The *Metro Map Maker* ™ Software Design Description

Author: Alexander Collado
Debugging Enterprises™
October, 2017
Version 1.0

Based on IEEE Std 830TM-1998 (R2009) document format

1. Introduction

This is the Software Design Description (SDD) for the Metro Map Maker ™ application. Note that this document format is based on the IEEE Standard 1016-2009 recommendation for software design.

1.1 Purpose

This document is to serve as the blueprint for the construction of the Metro Map Maker application. This design will use UML class diagrams to provide complete detail regarding all packages, classes, instance variables, class variables, and method signatures needed to build the application. In addition, UML Sequence diagrams will be used to specify object interactions post-initialization of the application, meaning in response to user interactions or timed events.

1.2 Scope

Metro Map Maker is an application that provides users the ability to create graphical representations of city subway systems. Provided frameworks called DesktopJavaFramework, PropertiesManager, and jTPS will be used in the design of the application. This design contains design descriptions for the development of the application. Note that Java is the target language for this software design.

1.3 Definitions, acronyms, and abbreviations

Framework – In an object-oriented language, a collection of classes and interfaces that collectively provide a service for building applications or additional frameworks all with a common need.

GUI – Graphical User Interface, visual controls like buttons inside a window in a software application that collectively allow the user to operate the program.

IEEE – Institute of Electrical and Electronics Engineers, the "world's largest professional association for the advancement of technology".

jTPS – A framework used for undoing and redoing actions taken within the application.

JSON – JavaScript Object Notation. The file format that will be used to save and load work within the application

Stylesheet – a static text file employed by HTML pages that can control the colors, fonts, layout and other style components in a Web page.

UML – Unified Modeling Language, a standard set of document formats for designing software graphically.

Use Case Descriptions – A formal format for specifying how a user will interact with a system.

1.4 References

IEEE Std 830TM-1998 (R2009) – IEEE Recommended Practice for Software Requirements Specification

1.5 Overview

This Software Design Description document provides a working design for the Metro Map Maker software application as described in the Metro Map Maker Software Requirements Specification. Note that all parties in the implementation stage must agree upon all connections between components before proceeding with the implementation stage. Section 2 of this document will provide the Package-Level Viewpoint, specifying the packages and frameworks to be designed. Section 3 will provide the Class-Level Viewpoint, using UML Class Diagrams to specify how the classes should be constructed. Section 4 will provide the Method-Level System Viewpoint, describing how methods will interact with one another. Section 5 provides deployment information like file structures and formats to use. Section 6 provides a Table of Contents, an Index, and References. Note that all UML Diagrams in this document were created using the VioletUML editor.

2. Package-Level Design Viewpoint

As mentioned, this design will encompass both the Metro Map Maker game application and the DesktopJavaFramework, PropertiesManager, and jTPS frameworks to be used in its construction. In building, we will heavily rely on the Java API to provide services. Following are descriptions of the components to be built, as well as how the Java API will be used to build them.

2.1 Metro Map Maker overview

The Metro Map Maker application will be designed, relying on the DesktopJavaFramework, PropertiesManager, and jTPS frameworks to support it. Figure 2.1 specifies all the components to be developed and places all classes in home packages.

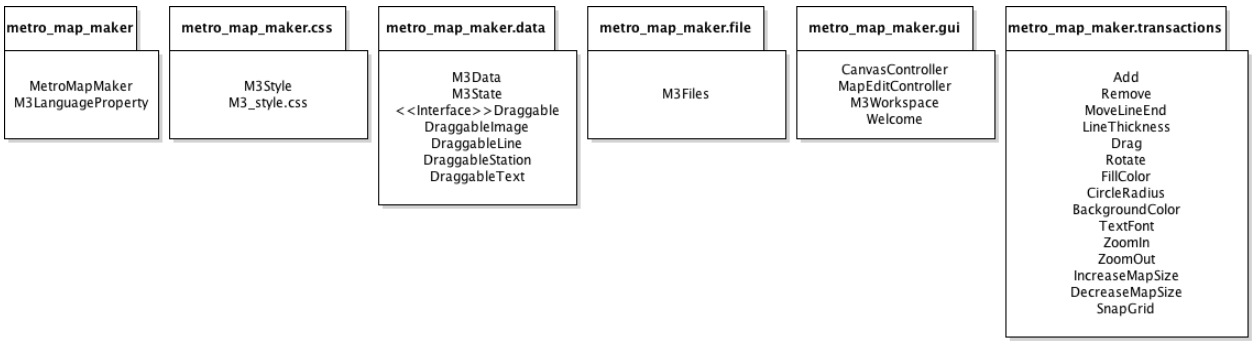


Figure 2.1: Design Packages Overview

2.2 Java API Usage

The Metro Map Maker application will be developed using the Java programming languages. As such, this design will make use of the classes specified in Figure 2.2.

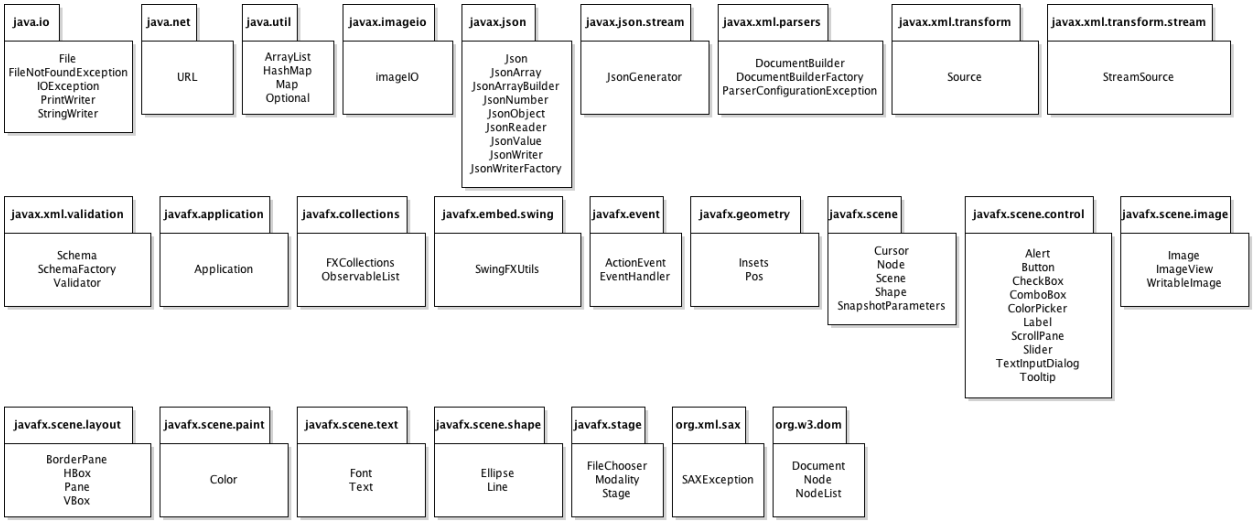


Figure 2.2: Java API Classes and Packages To Be Used

2.3 Java API Usage Descriptions Tables

2.1-2.25 below summarize how each of these classes will be used.

Class Interface	Use
File	For storing files including maps and images.
FileNotFoundException	Checks for errors in identifying files.
IOException	Checks for errors writing out data to files.
PrintWriter	Used in conjunction with StringWriter for writing a neat Json file.
StringWriter	Used in conjunction with PrintWriter for writing a neat Json file.

Table 2.1: Uses for classes in the Java API's java.io package

Class Interface	Use
URL	For setting up the stylesheet of the application.

Table 2.2: Uses for classes in the Java API's java.net package

Class Interface	Use
ArrayList	For creating transactions for undo/redo and for loading properties from XML files.
HashMap	For outputting to a Json file and for storing properties from XML files.

Map	For outputting to a Json file.
Optional	For creating show and wait dialog boxes.

Table 2.3: Uses for classes in the Java API's java.util package

Class Interface	Use
ImageIO	For exporting an image of the map.

Table 2.4: Uses for classes in the Java API's javax.imageio package

Class Interface	Use
Json	For listing complete information about the map.
JsonArray	For storing information in an array concerning Metro Lines and Stops in the map.
JsonArrayBuilder	For building the collection of Metro Lines and Stops in the map.
JsonNumber	For storing numerical information.
JsonObject	For storing complete information about the map.
JsonReader	For loading map information from a Json format .
JsonValue	For getting a JsonObject, JsonArray, or JsonNumber.
JsonWriter	For encoding data as Json.
JsonWriterFactory	For creating JsonWriter instances.

Table 2.5: Uses for classes in the Java API's javax.json package

Class Interface	Use
JsonGenerator	For writing Json data in a streaming way.

Table 2.6: Uses for classes in the Java API's javax.json.stream package

Class Interface	Use
DocumentBuilder	For loading XML documents.
DocumentBuilderFactory	For creating DocumentBuilder instances.
ParserConfigurationException	Checks for errors loading and parsing XML.

Table 2.7: Uses for classes in the Java API's javax.xml.parsers package

Class Interface	Use
Source	For creating a source input.

Table 2.8: Uses for classes in the Java API's javax.xml.transform package

Class Interface	Use
StreamSource	Used in conjunction with Source for parsing the XML document we want to check.

Table 2.9: Uses for classes in the Java API's javax.xml.transform.stream package

Class Interface	Use
Schema	For describing the elements in a XML document.
SchemaFactory	For creating Schema instances.
Validator	For checking the validity of a schema.

Table 2.10: Uses for classes in the Java API's java.xml.validation package

Class Interface	Use
Application	For initializing Metro Map Maker.

Table 2.11: Uses for classes in the Java API's javafx.application package

Class Interface	Use
FXCollections	For storing a list of Fonts.
ObservableList	For storing a list of stations or lines.

