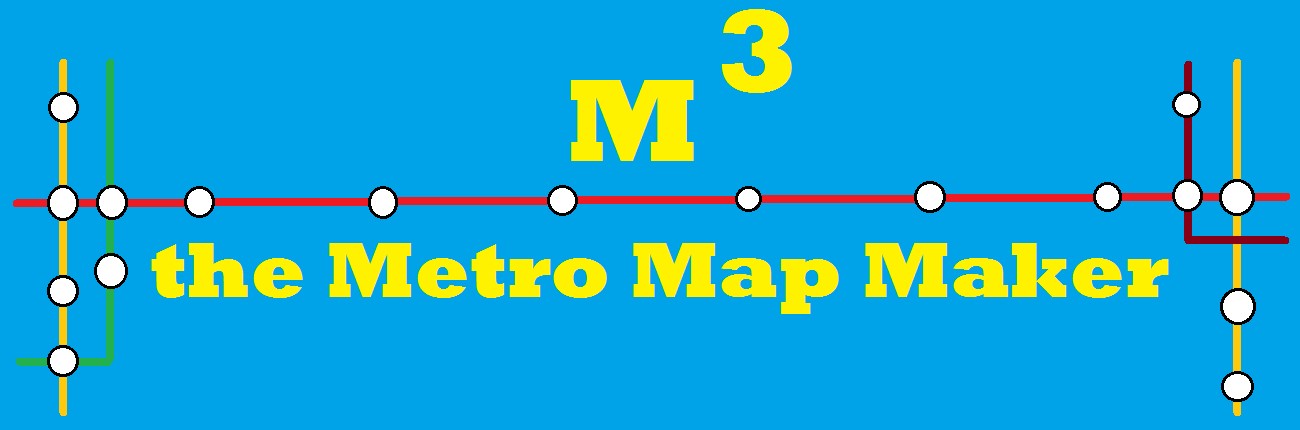
**The *Metro Map Maker* TM0**

**Software Requirements Specification**



**Author:** Yonghun Jeong

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

Copyright © 2017 Debugging Enterprises

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

**Table View**

1. Introduction 3
   1. Purpose 3
   2. Scope 3
   3. Definitions, acronyms, and abbreviations 4
   4. References 4
   5. Overview 4
2. Package-Level Design Viewpoint 5
   1. Metro Map Maker overview 5
   2. Java API Usage 6
   3. Java API Usage Descriptions 7
3. Class-Level Design Viewpoint 19
4. Method-Level Design Viewpoint 32
5. File Structure and Formats 48
6. Supporting Information 51

**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”.

**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 Recommended Practice for Software Requirements

Specification

**ORACLE SE DOCUMENT-** Definition and usage of API

**1.5 Overview**

This SRS will clearly define how the ***Metro Map Maker*** application should look and operate. Note that this is not a software design description (SDD), which would design how to construct the software using UML. This document does not specify how to build the appropriate technologies, it is simply an agreement concerning what to build. Section 2 of this document will provide the context for the project and specify all the conceptual design. Section 3 will present how the user interface should be laid out. Section 4 provides a Table of Contents, an Index, and References.

**2. Package-Level Design Viewpoint**

As mentioned, this design will encompass both the Metro Map Maker desktop application 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.

* 1. **Metro Map Maker overview**

The Metro Map Maker Applicion will be designed and developed in tandem. Figure 2.1 specifies all the components to be developed and places all classes in home packages. The Metro Map Maker Application consists of 3 partitions : DesktopJavaFramework, MetroMapMaker, PropertiesManager.

DesktopJavaFramework provides the start method that begins the program initialization, which delegates component initialization to the application-specific child class' hook function. This package also include the UI for the workspace, but does not include workspace.

MetroMapMaker is actual code for executing application. In this package, line,station, button, canvas will be implemented. The bunch of its frame is inherited from DesktopJavaFramework

PropertiesManager checkes exception that represents the occasion where an XML document does not validate against its schema. Also It is used for loading properties from XML files that can then be used throughout an application.

To collaborate and communicate with each other, many user interface API is used. For example, button, scroll bar, redo/undo button, textField will be made and initialized in mmmWorkspace. By using a setOnAction function in logoEditController class, diverse requests are executed. All the information in the application is managed by mmmData class. It has lines, stations, label, Shape. Those shape should be draggable. So I make a Draggable interface and other class implements Draggable class.

This Framework is made up of Model, View, Controller. I have some reasons that I use this framework. First of all, User communicate with application with only through View. 2. This Framework avoid the interference between View and Model 3. Controller have an important role in managing View and Model relation. The disadvantage of using this framework is that it may looks so complicated. Also, separating Model and View perfectly is very difficult. There are MVP(Model-View-Presenter), MVVM(Model-View, View Model) framework. Comparing to other framework, I used Model-View-Controller model. this framework is very good for reusability. Also, It is very good to extension. This is why U use this framework

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DesktopJavaFramework |  | MetroMapMaker |  | PropertiesManager |
| EMB000032782229 |  | EMB000032782234 |  | EMB000032782235 |

Figure 2. : Design Packages Overview

* 1. **Java API Usage**

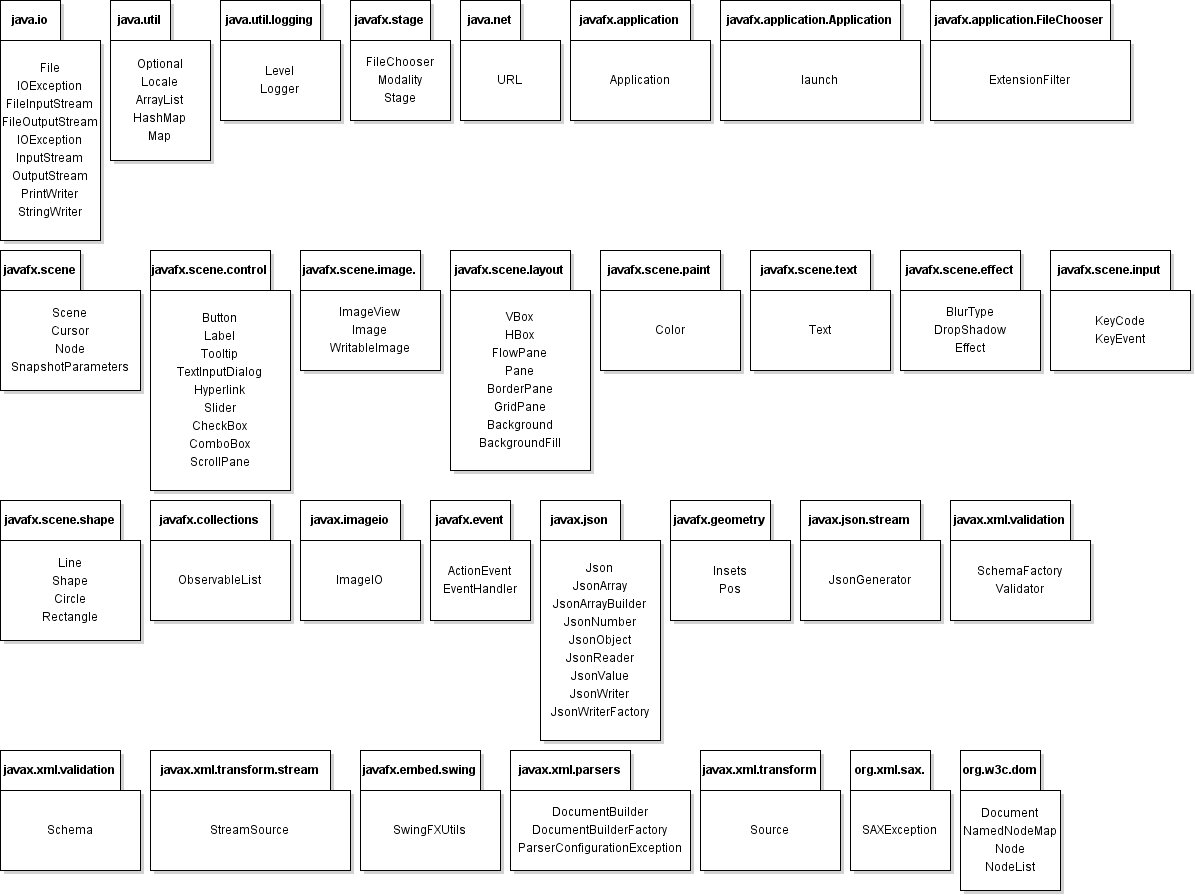
Both the Desktop Java framework and the Metro Map Maker application will be developed using the Java programming languages. As such, this design will make use of the classes specified in Figure 2.2.

Figure 2. Java API Classes and Packages To Be Used

* 1. **Java API Usage Descriptions**

Tables 2.1 – 2.17 below summarize how each of these classes will be used.

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ObservableList** | To save the all the information in the canvas such as line, station, circle, shape and so on. |

Table 2.1: Uses for classes in the Java API’s javafx.collections.ObservableList

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Pos** | To set of values for describing vertical and horizontal positioning and alignment. |
| **Insets** | To give a spare space into rectangular area |

Table 2.2: Uses for classes in the Java API’s javafx.geometry.Pos

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Button** | To A simple button control. The button control can contain text and/or a graphic. |
| **Label** | Label is a non-editable text control. A Label is useful for displaying text that is required to fit within a specific space, and thus may need to use an ellipsis or truncation to size the string to fit. |
| **Tooltip** | Tooltips are common UI elements which are typically used for showing additional information about a Node in the scenegraph when the Node is hovered over by the mouse. Any Node can show a tooltip. In most cases a Tooltip is created and its [text](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Tooltip.html#textProperty--) property is modified to show plain text to the user. However, a Tooltip is able to show within it an arbitrary scenegraph of nodes - this is done by creating the scenegraph and setting it inside the Tooltip [graphic](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Tooltip.html#graphicProperty--) property. |
| **TextInputDialog** | A dialog that shows a text input control to the user. |
| **Hyperlink** | An HTML like label which can be a graphic and/or text which responds to rollovers and clicks. When a hyperlink is clicked/pressed [isVisited()](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Hyperlink.html#isVisited--) becomes true. A Hyperlink behaves just like a [Button](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Button.html). When a hyperlink is pressed and released a [ActionEvent](https://docs.oracle.com/javase/8/javafx/api/javafx/event/ActionEvent.html) is sent, and your application can perform some action based on this event. |
| **Slider** | The Slider Control is used to display a continuous or discrete range of valid numeric choices and allows the user to interact with the control. It is typically represented visually as having a "track" and a "knob" or "thumb" which is dragged within the track. The Slider can optionally show tick marks and labels indicating the different slider position values. |
| **CheckBox** | A check box is a graphical component that can be in either an "on" (true) or "off" (false) state. Clicking on a check box changes its state from "on" to "off," or from "off" to "on."  The following code example creates a set of check boxes in a grid layout: |
| **ComboBox** | An implementation of the [ComboBoxBase](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ComboBoxBase.html) abstract class for the most common form of ComboBox, where a popup list is shown to users providing them with a choice that they may select from. For more information around the general concepts and API of ComboBox, refer to the [ComboBoxBase](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ComboBoxBase.html)class documentation. |
| **ScrollPane** | A Control that provides a scrolled, clipped viewport of its contents. It allows the user to scroll the content around either directly (panning) or by using scroll bars. The ScrollPane allows specification of the scroll bar policy, which determines when scroll bars are displayed: always, never, or only when they are needed. The scroll bar policy can be specified independently for the horizontal and vertical scroll bars.  The ScrollPane allows the application to set the current, minimum, and maximum values for positioning the contents in the horizontal and vertical directions. These values are mapped proportionally onto the [layoutBounds](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Node.html#layoutBoundsProperty--) of the contained node. |

Table 2.3: Uses for classes in the Java API’s javafx.scene.control

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Stage** | The JavaFX Stage class is the top level JavaFX container. The primary Stage is constructed by the platform. Additional Stage objects may be constructed by the application. |
| **Modality** | This enum defines the possible modality types for a Stage. |
| **FileChooser** | Provides support for standard platform file dialogs. These dialogs have look and feel of the platform UI components which is independent of JavaFX.  On some platforms where file access may be restricted or not part of the user model (for example, on some mobile or embedded devices), opening a file dialog may always result in a no-op (that is, null file(s) being returned). |

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

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Color** | The Color class is used to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a[ColorSpace](https://docs.oracle.com/javase/7/docs/api/java/awt/color/ColorSpace.html). Every color has an implicit alpha value of 1.0 or an explicit one provided in the constructor. The alpha value defines the transparency of a color and can be represented by a float value in the range 0.0 - 1.0 or 0 - 255. An alpha value of 1.0 or 255 means that the color is completely opaque and an alpha value of 0 or 0.0 means that the color is completely transparent. When constructing a Color with an explicit alpha or getting the color/alpha components of a Color, the color components are never premultiplied by the alpha component. |

Table 2.5: Uses for classes in the Java API’s javafx.scene.paint.Paint

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Circle** | The Circle class creates a new circle with the specified radius and center location measured in pixels Example usage. |
| **Line** | This Line represents a line segment in (x,y) coordinate space |
| **Shape** | The Shape interface provides definitions for objects that represent some form of geometric shape. The Shape is described by a [PathIterator](https://docs.oracle.com/javase/7/docs/api/java/awt/geom/PathIterator.html) object, which can express the outline of the Shape as well as a rule for determining how the outline divides the 2D plane into interior and exterior points. Each Shape object provides callbacks to get the bounding box of the geometry, determine whether points or rectangles lie partly or entirely within the interior of the Shape, and retrieve a PathIterator object that describes the trajectory path of the Shape outline. |
| **Rectangle** | A Rectangle specifies an area in a coordinate space that is enclosed by the Rectangle object's upper-left point (x,y) in the coordinate space, its width, and its height.  A Rectangle object's width and height are public fields. The constructors that create a Rectangle, and the methods that can modify one, do not prevent setting a negative value for width or height. |

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

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **KeyEvent** | An event which indicates that a keystroke occurred in a Node. |
| **KeyCode** | Set of key codes for [KeyEvent](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/input/KeyEvent.html) objects. |
| **MouseDragEvent** | Mouse drag events are delivered to potential gesture targets during full press-drag-release gestures. The difference among different gesture types is described at MouseEvent. |
| **MousePressEvent** | When mouse event occurs, the top-most node under cursor is picked and the event is delivered to it through capturing and bubbling phases described at EventDispatcher. |

