DataViliJ
Software Design Description

Author: Kevin Gray
110542955
March 2018

Abstract: This is a software design description for DataViliJ, a data visualization program being developed by CSE 219 students at Stony Brook University.

Table of Contents

1 Introduction

This is the Software Design Description (SDD) for DataViliJ. This document format is based on the IEEE Standard 1016-2009 recommendation for software design.

1.1 Purpose

This document is an outline for the design and implementation of the DataViliJ application. It will contain UML class diagrams to illustrate the system's architecture and class designs as well as UML sequence diagrams to demonstrate the interactions between class methods at runtime. This document is intended to satisfy Homework 3 requirements in CSE 219 Spring 2018 semester at Stony Brook University. The target audience is the grading T.A.s and the instructor (Professor Stark).

1.2 Scope

DataViliJ is a data visualization application designed to plot the results of two different machine learning algorithms. These are called clustering and classification, and they have already been developed for this product. In addition to the algorithms themselves, PropertyManager, Components, Templates, and Settings have all been previously constructed. This design will include descriptions only for the application, not these previously listed functionalities, unless changes have been made to them. This design assumes Java is the language being used for development.

1.3 Definitions, Acronyms, and Abbreviations

1) **Algorithm -** In this document, the term 'algorithm' will be used to denote an AI algorithm that can "learn" from some data and assign each data point a label

2) **Class Diagram -** a UML document that shows classes' relationships with one another in addition to their instance variables and methods
3) **Classification -** A type of AI algorithm that learns to assign new labels to instances based on how older instances were labeled. These algorithms calculate geometric objects that divide the x-y plane into parts. E.g., if the geometric object is a circle, the two parts are the inside and the outside of that circle; if the geometric object is a straight-line, then again, there two parts, one on each side of the line
4) **Clustering -** A type of AI algorithm that learns to assign labels to instances based purely on the spatial distribution of the data points
5) **Graphical User Interface (GUI) -** An interface that allows users to interact with the application through visual indicators and controls. A GUI has a less intense learning curve for the user, compared to text-based command line interfaces. Typical controls and indicators include buttons, menus, check boxes, dialogs, etc.
6) **IEEE -** Institute of Electrical and Electronics Engineers, is a professional association founded in 1963. Its objectives are the educational and technical advancement of electrical and electronic engineering, telecommunications, computer engineering and allied disciplines
7) **Java -** a class based, object oriented programming language
8) **Sequence Diagram -** a UML document that shows the dynamics of class interactions in a specific scenario
9) **Software Design Description (SDD) -** A written description of a software product, that a software designer writes in order to give a software development team overall guidance to the architecture of the project

10) **Unified Modeling Language -** A general-purpose, developmental modeling language to provide a standard way to visualize the design of a system

11) **User -** Someone who interacts with the DataViLiJ application via its GUI

1.4 References

1) **DataViliJ SRS -** Professor Banerjee's Software Requirements Specification for the DataViliJ application

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

1.5 Overview

This Software Design Description (SDD) will outline a means of implementation for the DataViliJ application, as described in the respective Software Requirements Specification (SRS). All parties in the implementation stage must agree on this description before moving forward. The following section (Section 2) is the Package-level Design Viewpoint, which will describe the packages being designed as well as the Java API being used. Section 3 will contain the Class-level Design Viewpoint, which will use UML Class Diagrams to outline class variables and functions. Section 4 will detail the Method-level Design Viewpoint through the use of UML Sequence Diagrams. Section 5 will show a sample of what the project files should look like. Finally, Section 6 will provide supporting information necessary to understand this implementation. UML Diagrams will be made using VioletUML.

2 Package-level Design Viewpoint

This section will list the packages to be developed for DataViliJ as well as the Java API being used. Definitions and intended uses of Java API will also be provided.

2.1 Software Overview

DataViliJ will contain few, if any, modifications to the following packages: gui.css, gui.icons, properties, xmlutil, vilij.components, vilij.propertymanager, vilij.settings, and vilij.templates. Figure 2.1 provides all packages being developed and utilized outside of the standard Java API.

