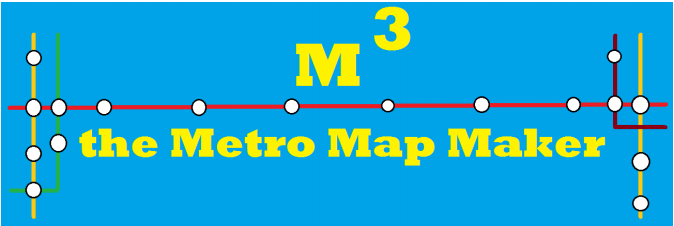
**The Metro Map Maker**

Software Design Description



Author:

Michael Spadafora

October, 2017

Version 1.0

Abstract: This document describes the software design for the MetroMapMaker, a tool which can be used to design metro map

**Table of Contents**

1. Introduction
   1. Purpose
   2. Scope
   3. Definitions, acronyms, and abbreviations
   4. References
   5. Overview
2. Package-Level Design Viewpoint
   1. Zombiquarium and Mini Game 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 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.

**1.1 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.

**1.2 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.

**1.3 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”. 3

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.

**1.4 References**

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

– Software Design Descriptions

**1.5 Overview**

This Software Design Description document provides a design for the MMM software application as described in the MMM Software Requirement Specification. Section 2 of this document provides 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, when completed for homework 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 an index and references. All UML Diagrams made using VioletUML.

**2.1 DesktopJavaFramework, MetroMapMaker, and PropertiesManager**

**Overviews**

The DesktopJavaFramework, MetroMapMaker, and Properties manager will be designed and developed in tandem. Figures 2.1.1-2.1.3 shows all the components to be developed



*Figure 2.1.1: Properties Manager outline*

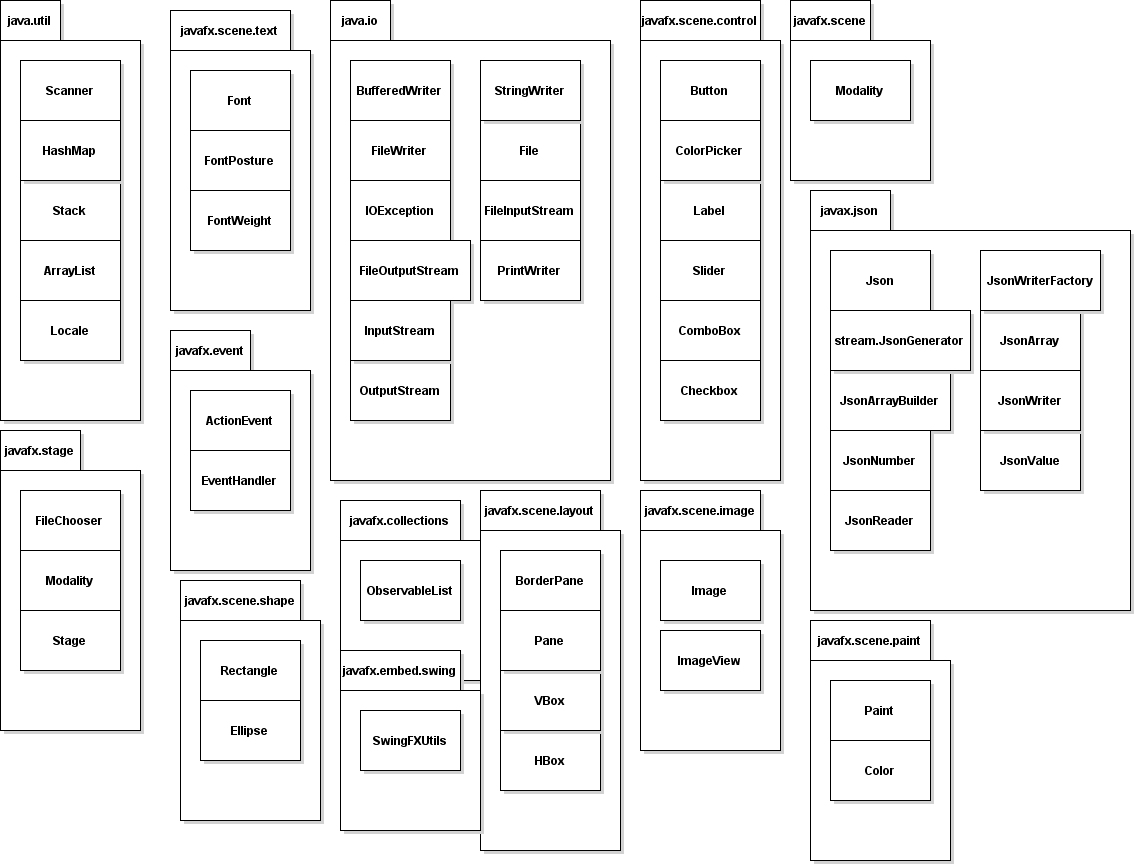


*Figure 2.1.2: DesktopJavaFramework outline*

*Figure 2.1.3: MetroMapMaker outline*

**2.2 Java API Usage**

All classes will be developed using the Java programming language. As such, the design will make use of the classes specified in figure 2.2

**

*Figure 2.2.1 and 2.2.2: Java API Classes and Packages to be used*

**2.3 Java API Usage Descriptions**

Tables 2.2.1-2.2.12 summarize how each of these class will be used

|  |  |
| --- | --- |
| **java.util** |  |
| **Class/Interface** | **Use** |
| Scanner | Read in information |
| HashMap | Used to store information |
| Stack | Used to store Transactions |
| ArrayList | Used to store information |
| Locale | Set the default locale for certain methods |

