The *Metro Map Maker* TM

Software Design Description

Author: Gordon Wu

November, 2017

Version 1.0

Abstract: This document describes the software design for the Metro Map Maker, an application used to construct maps of real world subway systems and then export them into a format that can be used by real Web sites.

Based on IEEE Std 1016TM-2009 document format

1 Introduction

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

* 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.

* 1. Definitions, acronyms, and abbreviations

Class Diagram – A UML document format that describes classes graphically. Specifically, it describes their instance variables, method headers, and relationships to other classes.

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.

* 1. References

IEEE Std 830TM-1998 (R2009) –IEEE Standard for Information Technology – Systems Design – Software Design Descriptions

The Metro Map Maker TM SRS – Debugging Enterprises’ Software Requirements Specification for the Metro Map Maker software application.

* 1. 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 provides deployment information like file structures and formats to use. Section 5 provides a Table of Contents, an Index, and References. Note that all UML Diagrams in this document were created using the VioletUML editor.

This GUI is highly recommended since it’s very easy to use for making a map. The programming will automatically help the user to find a path between 2 stations. One con is that this GUI is developed in a short period, so there might not be a lot of functionalities.

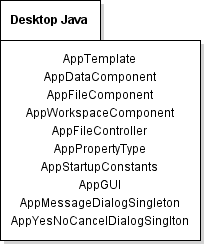
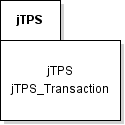
1. Package-Level Design Viewpoint

As mentioned, this design will encompass both the Metro Map Maker software application and the Desktop Java Framework to be used in its construction. In building both 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. Those API that preexisted in the Desktop Java Framework will not be described.

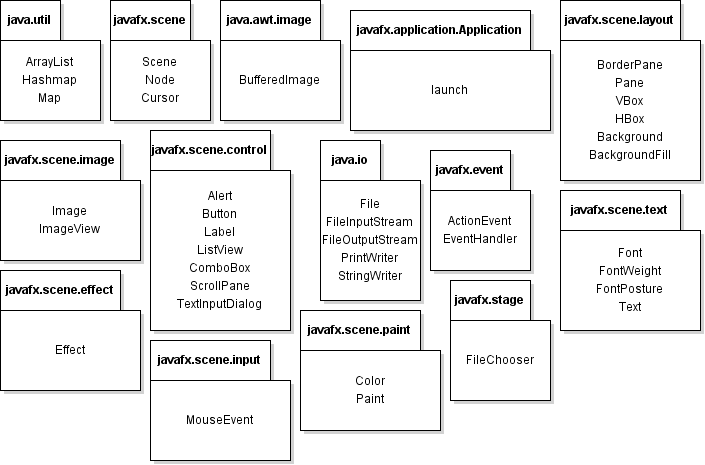
2.1 Metro Map Maker and Desktop Java Framework overview

The Metro Map Maker and Desktop Java framework will be designed and developed in tandem. Figure 2.1 specifies all the components to be developed and places all classes in the packages.

Desktop Java Framework jTPS MetroMapMaker



**Figure 2.1: Design Packages Overview**



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

2.3 Java API Usage Descriptions

Tables 2.1-2.14 below summarize how each of these classes will be used.

|  |  |
| --- | --- |
| Class/Interface | Use |
| ArrayList | For storing the station nodes and transaction nodes |
| HashMap | For storing the properties |

**Table 2.1 Uses for classes in the Java API’s java.util package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Scene | For editing the content in the stage |
| Cursor | For determining the position when mouse-clicked |
| Node | For store the station data and traction data |

**Table 2.2 Uses for classes in the Java API’s java.scene package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| BufferedImage | For import an image from a local file |

**Table 2.3 Uses for classes in the Java API’s java.awt.iamge package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Application | For launch the entire program |

**Table 2.4 Uses for classes in the Java API’s javafx.application package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| BorderPane | For organizing the AppGUI |
| Pane | For making the toolbar pane |
| VBOX | For making the workspace pane |
| HBOX | For Organizing the buttons |
| Background | For setting the background |
| BackgroundFill | For setting the color of the background |

**Table 2.5 Uses for classes in the Java API’s javafx.scene.layout package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Image | For making an image to the scene |
| ImageView | For showing images in the scene |

