THE METRO MAP MAKERTMO

Software Design Description

Author: Naim Youssif Traore

Debugging Enterprises TM

October, 2017

Version 1.0

**1 Introduction**

Finding your way around a new city can be challenging so many people look to the Internet for help. Cities with subway systems typically provide maps to help one navigate from one stop to another across a number of intersecting lines. These maps let one chart which lines to take and how many stops it will be before one arrives and the user can typically choose between multiple routes.

The ***Metro Map Maker (i.e. M3)*** application will provide the user with a set of tools to build graphical representations of city subway systems with named lines and named stops and intersecting lines and landmarks. It will also provide a means for calculating the best route to take to journey from one particular station to another. Finally, it will provide an export feature such that it may export a generated map and associated metro system information to a format that can be used by a corresponding Web application that will be able to make use of it.

* 1. **Purpose**

The purpose of this document is to specify how our ***Metro Map Maker*** program should look and operate. The intended audience for this document is all the members of the development team, those who will design the maps for use with the Web application, and the potential users of such an application. This document serves as an agreement among all parties and as a reference for how the map creation tool should ultimately be constructed. Upon completing the reading of this document, one should clearly visualize how the application will look and operate.

* 1. **Scope**

For this project the goal is for users to easily make and edit subway maps. There will be an emphasis on ease of use. Note that there will be a common export format that will be provided for exported subway system data such that all maps can be used by a uniform application.

**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”.

**JavaScript** – the default scripting language of the Web, JavaScript is provided to pages in the form of text files with code that can be loaded and executed when a page loads so as to dynamically generate page content in the DOM.

**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.

**References**

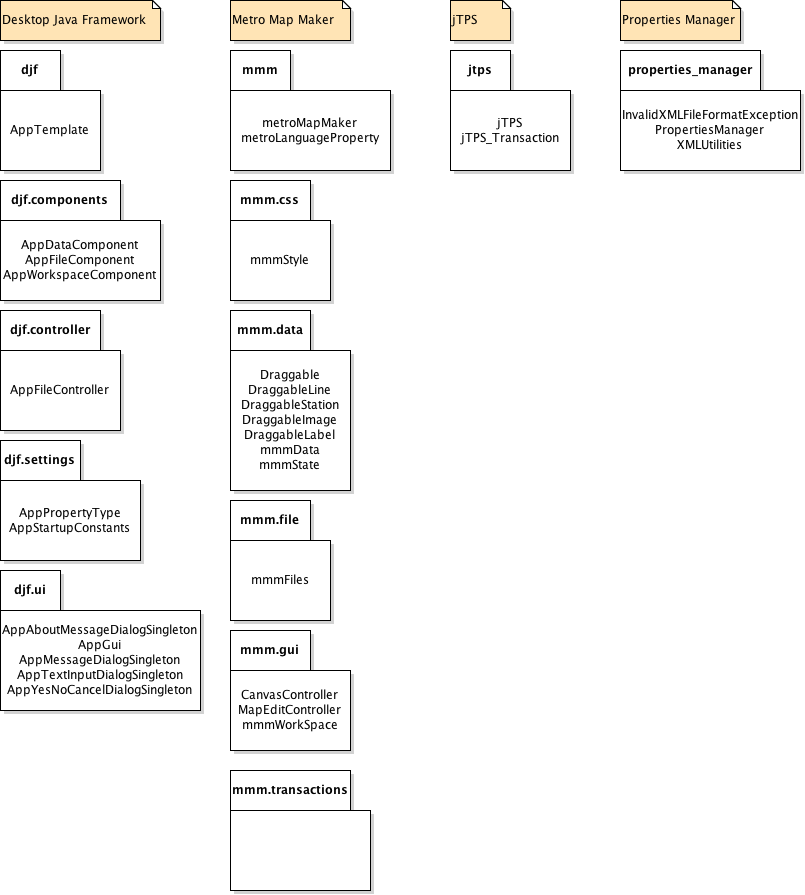
**IEEE Std 830TM-1998 (R2009) –** IEEE Recommended Practice for Software Requirements Specification