*Table 2.2.1: Uses for classes in the Java API’s java.util package*

|  |  |
| --- | --- |
| **java.io** |  |
| **Class/Interface** | **Use** |
| BufferedWriter | Writes text to a character output stream |
| StringWriter | Used to construct a string from output |
| FileWriter | Writes to character files |
| File | Access files |
| IoException | Thrown when errors in input/output occur |
| FileOutputStream | Used to write to a file |
| FileInputStream | Used to read from a file |
| PrintWriter | Prints objects to a text output stream |
| InputStream | Superclass of all classes that are an input stream of bytes |
| OutputStream | Superclass of all classes that are an output stream of bytes |

*Table 2.2.2: Uses for classes in the Java API’s java.io package*

|  |  |
| --- | --- |
| **javafx.event** |  |
| **class/interface** | **use** |
| ActionEvent | Specifies the component that causes an action |
| EventHandler | Provides support for dynamically generating event listeners whose methods execute a simple statement involving anincoming event object and a target object |

*Table 2.2.3: Uses for classes in the Java API’s javafx.event package*

|  |  |
| --- | --- |
| **javafx.stage** |  |
| **class/interface** | **use** |
| FileChooser | Allows a file to be chosen through a dialog box |
| Modality | Set weather or not an application will wait for a dialog box to be closed |
| Stage | The top level JavaFX container |

*Table 2.2.4: Uses for classes in the Java API’s java.stage package*

|  |  |
| --- | --- |
| **javafx.embed.swing** |  |
| **class/interface** | **use** |
| SwingFXUtils | Converts between Swing and JavaFX formats |

*Table 2.2.5: Uses for classes in the Java API’s java.embed.swing package*

|  |  |
| --- | --- |
| **javafx.scene.shape** |  |
| **class/interface** | **use** |
| Rectangle | A rectangle that will be used for images and lines |
| Ellipse | Will be used for stations |

*Table 2.2.6: Uses for classes in the Java API’s javafx.scene.shape package*

|  |  |
| --- | --- |
| **javafx.scene.paint** |  |
| **class/interface** | **use** |
| Paint | Used to fill shapes or backgrounds |
| Color | A color |

*Table 2.2.7: Uses for classes in the Java API’s javafx.scene.paint package*

|  |  |
| --- | --- |
| **javafx.scene.text** |  |
| **class/interface** | **use** |
| Font | Used to set the style of Texts |
| FontPosture | Used to set whether a text is italic or not |
| FontWeight | Sets whether a text is bold or not |

*Table 2.2.8: Uses for classes in the Java API’s javafx.scene.text package*

|  |  |
| --- | --- |
| **javafx.scene.control** |  |
| **class/interface** | **use** |
| Button | Used to control actionEvents |
| ColorPicker | Used to select colors |
| Label | Used as uneditable text that describes things |
| Slider | Sliding these controls things on a scale |
| ComboBox | Used to select from a list of items |
| Checkbox | Used to select options |

*Table 2.2.9: Uses for classes in the Java API’s javafx.scene.control package*

|  |  |
| --- | --- |
| **javafx.scene** |  |
| **class/interface** | **use** |
| Scene | Container for all content in the scene graph |

*Table 2.2.10: Uses for classes in the Java API’s javafx.scene package*

|  |  |
| --- | --- |
| **java.scene.layout** |  |
| **class/interface** | **use** |
| BorderPane | Lay out children in the top left right bottom and center. |
| Pane | Base class for layout panes |
| VBox | Lets items be layed out vertically |
| HBox | Lets items be layed out horizontally |

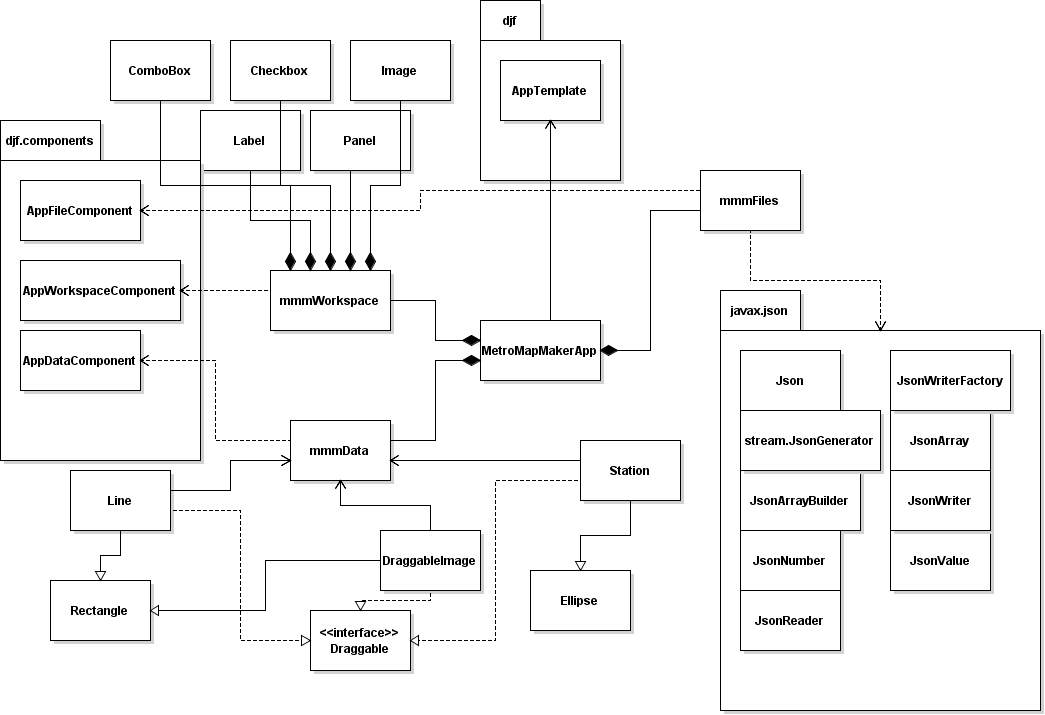
*Table 2.2.11: Uses for classes in the Java API’s java.scene.layout package*

|  |  |
| --- | --- |
| **javafx.scene.image** |  |
| **class/interface** | **use** |
| Image | Loads images from a specified URL |
| ImageView | Displays images |