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

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **HBox** | HBox lays out its children in a single horizontal row. If the hbox has a border and/or padding set, then the contents will be layed out within those insets. |
| **VBox** | VBox lays out its children in a single vertical column. If the vbox has a border and/or padding set, then the contents will be layed out within those insets. |
| **GridPane** | GridPane lays out its children within a flexible grid of rows and columns. If a border and/or padding is set, then its content will be layed out within those insets. |
| **BorderPane** | BorderPane lays out children in top, left, right, bottom, and center positions. |
| **FlowPnae** | FlowPane lays out its children in a flow that wraps at the flowpane's boundary.  A horizontal flowpane (the default) will layout nodes in rows, wrapping at the flowpane's width. A vertical flowpane lays out nodes in columns, wrapping at the flowpane's height. If the flowpane has a border and/or padding set, the content will be flowed within those insets. |
| **ScrollPane** | A Control that provides a scrolled, clipped viewport of its contents. It allows the user to scroll the content around either directly (panning) or by using scroll bars. |
| **Pane** | Base class for layout panes which need to expose the children list as public so that users of the subclass can freely add/remove children.  This class may be used directly in cases where absolute positioning of children is required since it does not perform layout beyond resizing resizable children to their preferred sizes. It is the application's responsibility to position the children since the pane leaves the positions alone during layout |
| **Background** | The Background of a [Region](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/Region.html). A Background is an immutable object which encapsulates the entire set of data required to render the background of a Region. Because this class is immutable, you can freely reuse the same Background on many different Regions. Please refer to JavaFX CSS Reference for a complete description of the CSS rules for styling the background of a Region. |
| **BackgroundFill** | The fill and associated properties that direct how to fill the background of a [Region](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/Region.html). Because BackgroundFill is an immutable object, it can safely be used in any cache, and can safely be reused among multiple Regions or multiple times in the same Region. |

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

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **File** | An abstract representation of file and directory pathnames. |
| **IOException** | Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations. |
| **OutputStream** | This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink. |
| **FileInputStream** | A FileInputStream obtains input bytes from a file in a file system. What files are available depends on the host environment.  FileInputStream is meant for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader |
| **FileOutputStream** | A file output stream is an output stream for writing data to a File or to a FileDescriptor. Whether or not a file is available or may be created depends upon the underlying platform. Some platforms, in particular, allow a file to be opened for writing by only one FileOutputStream (or other file-writing object) at a time. In such situations the constructors in this class will fail if the file involved is already open |
| **I/O exception** | Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations. |
| **inputerStream** | Applications that need to define a subclass of InputStream must always provide a method that returns the next byte of input |
| **PrintWriter** | Prints formatted representations of objects to a text-output stream. This class implements all of the print methods found in [PrintStream](https://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html). It does not contain methods for writing raw bytes, for which a program should use unencoded byte streams. |
| **StringWriter** | A character stream that collects its output in a string buffer, which can then be used to construct a string.  Closing a StringWriter has no effect. The methods in this class can be called after the stream has been closed without generating an IOException. |

Table 2.9: Uses for classes in the Java API’s java.io

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Path** | An object that may be used to locate a file in a file system. It will typically represent a system dependent file path. |
| **Paths** | This class consists exclusively of static methods that return a Path by converting a path string or URI. |
| **Files** | This class consists exclusively of static methods that operate on files, directories, or other types of files. |

Table 2.10: Uses for classes in the Java API’s java.nio.file

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ArrayList** | Resizable-array implementation of the List interface. |
| **HashMap** | Hash table based implementation of the Map interface. |
| **Optional** | A container object which may or may not contain a non-null value. If a value is present, isPresent() will return true and get() will return the value.  Additional methods that depend on the presence or absence of a contained value are provided, such as [orElse()](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElse-T-) (return a default value if value not present) and [ifPresent()](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html#ifPresent-java.util.function.Consumer-) (execute a block of code if the value is present). |
| **Locale** | A Locale object represents a specific geographical, political, or cultural region. An operation that requires a Locale to perform its task is called *locale-sensitive* and uses the Locale to tailor information for the user. For example, displaying a number is a locale-sensitive operation— the number should be formatted according to the customs and conventions of the user's native country, region, or culture. |
| **Logeer** | A Logger object is used to log messages for a specific system or application component. Loggers are normally named, using a hierarchical dot-separated namespace. Logger names can be arbitrary strings, but they should normally be based on the package name or class name of the logged component, such as java.net or javax.swing. In addition it is possible to create "anonymous" Loggers that are not stored in the Logger namespace. |

Table 2.11: Uses for classes in the Java API’s java.util

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **URLSyntaxException** | Checked exception thrown to indicate that a string could not be parsed as a URI reference. |
| **URL** | Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web. |

Table 2.12: Uses for classes in the Java API’s java.net

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Application** | The entry point for JavaFX applications is the Application class. The JavaFX runtime does the following, in order, whenever an application is launched: |

Table 2.13: Uses for classes in the Java API’s javafx.application

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Level** | The Level class defines a set of standard logging levels that can be used to control logging output. The logging Level objects are ordered and are specified by ordered integers. Enabling logging at a given level also enables logging at all higher levels. |
| **Logger** | A Logger object is used to log messages for a specific system or application component. Loggers are normally named, using a hierarchical dot-separated namespace. Logger names can be arbitrary strings, but they should normally be based on the package name or class name of the logged component, such as java.net or javax.swing. In addition it is possible to create "anonymous" Loggers that are not stored in the Logger namespace. |

Table 2.14: Uses for classes in the Java API’s java.util.logging

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Launch** | A Logger object is used to log messages for a specific system or application component. Loggers are normally named, using a hierarchical dot-separated namespace. Logger names can be arbitrary strings, but they should normally be based on the package name or class name of the logged component, such as java.net or javax.swing. In addition it is possible to create "anonymous" Loggers that are not stored in the Logger namespace. |

Table 2.15: Uses for classes in the Java API’s javafx.appication.Application

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ExtensionFilter** | Defines an extension filter, used for filtering which files can be chosen in a FileDialog based on the file name extensions. |

Table 2.16: Uses for classes in the Java API’s javafx.appication.FileChooser

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Scene** | The JavaFX Scene class is the container for all content in a scene graph. The background of the scene is filled as specified by the fill property. |
| **Cursor** | A class to encapsulate the bitmap representation of the mouse cursor. |
| **Node** | The Node interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Nodeinterface may have children. For example, Text nodes may not have children, and adding children to such nodes results in a DOMException being raised. |
| **SnapshotParameters** | Parameters used to specify the rendering attributes for Node snapshot. |

Table 2.17: Uses for classes in the Java API’s javafx.scene

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ImageView** | The ImageView is a Node used for painting images loaded with [Image](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html) class.  This class allows resizing the displayed image (with or without preserving the original aspect ratio) and specifying a viewport into the source image for restricting the pixels displayed by this ImageView. |
| **Image** | The abstract class Image is the superclass of all classes that represent graphical images. The image must be obtained in a platform-specific manner. |
| **WriableImage** | The WritableImage class represents a custom graphical image that is constructed from pixels supplied by the application, and possibly from PixelReader objects from any number of sources, including images read from a file or URL. |

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

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Text** | Provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages. |

Table 2.19: Uses for classes in the Java API’s javafx.scene.text

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **BlurType** | Represents the type of blur algorithm that is used to soften a Shadow effect. |
| **DropShadow** | A high-level effect that renders a shadow of the given content behind the content with the specified color, radius, and offset. |
| **Effect** | The abstract base class for all effect implementations. An effect is a graphical algorithm that produces an image, typically as a modification of a source image. An effect can be associated with a scene graph Node by setting the [Node.effect](https://docs.oracle.com/javafx/2/api/javafx/scene/Node.html#effectProperty()) attribute. Some effects change the color properties of the source pixels (such as [ColorAdjust](https://docs.oracle.com/javafx/2/api/javafx/scene/effect/ColorAdjust.html)), others combine multiple images together (such as [Blend](https://docs.oracle.com/javafx/2/api/javafx/scene/effect/Blend.html)), while still others warp or move the pixels of the source image around (such as [DisplacementMap](https://docs.oracle.com/javafx/2/api/javafx/scene/effect/DisplacementMap.html) or [PerspectiveTransform](https://docs.oracle.com/javafx/2/api/javafx/scene/effect/PerspectiveTransform.html)). All effects have at least one input defined and the input can be set to another effect to chain the effects together and combine their results, or it can be left unspecified in which case the effect will operate on a graphical rendering of the node it is attached to. |

Table 2.20: Uses for classes in the Java API’s javafx.scene.text

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ImageIO** | A class containing static convenience methods for locating ImageReaders and ImageWriters, and performing simple encoding and decoding. |

Table 2.21: Uses for classes in the Java API’s javafxi.mageio

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ActionEvent** | A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every ActionListener object that registered to receive such events using the component's addActionListener method. |
| **EventHandler** | This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference. |

Table 2.22: Uses for classes in the Java API’s javafx.event

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Json** | [JSON](http://json.org/) (JavaScript Object Notation) is a lightweight, text-based, language-independent data exchange format that is easy for humans and machines to read and write. JSON can represent two structured types: objects and arrays. An object is an unordered collection of zero or more name/value pairs. An array is an ordered sequence of zero or more values. The values can be strings, numbers, booleans, null, and these two structured types. |
| **JsonArray** | JsonArray represents an immutable JSON array (an ordered sequence of zero or more values). It also provides an unmodifiable list view of the values in the array.  A JsonArray object can be created by reading JSON data from an input source or it can be built from scratch using an array builder object. |
| **JsonArrayBuilder** | A builder for creating [JsonArray](https://docs.oracle.com/javaee/7/api/javax/json/JsonArray.html) models from scratch. This interface initializes an empty JSON array model and provides methods to add values to the array model and to return the resulting array. The methods in this class can be chained to add multiple values to the array. |
| **JsonNumber** | Implementations may use a [BigDecimal](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html?is-external=true) object to store the numeric value internally. The BigDecimal object can be constructed from the following types: [int](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html?is-external=true#BigDecimal-int-), [long](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html?is-external=true#BigDecimal-long-), [BigInteger](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html?is-external=true#BigDecimal-java.math.BigInteger-), [double](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html?is-external=true#valueOf-double-), and [String](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html?is-external=true#BigDecimal-java.lang.String-). Some of the method semantics in this class are defined using the BigDecimal semantics. |
| **JsonObject** | JsonObject class represents an immutable JSON object value (an unordered collection of zero or more name/value pairs). It also provides unmodifiable map view to the JSON object name/value mappings. |
| **JsonReader** | Reads a JSON [object](https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html) or an [array](https://docs.oracle.com/javaee/7/api/javax/json/JsonArray.html) structure from an input source.  The class [Json](https://docs.oracle.com/javaee/7/api/javax/json/Json.html) contains methods to create readers from input sources ([InputStream](http://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html?is-external=true) and [Reader](http://docs.oracle.com/javase/7/docs/api/java/io/Reader.html?is-external=true)).  The following example demonstrates how to read an empty JSON array from a string: |
| **JsonValue** | JsonValue represents an immutable JSON value. |
| **JsonWriter** | Writes a JSON [object](https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html) or [array](https://docs.oracle.com/javaee/7/api/javax/json/JsonArray.html) structure to an output source.  The class [Json](https://docs.oracle.com/javaee/7/api/javax/json/Json.html) contains methods to create writers from output sources ([OutputStream](http://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html?is-external=true) and [Writer](http://docs.oracle.com/javase/7/docs/api/java/io/Writer.html?is-external=true)). |
| **JsonWriterFactory** | Factory to create [JsonWriter](https://docs.oracle.com/javaee/7/api/javax/json/JsonWriter.html) instances. If a factory instance is configured with some configuration, that would be used to configure the created writer instances.  [JsonWriter](https://docs.oracle.com/javaee/7/api/javax/json/JsonWriter.html) can also be created using [Json](https://docs.oracle.com/javaee/7/api/javax/json/Json.html)'s createWriter methods. If multiple writer instances are created, then creating them using a writer factory is preferred. |

Table 2.23: Uses for classes in the Java API’s javax.json

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **JsonGenerator** | Writes JSON data to an output source in a streaming way. The class [Json](https://docs.oracle.com/javaee/7/api/javax/json/Json.html) contains methods to create generators for character or output streams ([Writer](http://docs.oracle.com/javase/7/docs/api/java/io/Writer.html?is-external=true) and [OutputStream](http://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html?is-external=true)). |

Table 2.24: Uses for classes in the Java API’s javafx.geometry.Pos

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **SchemaFactory** | [SchemaFactory](https://docs.oracle.com/javase/7/docs/api/javax/xml/validation/SchemaFactory.html) is a schema compiler. It reads external representations of schemas and prepares them for validation.  The [SchemaFactory](https://docs.oracle.com/javase/7/docs/api/javax/xml/validation/SchemaFactory.html) class is not thread-safe. In other words, it is the application's responsibility to ensure that at most one thread is using a [SchemaFactory](https://docs.oracle.com/javase/7/docs/api/javax/xml/validation/SchemaFactory.html) object at any given moment. Implementations are encouraged to mark methods as synchronized to protect themselves from broken clients |
| **Validator** | A validator object is not thread-safe and not reentrant. In other words, it is the application's responsibility to make sure that one [Validator](https://docs.oracle.com/javase/7/docs/api/javax/xml/validation/Validator.html) object is not used from more than one thread at any given time, and while the validate method is invoked, applications may not recursively call the validate method |
| **Schema** | A [Schema](https://docs.oracle.com/javase/7/docs/api/javax/xml/validation/Schema.html) object is thread safe and applications are encouraged to share it across many parsers in many threads.  A [Schema](https://docs.oracle.com/javase/7/docs/api/javax/xml/validation/Schema.html) object is immutable in the sense that it shouldn't change the set of constraints once it is created. In other words, if an application validates the same document twice against the same [Schema](https://docs.oracle.com/javase/7/docs/api/javax/xml/validation/Schema.html), it must always produce the same result. |

Table 2.25: Uses for classes in the Java API’s javafx.xml.validation

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **StreamSource** | Acts as an holder for a transformation Source in the form of a stream of XML markup. |

Table 2.26: Uses for classes in the Java API’s javafx.xml.transform.stream

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **SwingFXUtils** | This class provides utility methods for converting data types between Swing/AWT and JavaFX formats. |

Table 2.27: Uses for classes in the Java API’s javafx.embed.swing

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **DocumentBuilder** | Defines the API to obtain DOM Document instances from an XML document. Using this class, an application programmer can obtain a [Document](https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/Document.html) from XML. |
| **DocumentBuilder**  **Factory** | Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents. |
| **ParserConfiguration**  **Exception** | Indicates a serious configuration error. |

Table 2.28: Uses for classes in the Java API’s javax.xml.parsers

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Source** | An object that implements this interface contains the information needed to act as source input (XML source or transformation instructions). |

Table 2.29: Uses for classes in the Java API’s javax.xml.transfrom

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **SAXEception** | This class can contain basic error or warning information from either the XML parser or the application: a parser writer or application writer can subclass it to provide additional functionality. SAX handlers may throw this exception or any exception subclassed from it. |

Table 2.30: Uses for classes in the Java API’s org.xml.sax

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Document** | The Document interface represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data. |
| **NameNodeMap** | Objects implementing the NamedNodeMap interface are used to represent collections of nodes that can be accessed by name.  Note that NamedNodeMap does not inherit from NodeList.   NamedNodeMaps are not maintained in any particular order.  Objects contained in an object implementing NamedNodeMap may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a NamedNodeMap, and does not imply that the DOM specifies an order to these Nodes. |
| **Node** | The Node interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Nodeinterface may have children. For example, Text nodes may not have children, and adding children to such nodes results in a DOMException being raised. |
| **NodeList** | The NodeList interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented. NodeList objects in the DOM are live. |

Table 2.31: Uses for classes in the Java API’s org.xml.sax

**3. Class-Level Design Viewpoint**

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.