Table 2.12: Uses for classes in the Java API's javafx.collections package

Class Interface	Use
SwingFXUtils	For converting a JavaFX Image into a BufferedImage.

Table 2.13: Uses for classes in the Java API's javafx.embed.swing package

Class Interface	Use
ActionEvent	For getting information about an action event like which button was pressed.
EventHandler	For creating instances of EventListeners that respond to events such as action events.

Table 2.14: Uses for classes in the Java API's javafx.event package

Class Interface	Use
Insets	For specifying container sizes of message dialogs.
Pos	For specifying vertical and horizontal positioning of message dialogs.

Table 2.15: Uses for classes in the Java API's javafx.geometry package

Class Interface	Use
Cursor	For getting and setting the cursor type depending on the action being taken on the workspace.
Node	For creating Metro Lines and Stations on the map.
Scene	For containing all of the content of the stage's scene graph.

Shape	For creating Metro Lines and Stations on the map.
SnapshotParameters	For specifying the rendering attributes for node snapshots.

Table 2.16: Uses for classes in the Java API's javafx.scene package

Class Interface	Use
Alert	For creating the About dialog.
Button	For creating the user interface of the different functionalities of the map. Releases ActionEvents when pressed that are handled by EventHandlers.
CheckBox	For creating the snap-to-grid toggle button
ComboBox	For creating drop downs for Metro Lines, Metro Stations, and fonts.
ColorPicker	For selecting colors for Metro Lines, the background, and text.
Label	For giving titles to buttons and user interface in the workspace.
ScrollPane	For containing map editing user controls.
Slider	For changing Metro Line thickness and Metro Station circle radius.
TextInputDialog	For creating text such as the names of Metro Lines and Metro Stations.
Tooltip	For providing info on buttons when hovered on.

Table 2.17: Uses for classes in the Java API's javafx.scene.control package

Class Interface	Use
Image	For saving image backgrounds, overlays, and map exports.
ImageView	For viewing image backgrounds and overlays.
WritableImage	For creating an image from a snapshot.

Table 2.18: Uses for classes in the Java API's javafx.scene.image package

Class Interface	Use
BorderPane	For setting up the workspace.
HBox	For creating the top toolbar and all of the row toolbars that hold map editor controls.
Pane	For setting up the workspace.
VBox	For holding the row toolbars.

Table 2.19: Uses for classes in the Java API's javafx.scene.layout package

Class Interface	Use
Color	For setting the colors for Metro Lines, the background, and text.

Table 2.20: Uses for classes in the Java API's javafx.scene.paint package

Class Interface	Use
Font	For determining font family, size, weight, and posture.
Text	For displaying user created text.

Table 2.21: Uses for classes in the Java API's javafx.scene.text package

Class Interface	Use
Ellipse	For creating Metro Stations.
Line	For creating Metro Lines.

Table 2.22: Uses for classes in the Java API's javafx.scene.shape package

Class Interface	Use
FileChooser	For opening the file directory when choosing to save or load a map.
Modality	For making dialog boxes modal.
Stage	For initializing the application.

Table 2.23: Uses for classes in the Java API's javafx.stage package

Class Interface	Use
SAXException	Checks for invalid XML files.

Table 2.24: Uses for classes in the Java API's org.xml.sax package

Class Interface	Use
Document	For loading XML documents.
Node	For loading properties from XML documents.
NodeList	For accessing nodes in XML documents.

Table 2.25: Uses for classes in the Java API's org.w3c.dom package

3. Class-Level Design Viewpoint

As mentioned, this design will encompass the Metro Map Maker application and DesktopJavaFramework, PropertiesManager, and jTPS frameworks. The following UML Class Diagrams reflect this. Note that due to the complexity of the project, we present the class designs using a series of diagrams going from overview diagrams down to detailed ones.

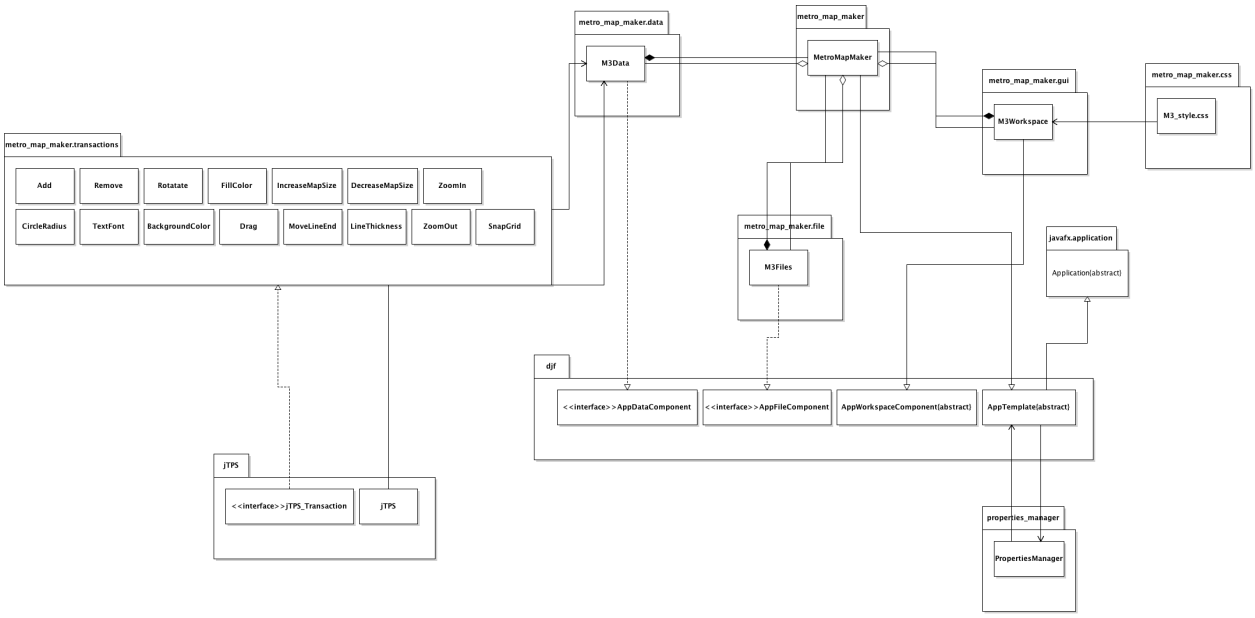


Figure 3.1: Metro Map Maker Overview UML Class Diagram

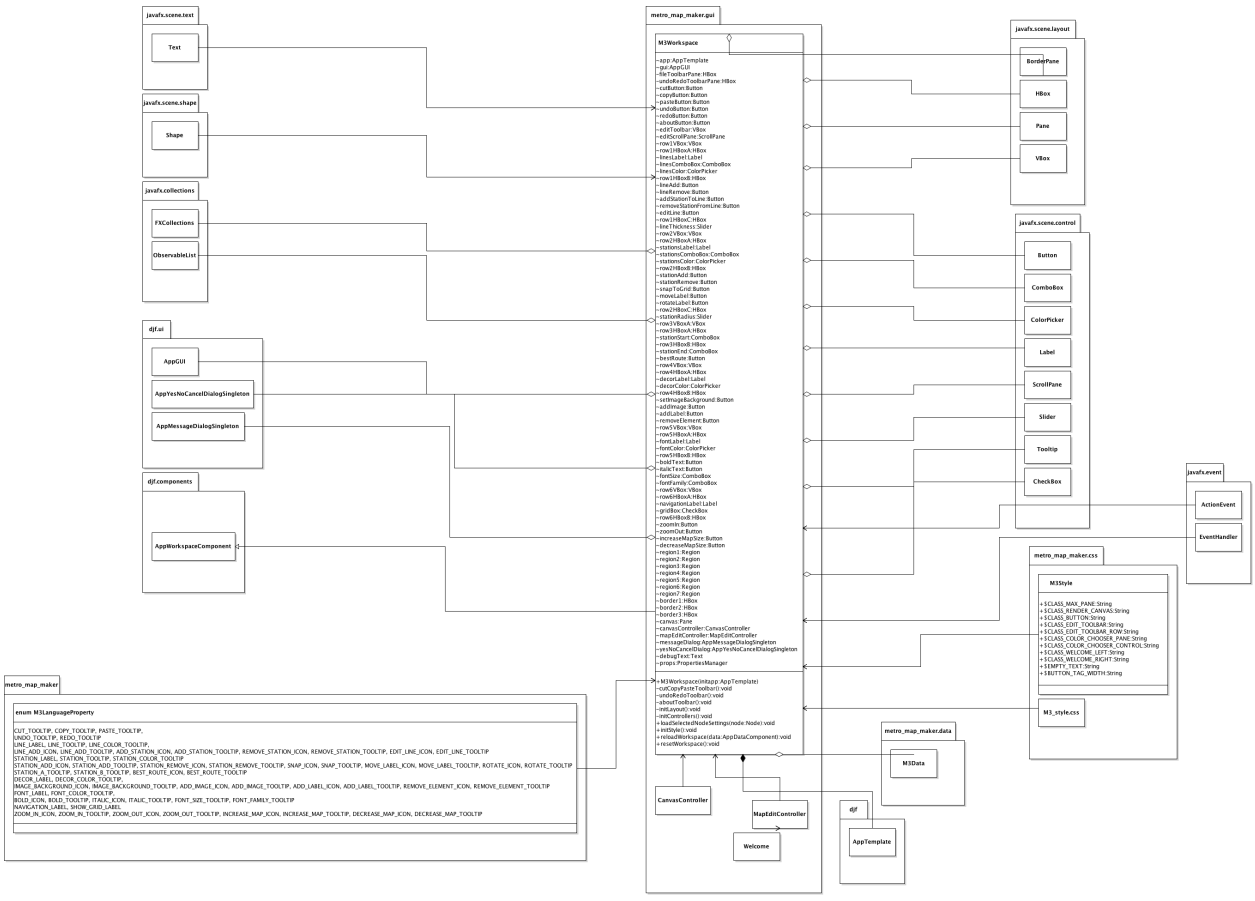


Figure 3.2: Detailed M3Workspace, M3Style, and M3LanguageProperty UML Class Diagrams

