in states: string_vector,
in input_alphabet: string_vector,
in tape_alphabet: string_vector,
in transition_function: transition_vector,
in initial_state: string_vector,
in blank_character: string_vector)

The **TM_Definition** constructor is used to initialize all attributes of the object. There is no other way to set the variables outside of the constructor.

view_definition( )

The **view_definition** procedure displays a formatted definition of the Turing machine to the monitor.

is_input_letter(in test_character: character): boolean

The **is_input_letter** method checks if a given character is in the input alphabet.

is_final_state(in current_state: string): boolean

The **is_final_state** method checks if a given state is included in the final states.

get_initial_state( ): string

The **get_initial_state** method returns the initial state of the definition.

get_blank_character( ): character

The **get_blank_character** method returns the blank character of the definition.

```
search_transition(in source_state:string,
                  in read_character:character,
                  out destination_state:string,
                  out write_character: character,
                  out move_direction:direction): boolean
```

The **search_transition** method searches for a transition that match the read character and current state and returns a boolean indicating success and returns by reference the destination state, write character and move direction .

3.5 Transition

3.5.1 Description

The Transition class is a member of the data layer of the application. It is instantiated as many as there are transition functions in the definition and its lifetime is duration of the application once it has been instantiated. The purpose of the object is to store and encapsulate each TM transition function.

3.5.2 Associations

Transition objects are a component of the TM_Definition object

3.5.3 Attributes

current_state: string

The attribute **current_state** is used to store the current state of a transition from the definition file.

read_character: character

The attribute **read_character** is used to store the read character of a transition from the definition file.

write_character: character

The attribute **write_character** is used to store the write character of a transition from the definition file.

destination_state: string

The attribute **destination_state** is used to store the destination state of a transition from the definition file.

move: character

The attribute **move** is used to store the direction of a transition from the definition file.

3.5.4 Methods

Transition(in current_state: string, read_character: character, write_character: character, move: character)

The **Transition** constructor is used to initialize all attributes of the object. There is no other way to set the variables outside of the constructor.

get_current_state( ): string

The **get_current_state** method returns the current state of a transition.

get_read_character( ): character

The **get_read_character** method returns the read character of a transition.

get_write_character( ): character

The **get_write_character** method returns the write character of a transition.

get_destination_state( ): string

The **get_destination_state** method returns the destination state of a transition.

get_direction( ): direction

The **get_direction** method returns the direction of a transition.

print_transition( )

The **print_transition** method prints the transition formatted.

3.6 TM_Facade

3.6.1 Description

The TM_Facade class is the primary member of the logic layer of the application. It is instantiated only once and its lifetime is duration of the application once it has been instantiated by the TM object. The purpose of the TM_Facade object is to manage the essential eleven commands of the application by maintaining them and delegating tasks to the other application objects once they have been called.

3.6.2 Associations

The TM_Operation object is composed of the Transition_Processor object and aggregates the Validator, TM_Definition, and Input_Strings objects. It is also a component of the TM object.

3.6.3 Attributes

tm_name: string

The **tm_name** attribute maintains the name of the Turing machine for the show method.

3.6.4 Methods

delete( )

The **delete** procedure allows user to delete an input string from the input string list.

exit( )

The **exit** procedure begins termination of the application, frees up dynamically allocated memory and returns the user the the shell command line.

help( )

The **help** procedure displays the list of possible commands to the user and a very brief description of each purpose.

insert( )

The **insert** procedure prompts the user for a string then sends to the string to the Validator object via the test_input_string message which relays the validated string to the Input_Strings object.

list( )

The **list** procedure sends a message to the Input_Strings object via the list_strings method which displays the list of valid input strings that are currently being stored.

quit( )

The **quit** procedure sends a message to the Transition_Processor object to stop operating on an input string via the quit_operation procedure.

run( )

The **run** procedure has two functions.

If the Transition_Processor object is currently operating on a string then it sends a message to it to perform a transition(s).