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

|  |  |
| --- | --- |
| Class/Interface | Use |
| Alert | For making the dialog about the information of the GUI |
| Button | For making the buttons of the GUI |
| Label | For color label and outline thinkness label |
| Slider | For changing the line thickness and station radius |
| ComboBox | For change the font family and font size |
| ScrollPane | For organizing the workspace pane |
| TextInputDialog | For editing the text |

**Table 2.7 Uses for classes in the Java API’s javafx.scene.control package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| File | For storing the map |
| FileInputStream | For loading a file |
| FileOutputStream | For outputting a file |
| PrintWriter | For writing the data to a file |
| StringWriter | For constructing a string |

**Table 2.8 Uses for classes in the Java API’s javafx.io package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| EventHandler | For setting an handler when the mouse is clicked |

**Table 2.9 Uses for classes in the Java API’s javafx.event package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Font | For setting the font |
| FontWeight | For setting the font weight |
| FontPosture | For setting the font posture |
| Text | For setting the text content |

**Table 2.10 Uses for classes in the Java API’s javafx.scene.text package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Effect | For highlighting the node when it’s selected |

**Table 2.11 Uses for classes in the Java API’s javafx.scene.effect package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| MouseEvent | For making the handler when the mouse is clicked |

**Table 2.12 Uses for classes in the Java API’s javafx.scene.input package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Color | For setting the color of (station, line, background, etc) |
| Paint | For setting the color fill |

**Table 2.13 Uses for classes in the Java API’s javafx.scene.paint package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| FileChooser | For choosing an local file |

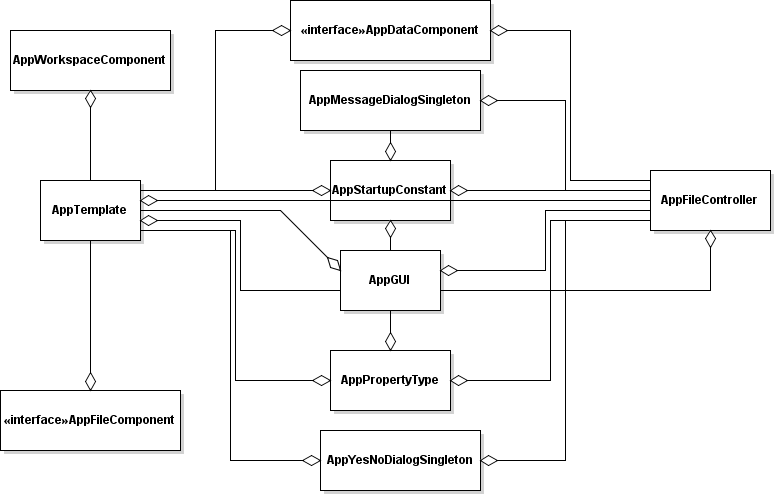
**Table 2.14 Users for classes in the Java API’s javafx.stage package**

1. Class-Level Design Viewpoint

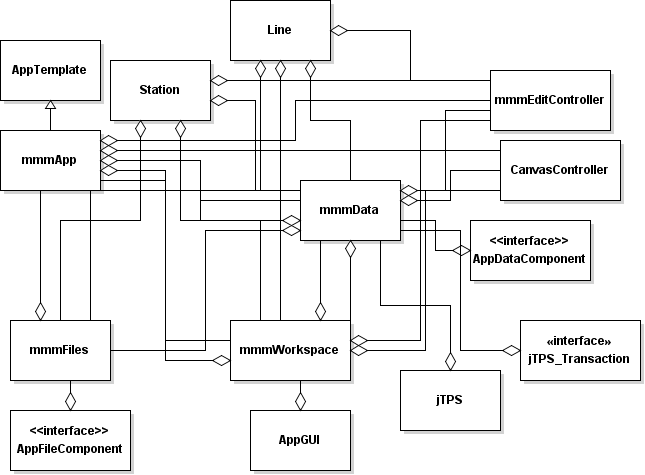
As mentioned, this design will encompass both the Metro Map Maker user interface application and the Desktop Java Framework (DJF). The AppGUI class inside the DJF plays a huge role in organizing the necessary such as the scene, pane, buttons, etc. The system launches from the mmmApp. The mmmWorkspace organizes the workspace components including those toolbar and the map view in the user interface. CanvasController senses the cursor whenever somewhere in the map view is clicked. mmmLogoEditController and the mmmData used for editing the map. Also, the function of mmmFiles is for saving and loading data. The jTPS package is used to record every transaction for the functions of undo and redo.