**Overview**

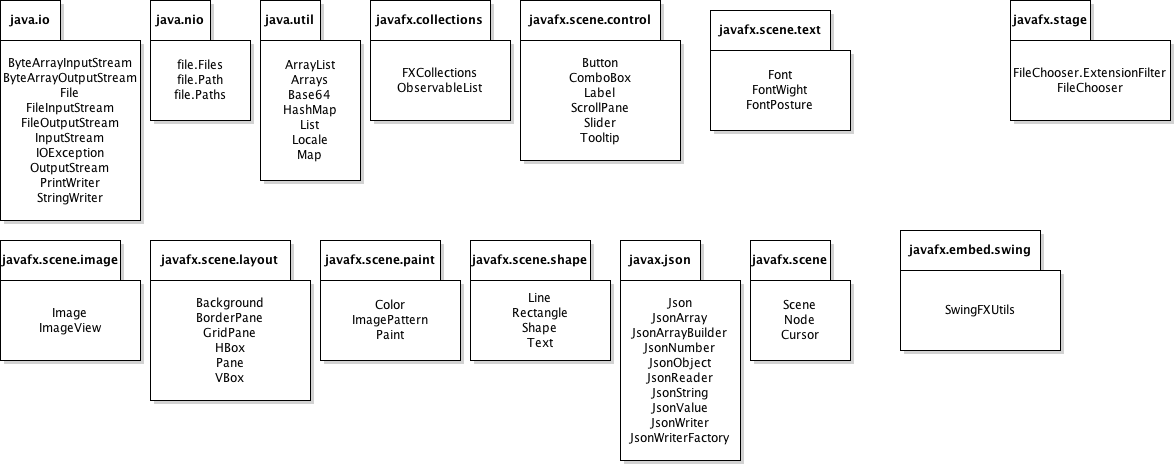
This Software Design Description document provides a working design for the Metro Map Maker software application as described in the Metro Map Maker Software Requirements Specification. Note that all parties in the implementation stage must agree upon all connections between components before proceeding with the implementation stage. Section 2 of this document will provide the Package - Level View point, 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.

1. **Package - Level Design Viewpoint** 
   1. **Desktop Java Framework, Metro Map Maker, jTPS and Properties Manager overview**

Figure 2.1 : Design Packages Overview



* 1. **Java API Usage**



**Figure 2.2:: Java API Classes and Packages to Be Used**

|  |  |
| --- | --- |
| Class/Interface | Use |
| ByteArrayInputStream | For storing bytes that may be read from the stream. |
| ByteArrayOutputStream | For writing data into a byte array. |
| File | For representing files and directory pathnames. |
| FileInputStream | For obtaining input bytes from a file in a file system. |
| FileOutputStream | For writing data to a File. |
| InputStream | For representing an input stream of bytes. |
| IOException | For signaling that an I/O exception of some sort has occurred. |
| OutputStream | For representing an output stream of bytes. |
| PrintWriter | For printing a formatted representation of objects to a text-output stream. |
| StringWriter | For collecting output in a string buffer, which can then be used to construct a string. |

Table 2.1: Uses for classes in the Java API’s java.nio.File package

|  |  |
| --- | --- |
| Class/Interface | Use |
| Files | For methods that operate on files. |
| Path | For locating a file in a file system |
| Paths | For getting a Path object from a path string or URI. |

Table 2.2: Uses for classes in the Java API’s java.io package

|  |  |
| --- | --- |
| Class/Interface | Use |
| ArrayList | For a resizable-array implementation of the List interface. |
| Arrays | For storing a non-resizable list. |
| Base64 | To encode strings to Base64 before storing them as JSON ojects. |
| HashMap | For a hash table based implementation of the Map interface. |
| List | For an ordered collection. |
| Map | For an object that maps keys to values. |
| Locale | For a representation of specific geographical, political, or cultural region. |

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

|  |  |
| --- | --- |
| Class/Interface | Use |
| FXCollections | For static methods that are 1:1 copies of java.util.Collections methods. |
| ObservableList | For a list that allows listeners to track changes when they occur. |

Table 2.4: Uses for classes in the Java API’s javafx.collections package

|  |  |
| --- | --- |
| Class/Interface | Use |
| Button | For making simple button controls. |
| ComboBox | For making multiple choice combo boxes. |
| Label | For a non-editable text control. |
| ScrollPane | For a control that provides a scrolled and clipped view of its contents. |
| Tooltip | For showing additional information about a control when the control is hovered over by the mouse. |

Table 2.5: Uses for classes in the Java API’s javafx.scene.control package

|  |  |
| --- | --- |
| Class/Interface | Use |
| Cursor | For encapsulating the bitmap representation of the mouse cursor. |
| Node | For a base class used as scene graph nodes. |
| Scene | For containing all content in a scene graph. |

Table 2.6: Uses for classes in the Java API’s javafx.scene package

|  |  |
| --- | --- |
| Class/Interface | Use |
| Image | For representing graphical images and for loading images. |
| ImageView | For painting images loaded with Image class. |

Table 2.7: Uses for classes in the Java API’s javafx.scene.image package

|  |  |
| --- | --- |
| Class/Interface | Use |
| BorderPane | For laying out children in top, left, right, bottom, and center positions. |
| GridPane | For laying out children within a flexible grid of rows and columns. |
| HBox | For laying out children in a single horizontal row. |
| Pane | For laying out panes. |
| VBox | For laying out children in a single vertical column. |

Table 2.8: Uses for classes in the Java API’s javafx.scene.layout package

|  |  |
| --- | --- |
| Class/Interface | Use |
| Line | For representing a line segment. |
| Rectangle | For representing a rectangle. |
| Text | For defining a node that displays text. |
| Shape | For creating a shape object. |