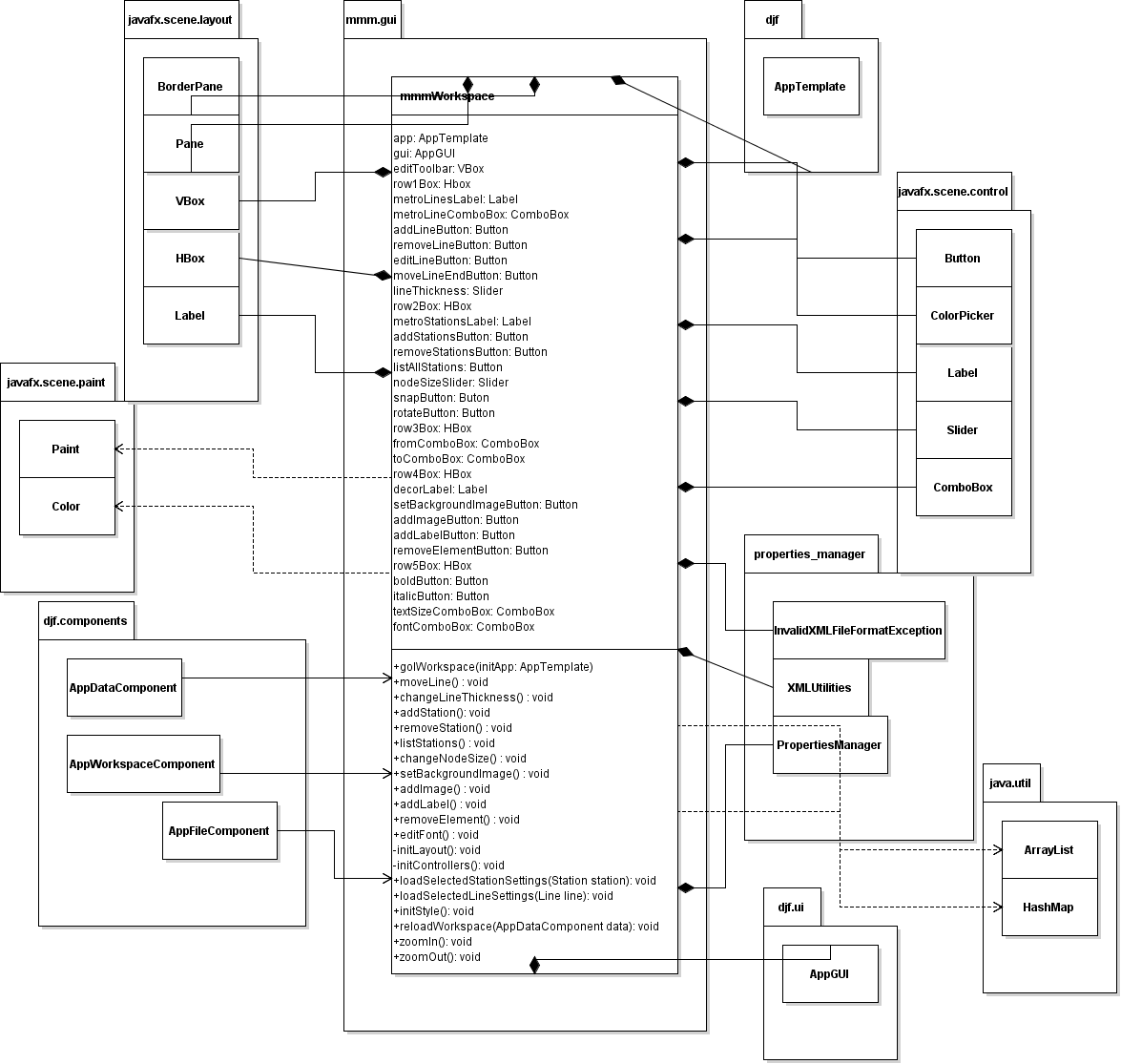
*Table 2.2.12: Uses for classes in the Java API’s java.scene.image package*

**3 Class-Level Design Viewpoint**

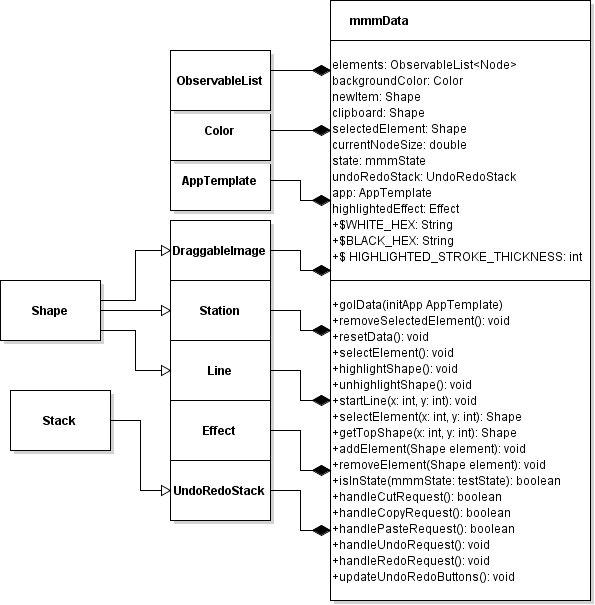
This design will encompass the MetroMapMaker application. The following UML Class diagrams reflect this.