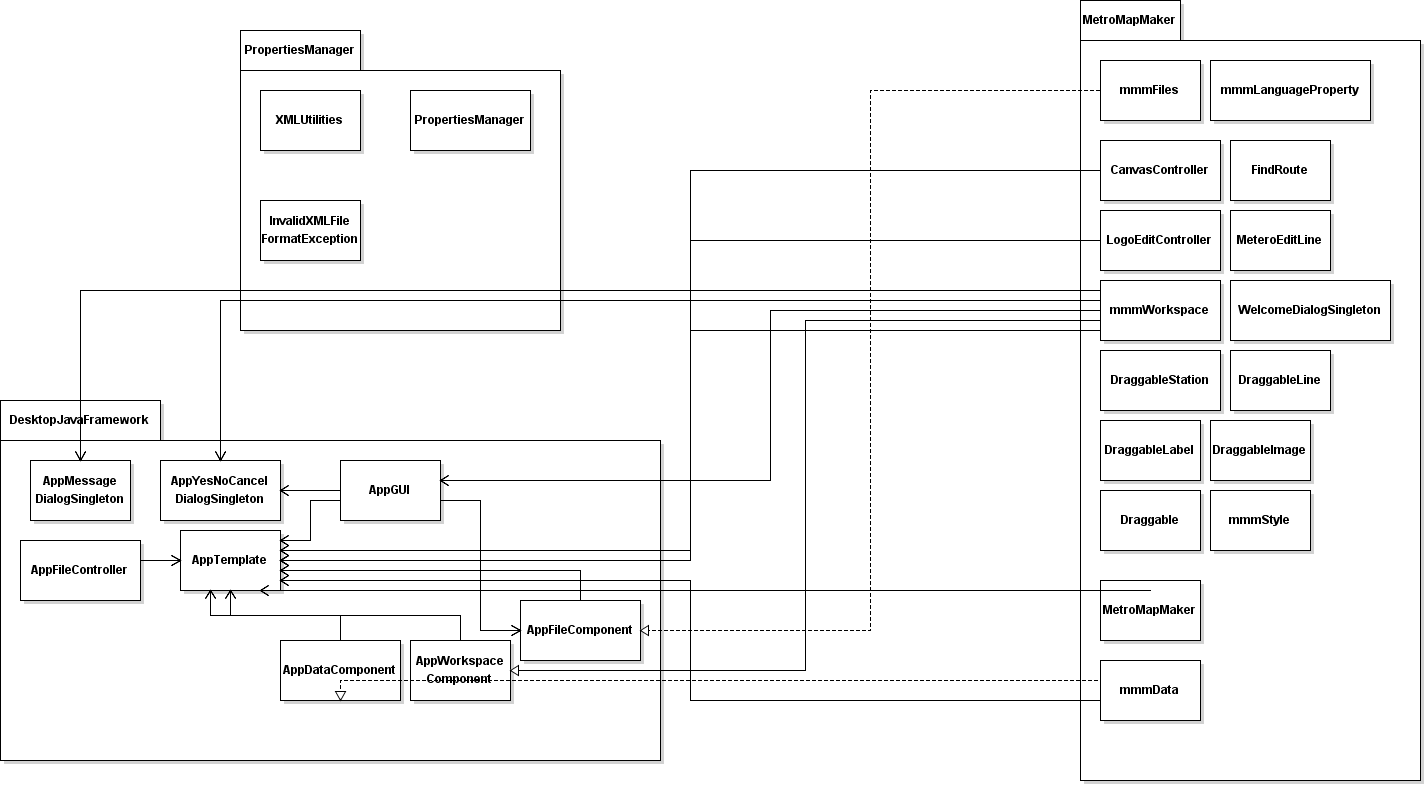
****

Figure 3.1 Metro Map Maker Framework Overview UML Class Diagram

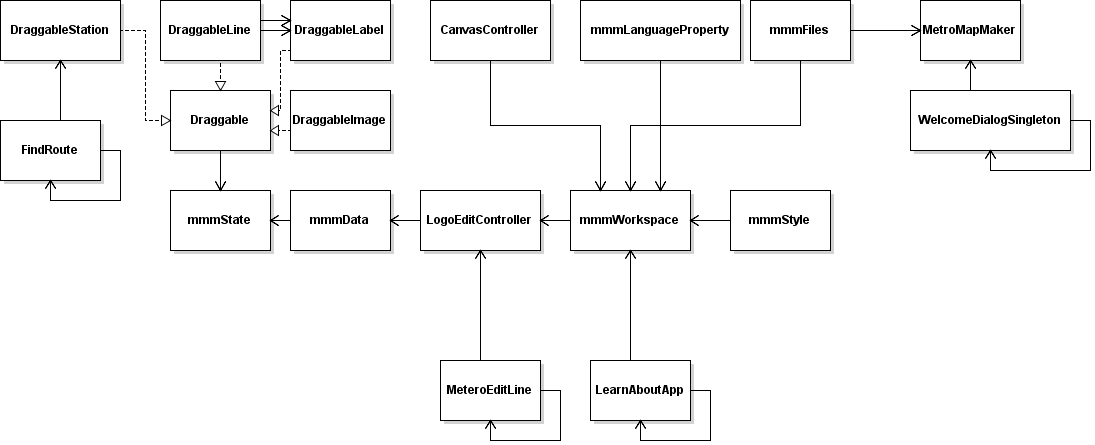


Figure 3.2: Metro Map Maker MetroMapMaker Overview UML Class

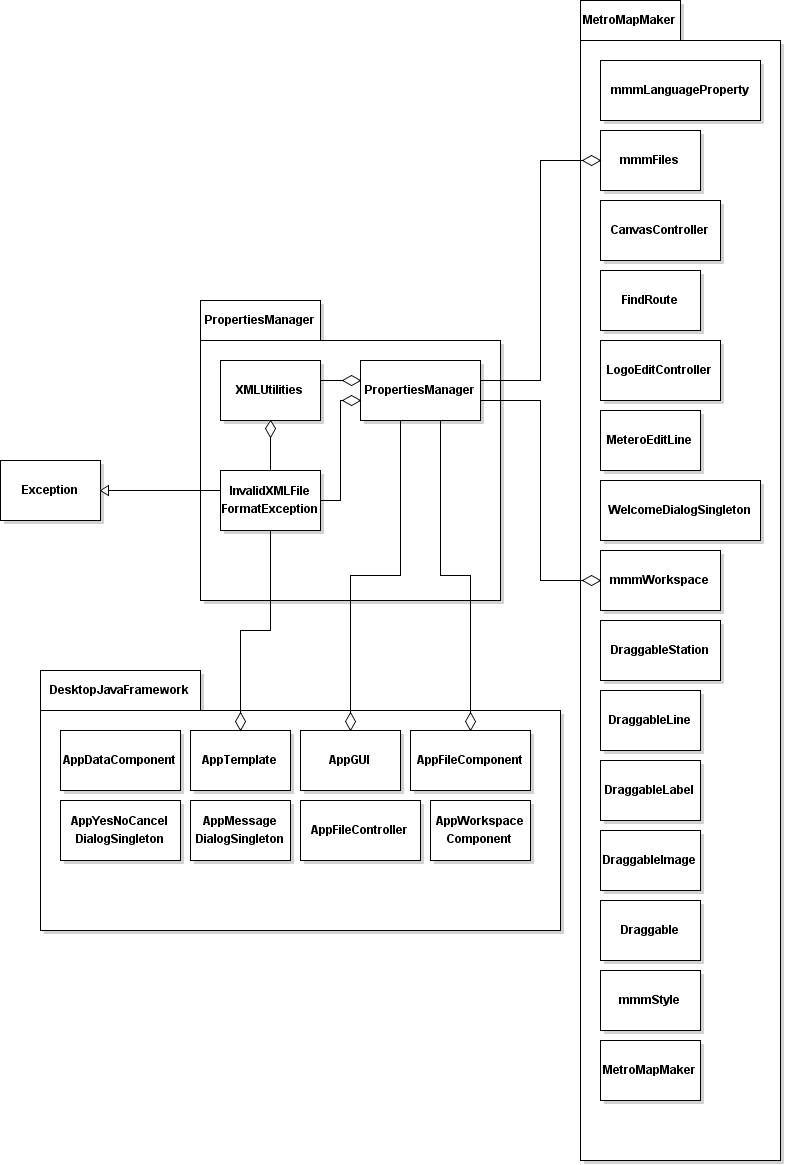


Figure 3.3 Metro Map Maker PropertiesManager Overview UML Class

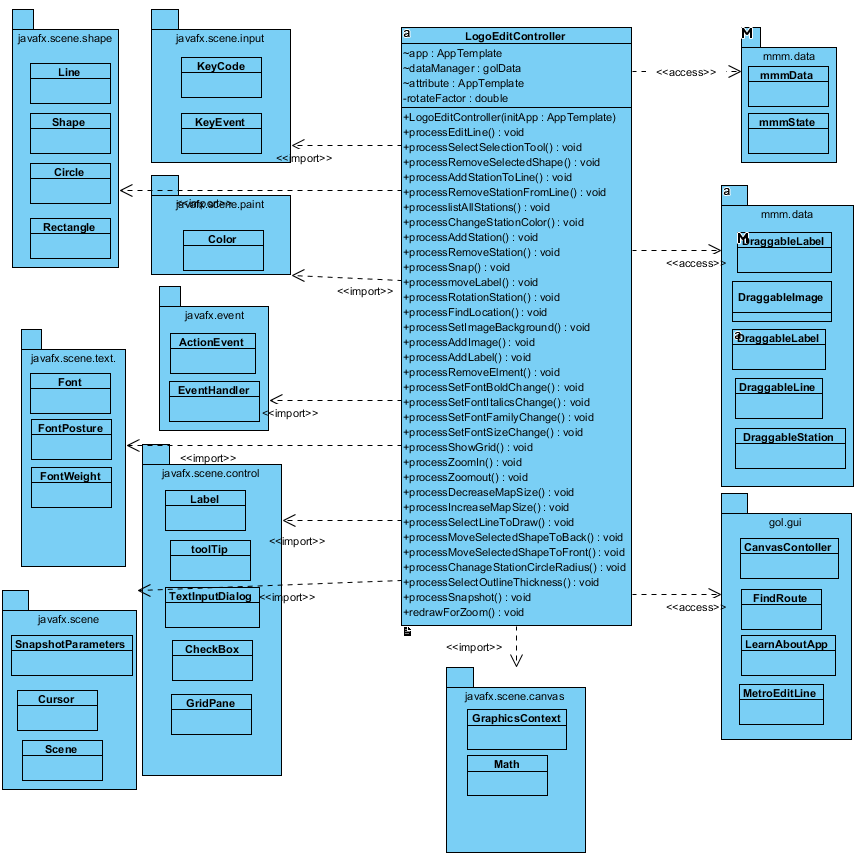


Figure 3.4 Detailed LogoEditController UML Class Diagram

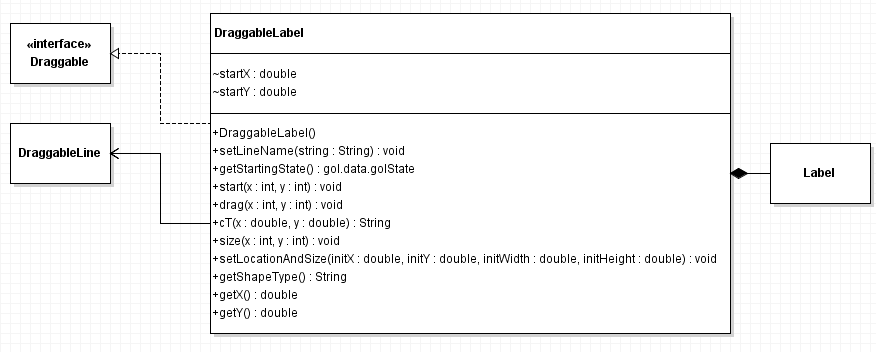


Figure 3.5 Detailed DraggableLabel UML Class Diagram

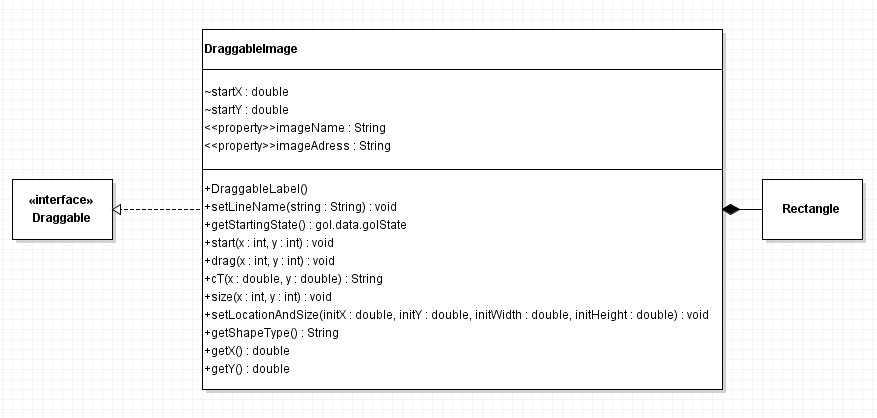


Figure 3.6 Detailed DraggableImage UML Class Diagram

