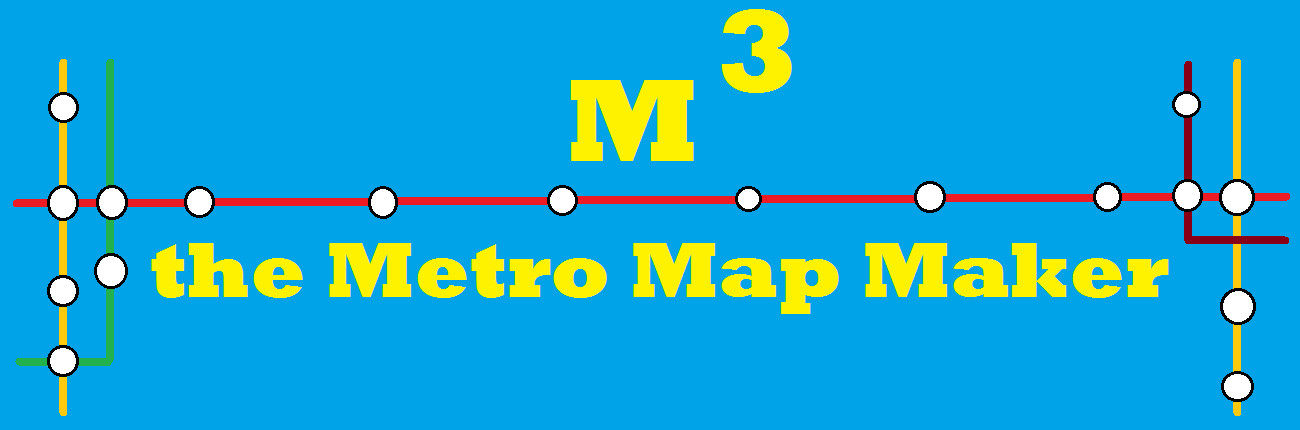
**The *Metro Map Maker* TM0**

**Software Design Description**



**Author:** Daeun Park

**Based on IEEE Std for IT - SDD (IEEE 1016-2009) document format**

**1 Introduction**

The ***Metro Map Maker (i.e. M3)*** application will provide a set of tools to build graphical representations of subway with named (intersecting) lines, stops and landmarks. It will also provide a calculating system, which gives the best route between two stations. Finally, it will provide an export feature such that it exports a generated map with related information which can be used by a corresponding Web application.

**1.1 Purpose**

The purpose of this software design document is to provide a low-level description of the Metro Map Maker (i.e M3) application, providing insight into the inner structure and design of each component described in the Software Requirements Specification. This document will cover class hierarchies and interactions, data flow and design, algorithmic models and user interface design.

**1.2 Scope**

For this project the goal is for users to provide functions that can be easily used to make and edit their own subway maps. Also, this application provides a common export format such that all maps can be used by a uniform application.

**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”.

**Java**– the default scripting language of the Web, Java 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.

**Model View Controller**– a software [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern), which divides a given application into three interconnected parts (Model, View and Controller).

**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 Diagram** – A UML document format that specifies how a user will interact with a system.

**User Interface** – the space where user directly interact with an application through specific features.

**1.4 References**

**IEEE Standard for Information Technology** - Systems Design - Software Design Descriptions (IEEE 1016-2009)

**1.5 Overview**

This SDD will specify how the ***Metro Map Maker*** application will be constructed, through inner structure and design of each component, which contribute to building functions described in SRS. In this application, there are three major classes: Workspace class which is the user interface, EditController class that notifies Data class to make changes, and Data that implements specific methods to modify data. To sum up, the EditController class updates the Data class, which modifies the Workspace class. This document is organized in general-to-specific order: Section 2 of this document will provide the general description of application and its Java API Usage, with package-level design viewpoint. Section 3 will present how classes will be organized and interact between them. Section 4 will present how each method will be implemented to build specific function. Section 5 provides a Table of Contents, an Index, and References. To sum up, classes (Section 3) contain methods to build specific functions (Section 4), which are based on Java API(Section 2).