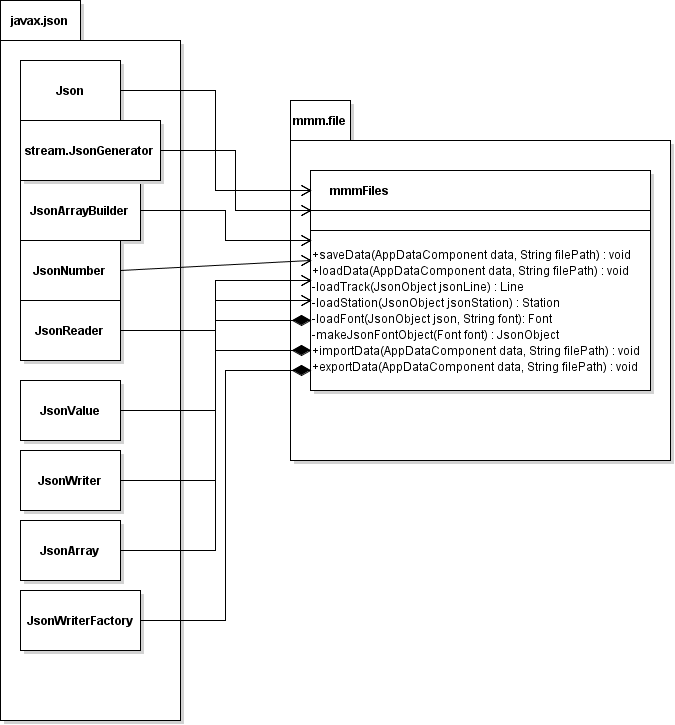
*Figure 3.1: Overview for MetroMapMaker*

**

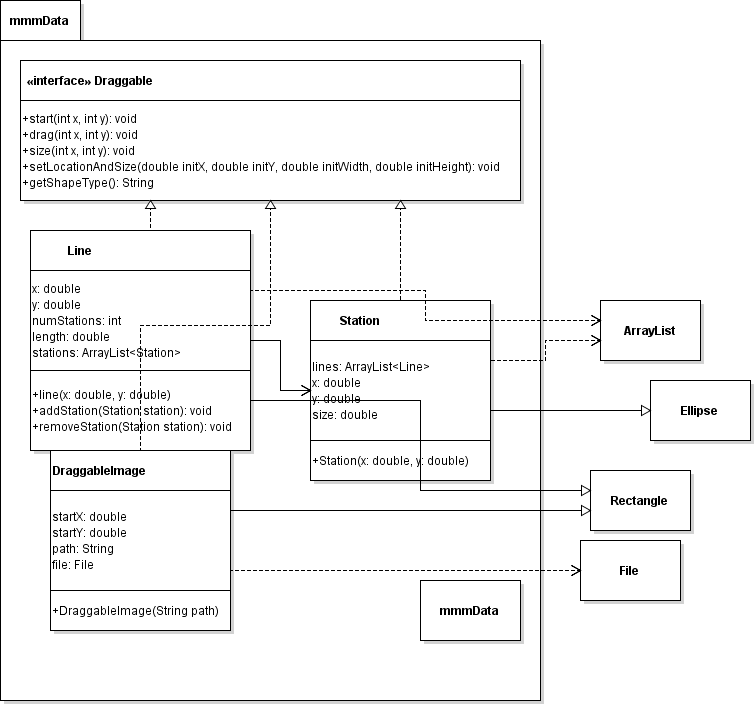
*Figure 3.2: Detailed view of for mmmWorkspace*