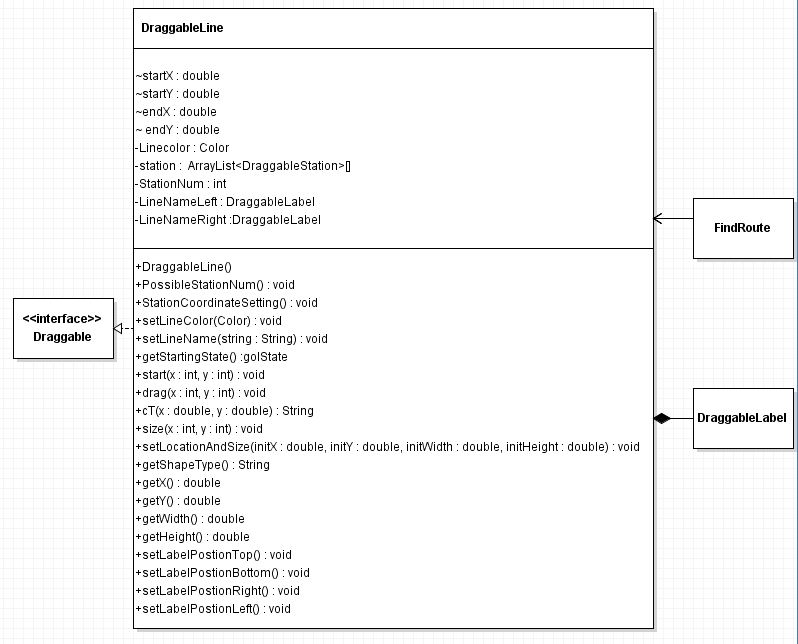


Figure 3.7 Detailed DraggableLine UML Class Diagram

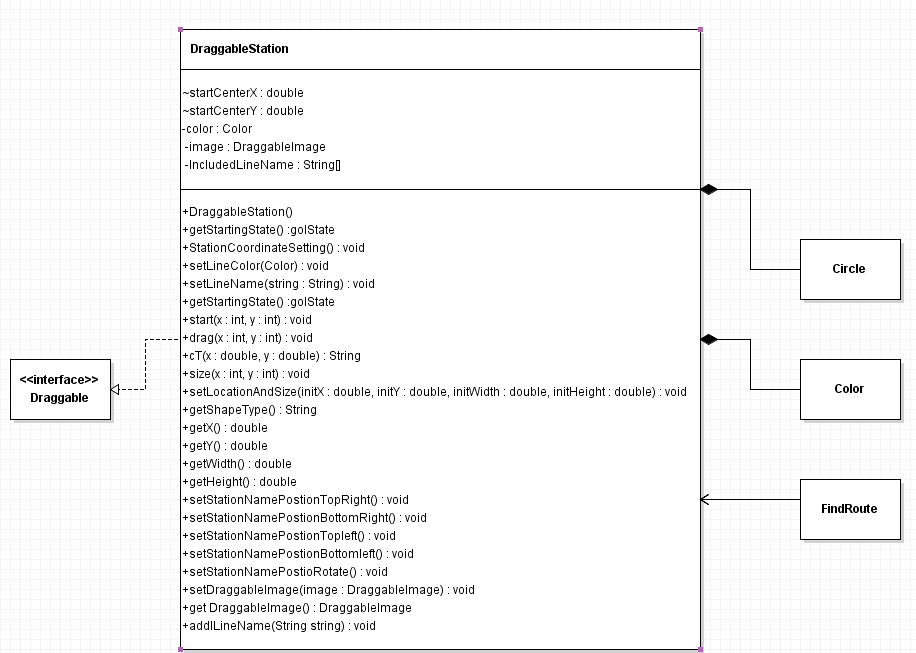


Figure 3.8 Detailed DraggableStation UML Class Diagram

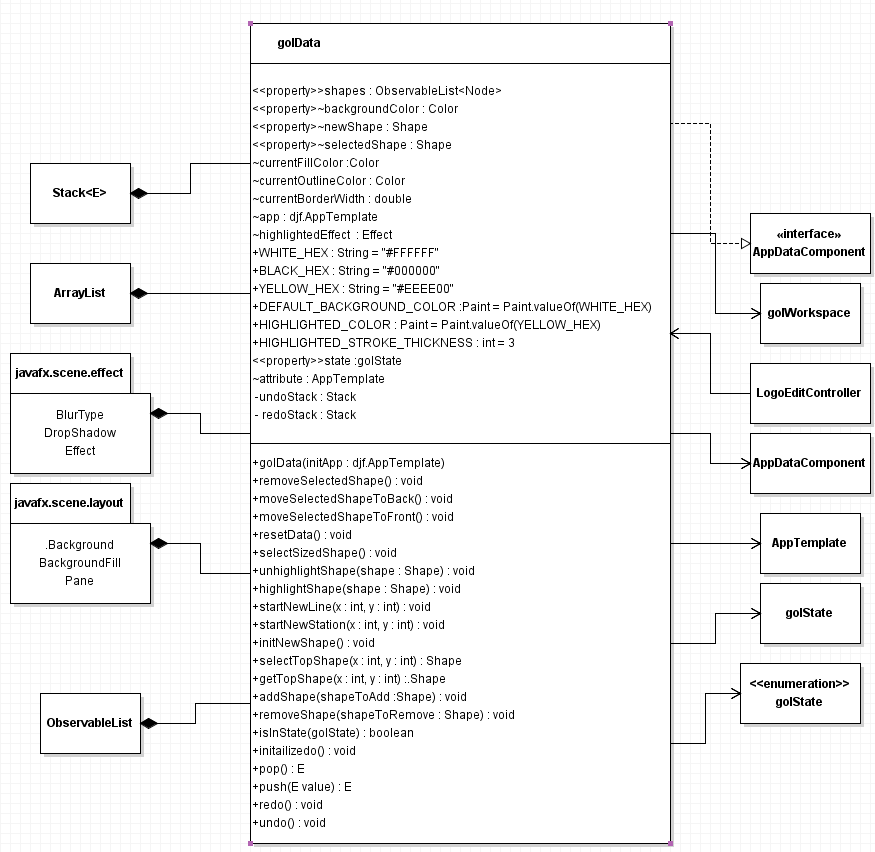


Figure 3.9 Detailed mmmData UML Class Diagram

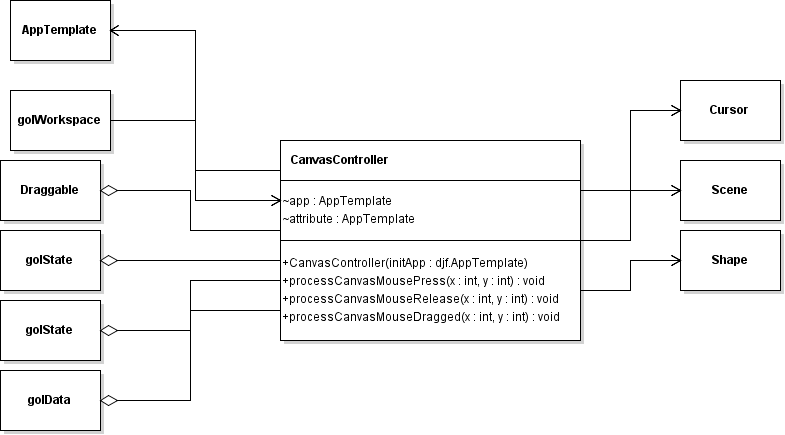


Figure 3.10 Detailed CanvasController UML Class Diagram

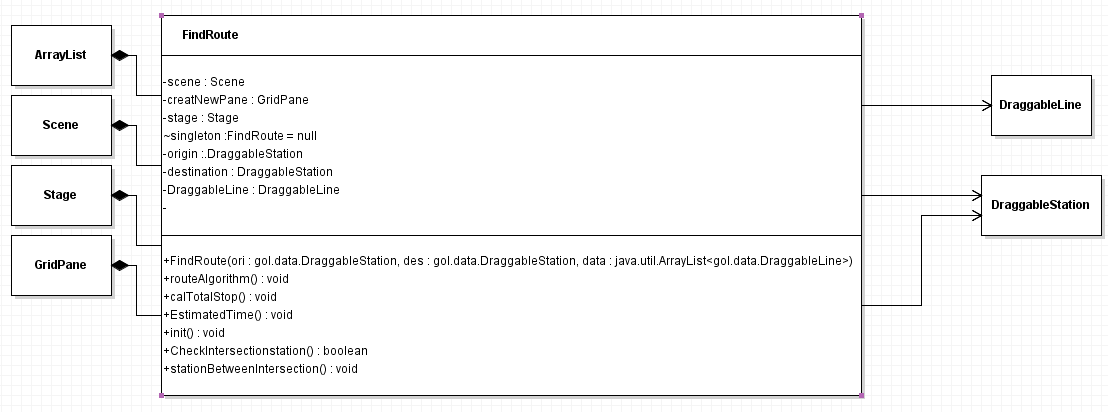


Figure 3.11 Detailed FindRoute UML Class Diagram

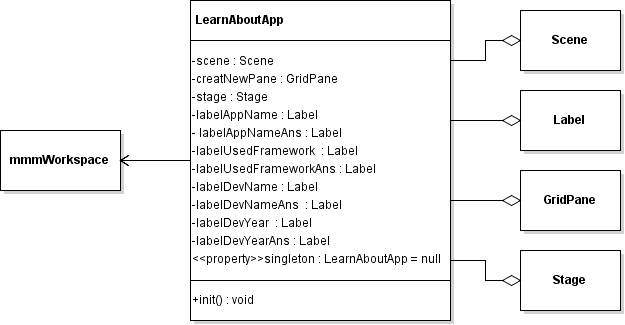


Figure 3.12 Detailed LearnAboutApp UML Class Diagram

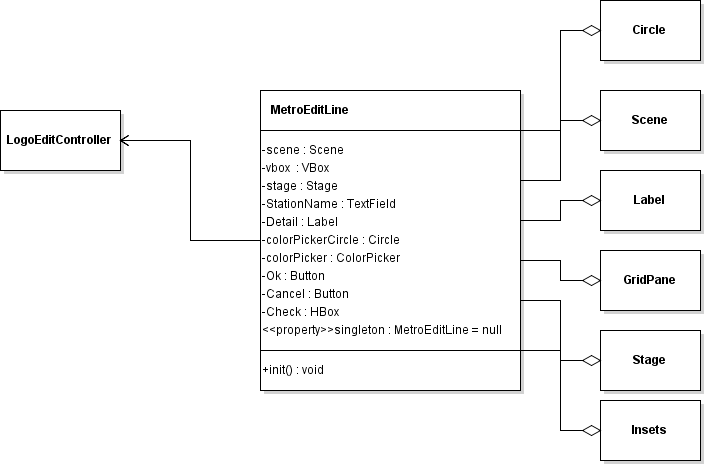