**2 Package-Level Design Viewpoint**

**2.1 Metro Map Maker overview**

The goal of ***Metro Map Maker*** is not only allowing users to make their own subway maps adapted to the personal needs, but also being a site where users can exchange their maps between them such that the entire community view the map and plot routes. Shape means lines and stations with corresponding JSON files describing the contents.

**2.2 Java API Usage**

This is the list of Java API used.

**javafx**

**javafx.beans.value**

- javafx.beans.value.ChangeListener;

- javafx.beans.value.ObservableValue;

**javafx.collections.ObservableList**

* javafx.collections.ObservableList;

[**javafx.scene**](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html)

* javafx.scene.paint
* [javafx.scene](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html).shape
* [javafx.scene](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html).effect
* [javafx.scene](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html).layout
* [javafx.scene](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html).text
* [javafx.scene](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html).image
* [javafx.scene](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html).Node
* [javafx.scene](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html).Cursor
* [javafx.scene](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html).Scene
* javafx.scene.shape.Shape

**javafx.stage**

* javafx.stage.FileChooser;

[**java.io**](https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html)

- java.io.FileInputStream;

- java.io.FileOutputStream;

- java.io.IOException;

- java.io.InputStream;

- java.io.OutputStream;

- java.io.PrintWriter;

- java.io.StringWriter;

- java.io.BufferedReader;

- java.io.BufferedWriter;

- java.io.File;

- java.io.FileReader;

- java.io.FileWriter;

[**java.util**](https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html)

* [java.util](https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html).HashMap
* [java.util](https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html).Map
* java.util.ArrayList
* java.util.List;
* java.util.Optional;

**javax**

javax.imageio

[**javax.json**](https://docs.oracle.com/javaee/7/api/javax/json/package-summary.html)

- javax.json.Json;

- javax.json.JsonArray;

- javax.json.JsonArrayBuilder;

- javax.json.JsonNumber;

- javax.json.JsonObject;

- javax.json.JsonReader;

- javax.json.JsonValue;

- javax.json.JsonWriter;

* javax.json.JsonWriterFactory;
* javax.json.stream.JsonGenerator;

**2.3 Java API Usage Descriptions**

**javafx**

**javafx.beans.value**

- ChangeListener and [**ObservableValue**](https://docs.oracle.com/javafx/2/api/javafx/beans/value/ObservableValue.html) interfaces are used in CanvasController class to detect changes of choices in comboBox of user interface.

[**javafx.collections**](https://docs.oracle.com/javafx/2/api/javafx/collections/package-summary.html)

- Contains the essential JavaFX collections and collection utilities. Used to store Node of shapes in ObservableList<Node>.

[**javafx.scene**](https://docs.oracle.com/javafx/2/api/javafx/scene/package-summary.html)

- Provides the core set of base classes for the JavaFX Scene Graph API. Used to visualize UI throughout the application.

**javafx.stage**

-Provides the top-level container classes for JavaFX content. FileChooser is used to let user choose the image file to add in Workspace.

[**java.io**](https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html)

-Provides for system input and output through data streams, serialization and the file system. Used to read, modify and save shapes with details.

[**java.util**](https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html)

-Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and etc. Mainly, ArrayList is used in Workspace class to store language choices. HashMap is used in Files to store shapes using keys.