If it is not currently operating on a string then it prompts the user for an index of an input string currently stored in the Input_Strings object. Once selected the string is sent from Input_Strings and loaded into the Tape object.

set( )

The **set** procedure sends a message to the Transition_Processor object to prompt and set a new number of transitions that will be performed each time the run command is called via the set_transition procedure.

show( )

The **show** procedure will display details related to the the application itself, the configuration settings, and status of the application. While doing so it will message the Transition_Processor via the show_tm_status procedure to display the appropriate information.

truncate( )

The **truncate** procedure sends a message to the Transition_Processor object to prompt and set a new number of cells to be displayed to the left and right of the tape head in the instantaneous description via the set_max_number_of_cells procedure.

view( )

The **view** procedure sends a message to the TM_Definition object via the view_definition procedure which displays a formatted definition of the Turing machine to the monitor.

3.7 Transition_Processor

3.7.1 Description

The Transition_Processor class is a member of the logic layer of the application. It is instantiated only once and its lifetime is duration of the application once it has been instantiated by the TM_Operation object. The purpose of the Transition_Processor object is handle all the logic required for the run command.

3.7.2 Associations

The Transition_Processor is composed of the Tape object and is a component of the TM_Operation object. It also aggregates the TM_Definition and the Input_Strings objects.

3.7.3 Attributes

current_state: string

The **current_state** attribute stores the current state for proper TM operation.

total_number_of_transitions: integer

The **total_number_of_transitions** attribute maintains the number of transitions that have been performed on an input strings up to this point.

original_input_string: string

The **original_input_string** attribute maintains the original input string that the Turing machine is or was operating on.

used: boolean

The **used** attribute keep track of whether or not the Turing machine has been used since the application start up.

operating: boolean

The **operating** attribute keeps track if the TM is currently operating on an input string or not.

accepted: boolean

The **accepted** attribute keeps track if the TM has accepted the current input string.

rejected: boolean

The **rejected** attribute keeps track if the TM has rejected the current input string.

max_cells: integer

The **max_number_of_cells** attribute keeps track of the maximum number of cells to the left and right of the tape head to display in the instantaneous description.

max_transitions: integer

The **transitions_per_run** attribute keeps track of the maximum number of transitions that can be performed for each time the run command is called.

3.7.4 Methods

view_instantaneous_description( )

The **view_ instantaneous_description** procedure displays the current number of transitions with a period then sends a message to the Tape object to display everything to the left of the tape head. Then it displays the current state in brackets then sends a message to the Tape object again to display everything to the right of the tape head.

perform_transitions( )

The **perform_transitions** procedure begins a loop of searching for a transition from the TM_Definition object then sending a message to update the Tape object. The loop is completed after the number of times specified by the transitions_per_run attribute or when the string is accepted or rejected, whichever comes first.

show_configuration_settings( )

The **show_configuration_settings** procedure displays the maximum number of transitions and the maximum number cells to display to the left and right of the tape head.

show_TM_status( )

The **show_TM_status** procedure displays whether or not the TM has completed operation on an input string. It also displays if it was accepted or rejected and how many transitions have been performed on the input string.

initialize(in input_string_index: integer)

The **quit_operation** procedure sets the operating attribute to false. If it is already false it displays an error.

quit_operation( )

The **quit_operation** procedure sets the operating attribute to false. If it is already false it displays an error.

set_transitions( )

The **set_transitions** procedure prompts for a new maximum number of transitions that can be performed for each time the run command is called. It also displays the current number of transitions to perform in the prompt.

set_max_number_of_cells( )

The **set_max_number_of_cells** procedure prompts for a new maximum number of cells to display to the left and right of the tape head. It also displays the current number of cells to display in the prompt.

3.8 Tape

3.8.1 Description

The Tape class is a member of the data layer of the application. It is instantiated only once and its lifetime is duration of the application once it has been instantiated by the Transition_Processor object. The tape of a Turing machine consists of an ordered sequence of cells that are 0-indexed.

3.8.2 Associations

Tape is a component of the Transition_Processor object and can only receive messages from it.