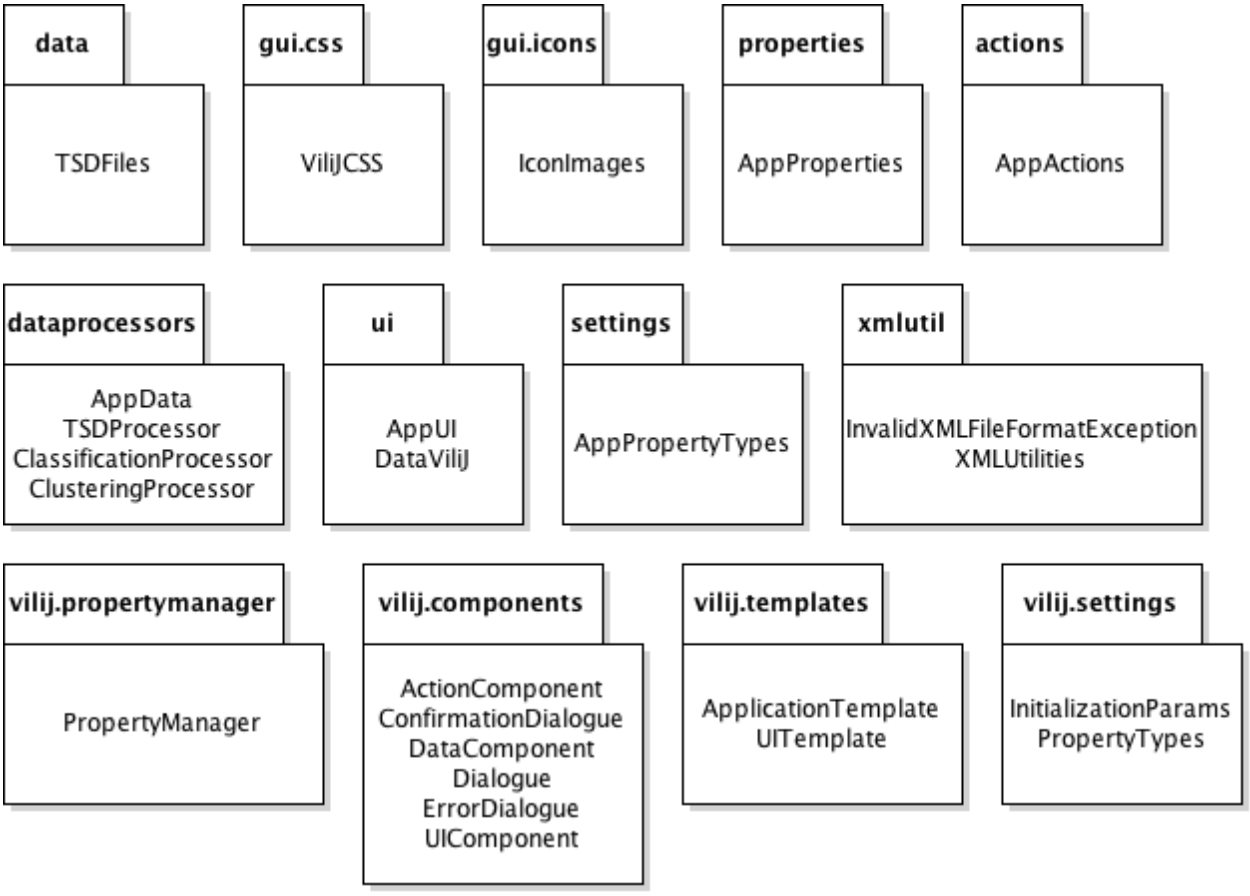


Figure 2.1 DataViliJ Package Overview

2.2 Java API Usage

Figure 2.2 specifies all Java API to be used in this project.