**javax**

**javax.imageio**

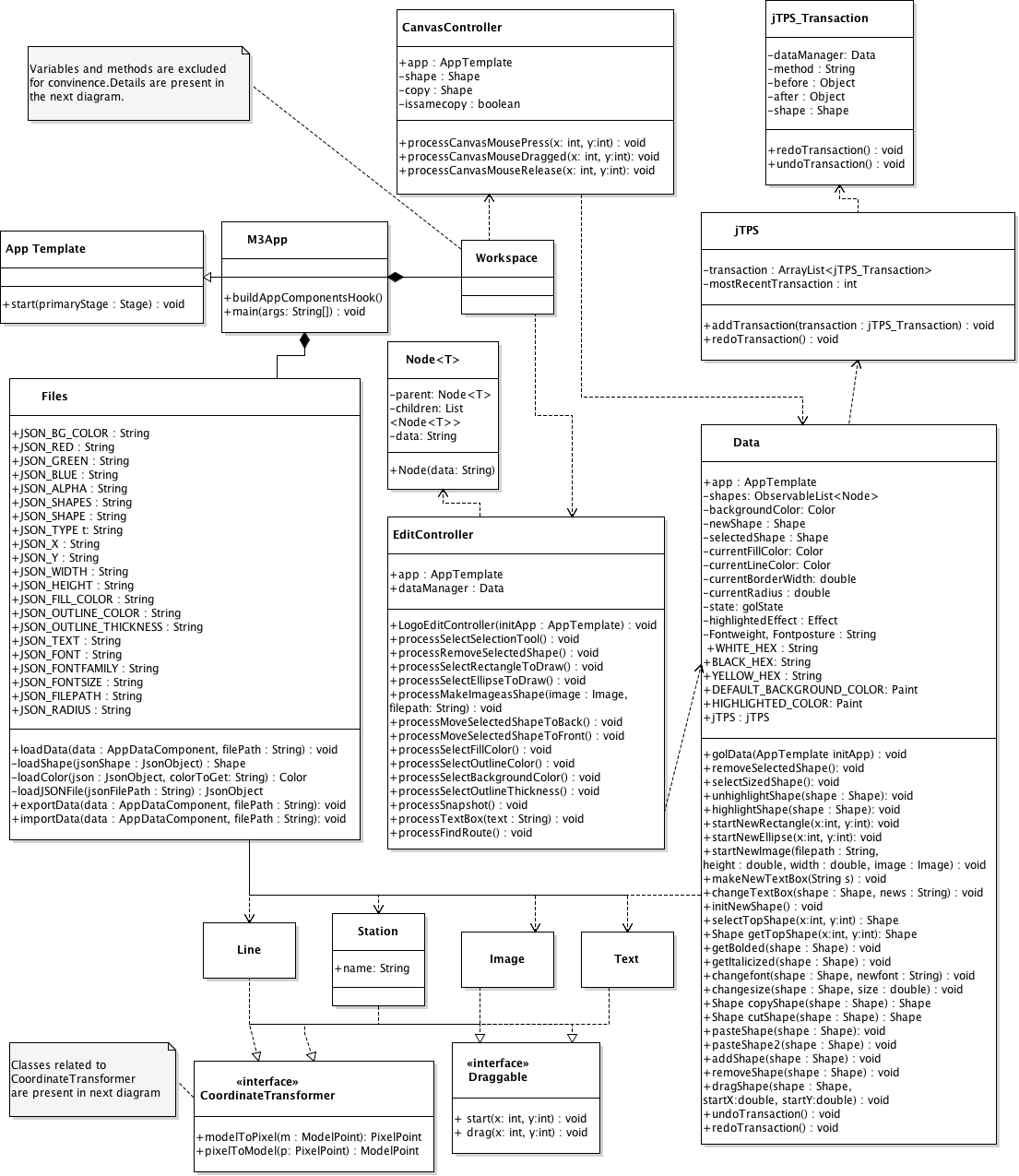
-Contains the basic classes and interfaces for describing the contents of image files, including metadata and thumbnails (IIOImage). In this application, this is used to control image writing process to process a snapshot in EditController class.

[**javax.json**](https://docs.oracle.com/javaee/7/api/javax/json/package-summary.html)

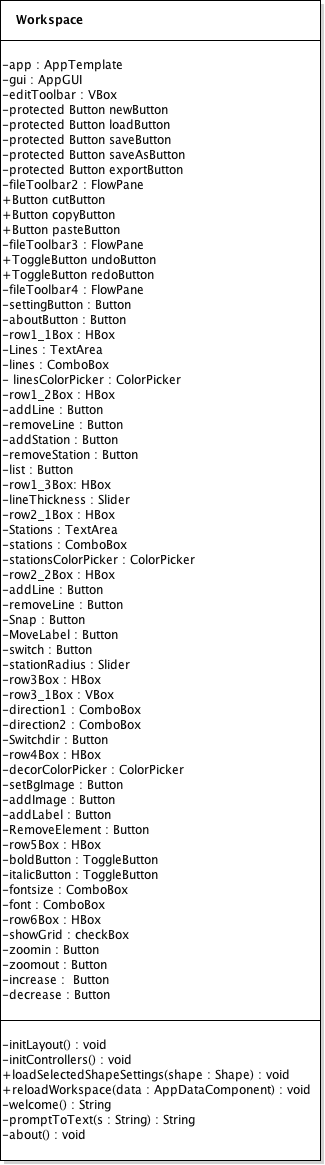
-Provides immutable object models for JSON object and array structures. In this application, lines, stations with details are stored in JSON formats with [JsonObject](https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html) and [JsonArray](https://docs.oracle.com/javaee/7/api/javax/json/JsonArray.html) in Files class.