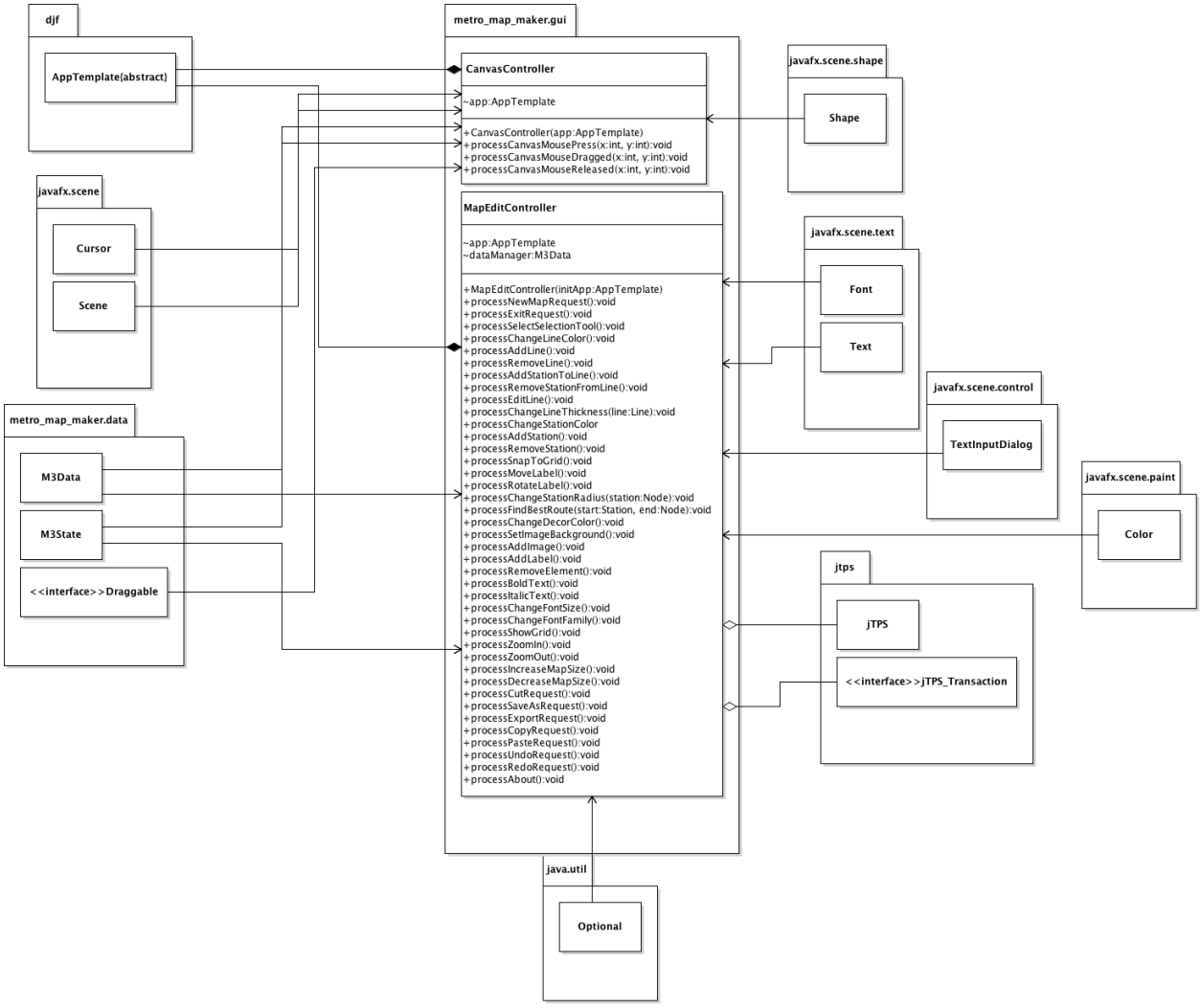


Figure 3.3: Detailed MapEditController and CanvasController UML Class Diagrams

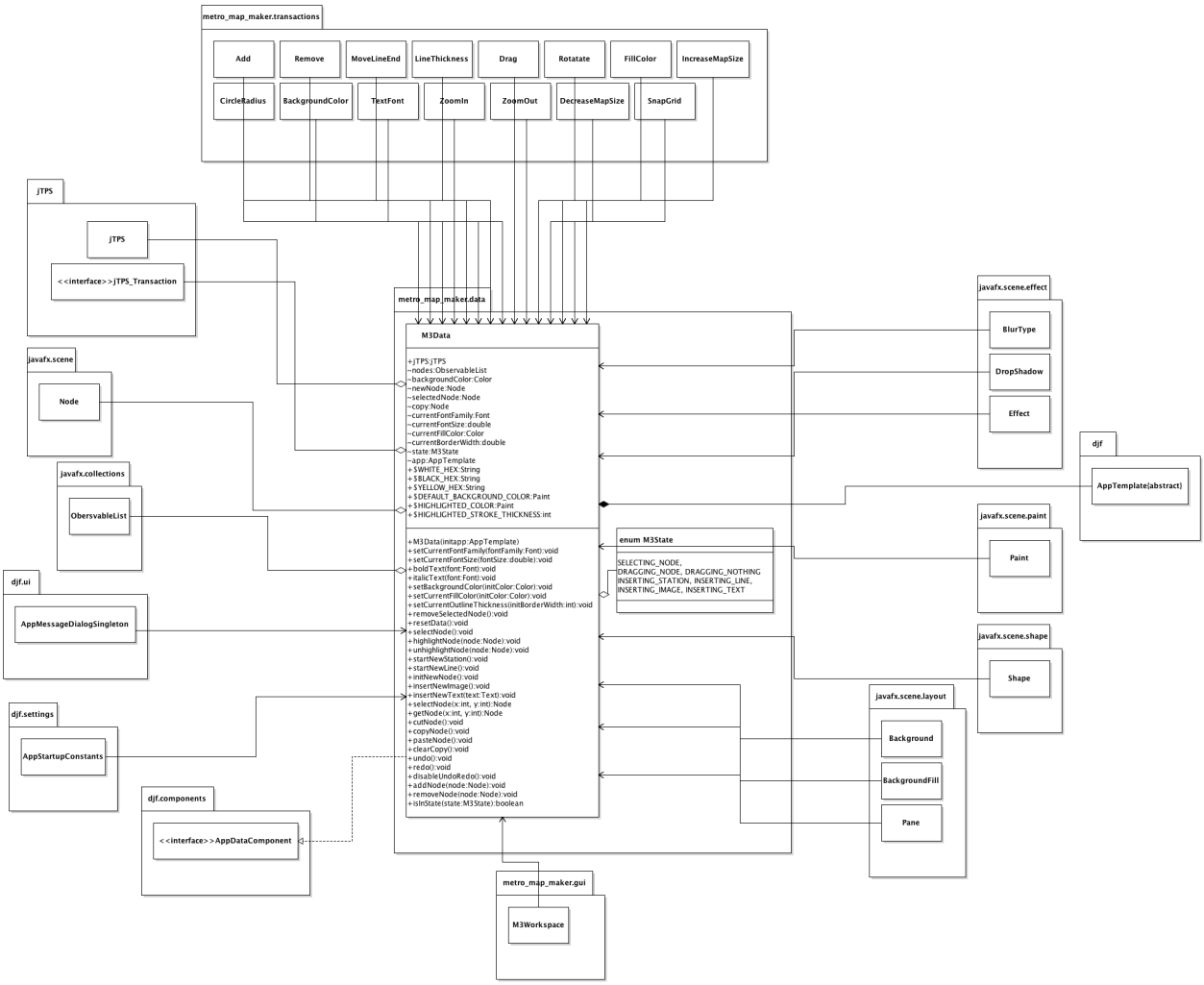


Figure 3.4: Detailed M3Data and M3State UML Class Diagrams

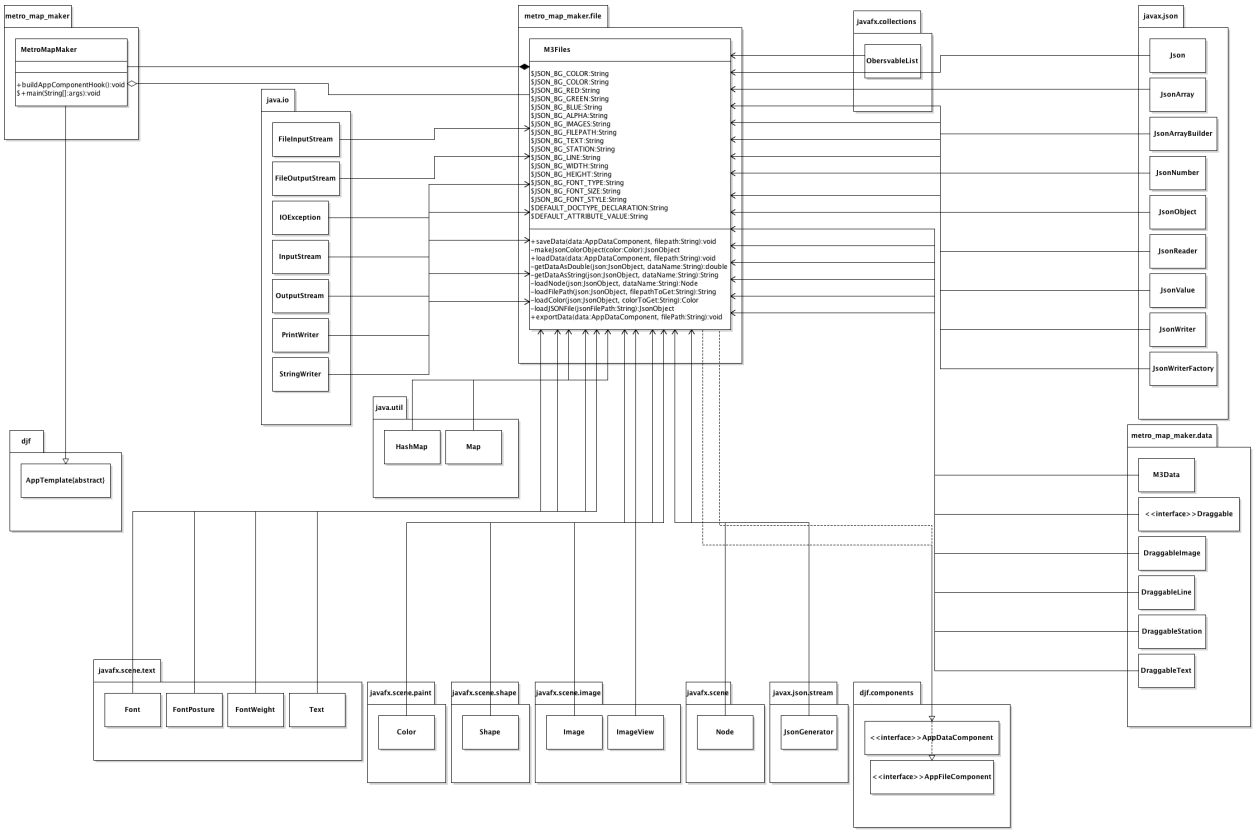