Table 2.9: Uses for classes in the Java API’s javafx.scene.shape package

|  |  |
| --- | --- |
| Class/Interface | Use |
| Font | For representation of fonts. |
| FontPosture | For italicizing text. |
| FontWeight | For bolding text. |

Table 2.9: Uses for classes in the Java API’s javafx.scene.text package

|  |  |
| --- | --- |
| Class/Interface | Use |
| Json | For creating JSON processing objects when saving or loading data. |
| JsonArray | For representing an immutable JSON array. |
| JsonArrayBuilder | For creating JsonArray models. |
| JsonNumber | For creating an immutable JSON number value. |
| JsonObject | For representing an immutable JSON object value. |
| JsonReader | For reading a JSON object or an array structure from an input source. |
| JsonString | For creating an immutable JSON string value. |
| JsonValue | For representing an immutable JSON value. |
| JsonWriter | For writing JSON object or array structure to an output source. |
| JsonWriterFactory | For creating JsonWriter instances. |

Table 2.10: Uses for classes in the Java API’s javafx.json package

|  |  |
| --- | --- |
| Class/Interface | Use |
| Application | For exending our AppTemplate to JavaFX Application class. |

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

|  |  |
| --- | --- |
| Class/Interface | Use |
| FileChooser | For providing support of standard file dialogs. |
| FileChooser.ExtensionFilter | For filtering, which files can be chosen in a FileDialog based on the file name extensions such. |
| Stage | For creating the top level of JavaFX container. |

