8/21/2018

Requirements Document

Braille Authoring App

Team Members:

Jamie Dishy
Jonas Laya
Samuel On
Paul Sison

BRAILLE CORP.

Submitted to:

Prof. Bill Tzerpos
EECS 2311

Contents

1 Introduction

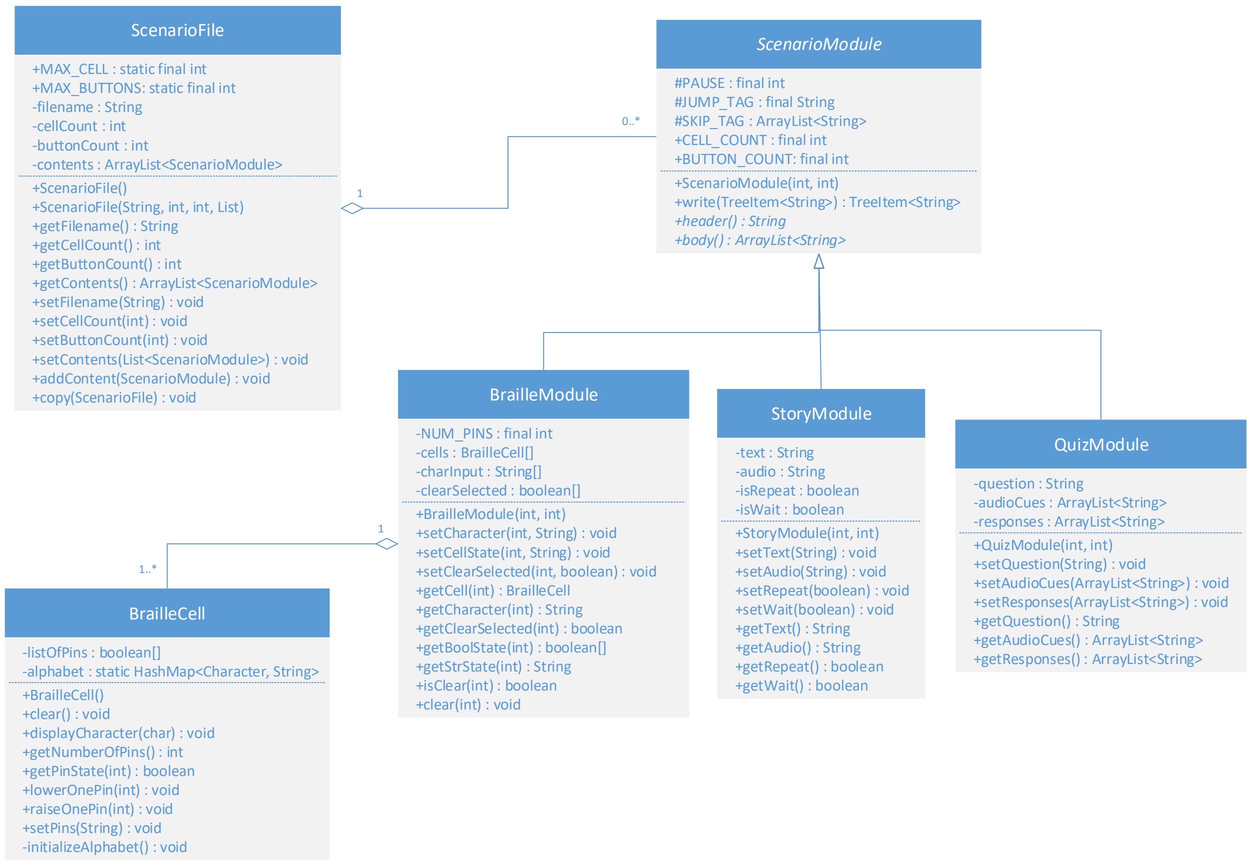
This design document is one of four documents that serves as final deliverables for the *Braille Authoring App* project submitted by Team no. 14, consisting of Jamie Dishy, Jonas Laya, Samuel On and Paul Sison, for the EECS 2031 *Software Development Project* course in the 2017–18 academic year.