Figure 3.13 Detailed MetroEditLine UML Class Diagram

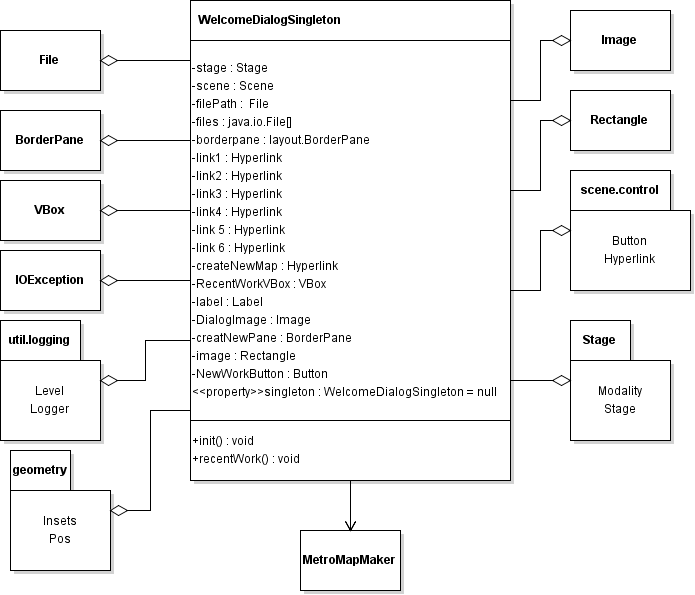


Figure 3.14 Detailed WelcomeDialogSingleton UML Class Diagram

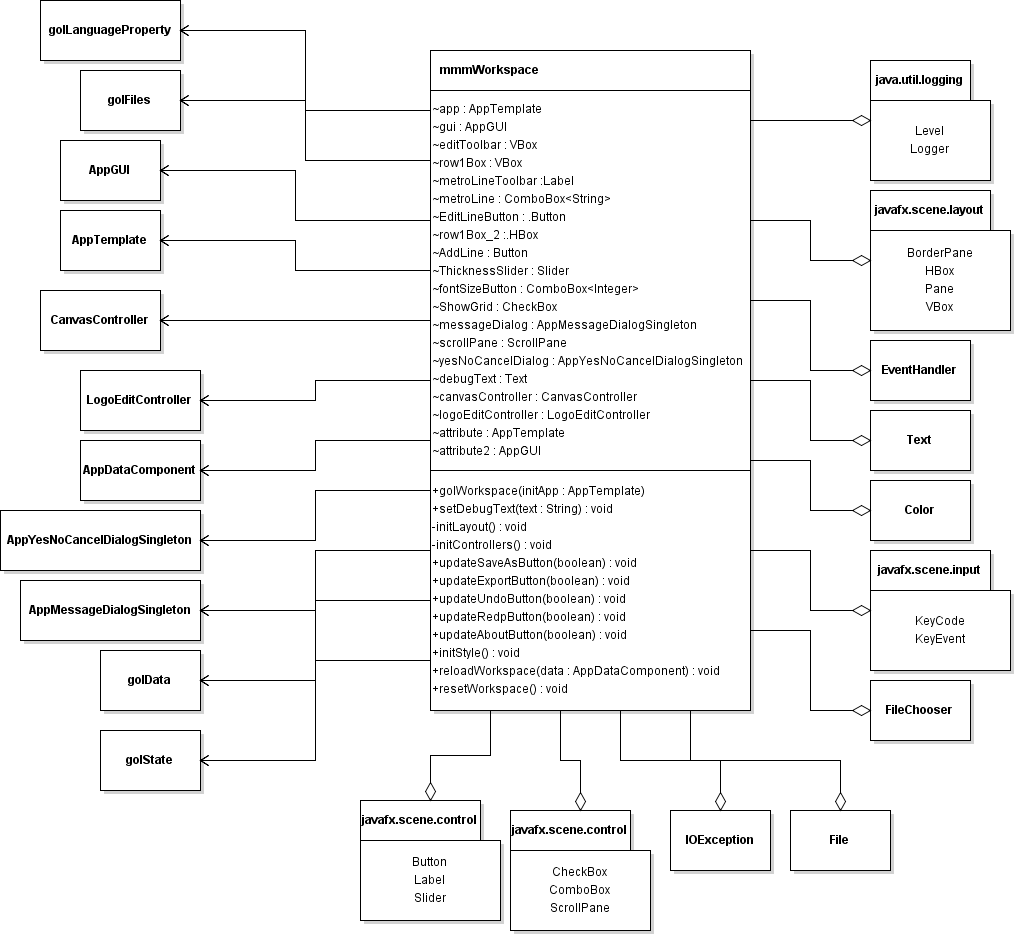


Figure 3.15 Detailed mmmWorkspace UML Class Diagram

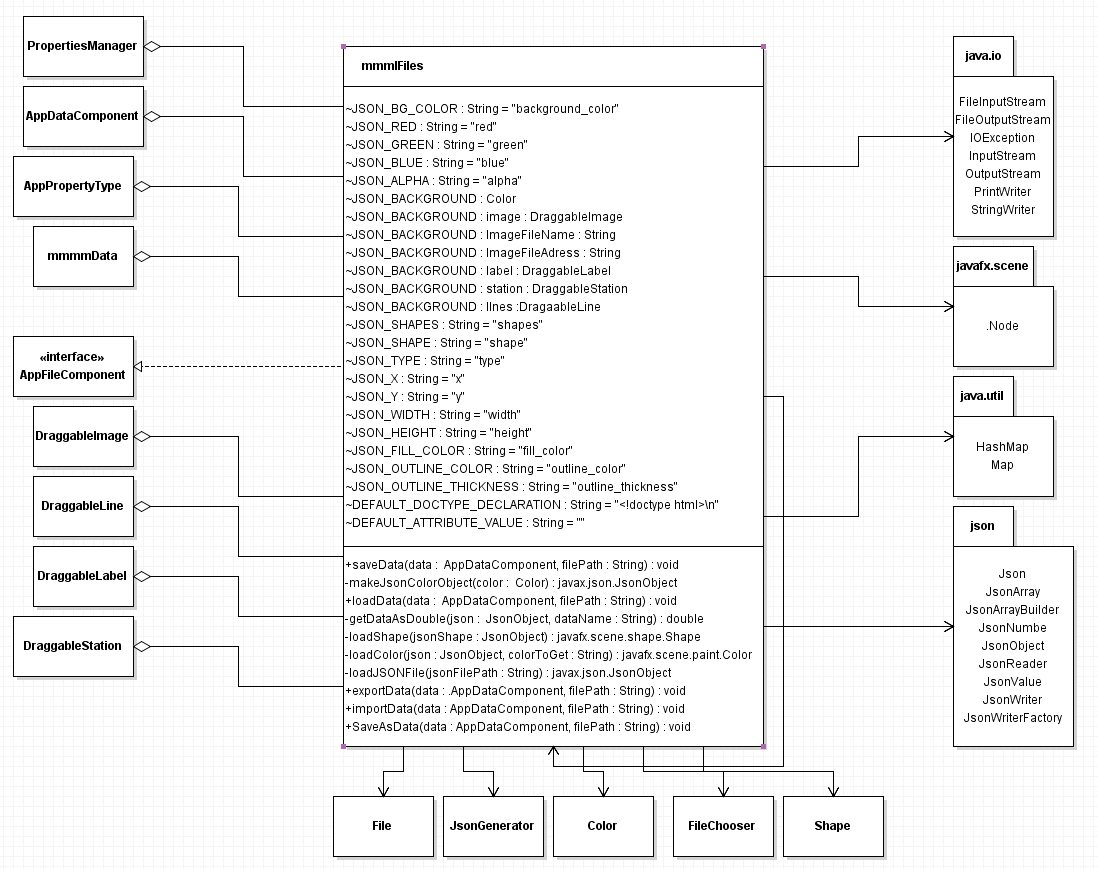


Figure 3.16 Detailed mmmFiles UML Class Diagram

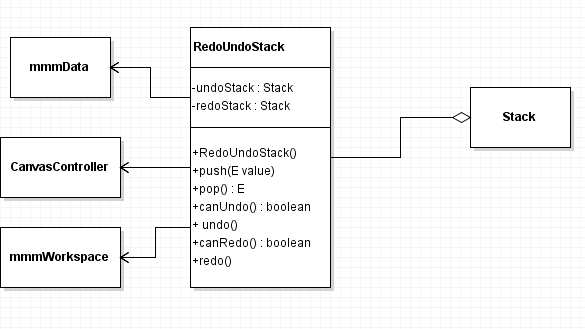


Figure 3.17 Detailed RedoUndoStack UML Class Diagram

**4. Method-Level Design Viewpoint**

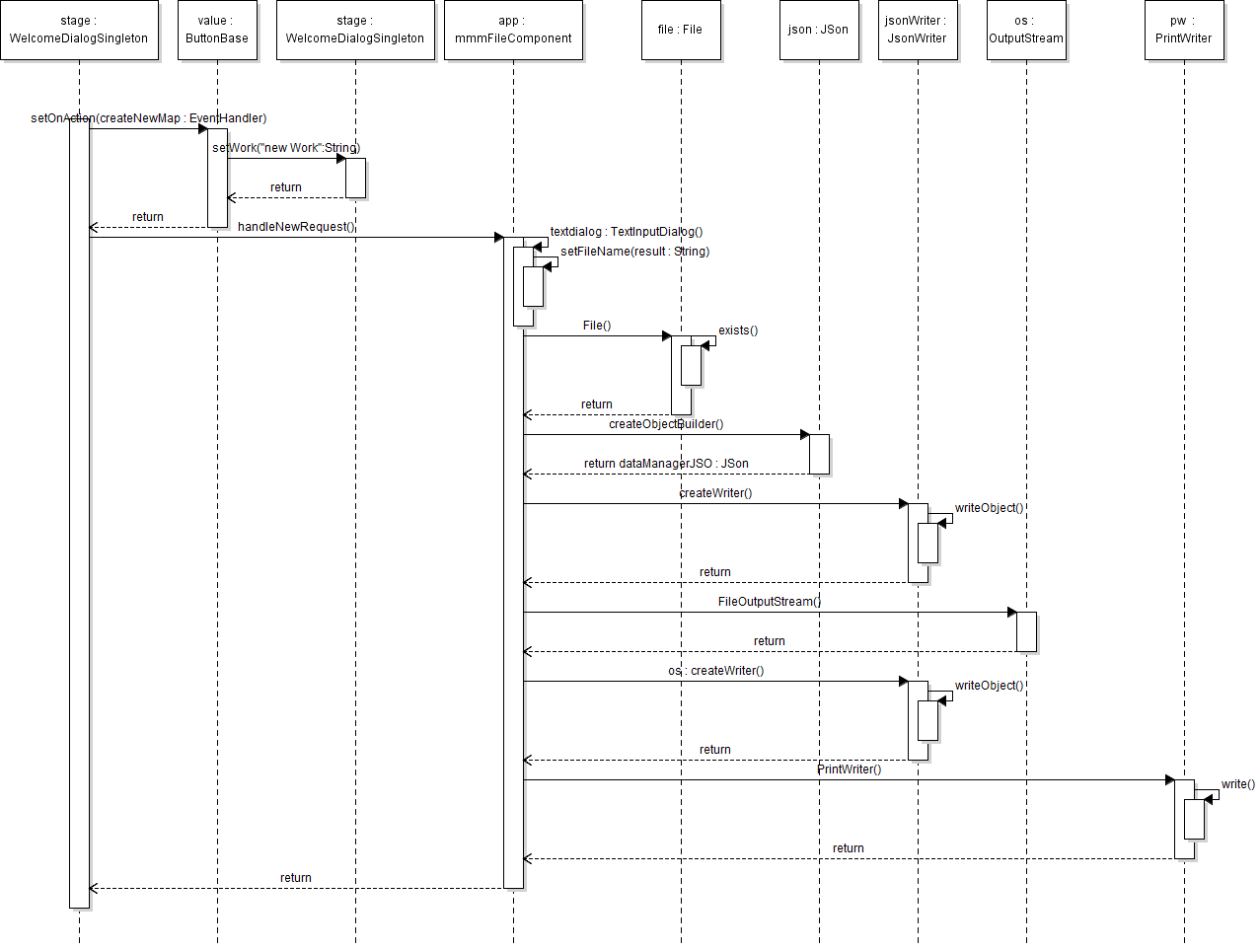
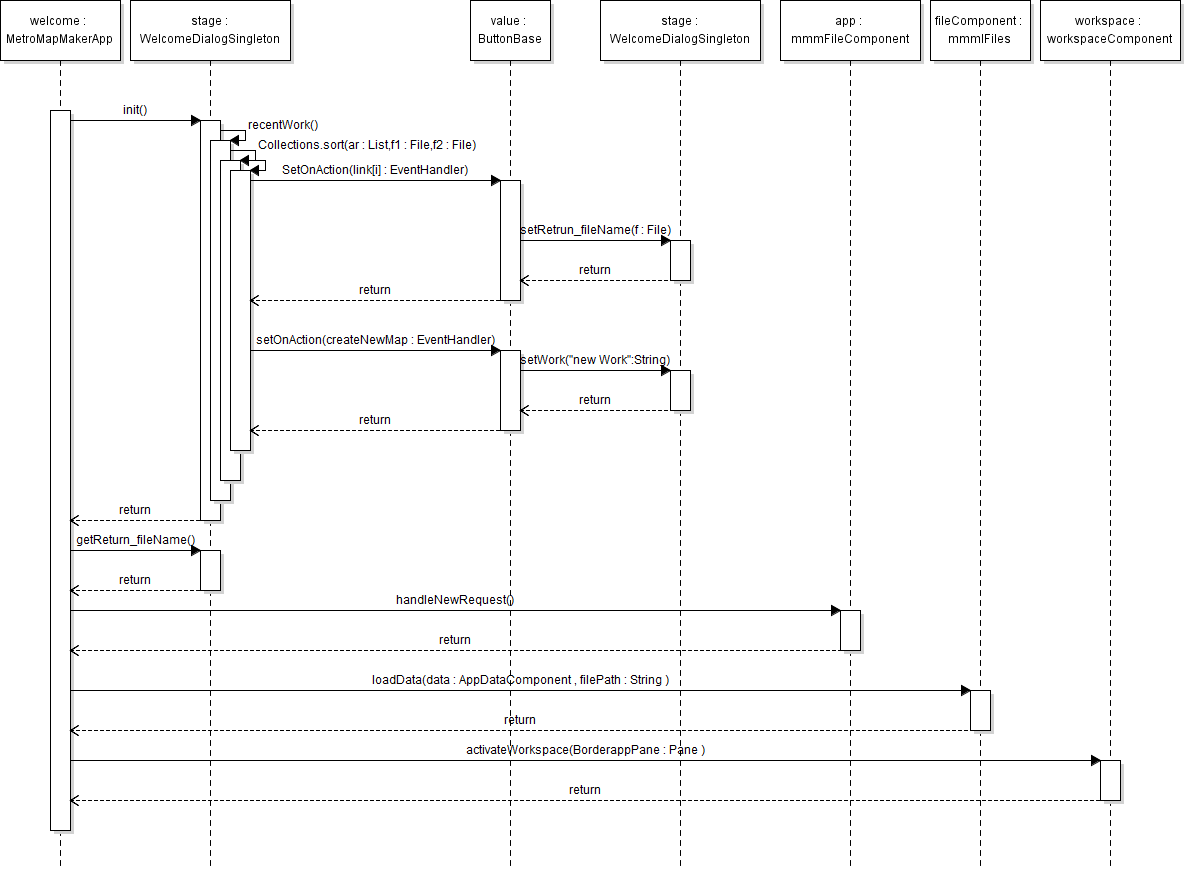
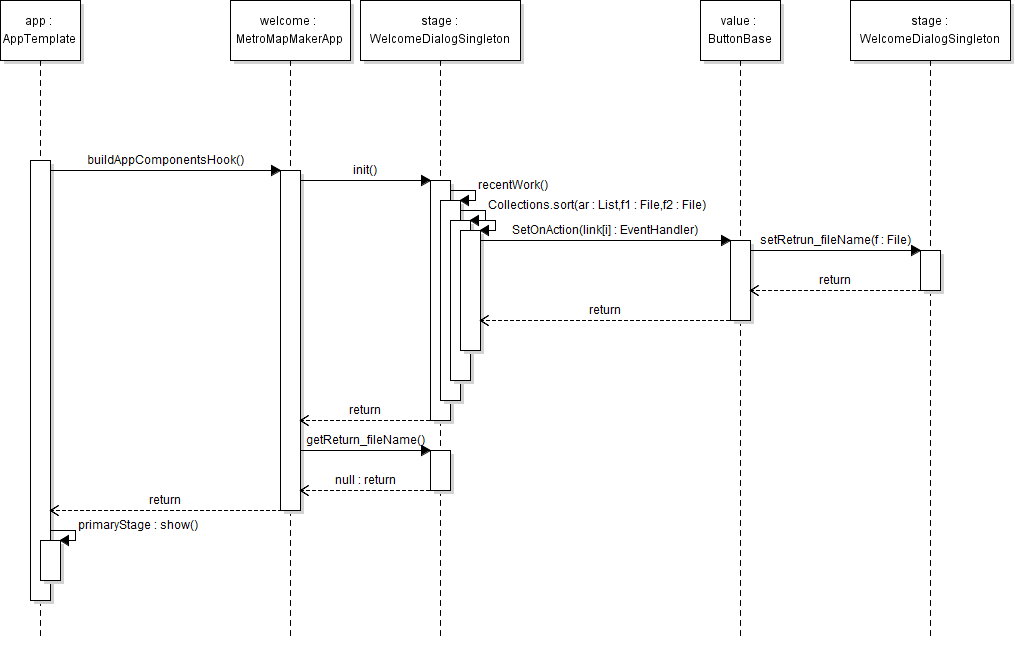
****