Table 2.12: Uses for classes in the Java API’s javafx.stage package

**3. Class-Level Design Viewpoint**

Figure 3.1: MetroMapMaker Overview UML Class Diagram

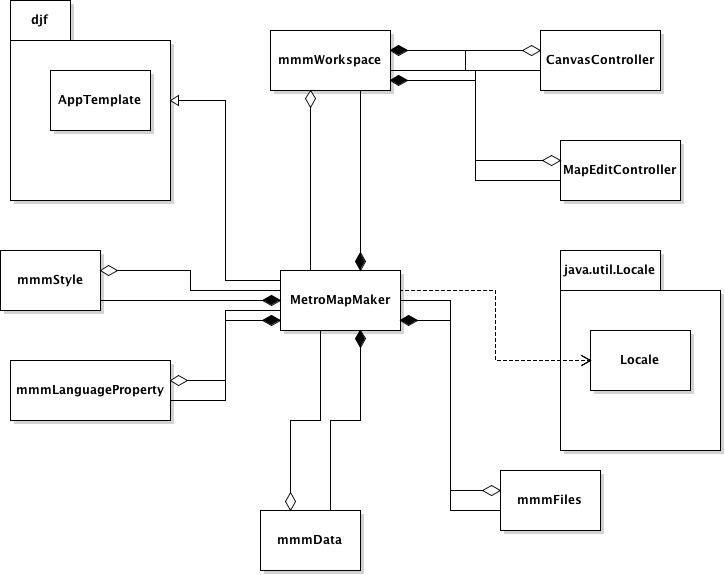


Figure 3.2: mmmFiles Overview UML Class Diagram

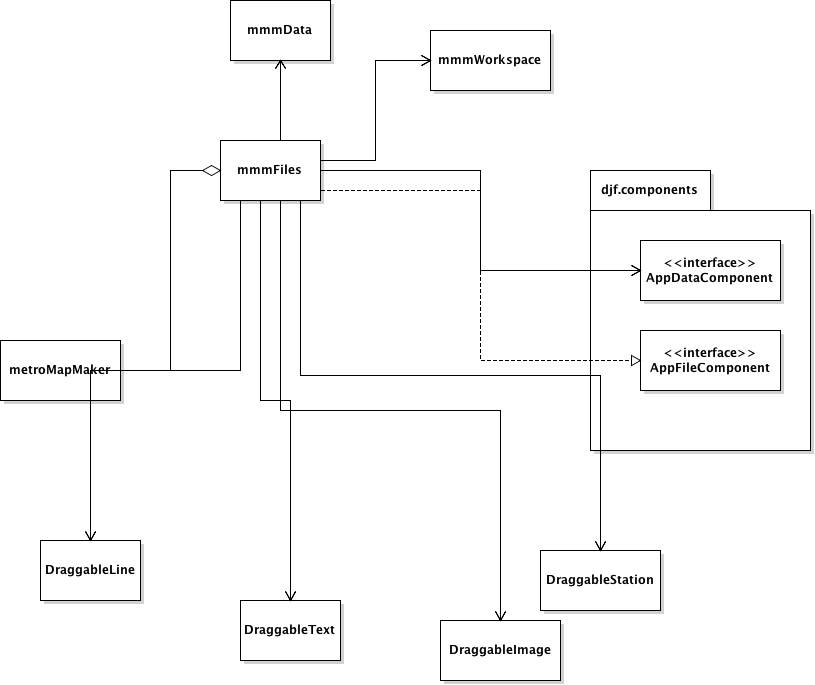


Figure 3.3: mmmData Overview UML Class Diagram