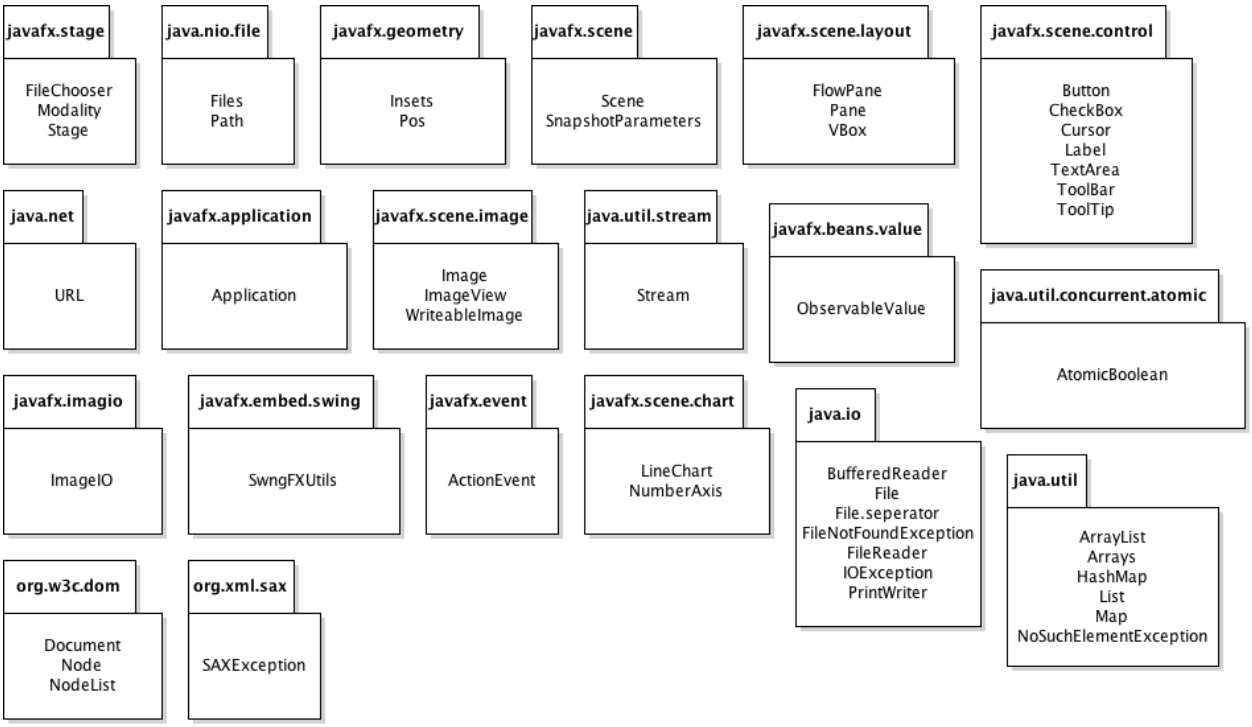


Figure 2.2 Java API Packages and Classes to be Used

2.3 Java API Usage Descriptions

Tables 2.3.1-2.3.20 describe how each class will be used.

Class/Interface	Usage
FileChooser	For allowing the user to specify a file to be saved/ loaded
Modality	For preventing messages from showing in other dialogue windows
Stage	The top level window that will hold the DataViliJ application

Table 2.3.1: Usage of javafx.stage

Class/Interface	Usage
Files	For checking if a specified file exists
Path	For providing a path to the active data file

Table 2.3.2: Usage of java.nio.file

Class/Interface	Usage
Insets	For creating an offset for a rectangular object (like a dialogue box)
Pos	For describing the position of an object (in this case a dialogue box)

Table 2.3.3: Usage of javafx.geometry

Class/Interface	Usage
Scene	For providing a container for the content that will be shown to the user
SnapshotParameters	For providing default values for rendering an image