In this document, we show the overall design hierarchy of the *Braille Authoring App* through class diagrams, and portray program flow and class interactions through sequence diagrams. We highlight important operations in the application by going through step-by-step some of the most common usage scenarios.

2 Class Diagrams

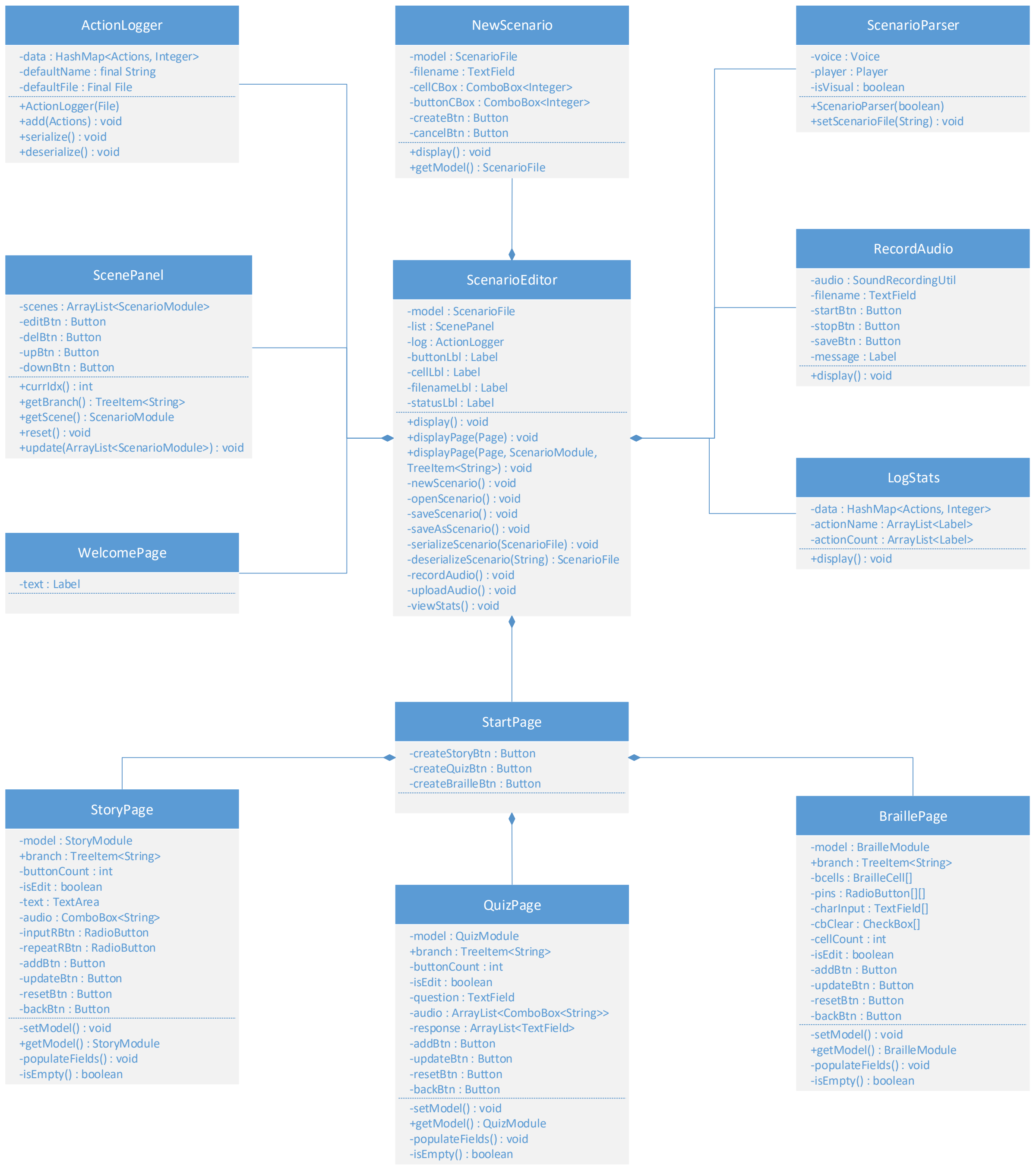
2.1 Model classes

These classes are the ones responsible for manipulating and storing all the data required by the application to build, write and edit Scenario files. In brief, Scenario files are represented by a `ScenarioFile` object, which can contain zero or more `ScenarioModule` objects. Note that `ScenarioModule` is an abstract class and is instantiated by three kinds of Module classes—`StoryModule`, `QuizModule`, and `BrailleModule`. Each of the Module class stores different kinds of information that corresponds to tags in a properly formatted Scenario file. Whenever the application executes a save operation, it is the state of the `ScenarioFile` that gets serialized into a binary file.