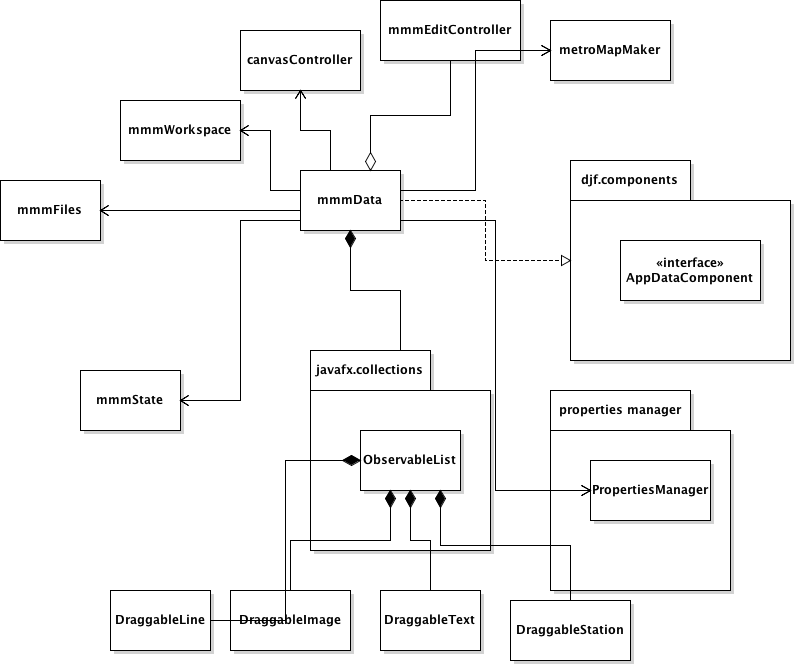


Figure 3.4: mmmEditController Overview UML Class Diagram

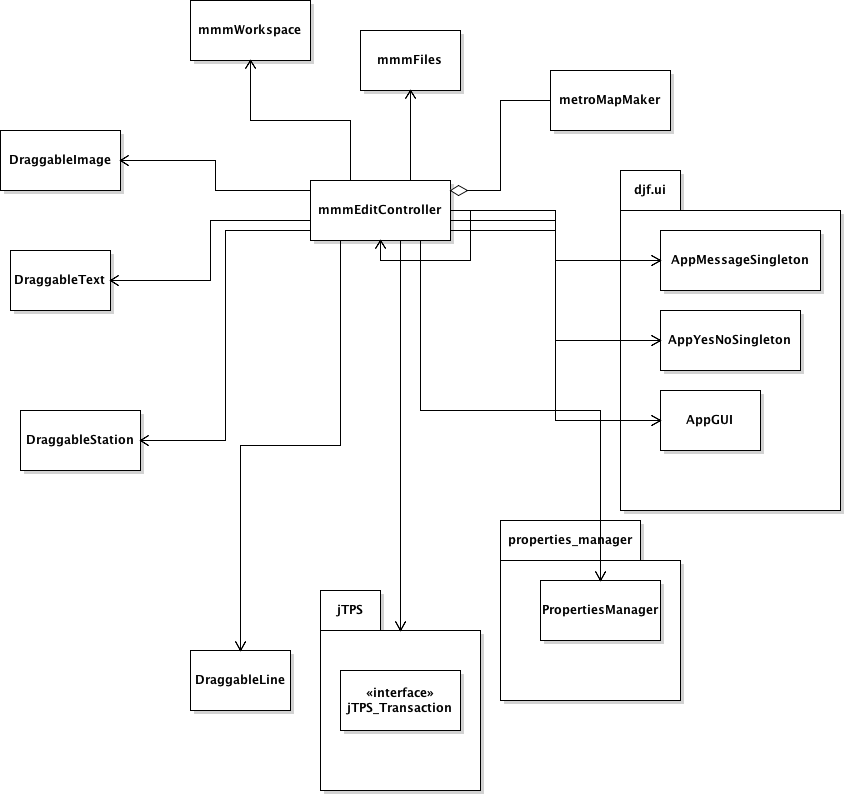


Figure 3.5: mmmWorkspace Overview UML Class Diagram

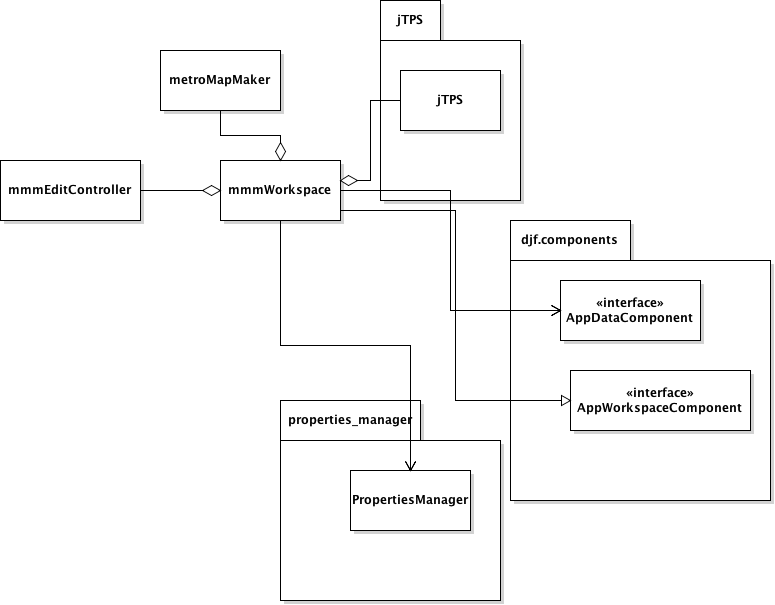
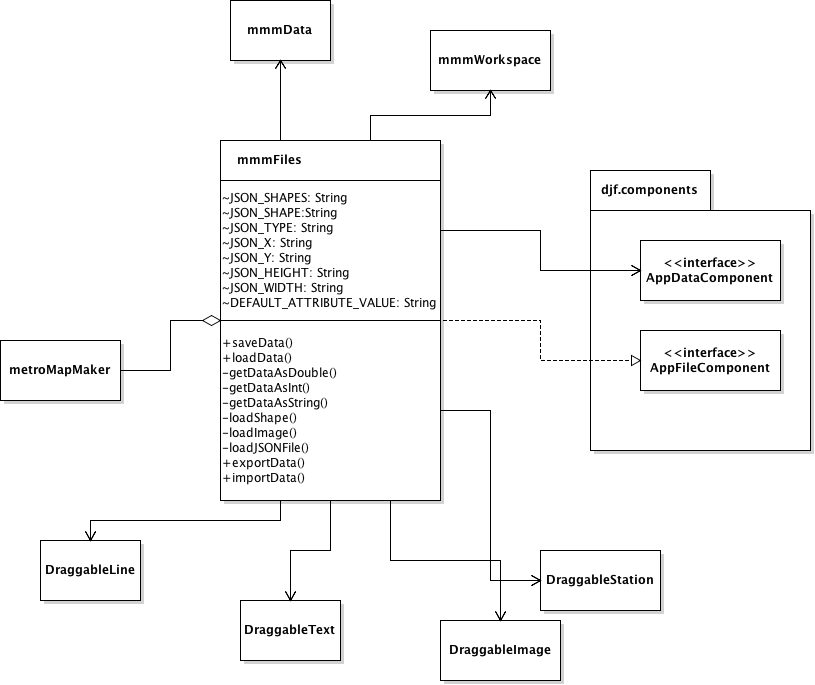
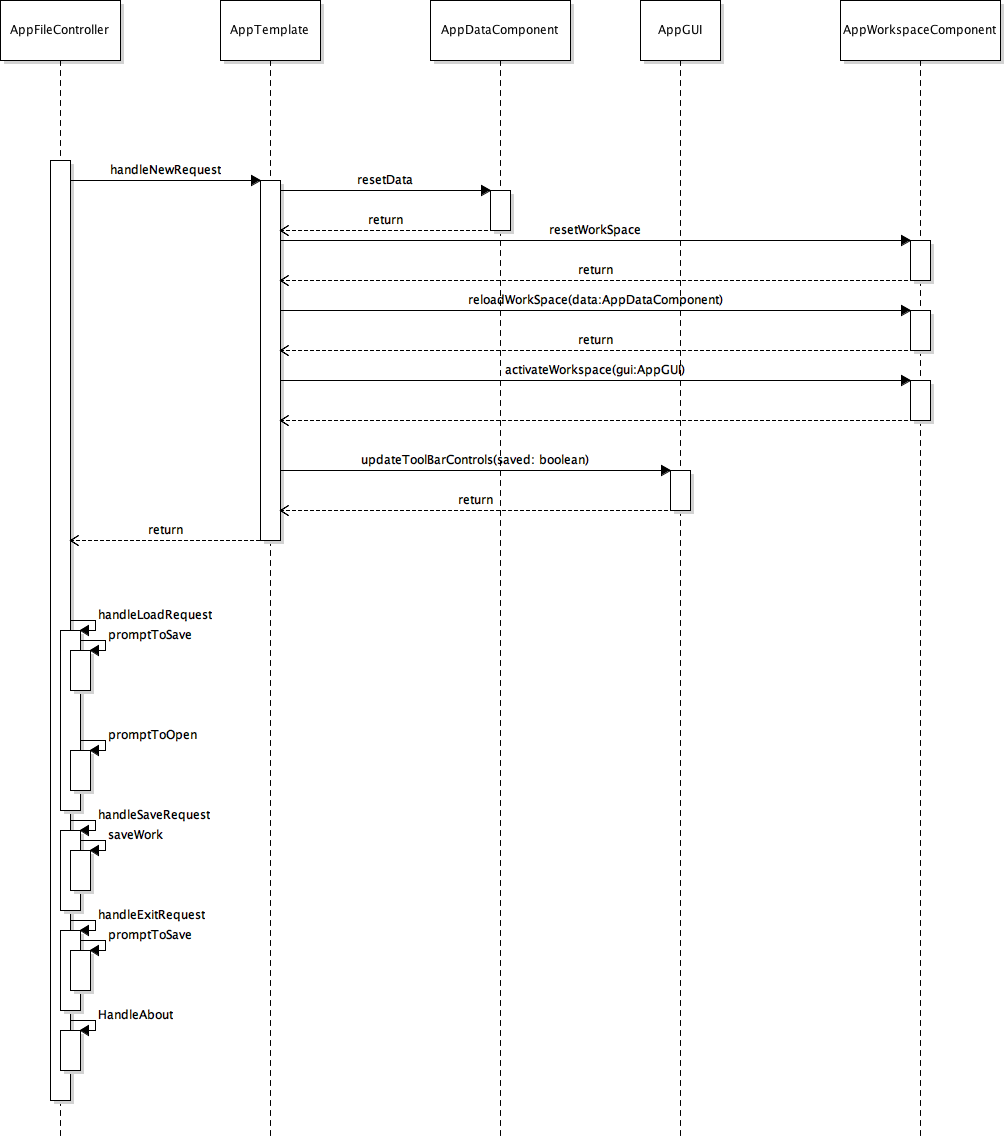