Table 2.3.4: Usage of javafx.scene

Class/Interface	Usage
FlowPane	For holding UI components, which are organized left->right, top->bottom
Pane	For holding UI components
VBox	For holding/organizing objects vertically

Table 2.3.5: Usage of javafx.scene.layout

Class/Interface	Usage
Button	For receiving input from the user in the form of a clickable button
Checkbox	For receiving input from the user in the form of a checkable box
Cursor	For setting changes in the user's cursor upon hovering over a data point
Label	For providing text descriptions of various components in the UI
TextArea	For the displaying/altering of user provided data

ToolBar	For storing several buttons at the top of the UI
ToolTip	For providing information about a component when it is hovered over

Table 2.3.6: Usage of javafx.scene.control

Class/Interface	Usage
URL	For getting the location of a file on the user's device

Table 2.3.7: Usage of java.net

Class/Interface	Usage
Application	For inheriting basic javafx application functionality

Table 2.3.8: Usage of javafx.application

Class/Interface	Usage
Image	For loading images for the buttons
ImageView	For showing images for the buttons
WriteableImage	For taking a picture of a BufferedImage and storing its pixels as a JavaFX object

Table 2.3.9: Usage of javafx.scene.image

Class/Interface	Usage
Stream	For breaking down a string containing user provided data into a more processable format

Table 2.3.10: Usage of java.util.stream

Class/Interface	Usage
ObservableValue	For determining if a change has taken place upon the creation of an ActionEvent

Table 2.3.11: Usage of javafx.beans.value

Class/Interface	Usage
AtomicBoolean	For setting a boolean value without interrupting a thread

Table 2.3.12: Usage of java.util.concurrent.atomic

Class/Interface	Usage
ImageIO	For writing a screenshot to a file

Table 2.3.13: Usage of javafx.imageio

Class/Interface	Usage
SwingFXUtils	For taking a picture of a JavaFX object and storing its pixels as a buffered image

Table 2.3.14: Usage of javafx.embed.swing

Class/Interface	Usage
ActionEvent	Used for determining when an action takes place and for dispatching an event handler

Table 2.3.15: Usage of javafx.event

Class/Interface	Usage
LineChart	For providing a basic chart for the display of data generated from both algorithms
NumberAxis	For providing the chart with a relevant/accurate X and Y axis

Table 2.3.16: Usage of javafx.scene.chart

Class/Interface	Usage
BufferedReader	For reading from a user provided file line by line
File	For providing a path for a file to be saved or loaded

File.separator	For providing a system dependent separator character for file paths
FileNotFoundException	For determining if a specified file cannot be found and throwing an error
FileReader	For reading from a file
IOException	For determining if there was an error reading from or writing to a file
PrintWriter	For writing to a file

Table 2.3.17: Usage of java.io

Class/Interface	Usage
ArrayList	For storing objects
Arrays	For storing objects
HashMap	Data structure that stores data using a key (accessor) and a value (data)
List	For storing objects
Map	For mapping keys to values
NoSuchElementException	For determining if a specified property in propertymanager does not exist