**

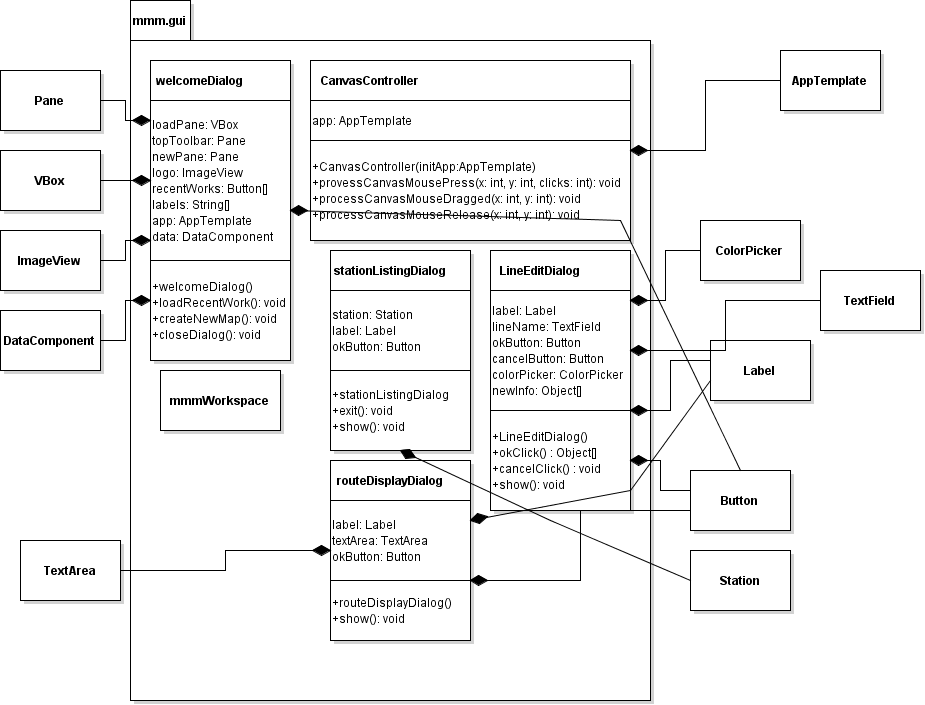
*Figure 3.3: Detailed view of mmmData*

**

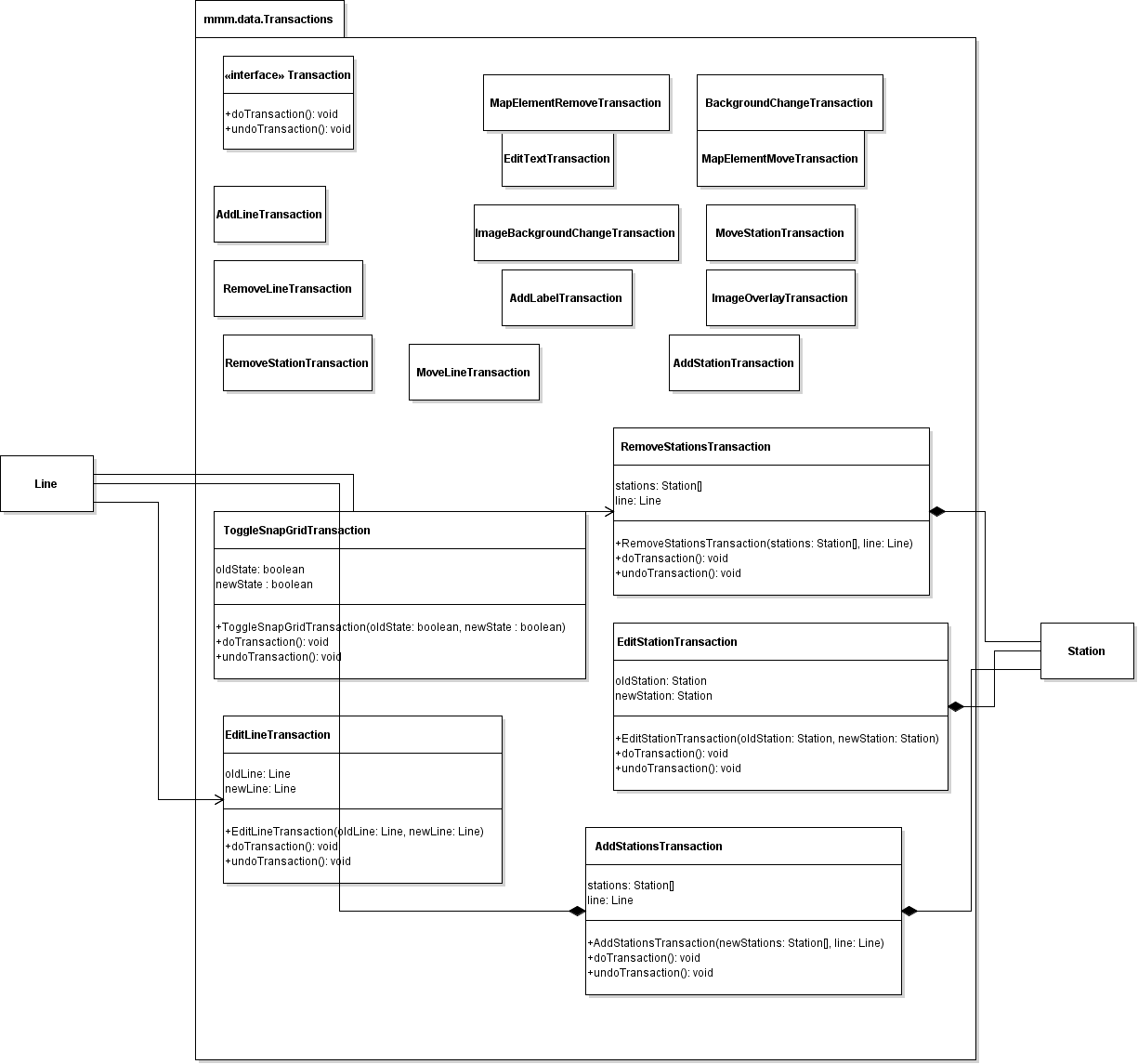
*Figure 3.4: Detailed view of mmmFiles*



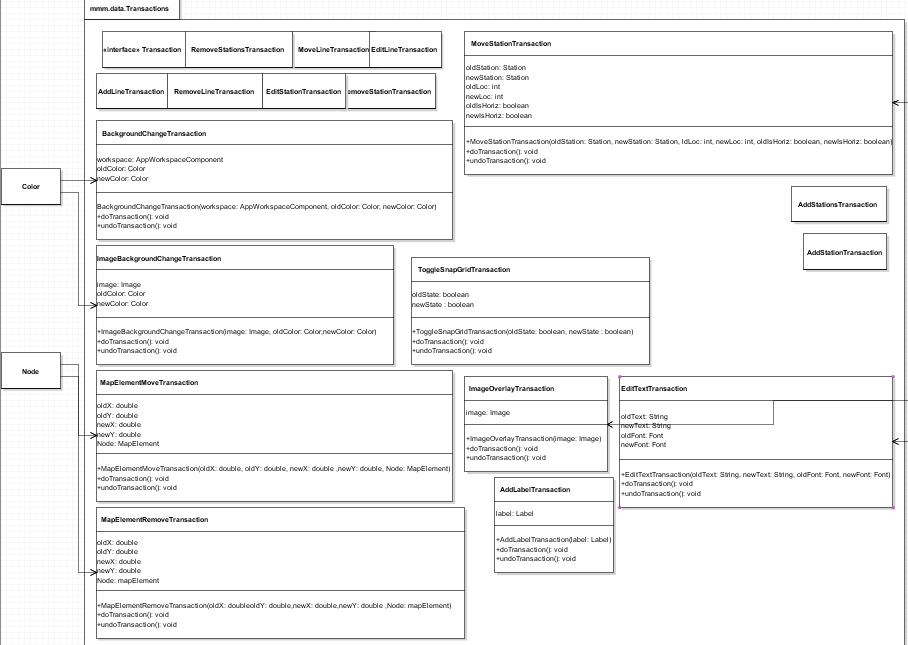
*Figure 3.5: Detailed view of the other classes in the mmmData package*