ScenarioFile
+MAX_CELL : static final int
+MAX_BUTTONS: static final int
-filename : String
-cellCount : int
-buttonCount : int
-contents : ArrayList<ScenarioModule>
+ScenarioFile()
+ScenarioFile(String, int, int, List)
+getFilename() : String
+getCellCount() : int
+getButtonCount() : int
+getContents() : ArrayList<ScenarioModule>
+setFilename(String) : void
+setCellCount(int) : void
+setButtonCount(int) : void
+setContents(List<ScenarioModule>) : void
+addContent(ScenarioModule) : void
+copy(ScenarioFile) : void
1
0..*
ScenarioModule
#PAUSE : final int
#JUMP_TAG : final String
#SKIP_TAG : ArrayList<String>
+CELL_COUNT : final int
+BUTTON_COUNT: final int
+ScenarioModule(int, int)
+write(TreeItem<String>) : TreeItem<String>
+header() : String
+body() : ArrayList<String>
BrailleModule
-NUM_PINS : final int
-cells : BrailleCell[]
-charInput : String[]
-clearSelected : boolean[]
+BrailleModule(int, int)
+setCharacter(int, String) : void
+setCellState(int, String) : void
+setClearSelected(int, boolean) : void
+getCell(int) : BrailleCell
+getCharacter(int) : String
+getClearSelected(int) : boolean
+getBoolState(int) : boolean[]
+getStrState(int) : String
+isClear(int) : boolean
+clear(int) : void
StoryModule
-text : String
-audio : String
-isRepeat : boolean
-isWait : boolean
+StoryModule(int, int)
+setText(String) : void
+setAudio(String) : void
+setRepeat(boolean) : void
+setWait(boolean) : void
+getText() : String
+getAudio() : String
+getRepeat() : boolean
+getWait() : boolean
QuizModule
-question : String
-audioCues : ArrayList<String>
-responses : ArrayList<String>
+QuizModule(int, int)
+setQuestion(String) : void
+setAudioCues(ArrayList<String>) : void
+setResponses(ArrayList<String>) : void
+getQuestion() : String
+getAudioCues() : ArrayList<String>
+getResponses() : ArrayList<String>
1
1..*
BrailleCell
-listOfPins : boolean[]
-alphabet : static HashMap<Character, String>
+BrailleCell()
+clear() : void
+displayCharacter(char) : void
+getNumberOfPins() : int
+getPinState(int) : boolean
+lowerOnePin(int) : void
+raiseOnePin(int) : void
+setPins(String) : void
-initializeAlphabet() : void