Figure 3.5: Detailed MetroMapMaker and M3Files UML Class Diagrams

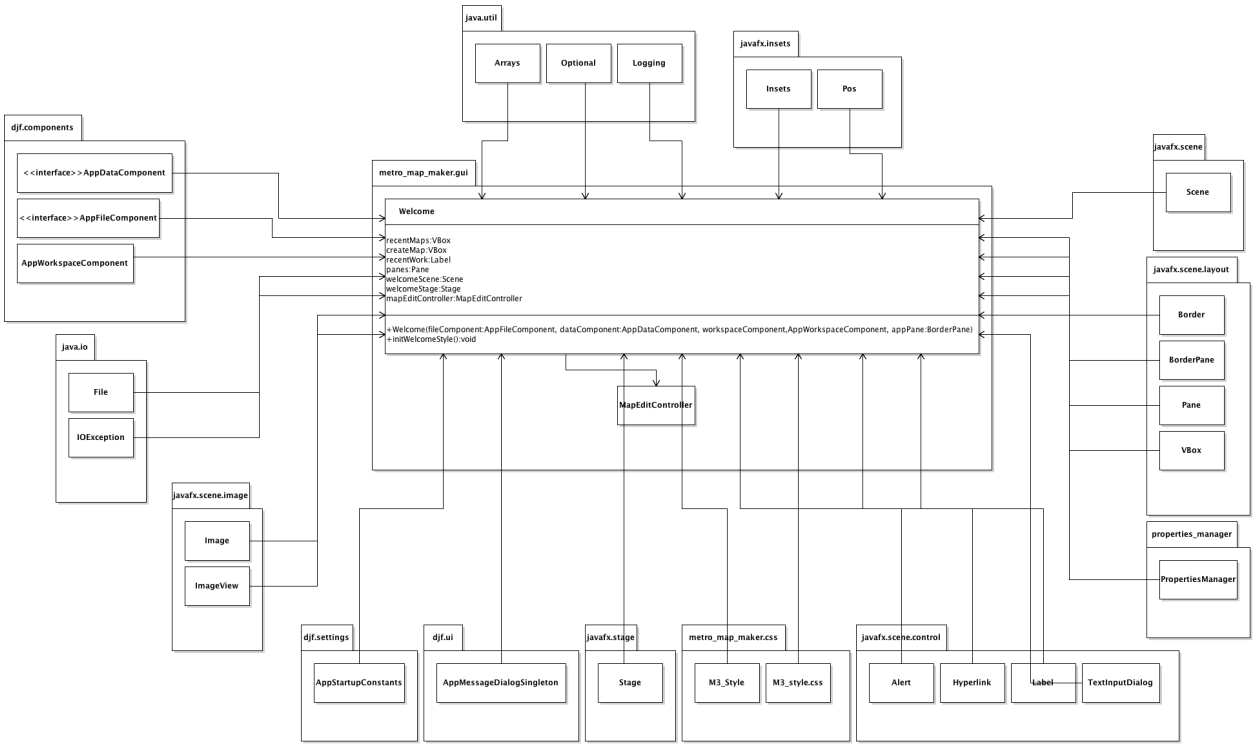


Figure 3.6: Detailed Welcome UML Class Diagram

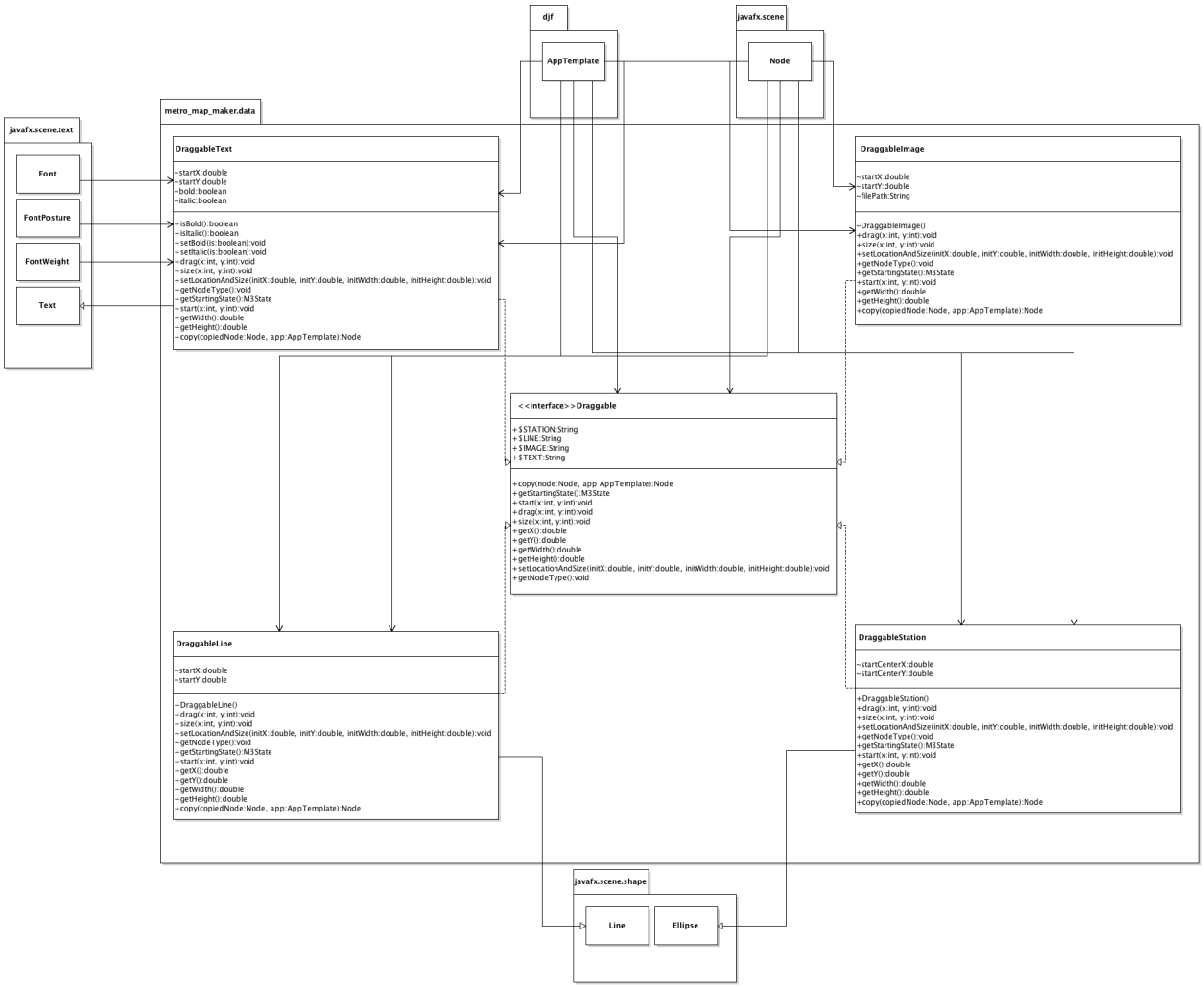
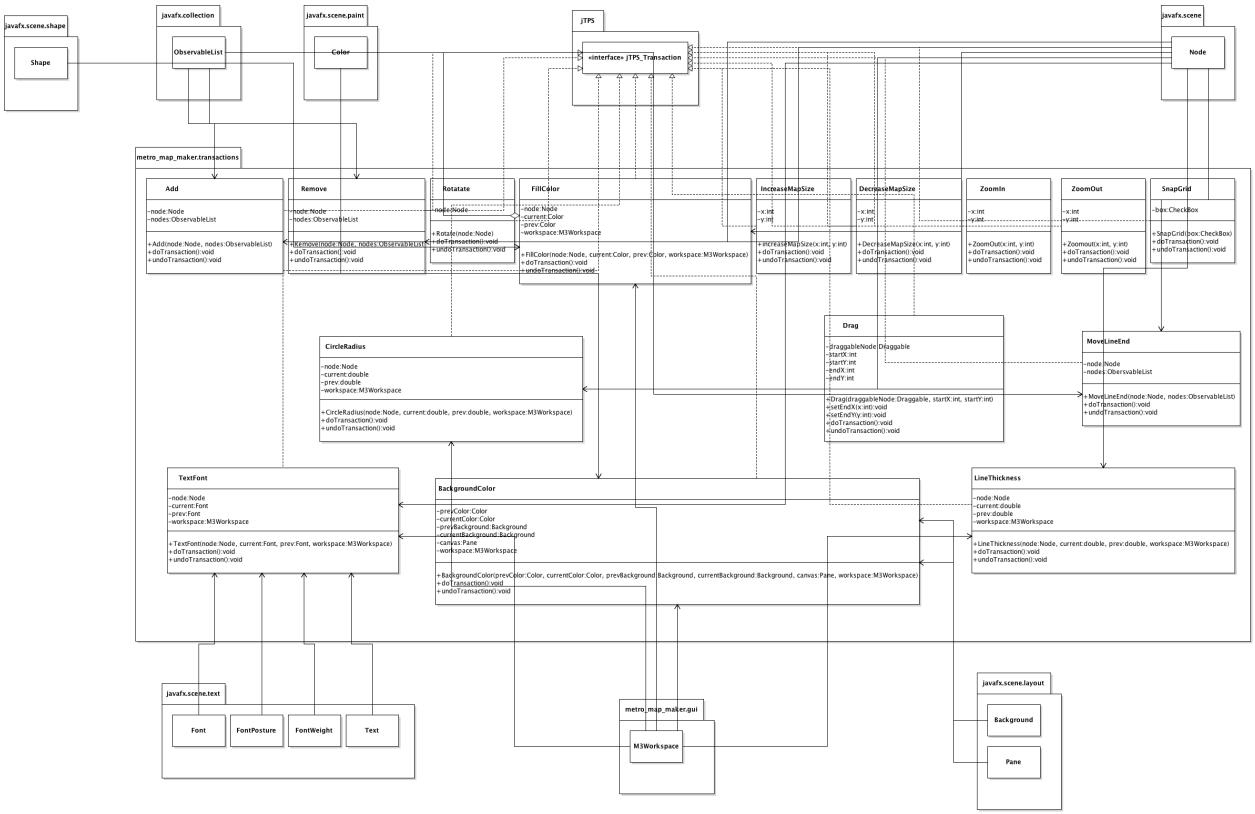