Table 2.3.18: Usage of java.util

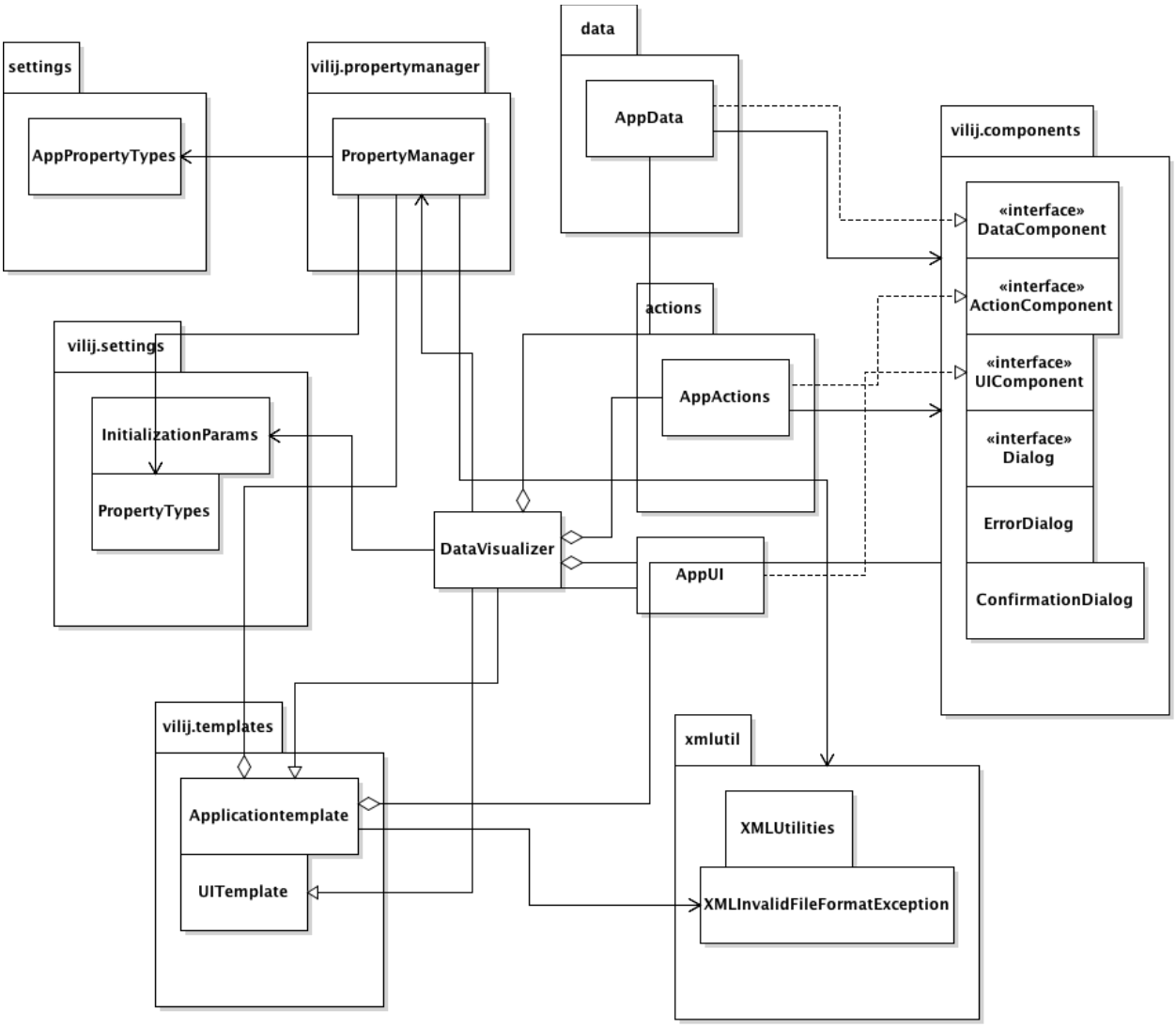
Class/Interface	Usage
Document	An object for XML Files to be loaded into and interpreted
Node	To break down XML tags into a usable object
NodeList	For organizing XML parent nodes

Table 2.3.19: Usage of org.w3c.dom

Class/Interface	Usage
SAXException	For handling XML formatting related exceptions

Table 2.3.20: Usage of org.xml.sax

3 Class-level Design Viewpoint



The following images are UML Class Diagrams describing the DataViliJ application.

Figure 3.1: DataViliJ Overview

This is a basic depiction of class interactions in the DataViliJ application. The following images will provide more detailed versions of the classes.