3.8.3 Attributes

cells: string = " "

The **cell** attribute is a dynamically growing string that is updated through the update method. It also can have the blank character appended to it if the current cell is past the length of the input string.

current_cell: integer = 0

The **current_cell** attribute maintains the location of the tape head on the tape.

3.8.4 Methods

initialize(in input_string:string)

The **initialize** procedure sets the cells attribute to a new input string and sets the current_cell back to 0.

update(in write_character:character, in move_direction:direction)

The **update** procedure writes the write character to the tape at the current tape cell and increments the current_cell attribute by one depending on the direction.

read_character( ):character

The **read_character** method returns the character from the cells attribute from the current cell.

is_first_cell( ):boolean

The **is_first_cell** method returns whether or not the current_cell attribute is 0;

is_last_cell( ):boolean

The **is_last_cell** method returns whether or not the current_cell attribute is at the end of the input string.

append_blank(in blank_character: character)

The **append_blank** procedure adds a blank character at the end of the string if the tape head is going to move past the end of the cells attribute.

left(in maximum_number_of_cells: integer):string

The **left** method returns the characters to the left of the tape head up to the maximum number of cells. If the number of cells to the left of the tape head exceed the maximum a “<“ to added to the beginning of the string and maximum minus one characters are returned with it.

right(in maximum_number_of_cells: integer):string

The **right** method returns the characters to the right of the tape head up to the maximum number of cells. If the number of cells to the right of the tape head exceed the maximum a “>“ to added to the end of the string and maximum minus one characters are returned with it.

3.9 Input_Strings

3.9.1 Description

The Input_Strings class is a member of the data layer of the application. It is instantiated only once and its lifetime is duration of the application once it has been instantiated by the Validator object. The purpose of the object is to store and encapsulate the Input_Strings of the TM so they can be sent to the Transition_Processor object.

3.9.2 Associations

The input_strings object is an aggregation of the TM_Facade, the Transition_Processor, and the Validator objects.

3.9.3 Attributes

input_strings: string_vector

The **input_strings** attribute holds a list of strings to be manipulated by the Transition_Processor object.

3.9.4 Methods

append_string(in input_string: string)

The **append_string** procedure appends a validated string to the input_strings attribute.

delete_string(in index: integer): boolean

The **delete_string** method removes a string from the input_strings attribute based on an index and returns based on success or failure.

list_strings( )

The **list_strings** procedure lists even string in the input_strings attribute with the appropriate index.

is_duplicate(in input_string: string): boolean

The **is_duplicate** method tests if a new string is a duplicate before it is appended to the input_strings.

is_string(in index: integer): boolean

The **is_string** method tests if there is a string at the given index and returns success or failure.

get_string(in index: integer): string

The **get_string** method returns a string from a given index.

4. User Interface

4.1 Command Line Invocation

Figure 4-1-1 Application Execution: Success

User wants to input myFirstTM.def and myFirstTM.str

```
./TM myFirstTM
myFirstTM loaded successfully!
Command:
```

Figure 4-1-2 Application Execution: Failure

User wants to input myBrokenTM.def and myBrokenTM.str

```
./TM myBrokenTM
Error: Initial state not listed in states
Error: Final_States: keyword not included in file
```

4.2 Help Command

Figure 4-2 Help Example Display

```
Command: h
(D)ELETE:       Delete input string from list
E(X)IT:         Exit application
(H)ELP:         Displays Commands
(I)NSERT:       Insert input string into list
(L)IST          List input strings
(Q)UIT          Quit operation of TM on input string
(R)UN:          Run TM on input string
S(E)T           Set maximum number of transitions to perform
SHO(W)          Show status of application
(T)RUNCATE:     Set truncation length of instantaneous descriptions
(V)IEW:         View TM
Command:
```

