Module Interface Specification for Flow

Team 9, min-cut
Ethan Patterson
Hussain Muhammed
Jeffrey Doan
Kevin Zhu
Chengze Zhao

January 22, 2026

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/ritual-17/flow/tree/main/docs/SRS-Meyer

Contents

3 Introduction

The following document details the Module Interface Specifications for Flow

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/ritual-17/flow.

4 Notation

The specification of Flow uses several derived data types, including sequences, strings, and tuples. Sequences are ordered lists containing elements of the same data type. Strings are sequences of characters. Tuples represent a finite collection of values, potentially of different types. Functions are described by specifying the data types of their inputs and outputs, and local functions are documented using their type signatures followed by their behavioural descriptions.

Several modules, most notably the Shape Interface Module, operate on geometric objects that are placed on a two-dimensional canvas. Shapes are treated abstractly and are not tied to a specific rendering or user interface implementation. A shape is represented conceptually as a tuple consisting of a shape type and a finite set of parameters. The shape type is drawn from a predefined set of supported shapes (e.g., rectangle, circle, text box), while the parameters describe properties such as position, size, and orientation.

Positions are expressed using Cartesian coordinates in a two-dimensional plane, typically represented as (x, y) pairs of real numbers. The canvas may be conceptually overlaid with a uniform grid that provides a visual reference and optional alignment structure for placing and manipulating shapes. The grid does not impose semantic constraints on shape placement, but may be used to support consistent positioning, snapping behaviour, or alignment operations. Dimensions such as width, height, or radius are represented using real-valued quantities. Collections of shapes are represented as sequences of shape tuples. Operations on shapes, such as insertion, deletion, movement, resizing, or transformation, are described in terms of updates applied to these abstract representations. Any constraints on valid coordinates or dimensions are specified explicitly in the relevant module interfaces.

5 Module Decomposition

Level 1	**Level 2**
Hardware-Hiding	6 Display Interface Module ?? Input Interface Module ?? File Interface Module
Behaviour-Hiding	?? Geometry State Parser Module ?? Geometry State Converter Module ?? Commands Parser Module ?? Mode Commands Module ?? Shape Interface Module ?? User Preference Module ?? Geometry State Parser Module
Software Decision	?? Text Buffer Module ?? Geometry State Module ?? Geometry State Mutator Module ?? Undo Redo Module ?? User Persistence Module

Table 1: Module Hierarchy

6 MIS of Display Interface Module

6.1 Module

Display for the system

This is the module that will handle the display for the system. This is a module that when called updates the display.

6.2 Uses

?? Geometry State Module

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
display	Shapes	Display to screen	-
resize	width, length	Resize the screen as needed	-
status		Show status	-

6.4 Semantics

6.4.1 State Variables

N/A

6.4.2 Environment Variables

- System Screen
- Rendering API

6.4.3 Assumptions

6.4.4 Access Routine Semantics

display(Shapes):

- output: Outputs Shapes visually on the screen

resize(length,width):

- output: Outputs resized version of the current screen

status():

- output: Outputs current status of the program ex waiting fr input, rendering etc.

6.4.5 Local Functions

N/A

7 MIS of Input Interface Module

7.1 Module

Input Module. Deals with User inputs and passes them on to the **??** Mode Commands Module

7.2 Uses

?? Mode Commands Module

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_keyboard_inputs		keyboard_inputs	-
get_mouse_inputs		mouse_inputs	-

7.4 Semantics

7.4.1 State Variables

7.4.2 Environment Variables

- Keyboard API
- Mouse API

7.4.3 Assumptions

7.4.4 Access Routine Semantics

get_keyboard_inputs():

- output: Data containing all current keyboard inputs. All keys will be off if the user is not using a keyboard.

get_mouse_inputs():

- output: Data containing mouse location and inputs. All inputs are off if the user does not have a mouse.

7.4.5 Local Functions

N/A

8 MIS of File Interface Module

8.1 Module

?? Geometry State Parser Module

8.2 Uses

?? Mode Commands Module

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
open	file_path	opened_file	Invalid File
save	file_path, opened_note	success	Saving Error

8.4 Semantics

8.4.1 State Variables

N/A

8.4.2 Environment Variables

- File System

8.4.3 Assumptions

N/A

8.4.4 Access Routine Semantics

open(file_path):

- output: The opened file
- exception: Invalid File if file is missing or of an incorrect type.

save(file_path, opened_note):

- output: If the Module was successful in saving opened_note to file_path
- exception: No Space if there is not enough space to save the file.

8.4.5 Local Functions

N/A

9 MIS of Geometry State Parser Module

9.1 Module

File parser takes the file input and parses it into data structure containing all shapes which is then stored by the **??** Geometry State Module which holds all shape data.