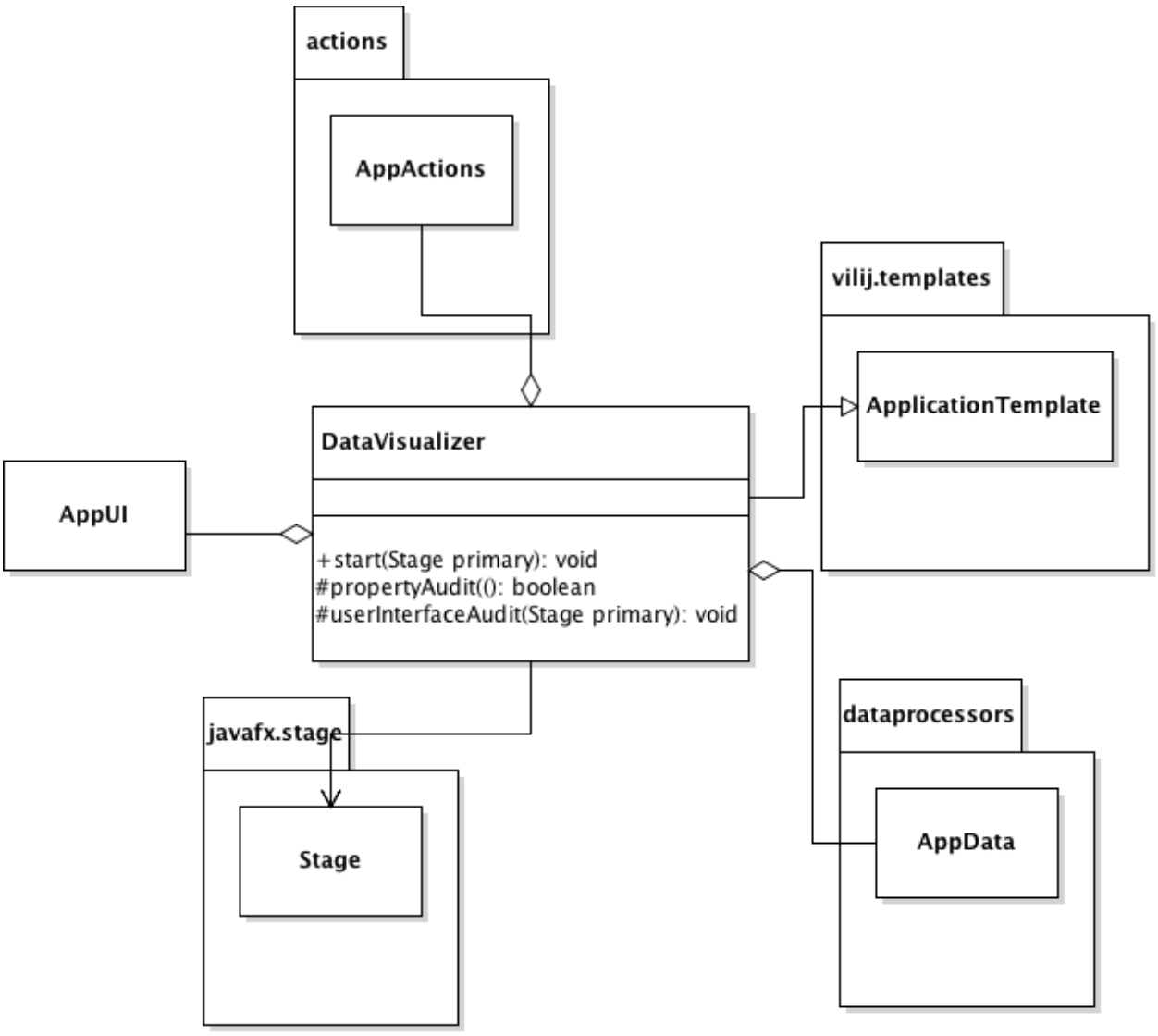


Figure 3.2: DataVisualizer Class Design

DataVisualizer is the main class from which DataViliJ is run.