2.1 Model classes

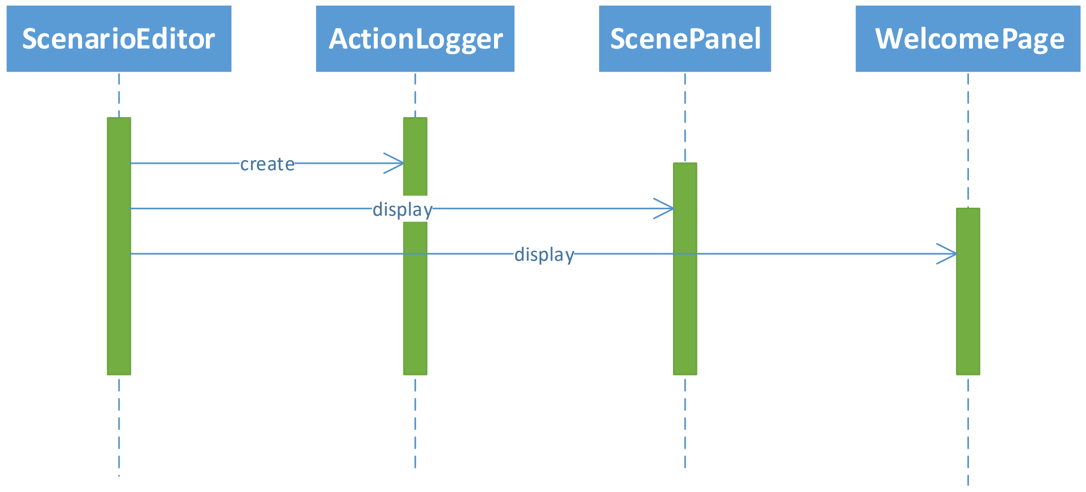
The application runs by instantiating a `ScenarioEditor` class, which is responsible for displaying everything else in the application, from dialog boxes to other pages. Note that an instance of a `ScenarioEditor` is associated with a single `ScenarioFile` object.



3 Sequence Diagrams

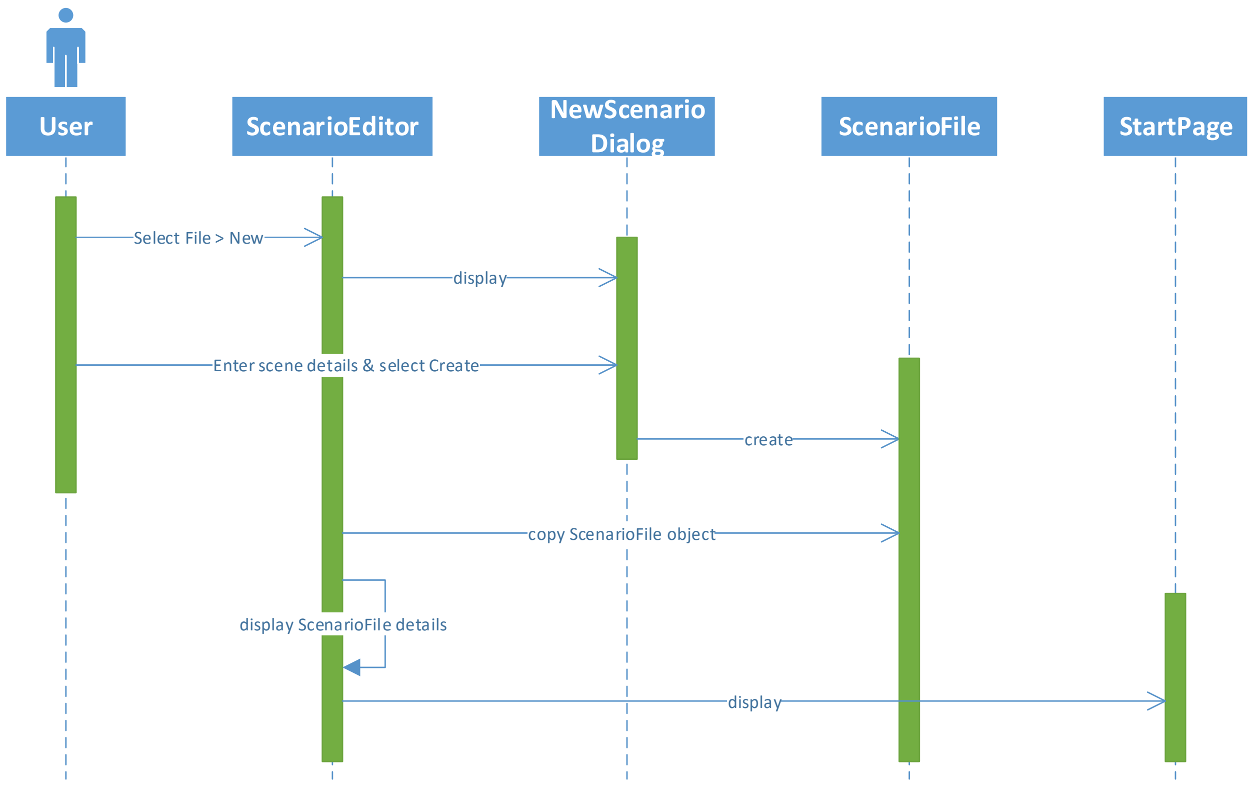
3.1 Start-up

Here are some initializations that happen during start-up of the application. Although a `ScenarioFile` object is associated per `ScenarioEditor`, it is not created until the user creates a new scenario, or opens an existing scenario file.