4.3 Show Command

Figure 4-3-1 Show: Has Not Operated

```
Command: w
COURSE: Computer Science 322: Software Engineering
SEMESTER: Spring
YEAR: 2017
INSTRUCTOR: Neil Corrigan
AUTHOR: Devan Farrell
VERSION: 0.0.1
CONFIGURATION SETTINGS:
        Max number of transitions: 1
        Max number of cells to the left and right of the tape head: 32
TM name: ANBR
        Status of TM:
        TM has not operated on an input string
Command:
```

Figure 4-3-2 Show: Currently Operating

```
Command: w
COURSE: Computer Science 322: Software Engineering
SEMESTER: Spring
YEAR: 2017
INSTRUCTOR: Neil Corrigan
AUTHOR: Devan Farrell
VERSION: 0.0.1
CONFIGURATION SETTINGS:
        Max number of transitions: 1
        Max number of cells to the left and right of the tape head: 32
TM name: ANBR
Status of TM:
        TM is currently running on an input string
        Input string ‘aab’ has undergone 15 transitions
Command:
```

Figure 4-3-3 Show: Completed Operation

```
Command: w
COURSE: Computer Science 322: Software Engineering
SEMESTER: Spring
YEAR: 2017
INSTRUCTOR: Neil Corrigan
AUTHOR: Devan Farrell
VERSION: 0.0.1
CONFIGURATION SETTINGS:
        Max number of transitions: 1
        Max number of cells to the left and right of the tape head: 32
TM name: ANBR
Status of TM:
        TM has completed operation on an input string
        Input string ‘aab’ was accepted in 15 transitions
Command:
```

4.4 View Command

Figure 4-4 View Example Display

```
Command: v

This is a description

M           =       { Q, Σ, Γ, δ, $q_0$, B, F }

Q           =       { s0, s1, s2, s3, s4 }

Σ           =       { a, b }

Γ           =       { a, b, Y , X, - }


δ(s0, a)    =       { s1, X, R }

δ(s0, Y)    =       { s3, Y, R }

δ(s1, a)    =       { s1, a, R }


$q_0$          =       s0

B           =       -

F           =       { s4 }

Command:
```

4.5 List Command

Figure 4-5 List Example Display

```
Command: l

[1] aaa

[2] bbb

[3]

[4] aaab
```

4.6 Insert Command

Figure 4-6-1 Insert: Success

```
Command: i

Input string: aab

Input string added to list!

Command:
```

Figure 4-6-2 Insert: Empty String

```
Command: i

Input string: \

Input string added to list!

Command:
```

Figure 4-6-3 Insert: Duplicate String

```
Command: i

Input string: aab

Error: String is a duplicate

Command:
```

Figure 4-6-4 Insert: String Contains Invalid Characters

```
Command: i

Input string: nicgriwgoskg3456y

Error: input string contains characters not in sigma

Command:
```

4.7 Delete Command

Figure 4-7-1 Delete: Successful

```
Command: d

Input string number: 1

Input string deleted!

Command:
```

Figure 4-7-2 Delete: Failed

```
Command: d

Input string number: 10000

Error: string not in list!

Command:
```

Figure 4-7-3 Delete: Not a positive integer

```
Command: d

Input string number: aab

Error: input was not a positive integer

Command:
```

4.8 Set Command

Figure 4-8-1 Set: Successful

```
Command: e

Set maximum number of transitions[1]: 10

Maximum set to 50!

Command:
```

Figure 4-8-2 Set: Failure

```
Command: e

Set maximum number of transitions[10]: -10

Error: input was not a positive integer

Command:
```

4.9 Truncate Command

Figure 4-9-1 Truncate: Successful

```
Command: t

Set maximum number of cells[32]: 50

Maximum set to 50!

Command:
```

Figure 4-9-2 Truncate: Failure

```
Command: e

Set maximum number of cells[50]: -10

Error: input was not a positive integer

Command:
```

4.10 Run Command

Figure 4-10-1 Run: Initialize String Failure

```
Command: r
Select input string to operate on: 10000
Error: string not in list!
Command:
```

Figure 4-10-2 Run: Initialize Input Failure

```
Command: r
Select input string to operate on: aab
Error: invalid input format, enter positive integer
Command:
```