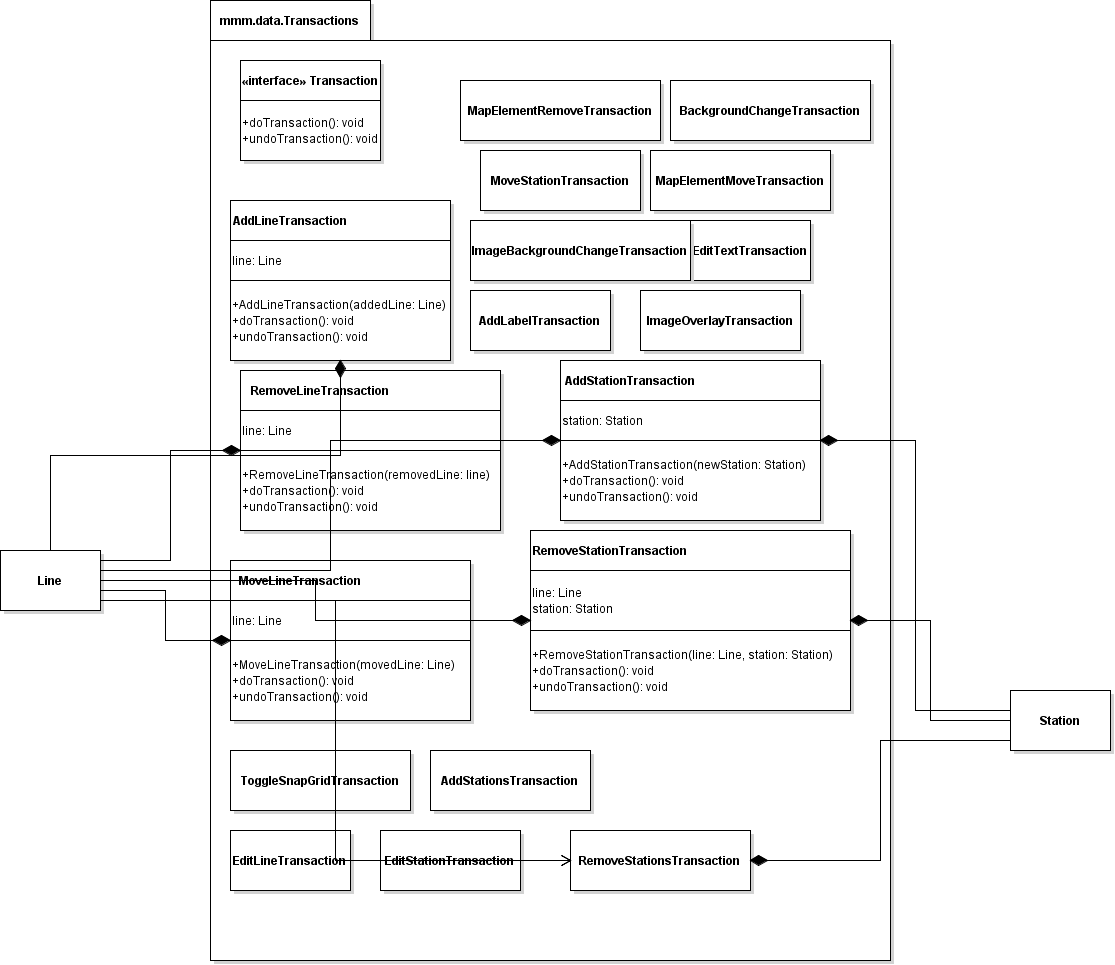
*Figure 3.6: Detailed view of the other classes in the mmm.gui package*