Figure 3.6: MetroMapMaker Detailed UML Class Diagram

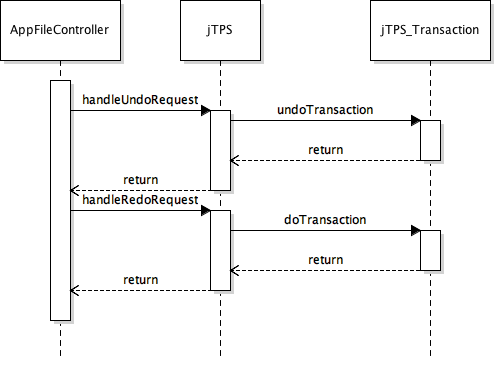


**4 . Method - Level Design Viewpoint**

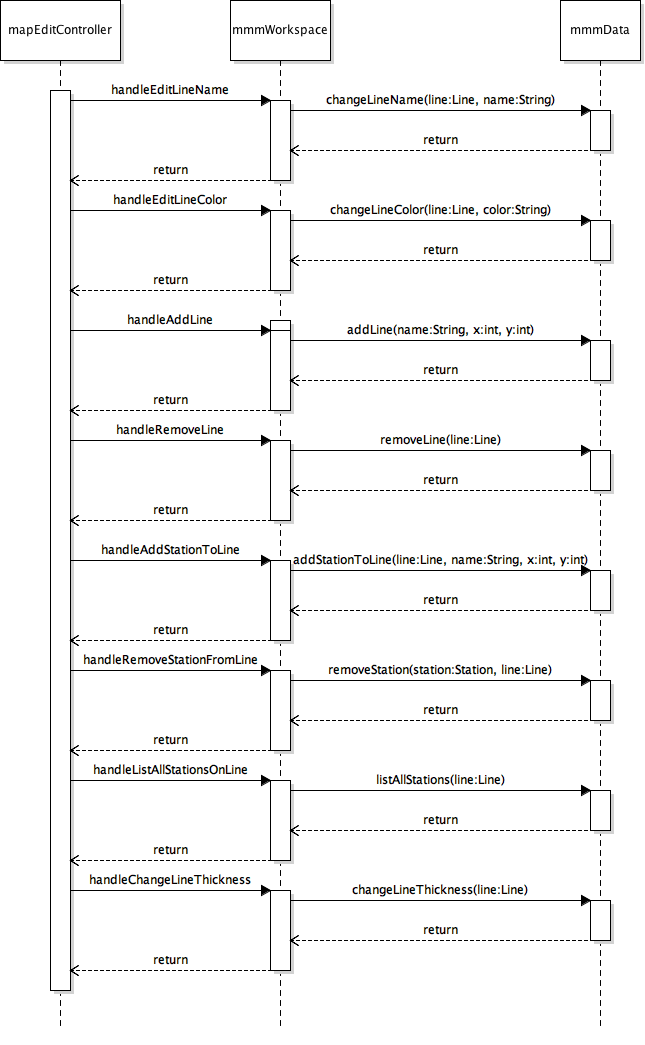
Now that the general architecture of the classes has been determined, it is time to specify how data will flow through the system. The following UML Sequence Diagrams describe the methods called within the code to be developed to provide the appropriate event responses.