Figure 4-10-3 Run: Initialize Success

```
Command: r
Select input string to operate on: 1
0. [s0]aabaab
1. a[s0]bab
Command:
```

Figure 4-10-4 Run: Perform Transitions

```
Command: r
2. [s1]aYaab
Command: r
3. X[s0]Y
Command: r
4. XY[s0]
Command:
```

Figure 4-10-5 Run: Accepted

```
Command: r
5. XY[s3]
ab is accepted in 5 transitions
Command:
```

Figure 4-10-6 Run: Rejected

```
Command: r
5. XY[s3]
ab is rejected in 5 transitions
Command:
```

4.11 Quit Command

Figure 4-11-1 Quit: Success

```
Command: q
aab not accepted or rejected in 1 transition
Command:
```

Figure 4-11-2 Quit: Failure

```
Command: q
Error: nothing to quit!
Command:
```

4.12 Exit Command

Figure 4-12-1 Exit: Success

```
Command: x
Input string file successfully overwritten!
```

Figure 4-12-2 Exit: Failure

```
Command: x
Error: Input string failed to be overwritten!
```

5. Files

5.1 Turing Machine Definition File

Figure 5-1-1 Valid Definition File 1

```
This Turing machine accepts the language of one or more a's followed by the
same number of b's.

STATES: s0 s1 s2 s3 s4

INPUT_ALPHABET: a b

TAPE_ALPHABET: a b X Y -

TRANSITION_FUNCTION:

s0 a s1 X R
s0 Y s3 Y R
s1 a s1 a R
s1 b s2 Y L
s1 Y s1 Y R
s2 a s2 a L
s2 X s0 X R
s2 Y s2 Y L

INITIAL_STATE: s0

BLANK_CHARACTER: -

FINAL_STATES: s4
```

Figure 5-1-2 Valid Definition File 2

```
The minimumalistic TM

STATES: read end

INPUT_ALPHABET: a

TAPE_ALPHABET: a -

TRANSITION_FUNCTION:

read a begin a R
read - end - L

INITIAL_STATE: read

BLANK_CHARACTER: -

FINAL_STATES: end
```

Figure 5-1-2 Erroneous Definition File 1

```
STATES:                      States declared in δ, q0, and F not included in Q

INPUT_ALPHABET: a b

TAPE_ALPHABET: a X Y -       ‘b’ is declared in Σ but not in Γ

TRANSITION_FUNCTION:

s0 a s1 X R
s0 Y s3 Y                    Function is missing a move direction
s1 a s1 a R
s1 b s2 Y L
s1 Y s1 Y R
s2 a s2 a L
s2 X s0 X R
s2 Y s2 Y L

INITIAL_STATE: s0

BLANK_CHARACTER: -

FINAL_STATES:                Not an error that no final states are defined
```

Figure 4-2-3 Erroneous Definition File 2

STATES: s0 s1 s2 s3 s4	
INPUT_ALPHABET: a b	
TAPE_ALPHABET: a b X Y	Character in B not included in Γ
TRANSITION_FUNCTION:	
s1 X R s0 Y s3 Y R s1 a s1 a R s1 b s2 Y L s1 Y s1 Y R s2 a s2 a L s2 X s0 X R s2 Y s2 Y L	Current state and read state not included. May cause cascading errors throughout the transition_function: section of the document
INITIAL_STATE:	No initial state declared
BLANK_CHARACTER: -	
	Is an error that final_states: keyword is not included

5.2 Input String File

Figure 5-2-1 Input String File based on Figure 5-1-1

aabb	
\	valid, empty string
ab	
abc	Not valid, contains characters not in sigma
aaabb	
aaaaaa	valid even though it will rejected by TM
aXYb	Not valid, all characters are in gamma but not in sigma
bb	
\	invalid, it is a duplicate
ab	invalid, it is a duplicate
ababab-	Not valid, blank character not in sigma

References

Farrell, D. C. (2017). Requirement Specification for NifflerTM: A Command-line Turing Machine. Unpublished manuscript.