The find track method uses the idea of graph to implement. Stations stand for the vertexes. If two stations are next to each other, these two station will be connected with an edge. So the programing will recursively check the next stations and eventually finds the destination. A counter will be used for counting the number of stations will be need to go through to reach the destination. If this destination is found with the smallest value of the counter, the path (edges) will be shown.

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.



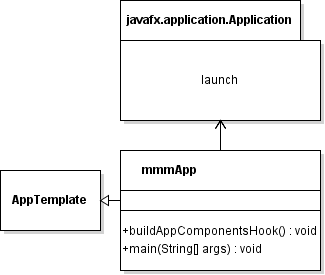
**Figure 3.1: Desktop Java Framework Overview UML Class Diagram**



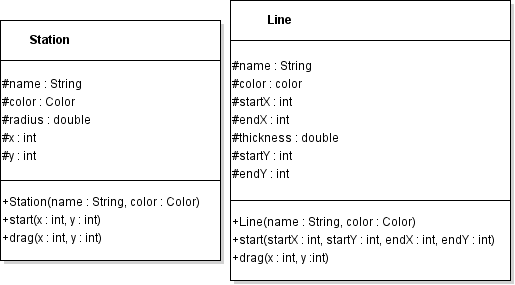
**Figure 3.2: Metro Map Maker Overview UML Class Diagram**



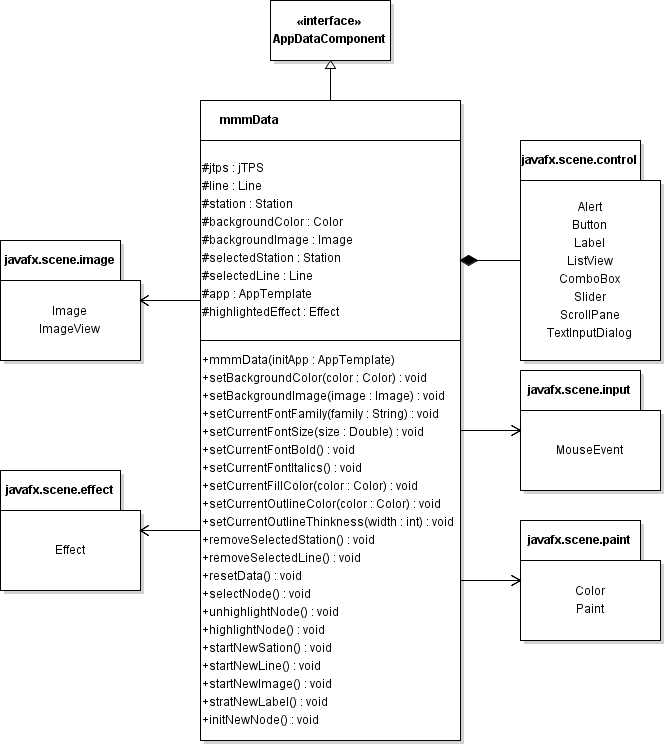
**Figure 3.3: mmmWorkspace UML Class Diagram**



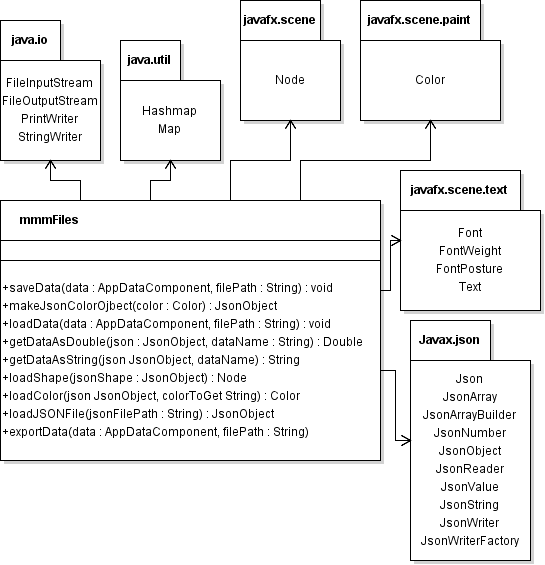
**Figure 3.4: mmmApp UML Class Diagram**