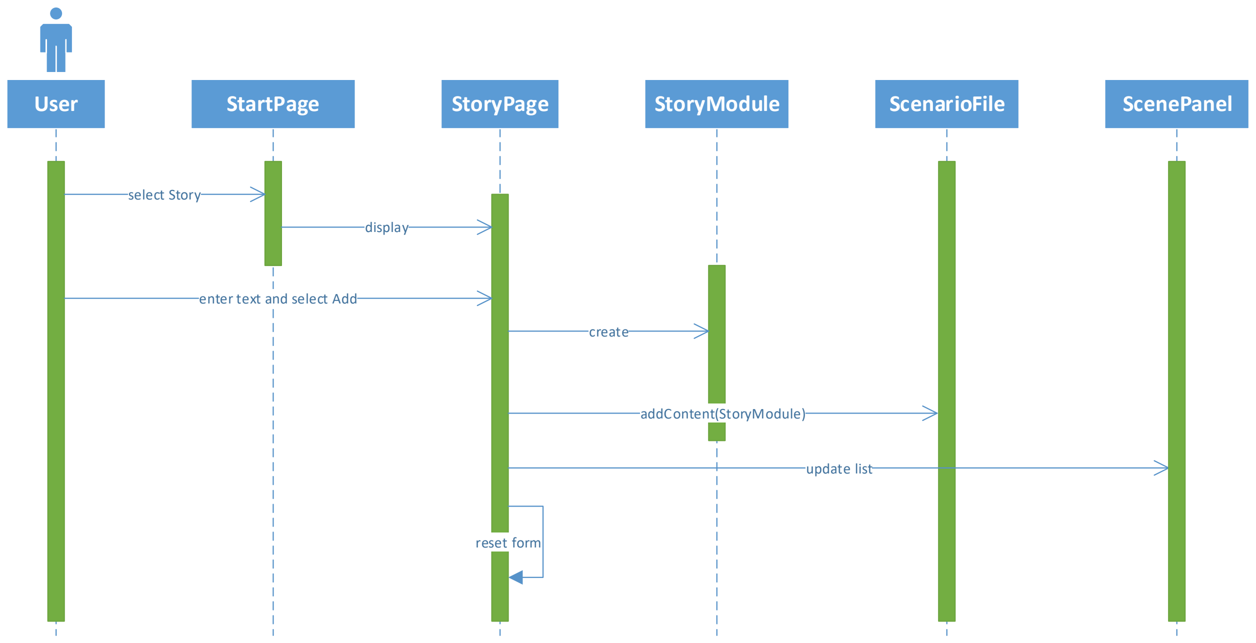
3.2 New Scenario

Here we show how a `ScenarioFile` object is created by the `NewScenarioDialog`. Reference to this `ScenarioFile` object is then passed on to `ScenarioEditor`. The `ScenarioEditor` window then displays on its status bar the number of buttons and braille cells and the filename that the `ScenarioFile` has for the user. Afterwards, `ScenarioEditor` displays the `StartPage` so the user can start building the `ScenarioFile`.