Figure 3.7: Detailed Draggable, DraggableImage, DraggableLine, DraggableStation, and DraggableText UML Class Diagrams



Figured 3.8: Detailed Transactions UML Class Diagrams

4. Method-Level Design Viewpoint

Now that the general architecture of the classes has been determined, it is time to specify how data will flow through the system. The following UML Sequence Diagrams describe the methods called within the code to be developed in order to provide the appropriate event responses.

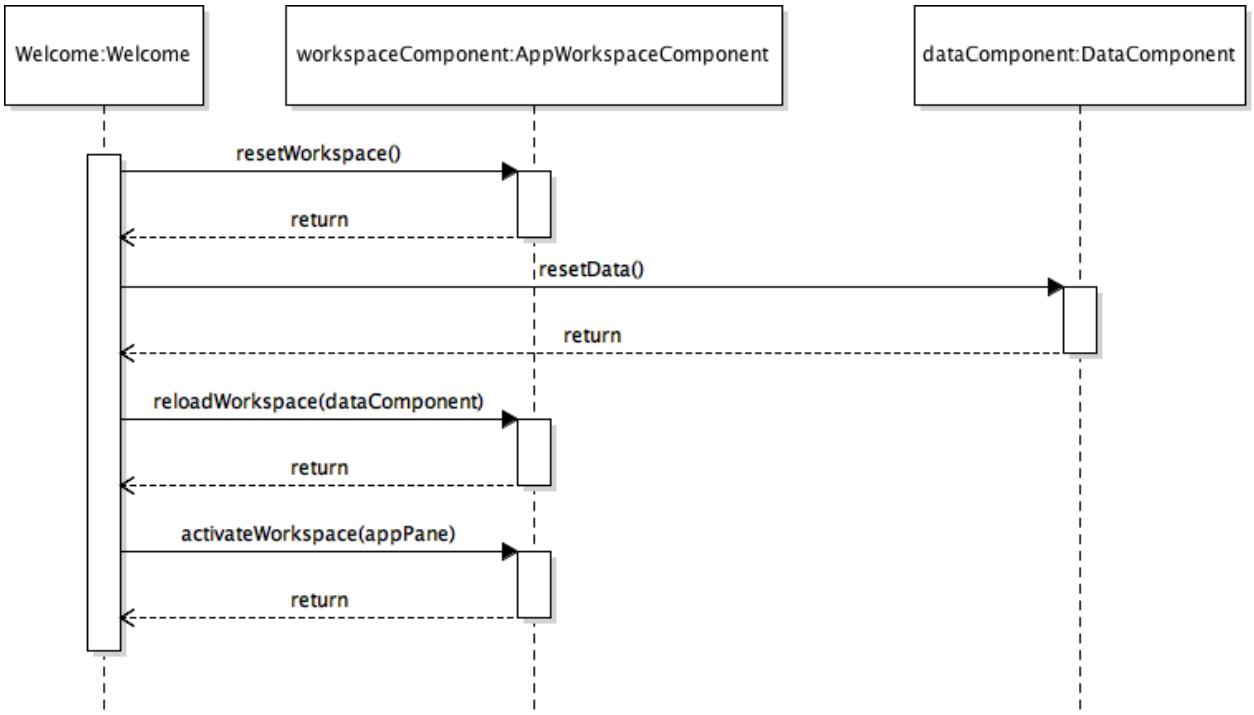


Figure 4.1: Use Case 2.1 Sequence Diagram

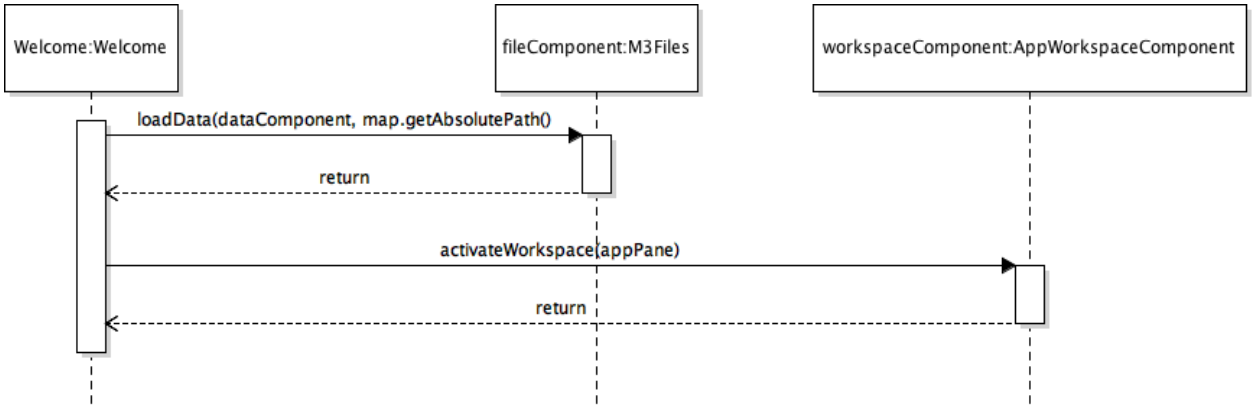


Figure 4.2: Use Case 2.2 Sequence Diagram

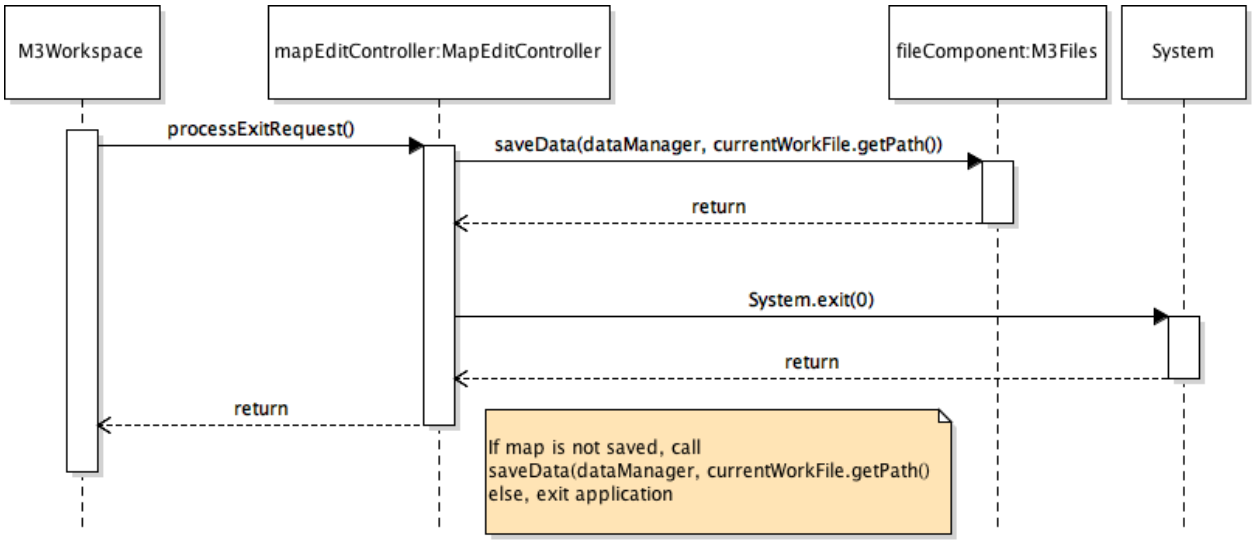