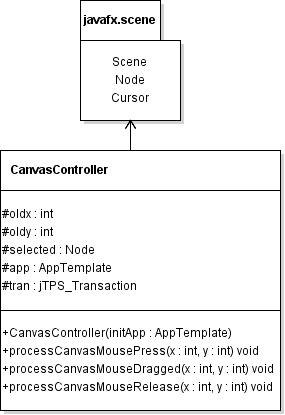
**Figure 3.5: Station &Line UML Class Diagram**



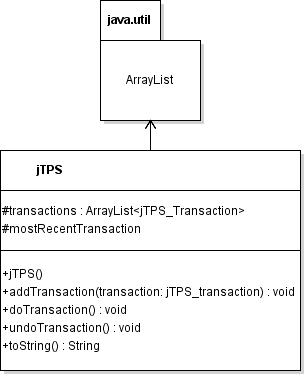
**Figure 3.6: mmmWorkspace UML Class Diagram**



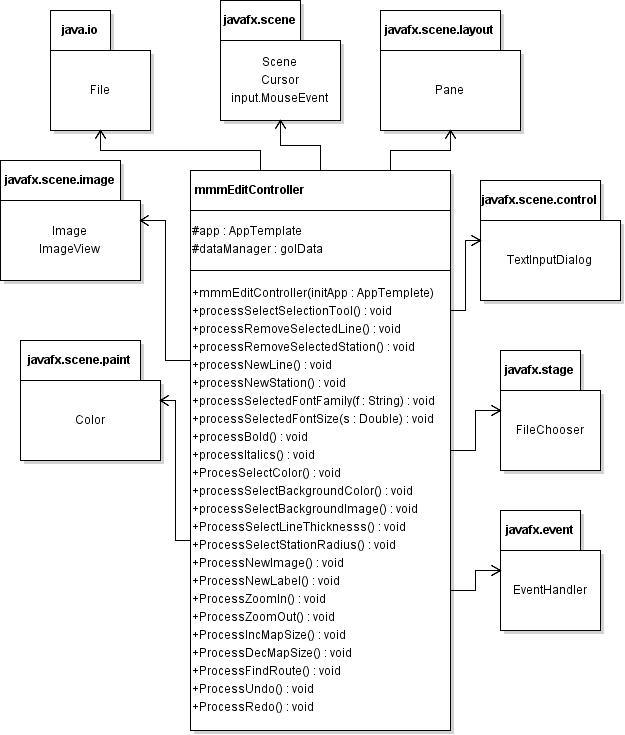
**Figure 3.7: mmmFiles UML Class Diagram**



**Figure 3.8: CanvasController UML Class Diagram**



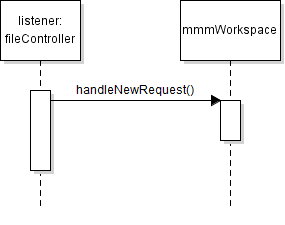
**Figure 3.9: jTPS UML Class Diagram**



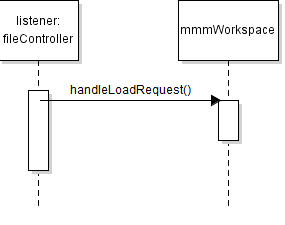
**Figure 3.10: mmmEditController Class Diagram**

4 Method-Level Design Viewpoint

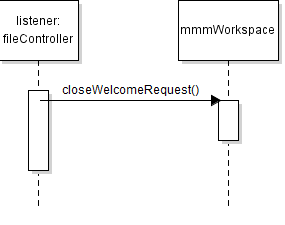
Now that the general architecture of the classes has been determined, it is time to specify how data will flow through the system. The following UML Sequence Diagrams describe the methods called within the code to be developed in order to provide the appropriate event responses.