**3 Class-Level Design Viewpoint**

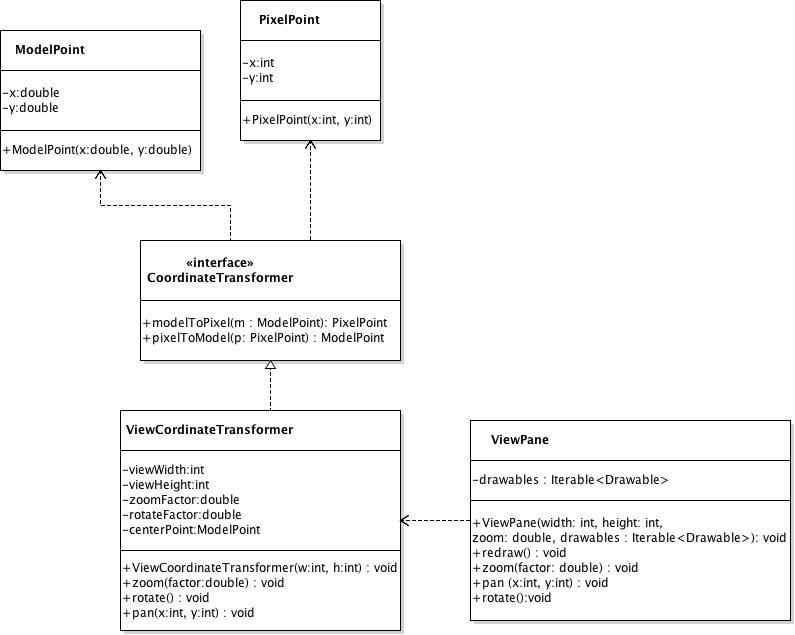
At first, the design with Line class and Stations class, interacting directly with Workspace class was considered. But the problem was that classes were too dependent to each other, with high coupling. To fix this problem of high coupling, classes are designed in Model View Controller structure (MVC), such that the controller updates the models, and the models modify the view (Figure 1). In addition, two interfaces Draggable and CoordinateTransformer allow the application to drag shapes, to zoom in/out and rotate the view (Figure 3). In this application, EditController class updates Data class, which modifies Workspace class (Figure 2). The dissociation of the data and the user interface makes the application more cohesive but difficult to implement.



**Figure 1. General Class Diagram**

****

**Figure 2. Workspace Class Diagram**

****

**Figure 3. ViewCoordinateTransformer Class Diagram**

**4 Method-Level Design Viewpoint**

**5 File Structure and Formats**

In this application, files are stored in JSON format. Each data (type, font, text, size and etc) is boxed in JSON Object using attributes, and stored in JsonArray with ObjectBuilder. To read data in JSON files, the application uses JsonReader to read data by unboxing JSON Object in Files class. Writing and exporting JSON files works in the same way. To write and export JSON files, the application boxes data, creating JsonObject and storing it in [JsonArray](https://docs.oracle.com/javaee/7/api/javax/json/JsonArray.html).

**6 Supporting Information**

Note that this document should serve as a base design for the software engineers to guide them in the future stages of the development process, so we’ll provide a table of contents to help quickly find important sections.

**6.1 Table of contents**

1. Introduction

1. Purpose

2. Scope

3. Definitions, acronyms, and abbreviations

4. References

5. Overview

2. Package-Level Design Viewpoint

1. Metro Map Maker overview

2. Java API Usage

3. Java API Usage Descriptions

3. Class-Level Design Viewpoint

4. Method-Level Design Viewpoint

5. File Structure and Formats

6. Supporting Information

1. Table of Contents

2. Appendixes

6.2 Appendixes

N/A