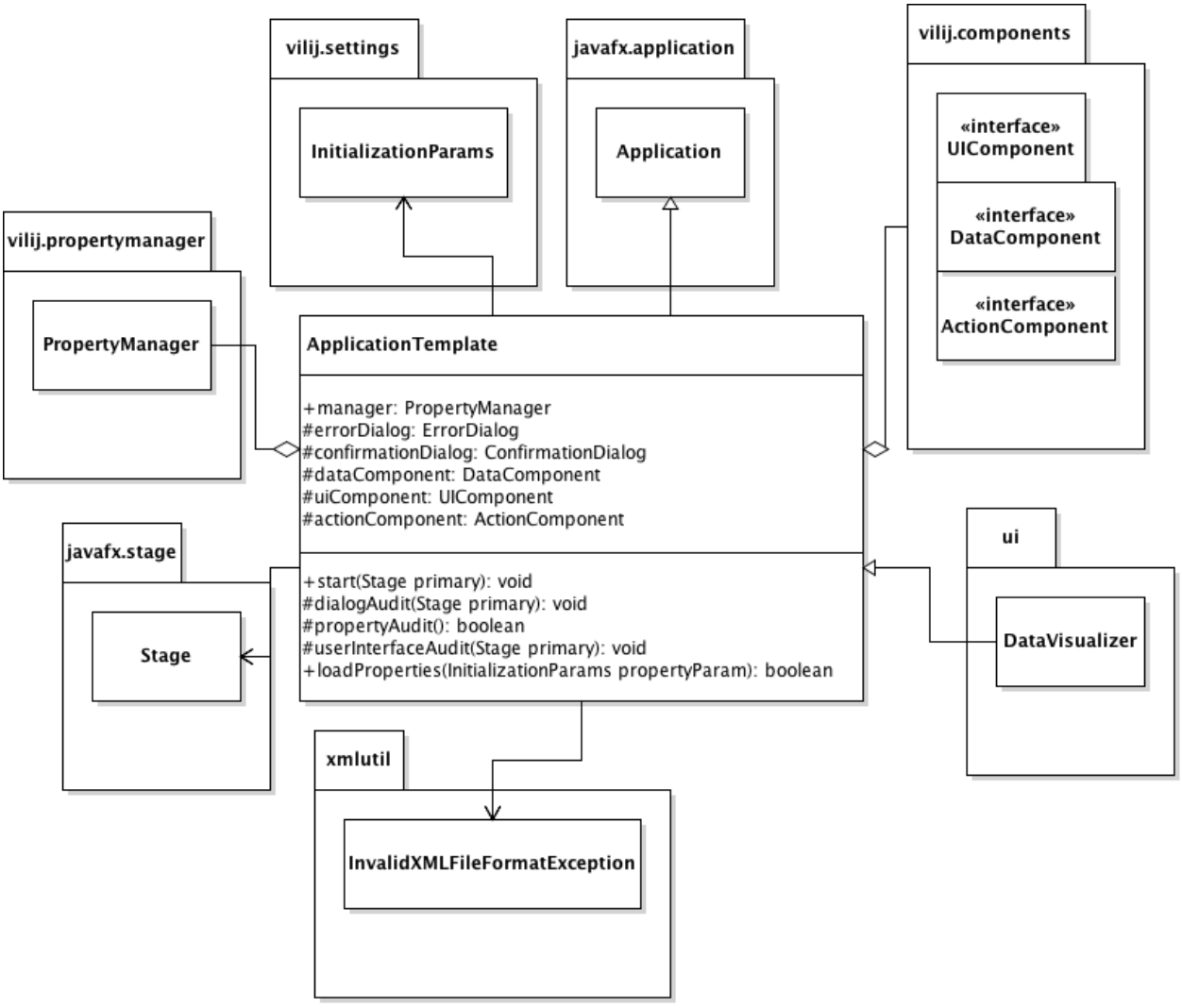


Figure 3.3: ApplicationTemplate Class Design

This class describes the structure of the DataViliJ application. DataVisualizer extends ApplicationTemplate.