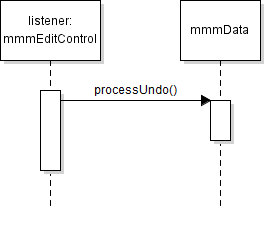
**Figure 4.1: handleNewRequest UML Sequence Diagrams**



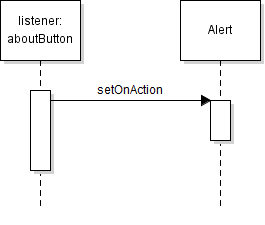
**Figure 4.2: handleNewRequest UML Sequence Diagrams**



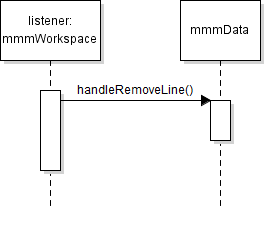
**Figure 4.3: closeWelcomeRequest UML Sequence Diagrams**



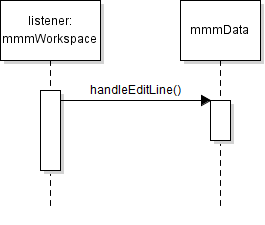
**Figure 4.4: processUndo UML Sequence Diagrams**



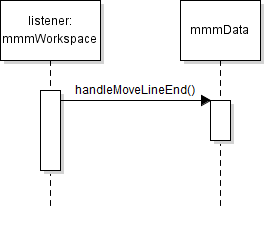
**Figure 4.5: aboutButtonAction UML Sequence Diagrams**



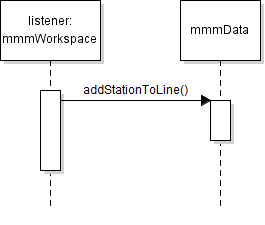
**Figure 4.6: handleRemoveLine UML Sequence Diagrams**



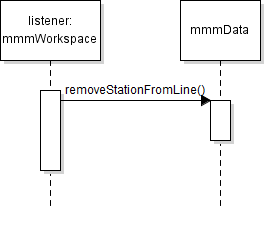
**Figure 4.7: handleEditLine UML Sequence Diagrams**



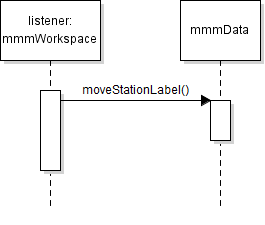
**Figure 4.8: handleMoveLineEnd UML Sequence Diagrams**



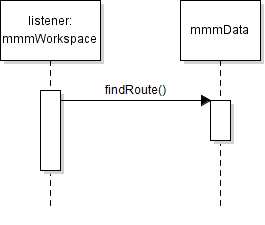
**Figure 4.9: addStationToLine UML Sequence Diagrams**



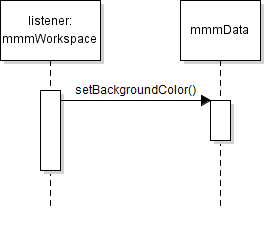
**Figure 4.10: removeStationFromLine UML Sequence Diagrams**



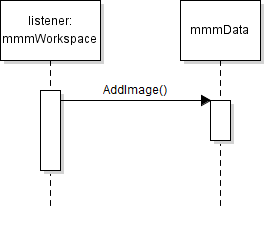
**Figure 4.11: moveStationLabel UML Sequence Diagrams**



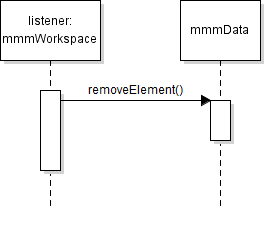
**Figure 4.12: findRoute UML Sequence Diagrams**



**Figure 4.13: setBackgroundColor UML Sequence Diagrams**



**Figure 4.14: addImage UML Sequence Diagrams**



**Figure 4.15: removeElement UML Sequence Diagrams**

1. File Structure and Formats

Note that the Desktop Java Framework 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 Metro Map Maker application. Note that all necessary data and art files must accompany this program.

The data of this application are stored in the mmmFiles, using JsonFile. Every time when a file is being saved, the data such as color, coordination of lines and stations are converted to Json Object. The JsonArrayBuilder is imported so it can organize the data in an array. The JsonWriter writes the JsonObject to a file which will be saved in the “work” folder insider the metroMapMaker project.

Last but not least, the record about the recent saved files are store in “recent.txt” in the metroMapMaker project. The name of each saved file is stored in lines in the text file.

1. Table of Contents

1. Introduction 2

1. Purpose 2

2. Scope 2

3. Definitions, acronyms, and abbreviations 2

4. References 2

5. Overview 3

2. Package-Level Design Viewpoint 3

3. Class-Level Design Viewpoint 7

4. Method-Level Design Viewpoint 15

5. File Structure and Formats 20