9.2 Uses

?? Geometry State Module

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
parse	opened_file	shapes	Invalid File

9.4 Semantics

9.4.1 State Variables

N/A

9.4.2 Environment Variables

N/A

9.4.3 Assumptions

N/A

9.4.4 Access Routine Semantics

parse(opened_file):

- output: Data structure containing all shapes from the given file.
- exception: Invalid File: if the file given is able to be parsed.

9.4.5 Local Functions

10 MIS of Geometry State Converter Module

10.1 Module

Note converter that converts notes to files of different file types, sending it to the file interface module

10.2 Uses

?? File Interface Module

10.3 Syntax

10.3.1 Exported Constants

Legal file types

- PDF
- fnote

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
convert	opened_note, file_type	file	Invalid file_type

10.4 Semantics

10.4.1 State Variables

10.4.2 Environment Variables

10.4.3 Assumptions

10.4.4 Access Routine Semantics

convert(opened_note, file_type):

- output: file that is ready to be saved on the computer.
- exception: Invalid file_type if the given file type is not compatable with the ones implemented

10.4.5 Local Functions

convert_*(opened_note) functions that do the converting once the file type has been determined. (ex convert_pdf)

11 MIS of Commands Parser Module

11.1 Module

Command Parser, converts keyboard inputs into commands and passes them onto the shape mutator.

11.2 Uses

?? Geometry State Mutator Module **??** User Preference Module

11.3 Syntax

11.3.1 Exported Constants

Command names/ids ex ctrl + s -¿ ¡C-s¿, d + d -¿ "dd"

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
parse_commands	keyboard_inputs, mouse_inputs	parsed$_{i}nput$	-

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Environment Variables

N/A

11.4.3 Assumptions

Inputs have been already cleaned by the other modules.

11.4.4 Access Routine Semantics

parse_commands(keyboard_inputs, mouse_inputs):

- output: commands, which would be used by the Geometry State Mutator Module **??**

11.4.5 Local Functions

N/A

12 MIS of Mode Commands Module

12.1 Module

Main Module contains the commands usable in the current mode.

12.2 Uses

N/A

12.3 Syntax

12.3.1 Exported Constants

Modes:

- normalMode
- insertMode
- visualMode
- commandMode
- lineMode
- textMode

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_commands	mode	commands	-

12.4 Semantics

12.4.1 State Variables

N/A

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

Mode given will always be a valid mode.

12.4.4 Access Routine Semantics

get_commands(mode):

- output: commands that can be run in the given mode

12.4.5 Local Functions

N/A

13 MIS of Shape Interface Module

13.1 Module

Module that contains information about shapes.

13.2 Uses

13.3 Syntax

13.3.1 Exported Constants

Shapes: Types of shapes

- Rectangle
- circle
- Triangle
- Text Box

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
create_shape	shape_type, points	shape	Invalid Points
move_shape	shape, new_points	-	Invalid Points

13.4 Semantics

13.4.1 State Variables

N/A

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

N/A

13.4.4 Access Routine Semantics

create_shape(shape_type, points):

- output: Shape with type shape_type using points for location and size.
- exception: Invalid points if the points are not possible for the type of shape.

move_shape():

- transition: changes the points of the shape to the new points
- exception: Invalid points if the points are not possible for the type of shape.

13.4.5 Local Functions

N/A

14 MIS of User Preference Module

14.1 Module

User preferences, provides an interface for creating and manipulating user preferences in the system.

14.2 Uses

6 Display Interface Module **??** User Persistence Module

14.3 Syntax

14.3.1 Exported Constants

N/A

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_command_prefrences	mode	commands	-
modify	-	-	-
get_theme_prefrences	-	theme	-

14.4 Semantics

14.4.1 State Variables

Command preferences: users defined commands Theme preferences: user theme prefrences

14.4.2 Environment Variables

14.4.3 Assumptions

14.4.4 Access Routine Semantics

get_command_prefrences(mode):

- output: Any user defined preferences for the current mode

modify():

- transition: Has the user input their preferences where the moudles saves it to a file.

get_theme_prefrences():

- output: Theme preferences for the system

14.4.5 Local Functions

15 MIS of Text Buffer Module

15.1 Module

TextBuffer

15.2 Uses

Geometry State Module (**??**)

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
createText	location, text, formatting	TextID	Invalid Location
modifyText	TextID, text	-	Text Not Found
moveText	TextID, newLocation	TextObject	Text Not Found
deleteText	TextID	-	Text Not Found

15.4 Semantics

15.4.1 State Variables

text : MapTextID, TextObject
Where TextObject contains:

- position : location
- content : string

15.4.2 Environment Variables

Screen (for rendering text)

15.4.3 Assumptions

TextIDs are unique and immutable.