Figure 4.3: Use Case 2.3 Sequence Diagram

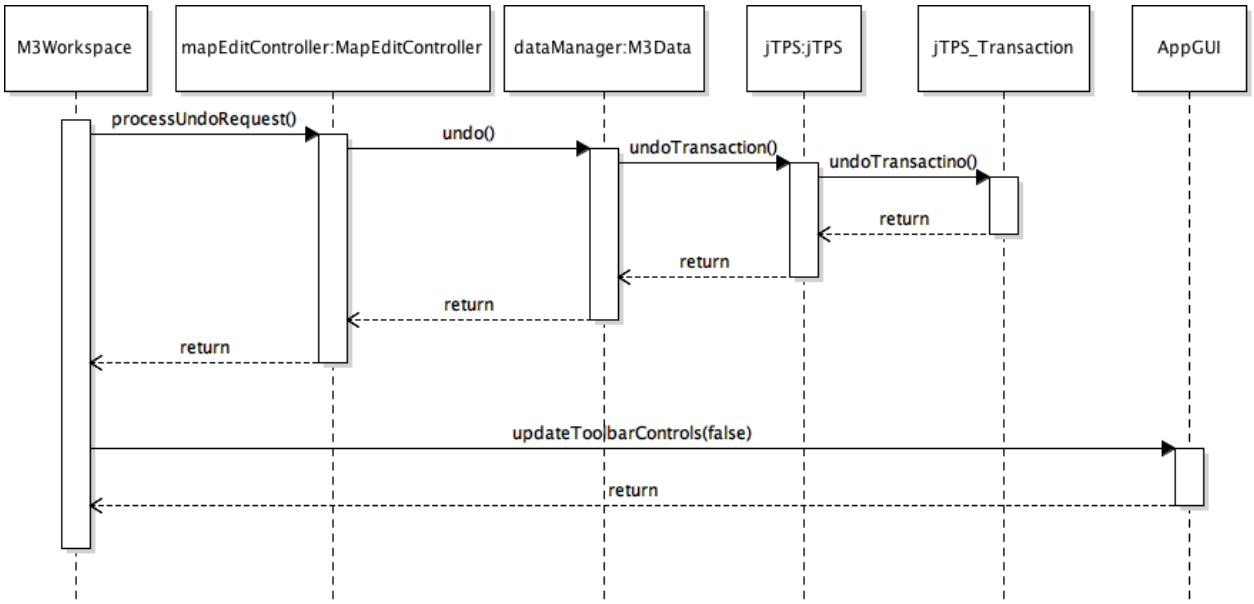


Figure 4.4: Use Case 2.9 Sequence Diagram

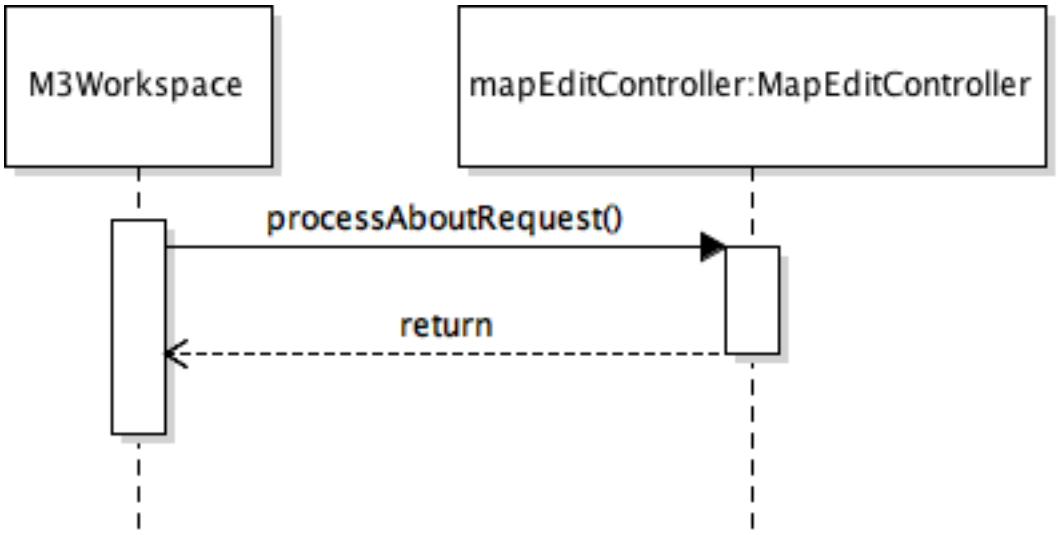


Figure 4.5: Use Case 2.11 Sequence Diagram

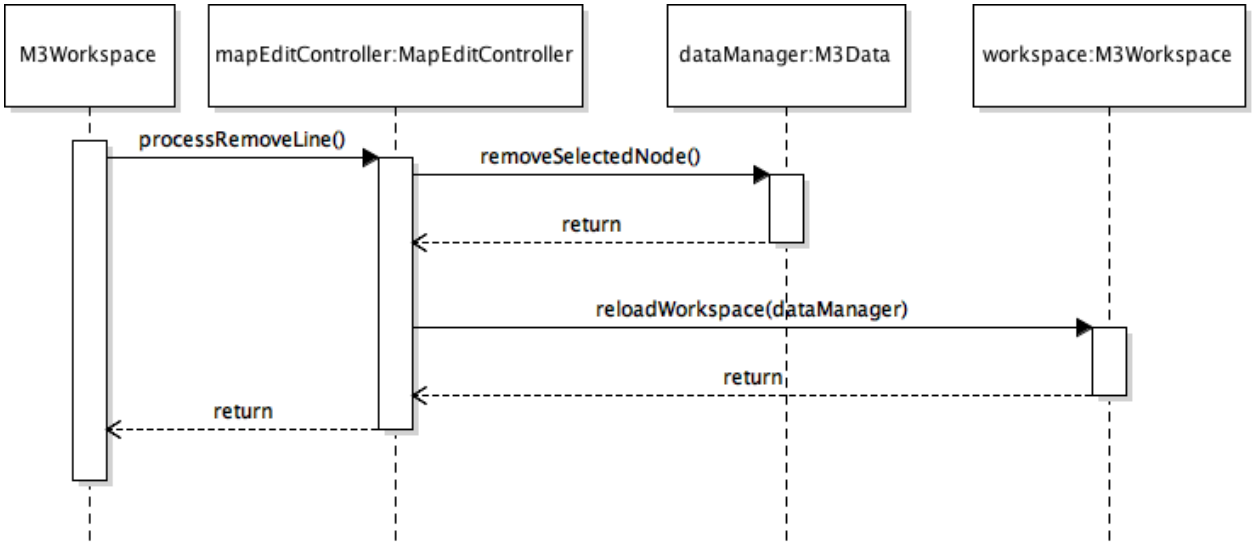


Figure 4.6: Use Case 2.13 Sequence Diagram

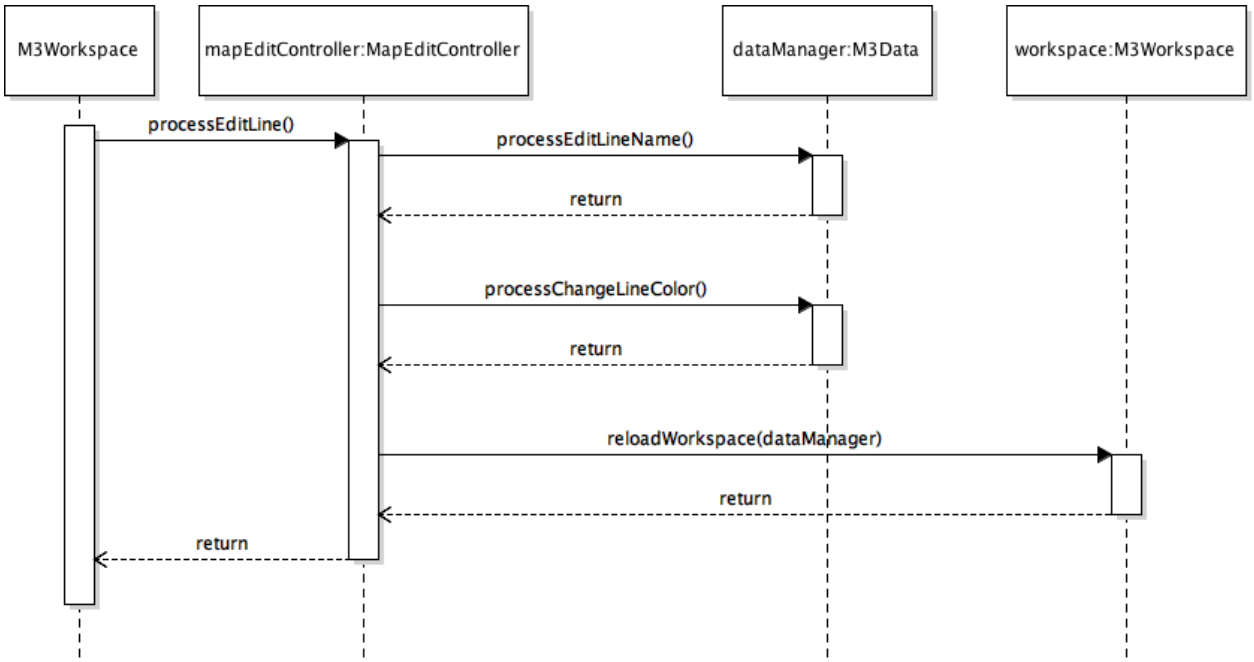


Figure 4.7: Use Case 2.14 Sequence Diagram

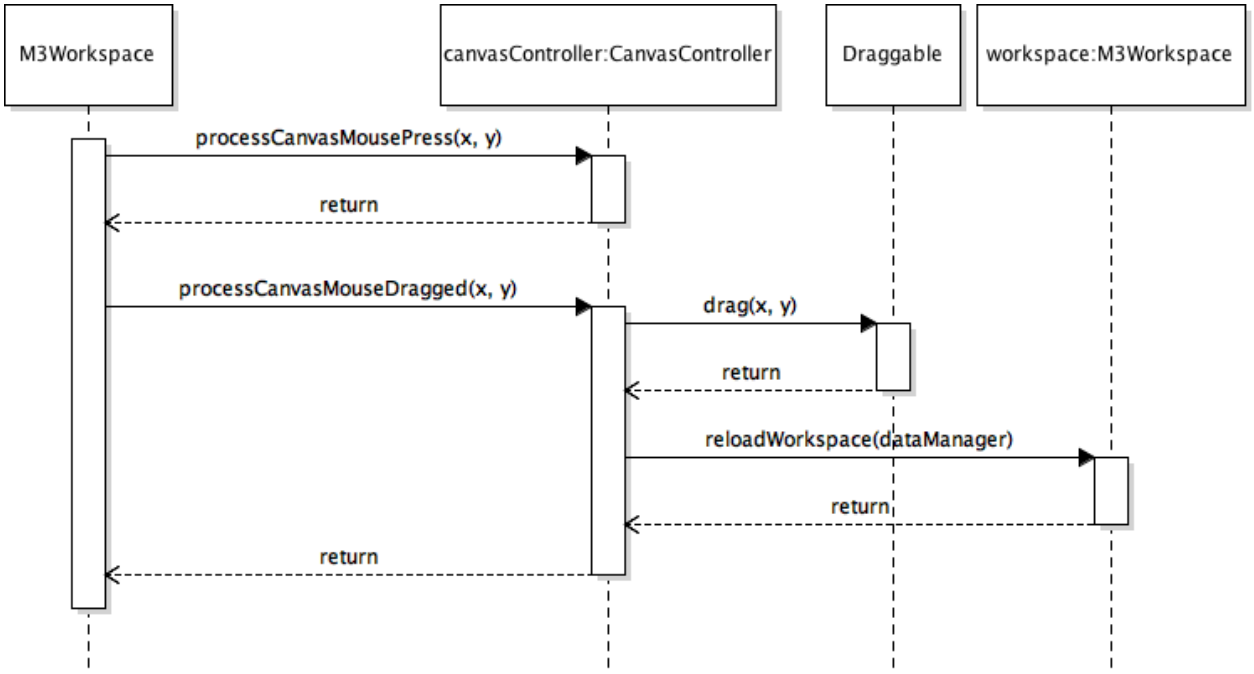


Figure 4.8: Use Case 2.15 Sequence Diagram

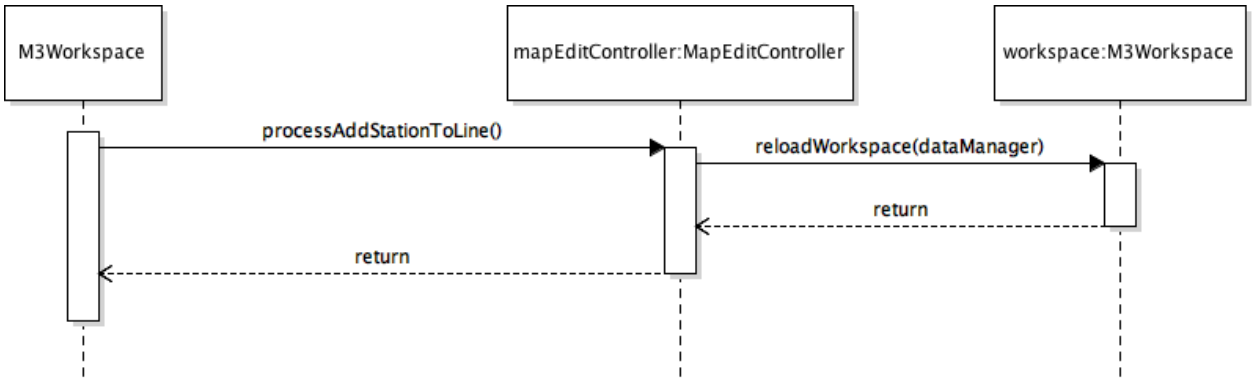


Figure 4.9: Use Case 2.16 Sequence Diagram

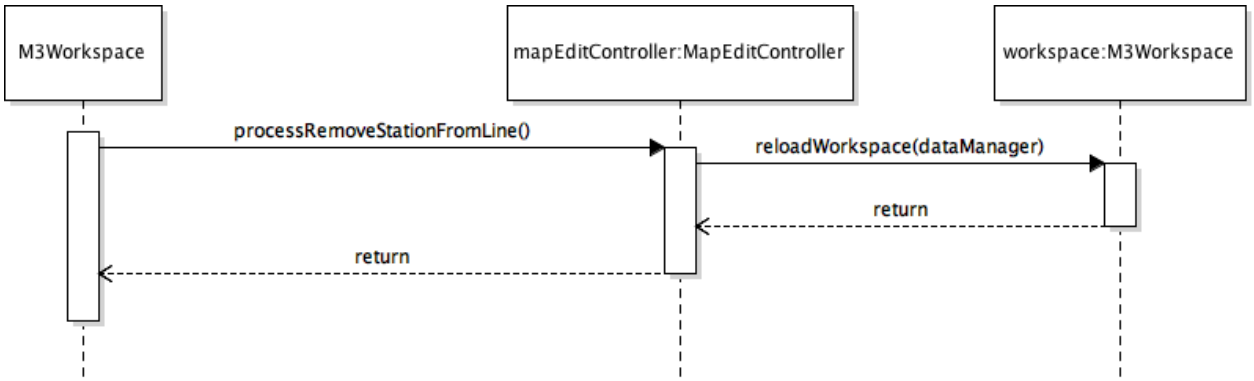


Figure 4.10: Use Case 2.17 Sequence Diagram

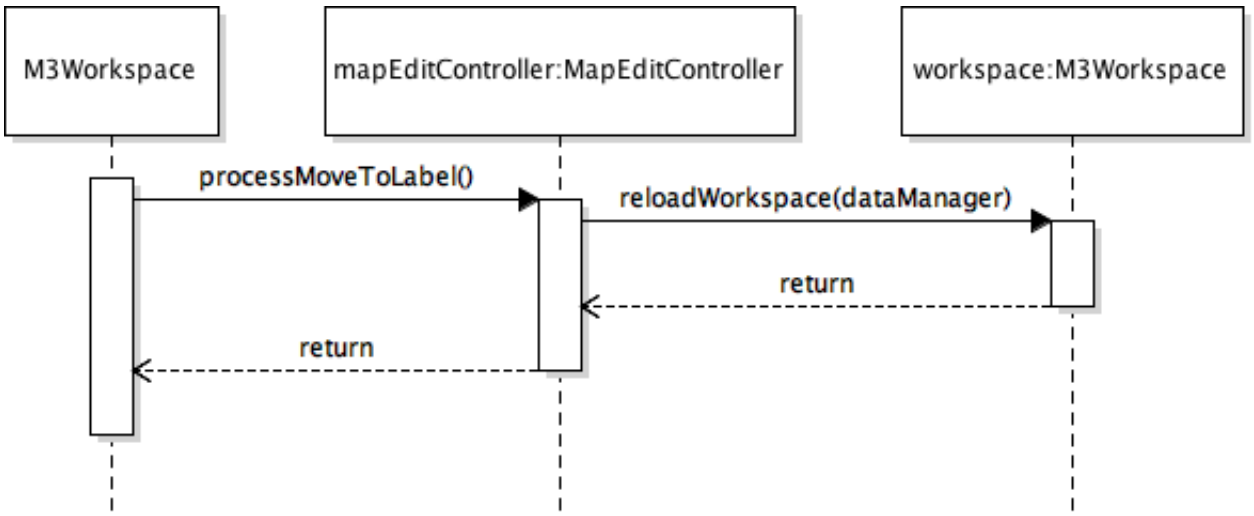