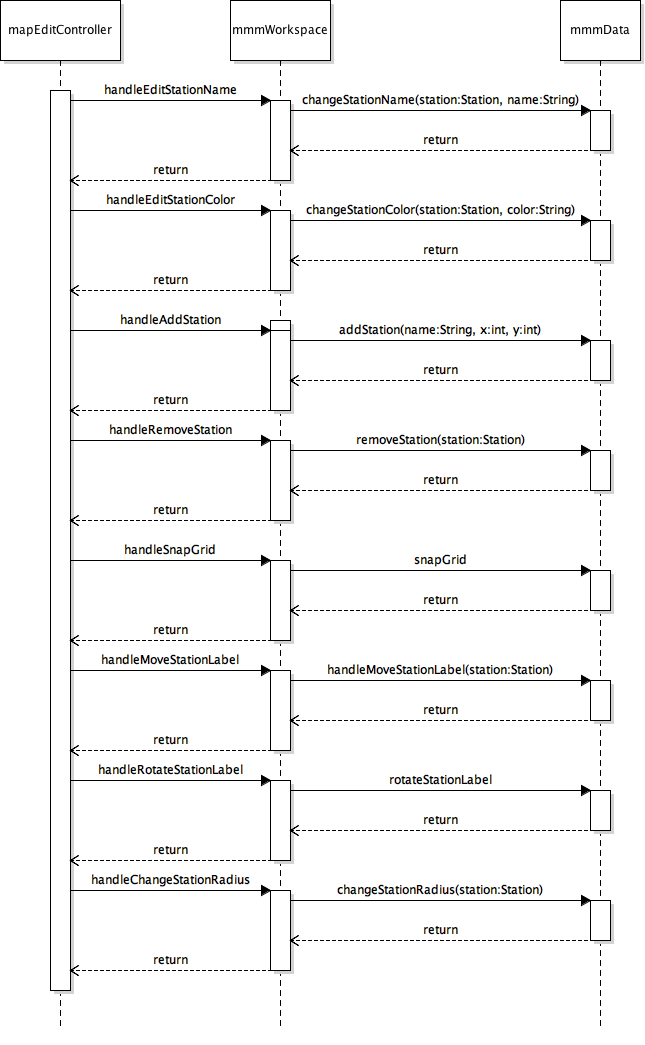
**Figure 4.1 AppFileController UML Sequence Diagram**

****

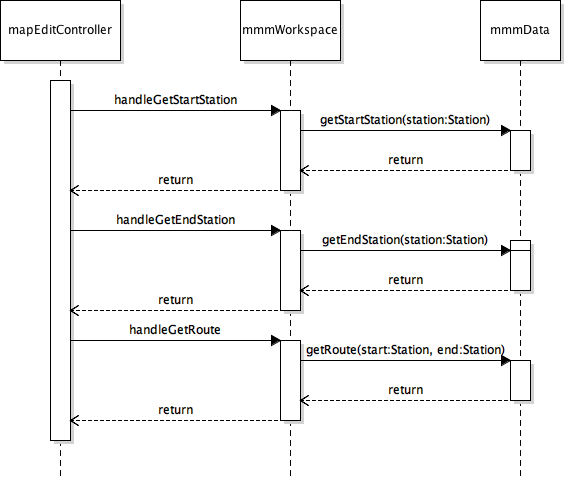
**Figure 4.2 AppFileController UML Sequence Diagram**



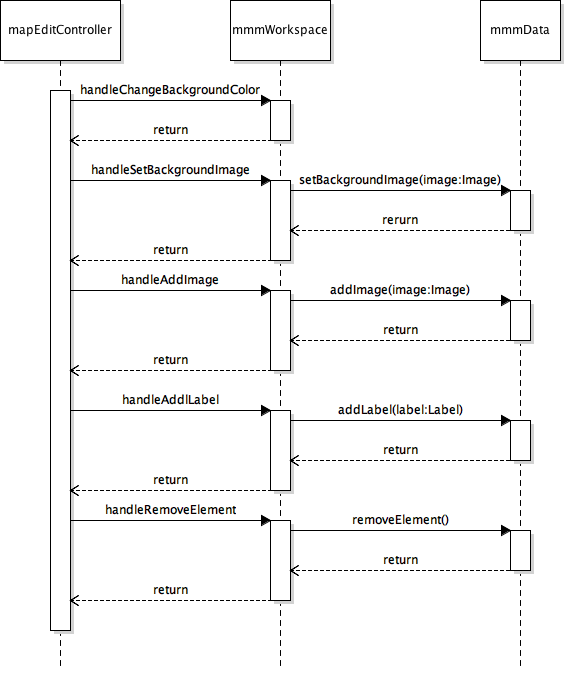
**Figure 4.3 CourseController UML Sequence Diagram**