15.4.4 Access Routine Semantics

createText(location, text, formatting):

- transition: Text box at location with text and formatting
- output: TextID of the created text box
- exception: Invalid Location

modifyText(TextID, text):

- transition: Text in text box is modified to now contain text with formatting
- exception: Text Not Found

moveText(TextID, newLocation):

- transition: text box is moved to location.
- exception: Text Not Found

deleteText(TextID):

- transition: text box is removed from the text buffer
- exception: Text Not Found

15.4.5 Local Functions

generateTextID() : TextID

16 MIS of Geometry State Module

16.1 Module

GeometryState (**??**)

16.2 Uses

None

16.3 Syntax

16.3.1 Exported Constants

ShapeType = Circle, Rectangle, Triangle, TextBox
LineType = Line, Arrow

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
getShapes	-	SetShape	-
getText	-	SetTextBox	-
getLines	-	SetLine	-

16.4 Semantics

16.4.1 State Variables

- Shapes : SetShape
- TextBoxes : SetTextBox
- Lines : SetLine

16.4.2 Environment Variables

None

16.4.3 Assumptions

Geometry is immutable except via Geometry State Mutator Module (**??**).

16.4.4 Access Routine Semantics

getShapes():

- output: Data Structure containing all shapes

getText():

- output: Data structure containing all text boxes

getLines():

- output: Data structure containing all lines

16.4.5 Local Functions

None

17 MIS of Geometry State Mutator Module

17.1 Module

GeometryStateMutator (**??**)

17.2 Uses

Undo Redo Module **??** Geometry State Module **??**

17.3 Syntax

17.3.1 Exported Constants

N/A

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
addShape	shapeSpec	GeometryID	Invalid Spec
addLine	lineSpec	GeometryID	Invalid Spec
delete	GeometryID	-	Not Found
resize	GeometryID	-	Not Found
move	GeometryID, location	-	Not Found, Invalid Transform

17.4 Semantics

17.4.1 State Variables

None

17.4.2 Environment Variables

None

17.4.3 Assumptions

Mutations must be recorded by Undo Redo Module.

17.4.4 Access Routine Semantics

addShape(shapeSpec):

- transition: Adds the shape to the canvas
- output: GeometryID of the added shape
- exception: Invalid Spec if the shapeSpec is invalid

addLine(lineSpec):

- transition: Adds the line to the canvas
- output: GeometryID of the added line
- exception: Invalid Spec if the lineSpec is invalid

delete(GeometryID):

- transition: Deletes the shape/line/textbox with the given GeometryID
- exception: Not Found if the GeometryID does not exist

resize(GeometryID):

- transition: Resizes the shape/line/textbox with the given GeometryID
- exception: Not Found if the GeometryID does not exist

move(GeometryID, location):

- transition: Moves the shape/line/textbox with the given GeometryID to location
- exception: Not Found if the GeometryID does not exist
- exception: Invalid Transform if the location is invalid

17.4.5 Local Functions

None

18 MIS of Undo Redo Module

18.1 Module

UndoRedoModule (**??**)

18.2 Uses

Geometry State Module **??**

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
undo	-	-	Cant Undo
redo	-	-	Cant Redo
addCommand	command	-	-

18.4 Semantics

18.4.1 State Variables

- undoStack : StackCommand
- redoStack : StackCommand

18.4.2 Environment Variables

None

18.4.3 Assumptions

Commands fully capture inverse operations.

18.4.4 Access Routine Semantics

undo():

- transition: Pops command from undoStack, executes its inverse, and pushes it onto redoStack
- exception: Cant Undo if undoStack is empty

redo():

- transition: Pops command from redoStack, re-executes it, and pushes it onto undoStack
- exception: Cant Redo if redoStack is empty

addCommand(command):

- transition: Pushes command onto undoStack and clears redoStack

18.4.5 Local Functions

N/A

19 MIS of User Persistence Module

19.1 Module

UserPersistenceModule (**??**)

19.2 Uses

User Preference Module **??**

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
save	FilePath	-	IOError
load	FilePath	User Themes and Saves	IOError

19.4 Semantics

19.4.1 State Variables

None

19.4.2 Environment Variables

File System

19.4.3 Assumptions

File format is versioned and backward compatible.

19.4.4 Access Routine Semantics

save(commands,theme):

- transition: Saves user preferences to file at FilePath
- exception: IOError if there is an issue writing to the file

load(FilePath):

- output: User Themes and Saves loaded from file at FilePath
- exception: IOError if there is an issue reading from the file

19.4.5 Local Functions

None

References

20 Appendix

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)