Figure 4.11: Use Case 2.23 Sequence Diagram

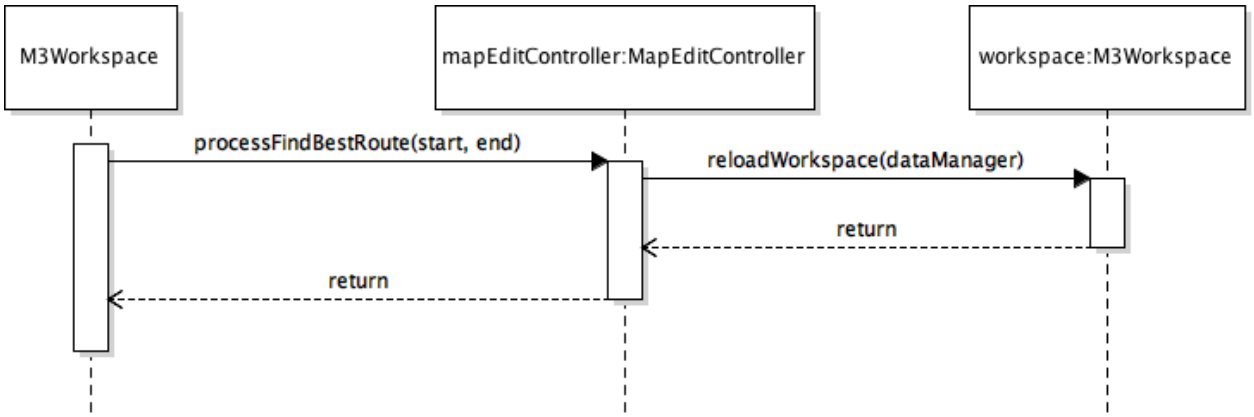


Figure 4.12: Use Case 2.27 Sequence Diagram

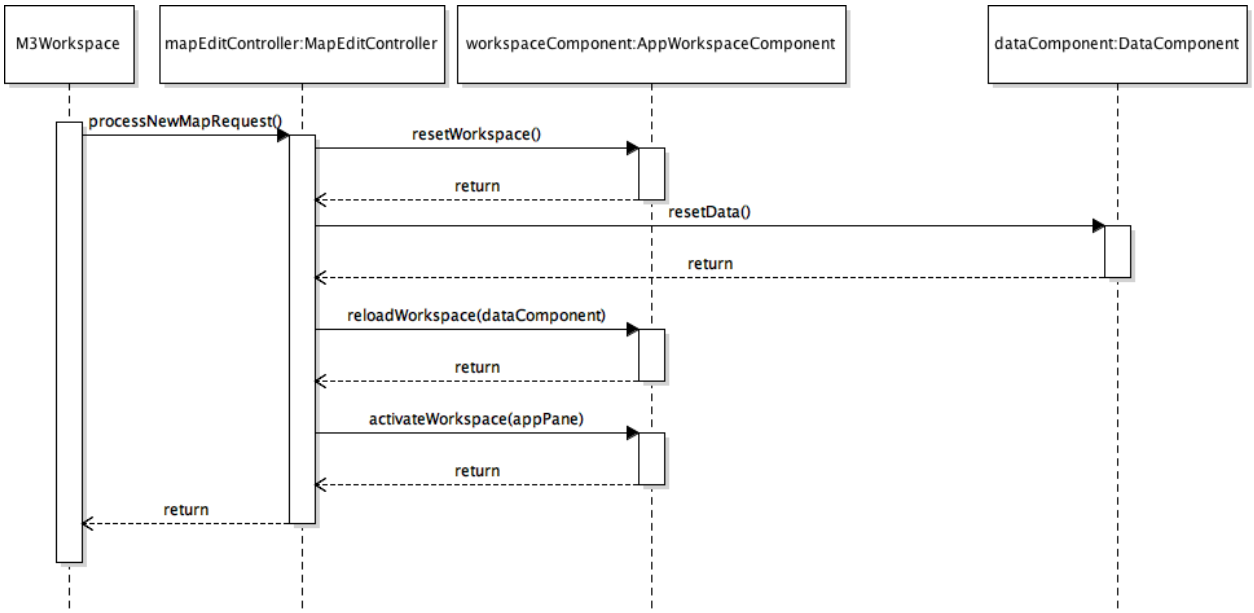


Figure 4.13: Use Case 2.4 Sequence Diagram

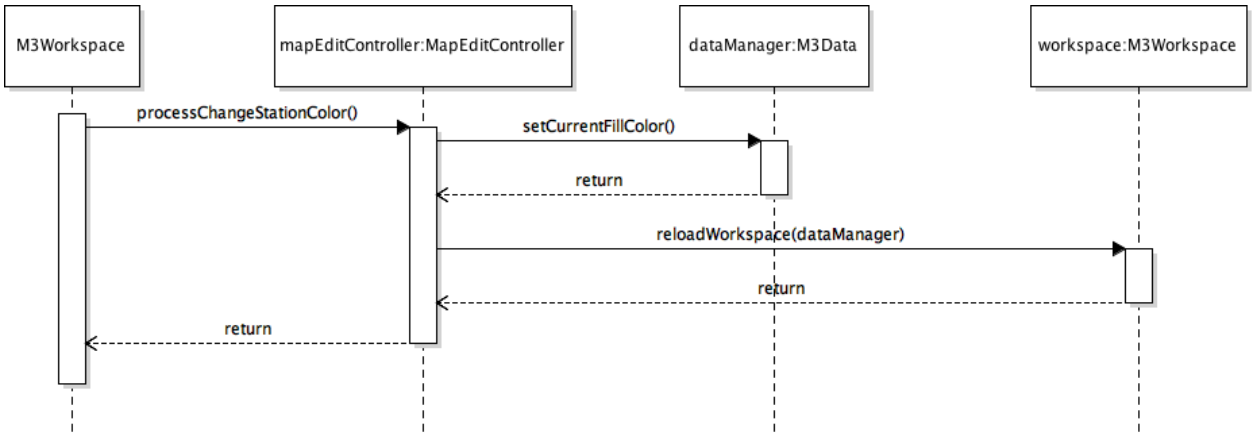


Figure 4.14: Use Case 2.25 Sequence Diagram

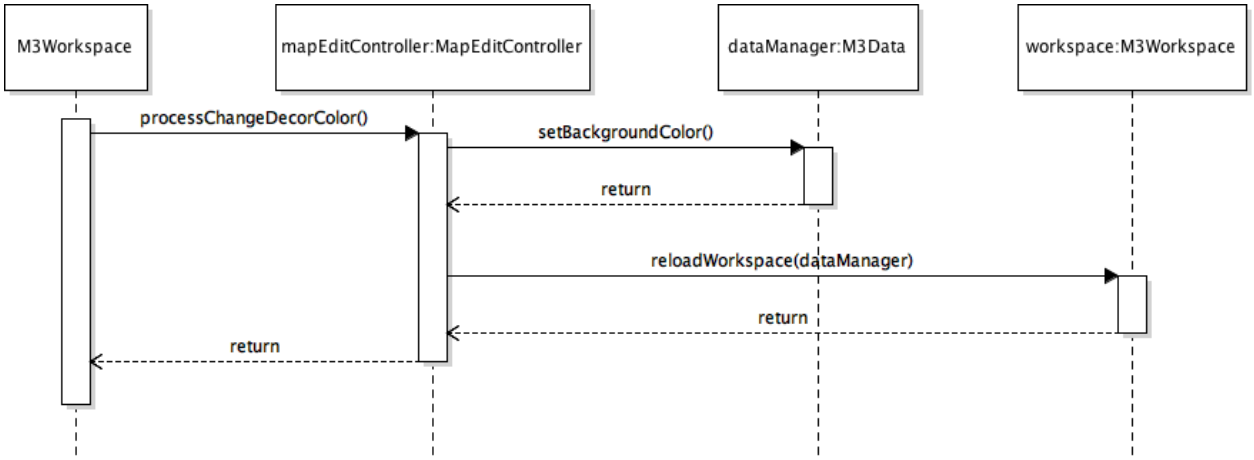


Figure 4.15: Use Case 2.28 Sequence Diagram

5. File Structure and Formats

Note that the DesktopJavaFramework will be provided inside DesktopJavaFramework.jar, a Java ARchive file that will encapsulate the entire framework. This should be imported into the necessary project for the MetroMapMaker application and will be included in the