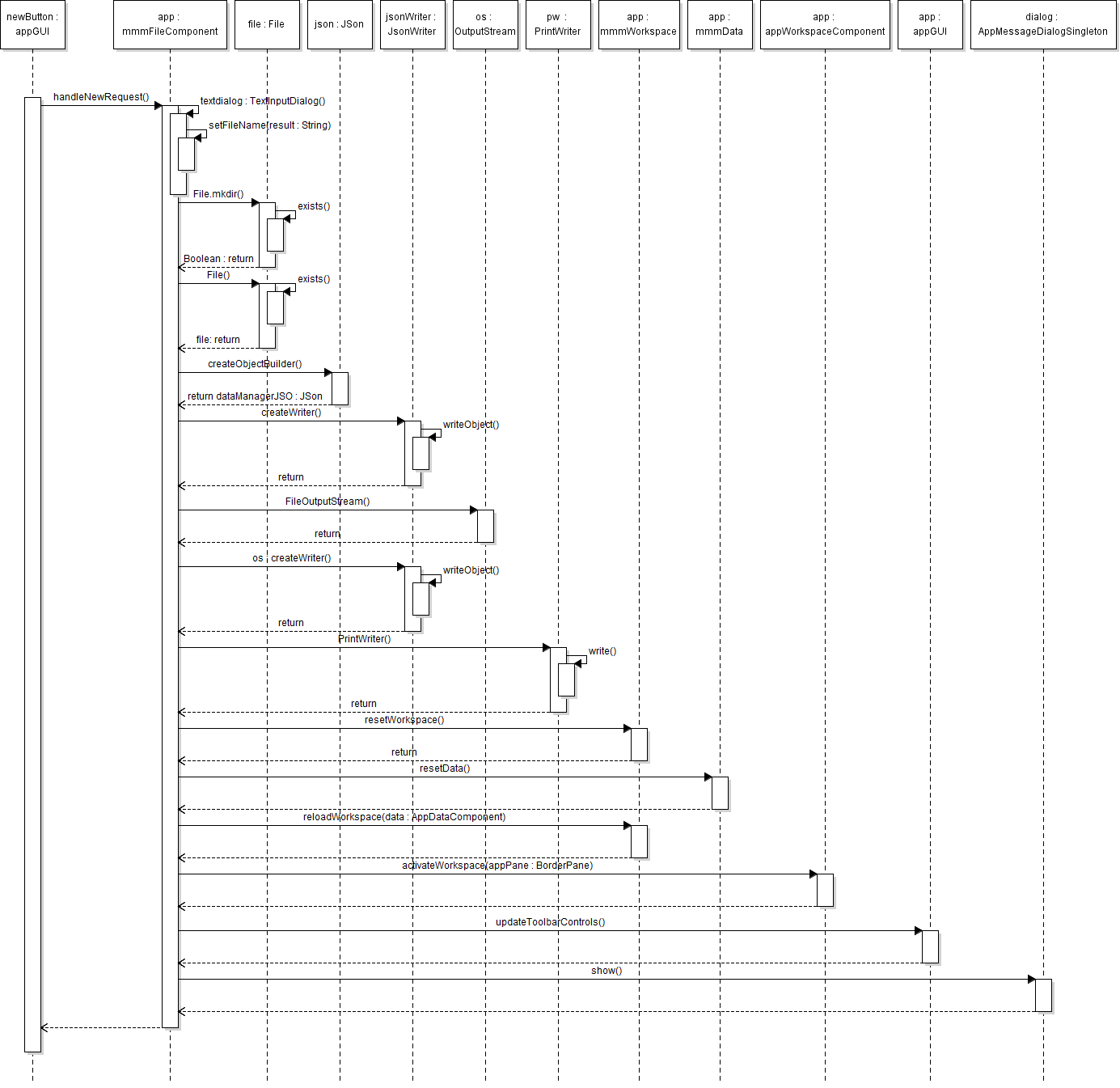
Figure 4.1: UML Sequence Diagram of Use Case 1: Create New Map



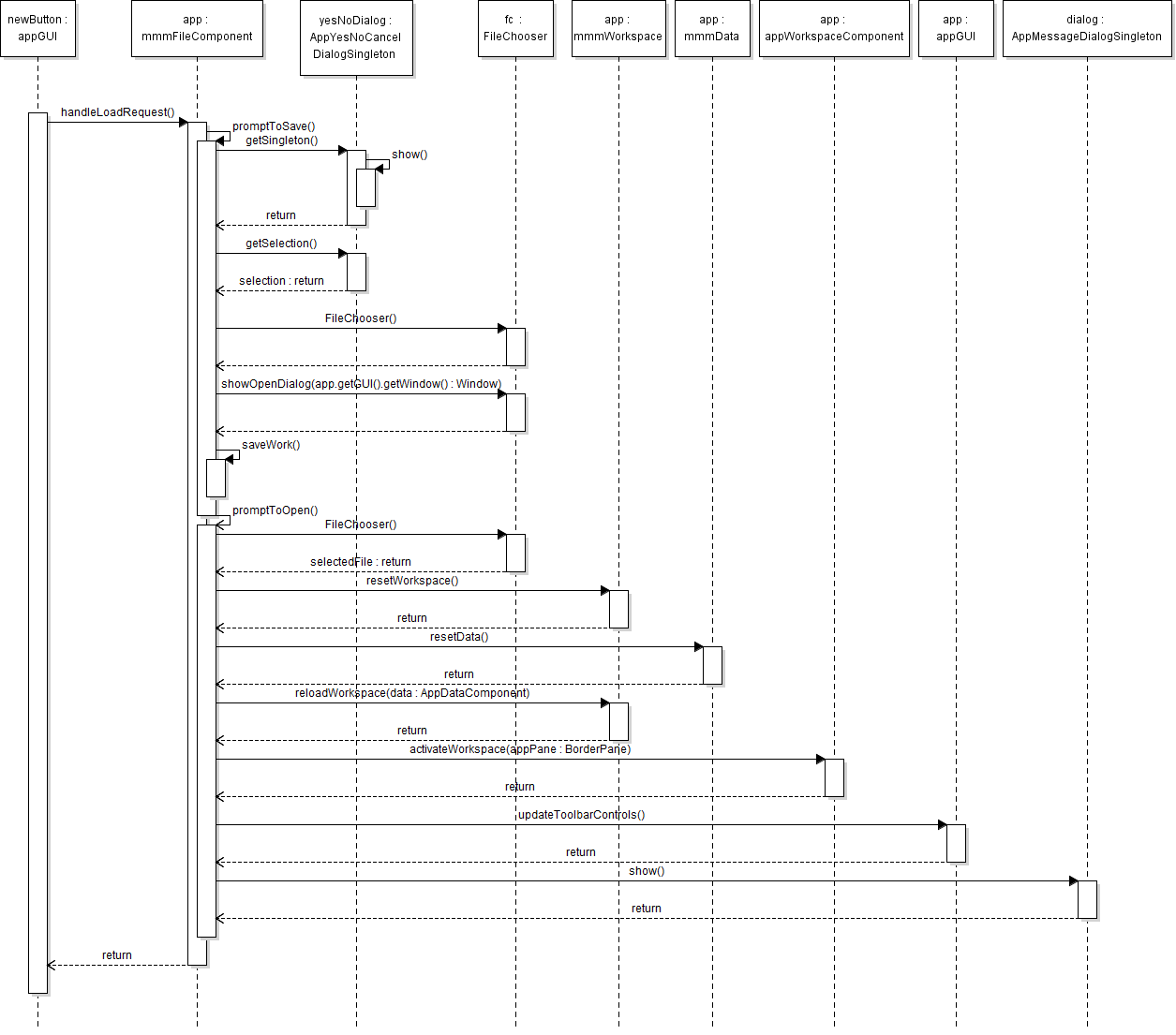
**Figure 4.2: UML Sequence Diagram of Use Case 2: Select Recent Map to Load**

****

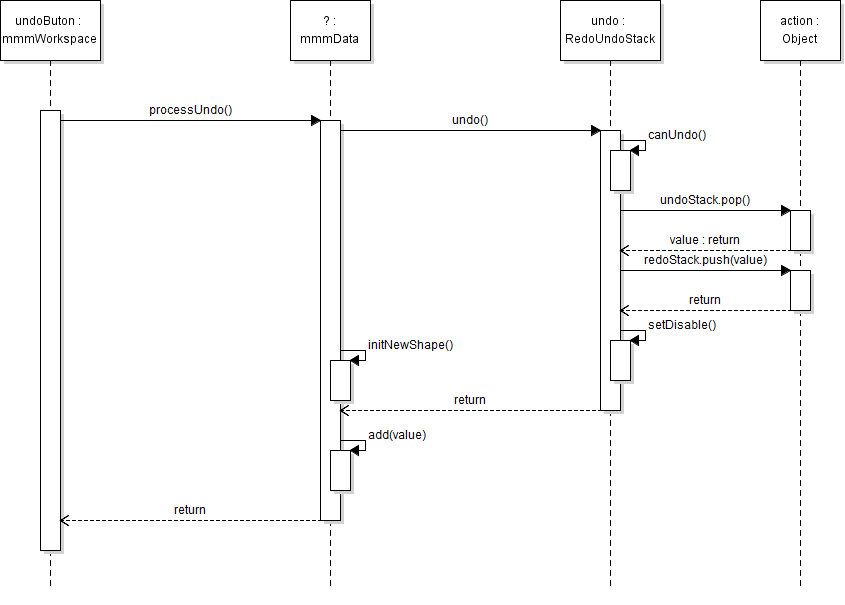
**Figure 4.3: UML Sequence Diagram of Use Case 3: Close Welcome Dialog**

****

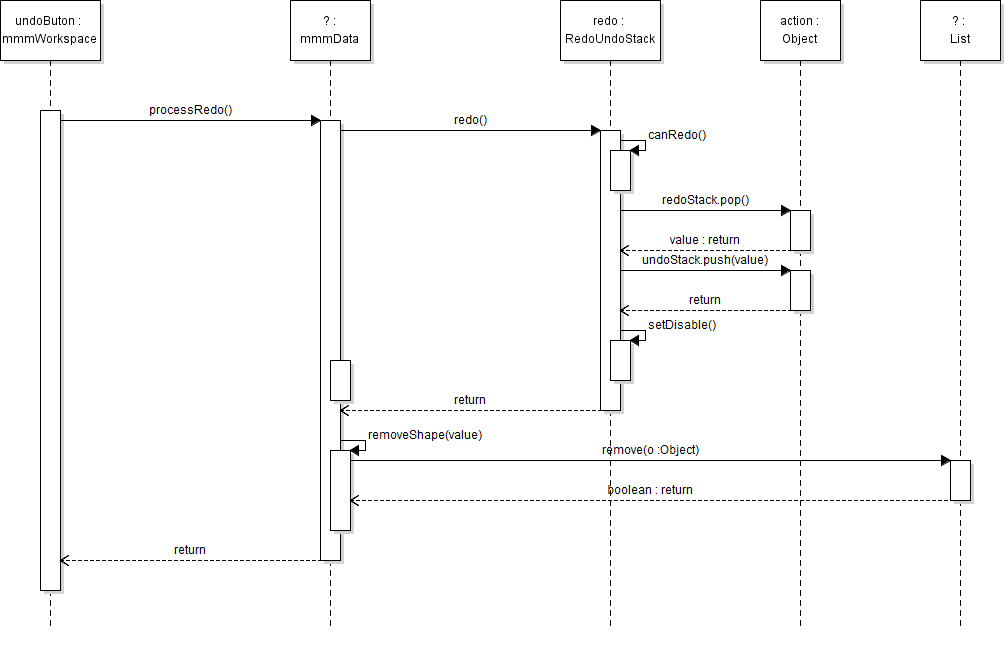
**Figure 4.4: UML Sequence Diagram of Use Case 4: Create New Map**

****

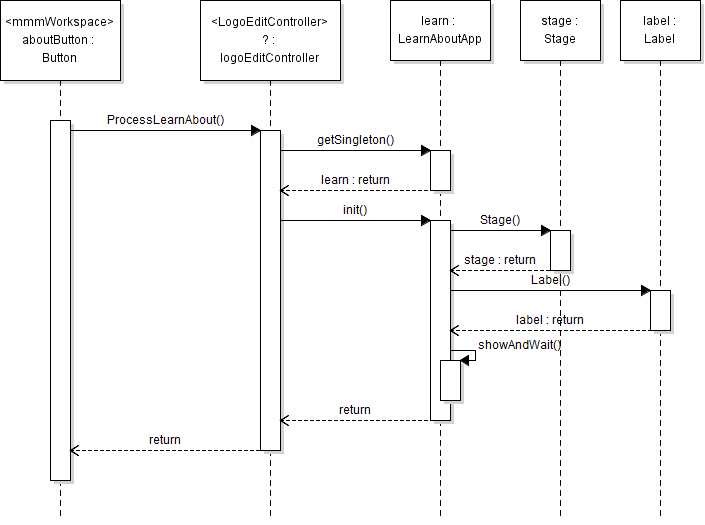
**Figure 4.5: UML Sequence Diagram of Use Case 5: Load Map**