*Figure 3.7 detailed view of the first third of the Transaction classes*

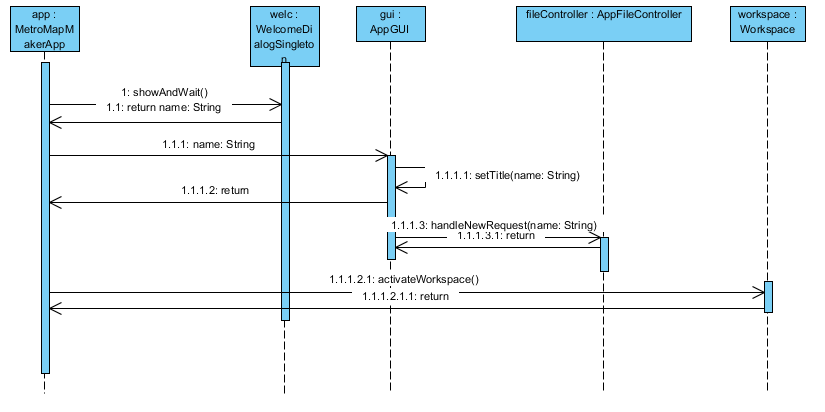


*Figure 3.8 detailed view of one third of the Transaction Classes*

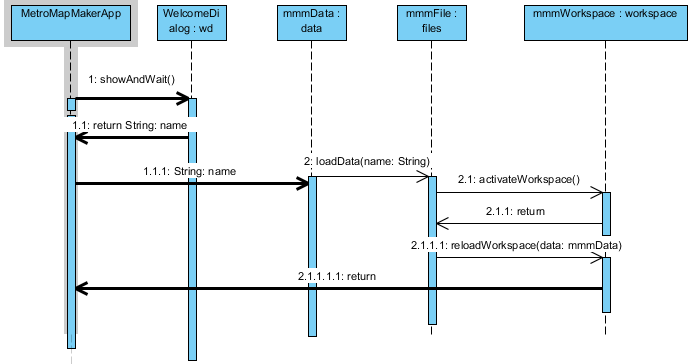
**

*Figure 3.9 detailed view of the last third of the Transaction Classes*

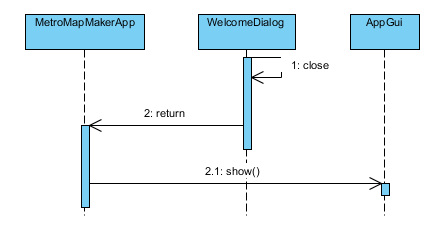
**4 Method-level Design Viewpoint**

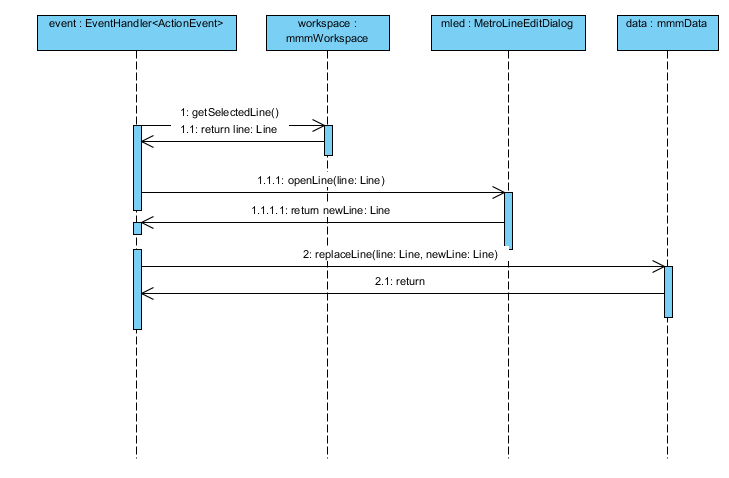
****

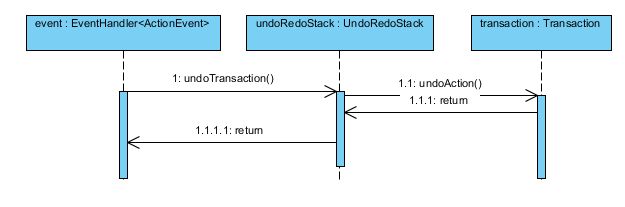
*Figure 4.1, detailing case 2.1: create new map*



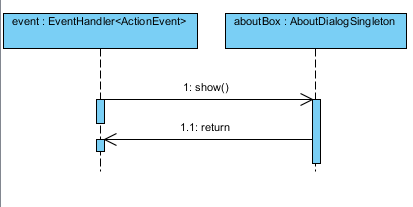
*Figure 4.2, detailing case 2.2: select recent map to load*

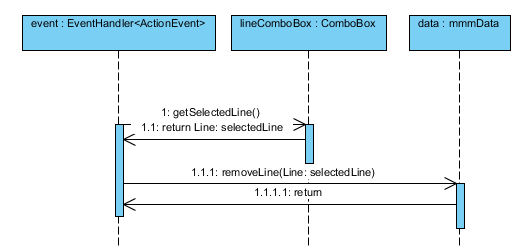


*Figure 4.3, detailing case 2.3: close welcome dialog*

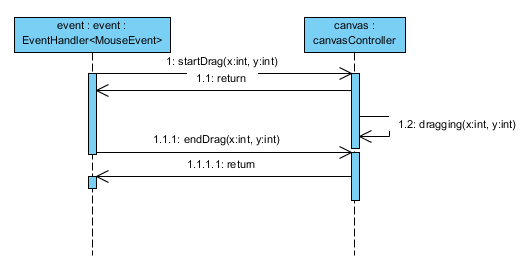


*Figure 4.4, detailing case 2.9: undo edit*

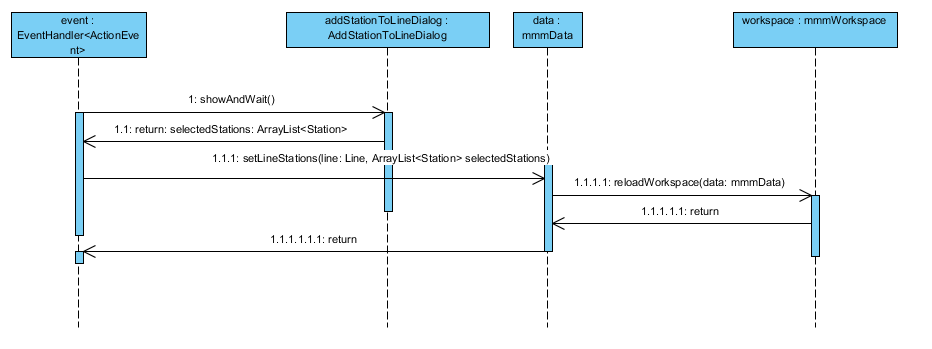


*Figure 4.5, detailing case 2.11: learn about application*

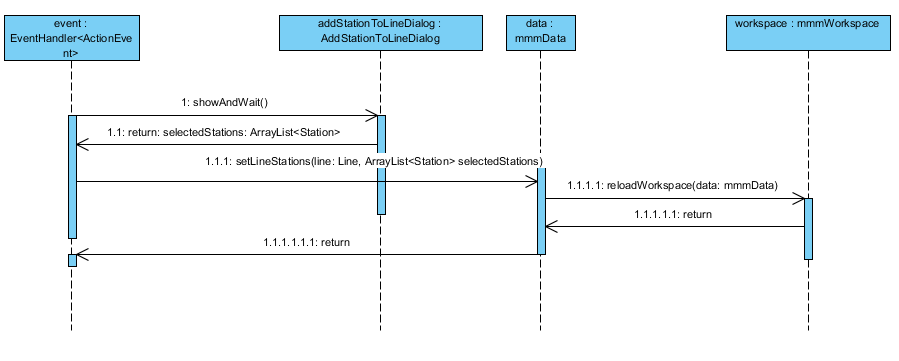
*Figure 4.6, detailing case 2.13: edit line*