deployment of a single, executable JAR file titled MetroMapMaker.jar. Note that all necessary data and art files must accompany this program. Figure 5.1 specifies the necessary file structure the launched application should use. Note that all necessary images should of course go in the image directory. Also note that this application will also use CSS, JSON, and XML file formats.

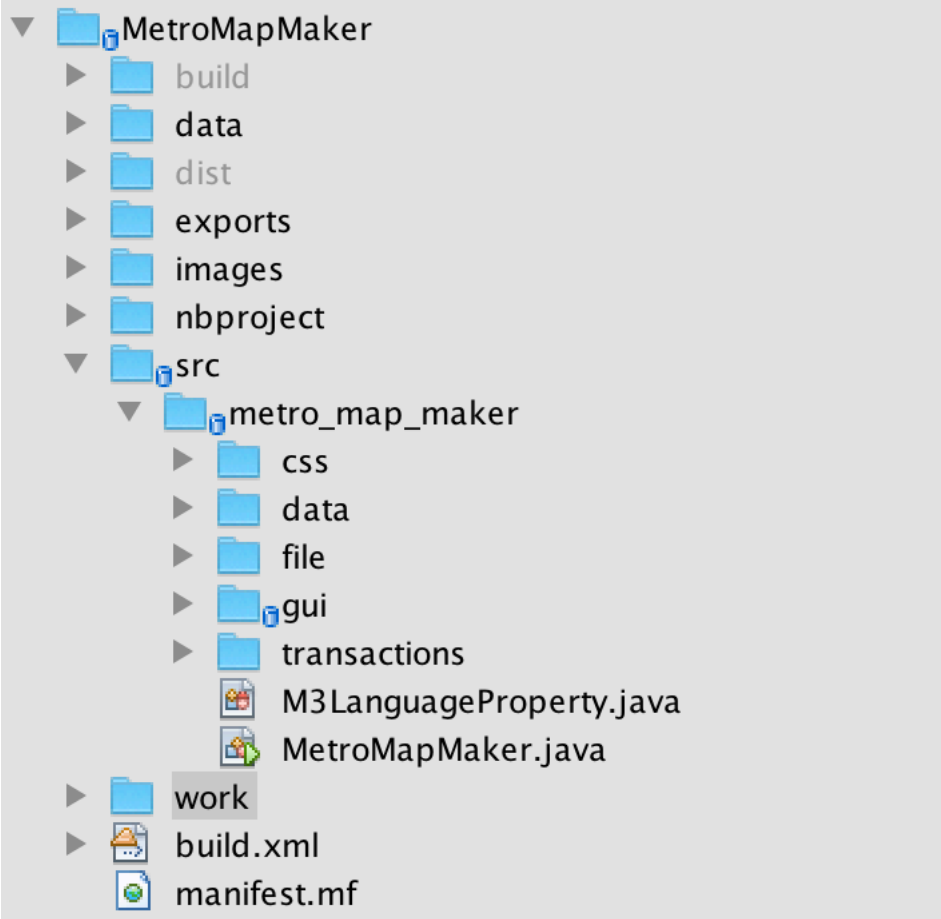


Figure 5.1: Metro Map Maker File Structure

6. Supporting Information

Note that this document should serve as a reference for those implementing the code, so we'll provide a table of contents to help quickly find important sections.

6.1 Table of contents

6.2 Appendixes

N/A