****

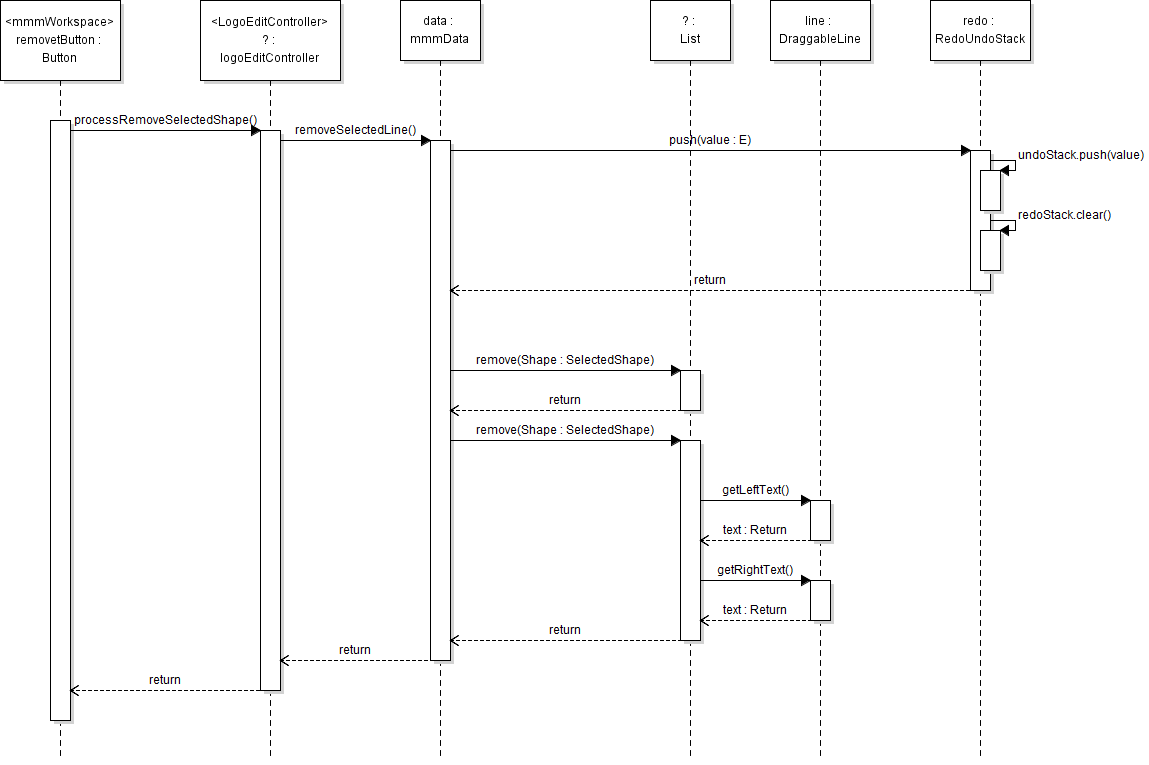
**Figure 4.6: UML Sequence Diagram of Use Case 9: Undo Edit**

****

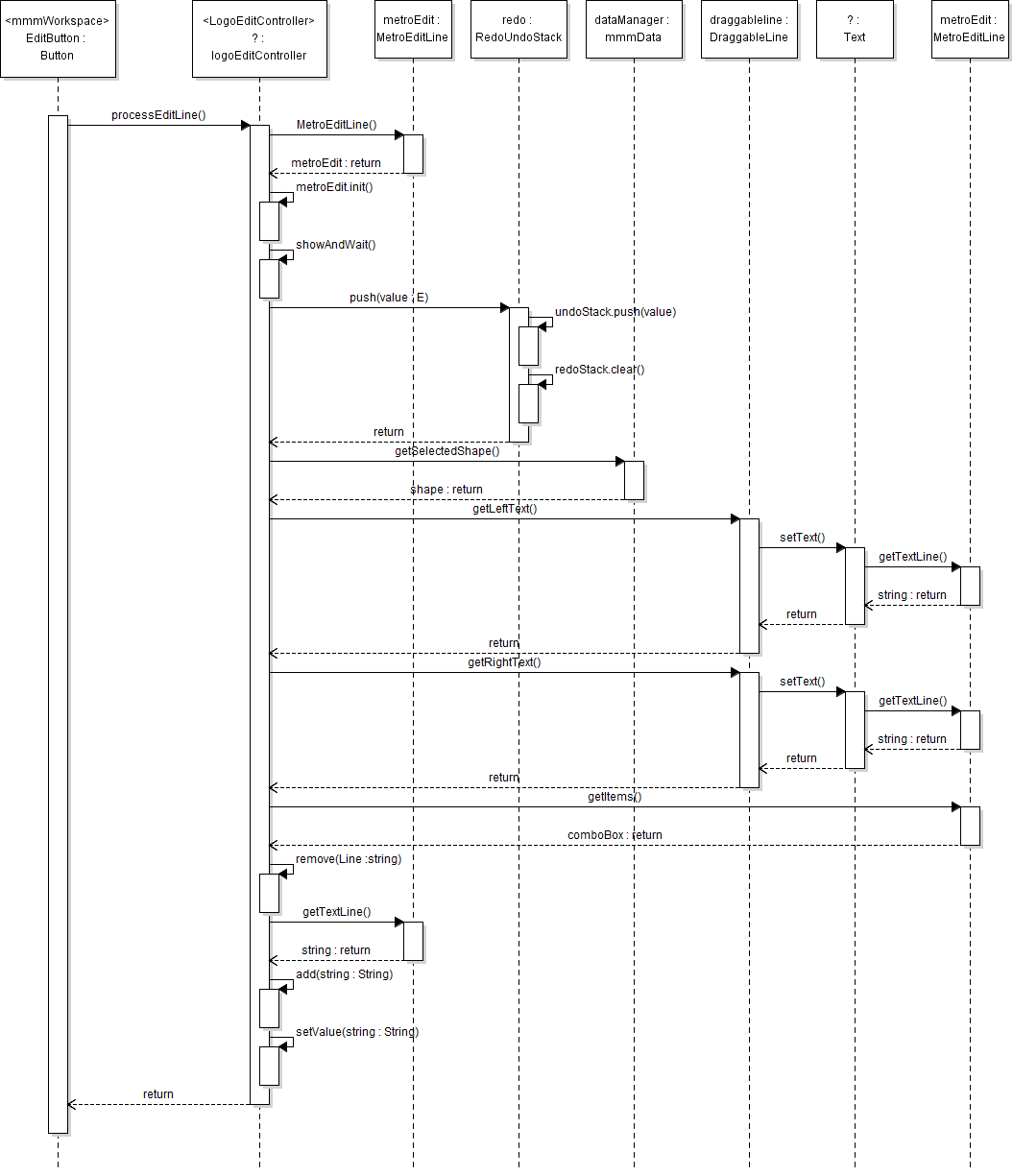
**Figure 4.7: UML Sequence Diagram of Use Case 10: Redo Edit**

****

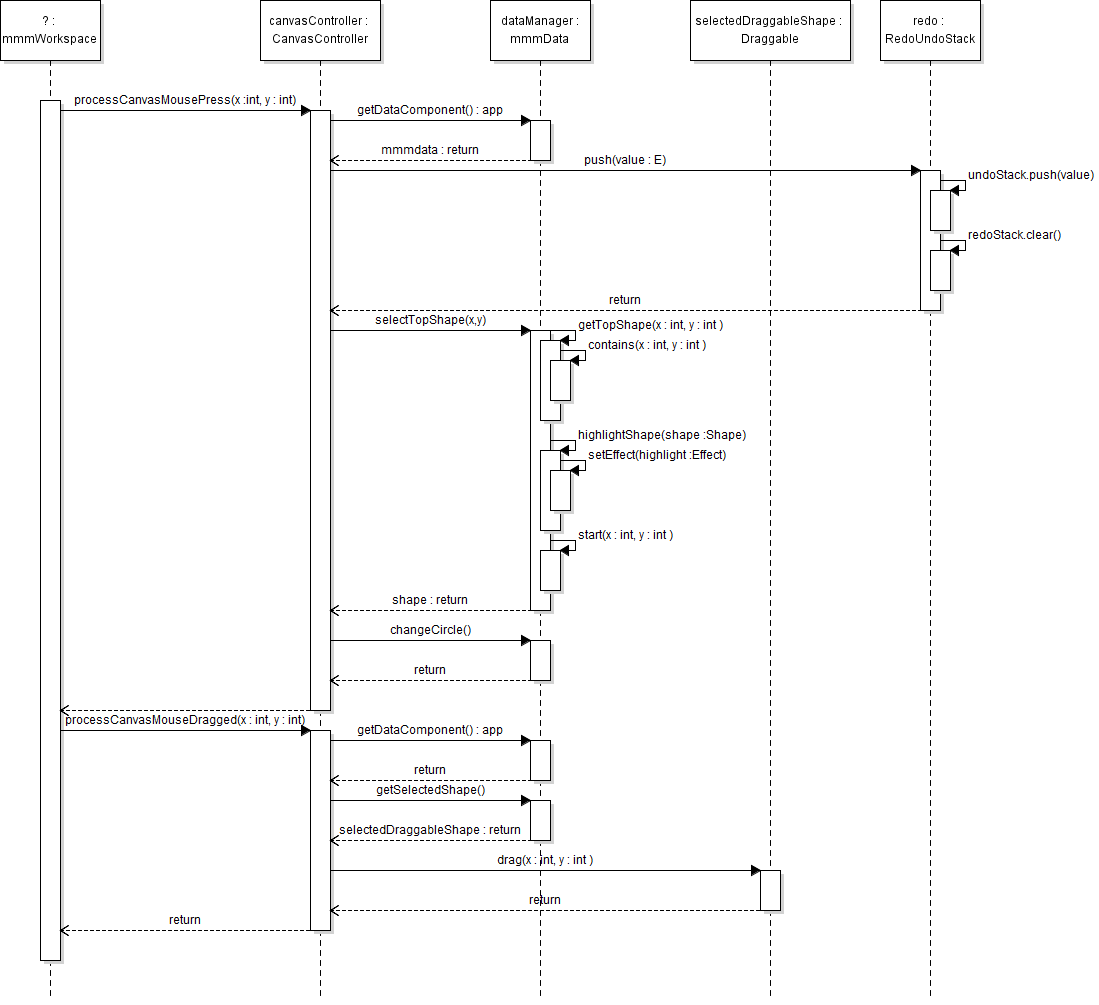
**Figure 4.8: UML Sequence Diagram of Use Case 11: Learn About Application**

****

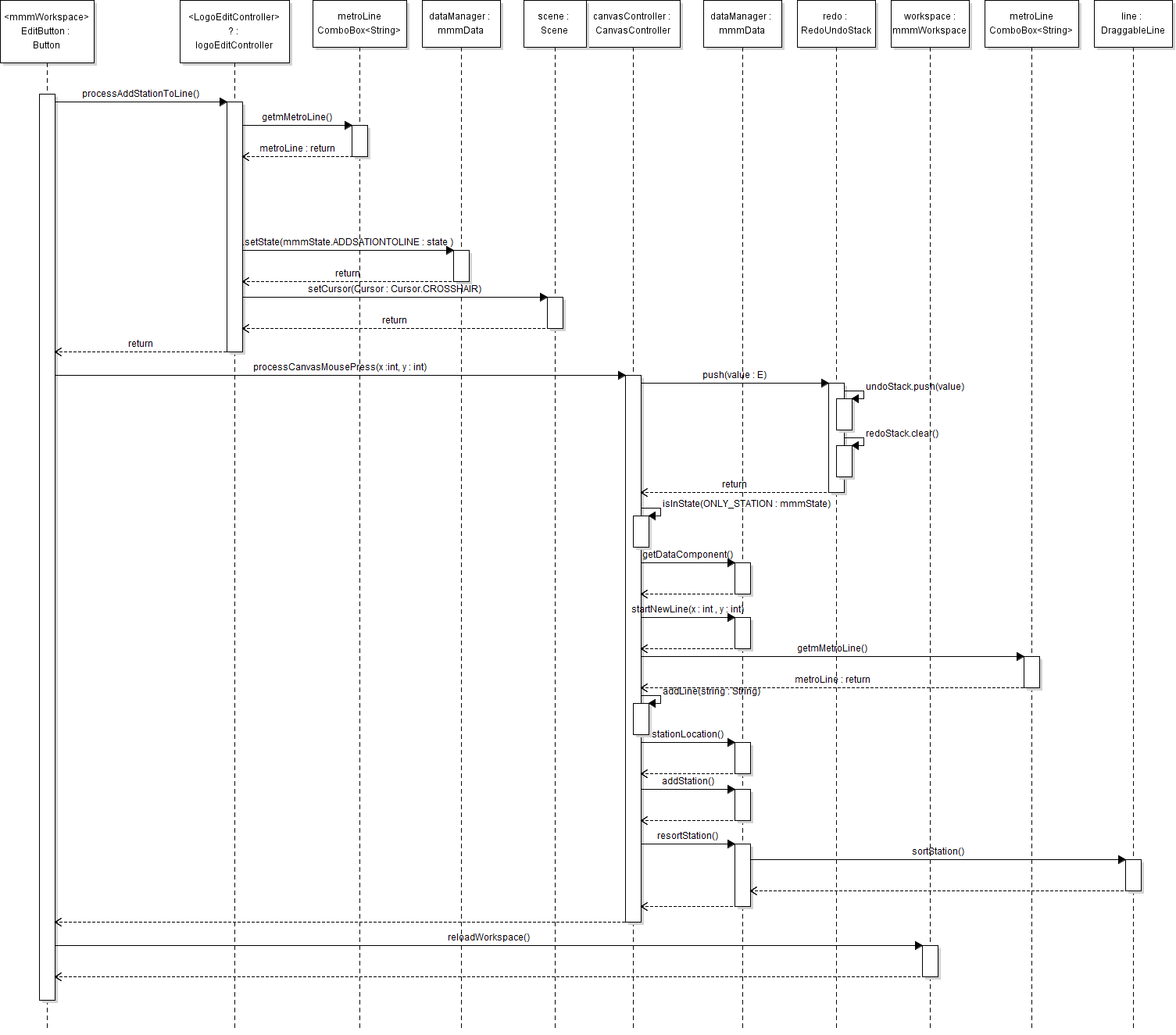
**Figure 4.9: UML Sequence Diagram of Use Case 13: Remove Line**