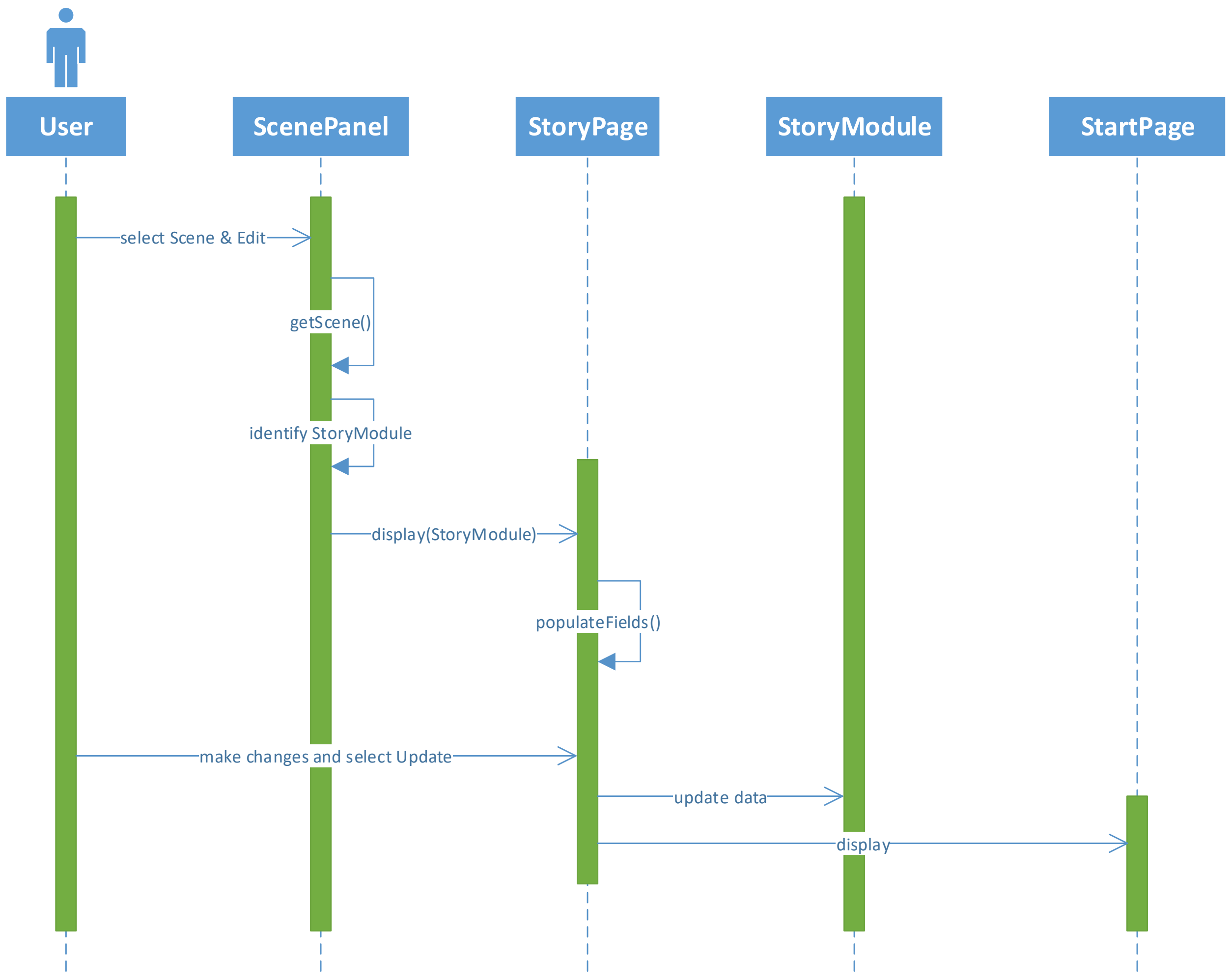
3.2 Adding Story

Here we assume that the `StartPage` is already being displayed to the User. Also, note that the `ScenarioFile` and `ScenePanel` already persists. The `StoryModule` object that was created is not lost but added to the `ScenarioFile`.



3.4 Editing Scenes

Here we assume that a `StoryModule` has already been added to the `ScenarioFile` object, hence the lifeline. The user can edit the object by selecting the `StoryModule` scene from the `ScenePanel` and selecting Edit. The `ScenePanel` retrieves the `StoryModule` object from its list and passes it to `StoryPage`. The `StoryPage` displays the data stored in `StoryModule` so the user can make the desired changes. The same process takes place when editing a `QuizModule` and a `BrailleModule`.


User
ScenePanel
StoryPage
StoryModule
StartPage
select Scene & Edit
getScene()
identify StoryModule
display(StoryModule)
populateFields()
make changes and select Update
update data
display