**

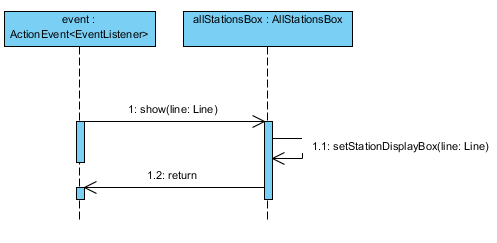
*Figure 4.8, detailing case 2.15: move line end*

**

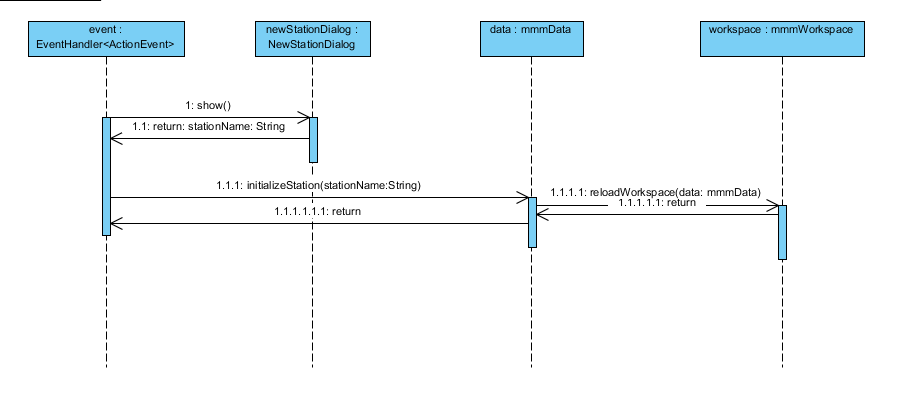
*Figure 4.9, detailing case 2.16: add stations to line*

**

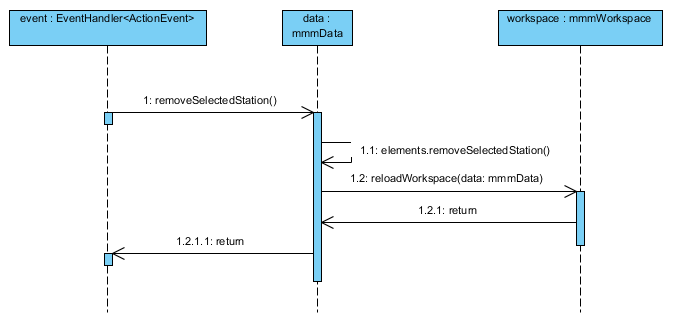
*Figure 4.10, detailing case 2.17: remove stations from line*

**

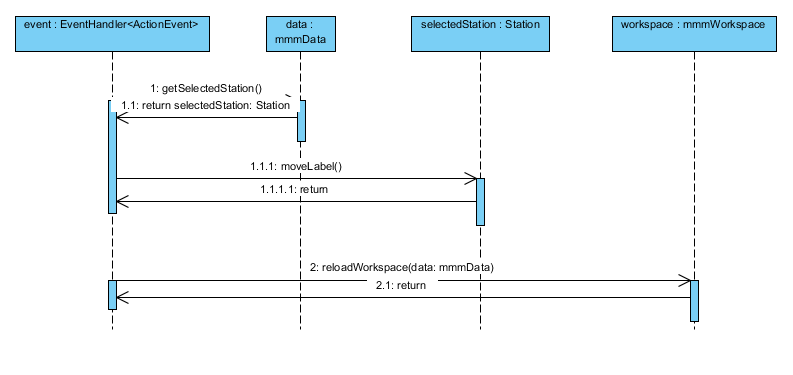
*Figure 4.11, detailing case 2.18: list all stations in line*

**

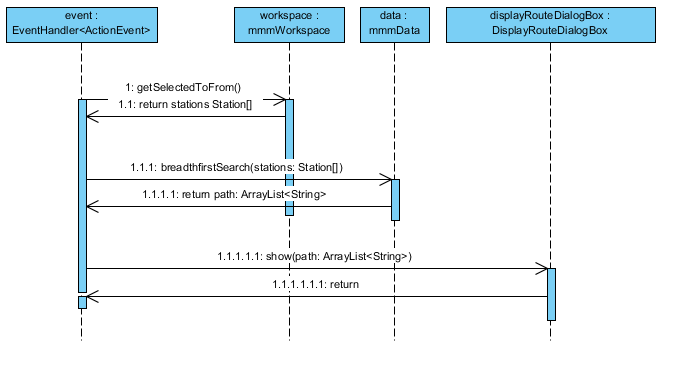
*Figure 4.12, detailing case 2:20: add new station*

**

*Figure 4.13, detailing case 2.21:remove station*

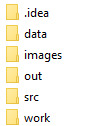
**

*Figure 4.14, detailing case 2.23: move station label*

**

*Figure 4.15, detailing case 2.27: find route*

**5 File Structure and Formats**



Images will be stored in the images folder and work will be stored in the work folder.

We will be loading from XML files with JSON objects.

Style classes are stored within a CSS.