****

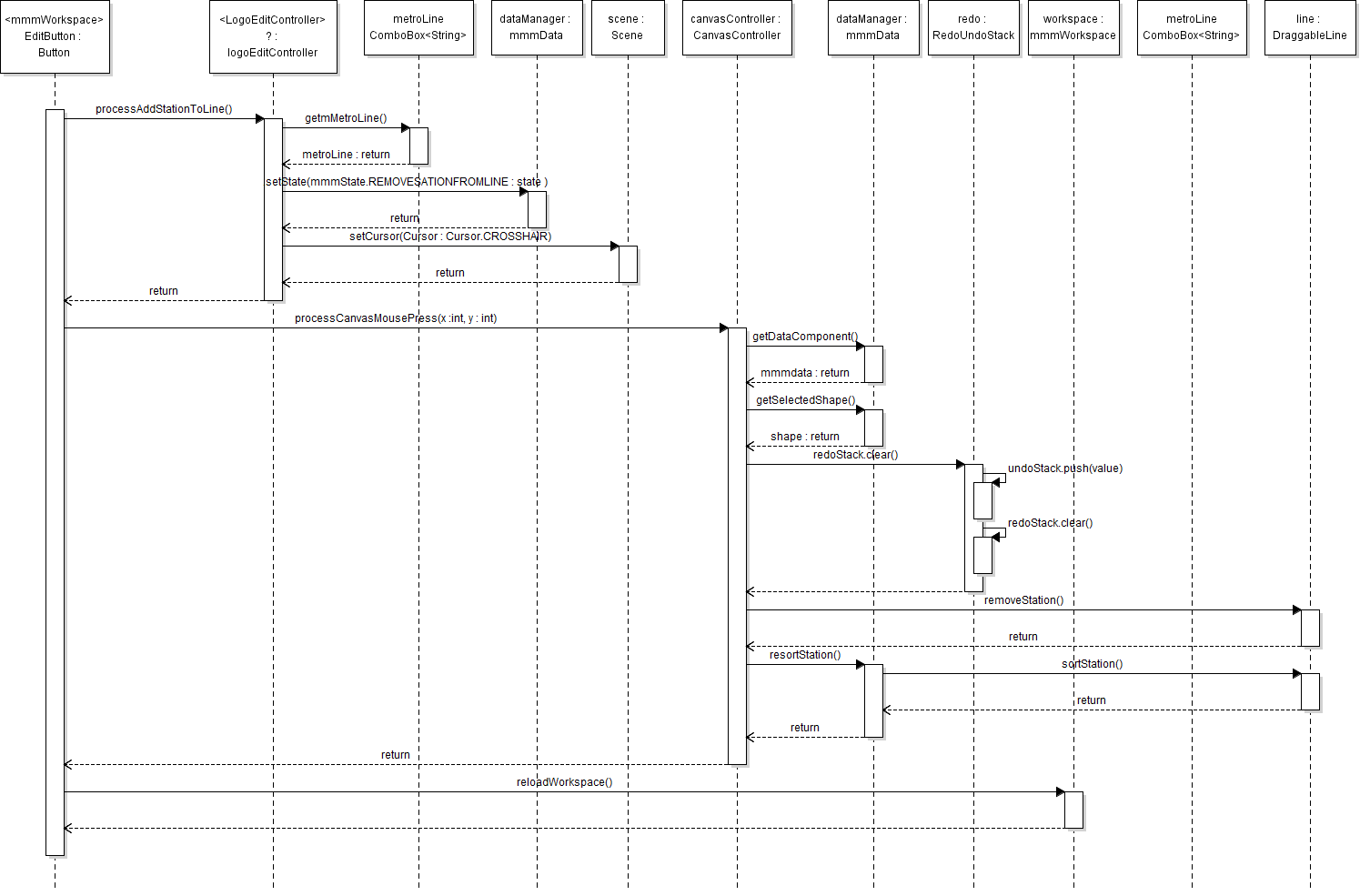
**Figure 4.10: UML Sequence Diagram of Use Case 14: Edit Line**

****

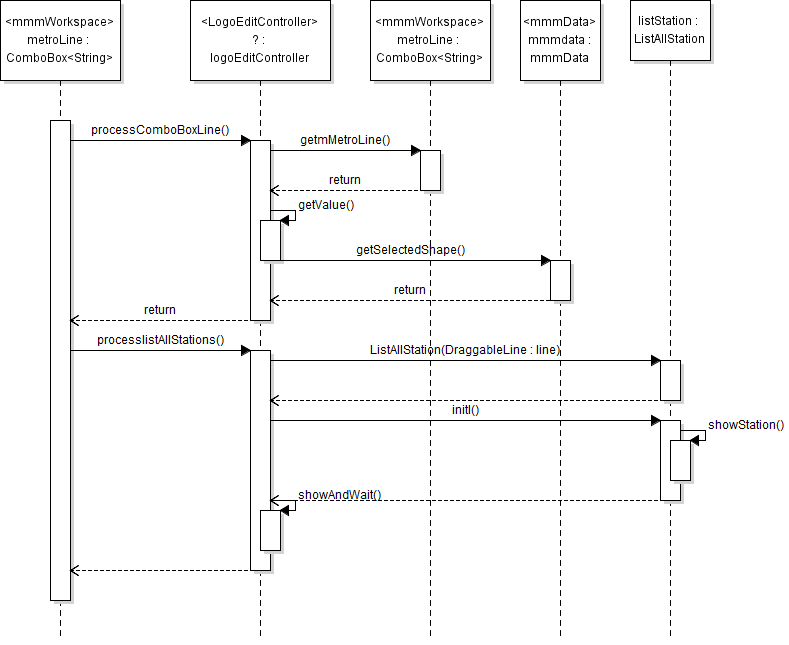
**Figure 4.11: UML Sequence Diagram of Use Case 15: Move Line End**

****

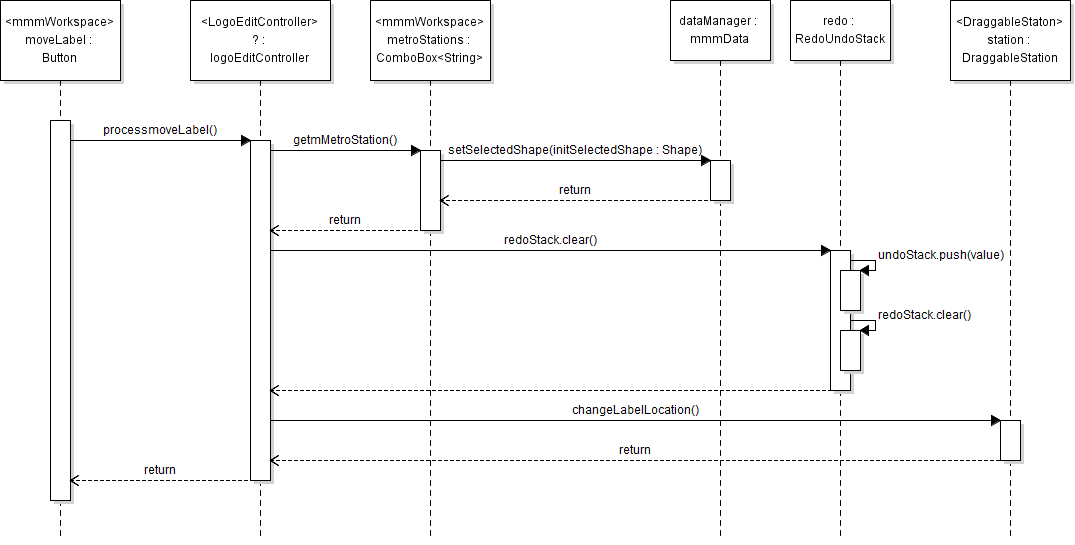
**Figure 4.12: UML Sequence Diagram of Use Case 16: Add Stations to Line**

****

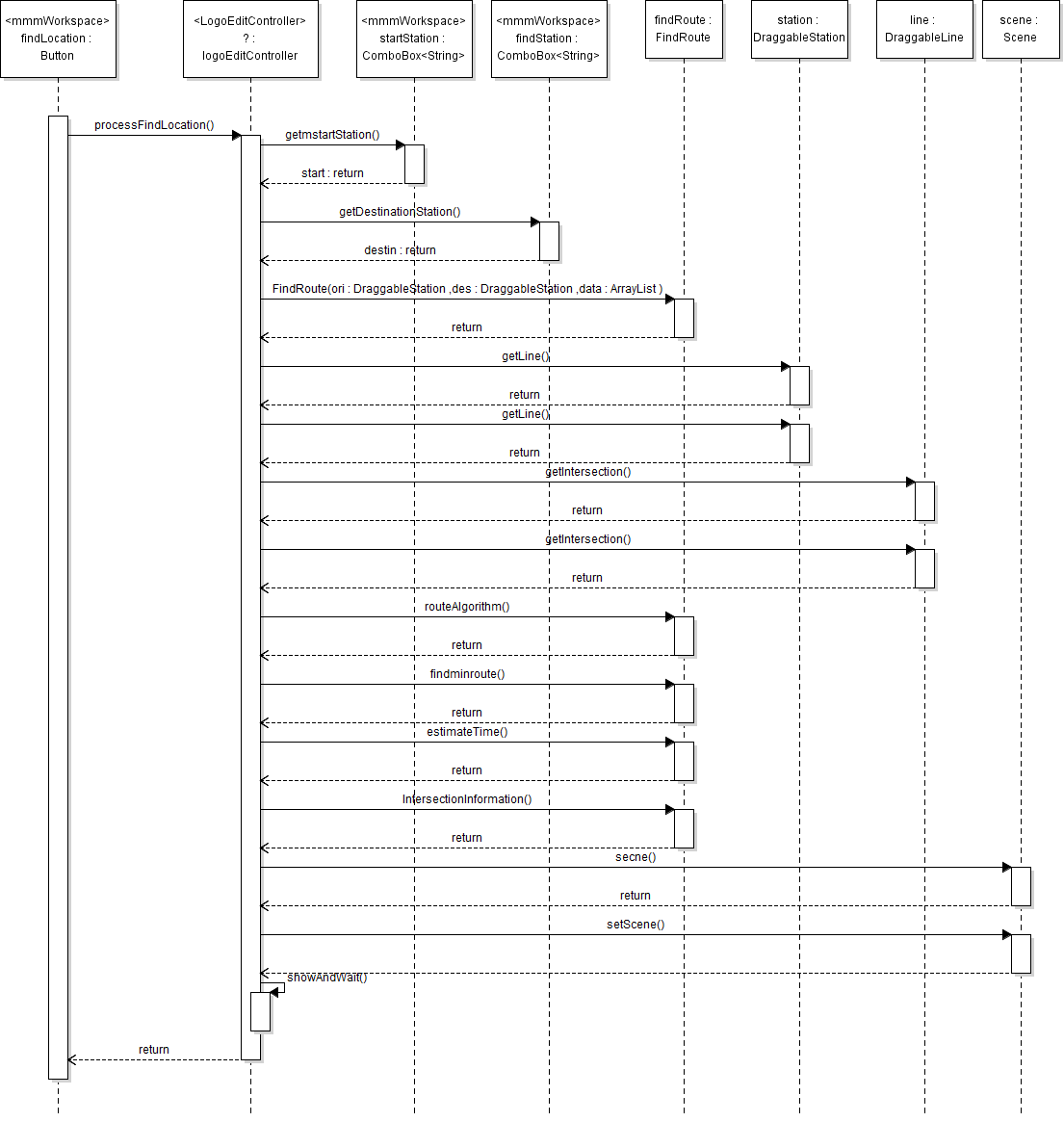
**Figure 4.13: UML Sequence Diagram of Use Case 17: Remove Stations From Line**

****

**Figure 4.14: UML Sequence Diagram of Use Case 18: List All Stations in Line**

****

**Figure 4.15: UML Sequence Diagram of Use Case 23: Move Station Label**

****

**Figure 4.16: UML Sequence Diagram of Use Case 27: Find Route**

**5. File Structure and Formats**

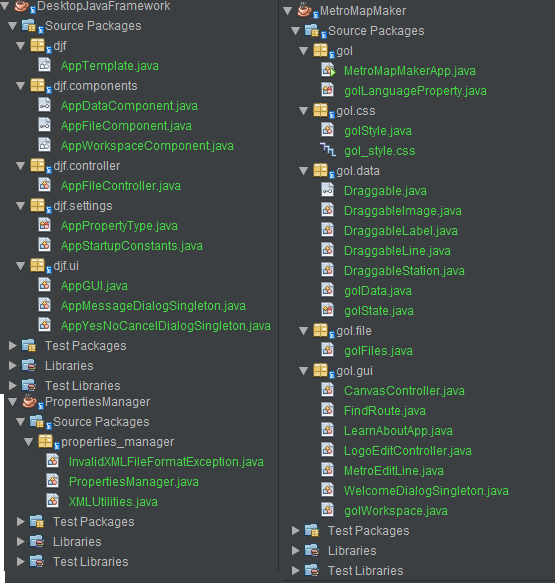
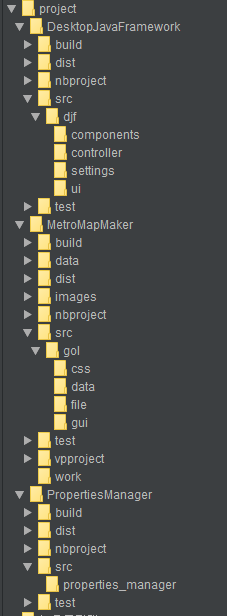


Figure 5.1 . File Structure and Formats

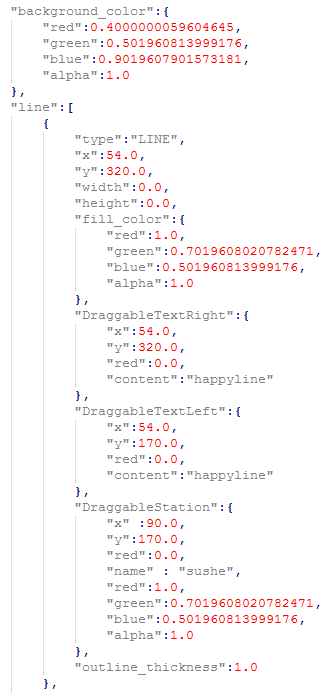


Figure 5.2 . ‘data.json’ File Contents



Figure 5.3 . Application Property File Contents and Formats



Figure 5.4 . Application Property Schema Content and File Formats

**6. Supporting Information**

Note that this document should serve as a reference for the designers and coders in the future stages of

the development process, so we’ll provide a table of contents to help quickly find important sections.

When this program read and write files, the function exportData(), importData() function in mmmfiles class is called , All information is saved as the from of JSON files. Those functions check every data type and save the information including the address of image and name of images. Also the text’s size, italics, Bold, font family also saved as Boolean. When export the data, there is a processSnapshot() function in mmmEditController. By using this method, the snapshot will be saved in default storage. When load an file, It also applied to uploaded file as the same way.

**6.1 Table of contents**

1. 1 Introduction 3
   1. Purpose 3
   2. Scope 3
   3. Definitions, acronyms, and abbreviations 4
   4. References 4
   5. Overview 4
2. Package-Level Design Viewpoint 5
   1. Metro Map Maker overview 5
   2. Java API Usage 6
   3. Java API Usage Descriptions 7
3. Class-Level Design Viewpoint 19
4. Method-Level Design Viewpoint 32
5. File Structure and Formats 48
6. Supporting Information 51

6.2 **Appendixes**

N/A