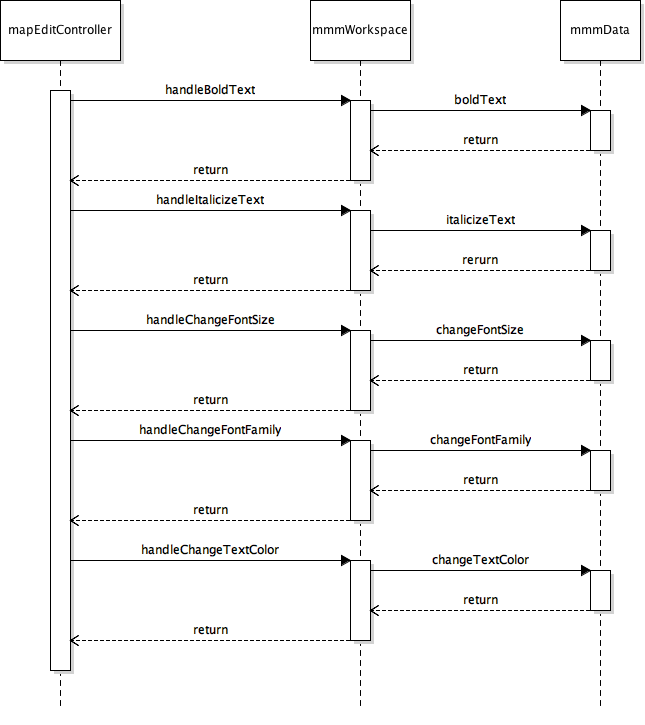
**Figure 4.4 CourseController UML Sequence Diagram**



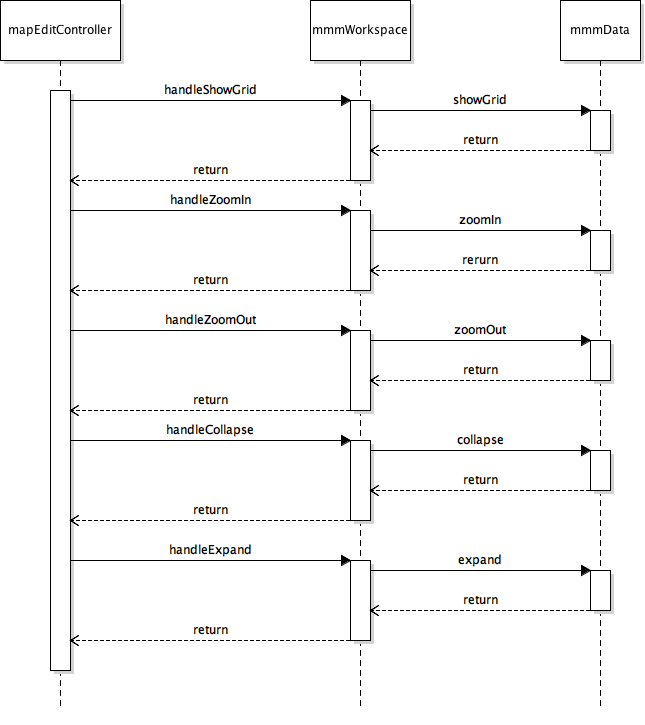
**Figure 4.5 CourseController UML Sequence Diagram**



**Figure 4.6 CourseController UML Sequence Diagram**



**Figure 4.7 CourseController UML Sequence Diagram**



**Figure 4.8 CourseController UML Sequence Diagram**

5. File Structure and Formats

Note that the DesktopJavaFramework, jTPS and PropertiesManager framework will be provided inside DesktopJavaFramework.jar, jTPS.jar and PropertiesManager.jar respectively, these are Java archive files which will encapsulate the entire frameworks. This should be imported into the necessary project for the Map Maker application and will be included in the deployment

of a single, executable JAR file titled MetroMapMaker.jar. Note that all necessary data files must accompany this program.

6. Supporting Information

Note that this document sho uld 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

1. Introduction 2

1. Purpose 2

2. Scope 2

3. Definitions, acronyms, and abbreviations 2

4. References 3

5. Overview 3

2. Package - Level Design Viewpoint 4

1. Zombiquarium and Mini Game overview 4

2. Java API Usage 5

3. Java API Usage Descriptions 5

3. Class - Level Design View point 8

4. Method - Level Design Viewpoint 14

5. File Structure and Formats 17

6.Supporting Information 18

1.Table of contents 18

2. Appendixes 18