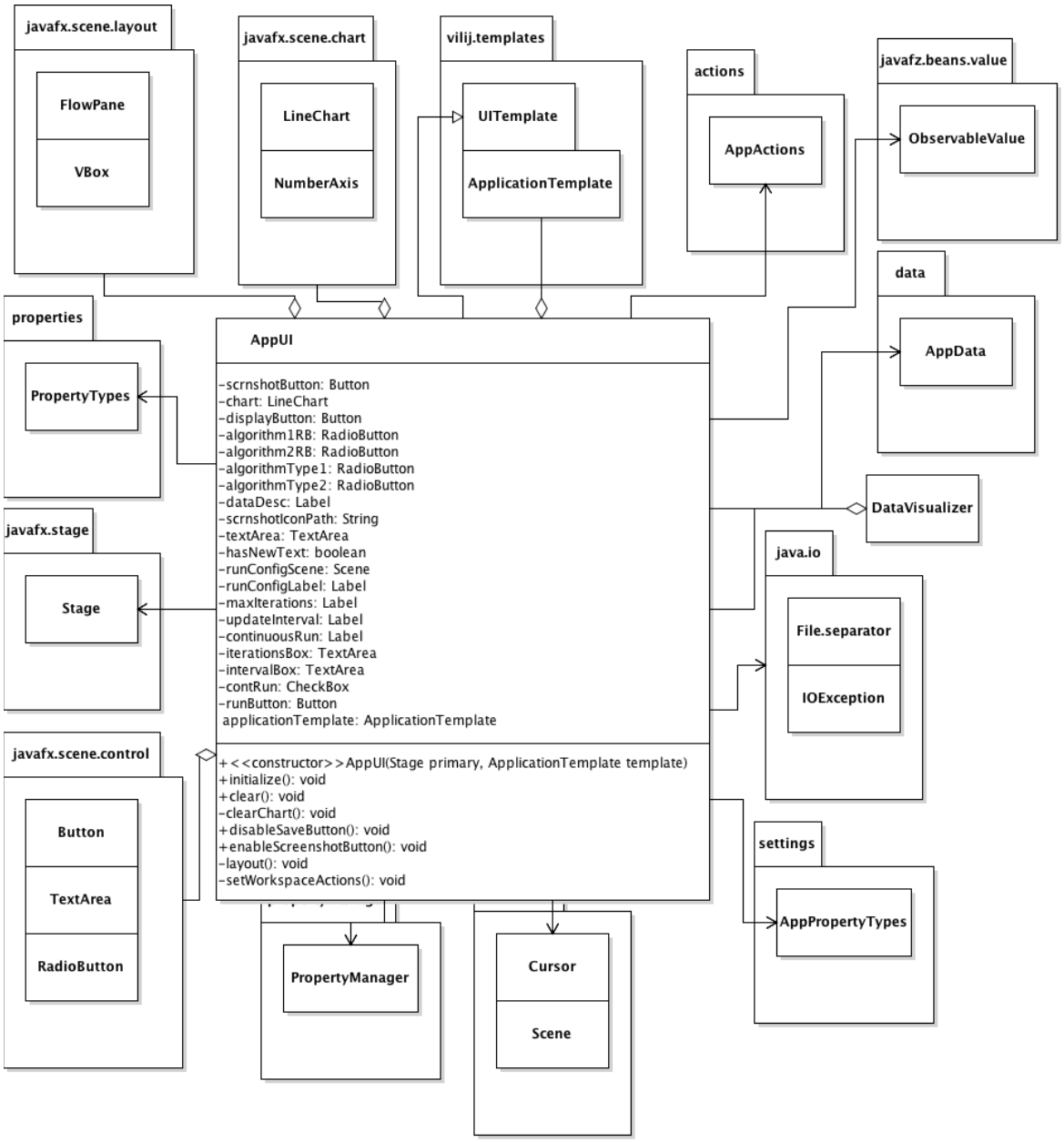


Figure 3.4: AppUI Class Design

AppUI is the source of this application's GUI.

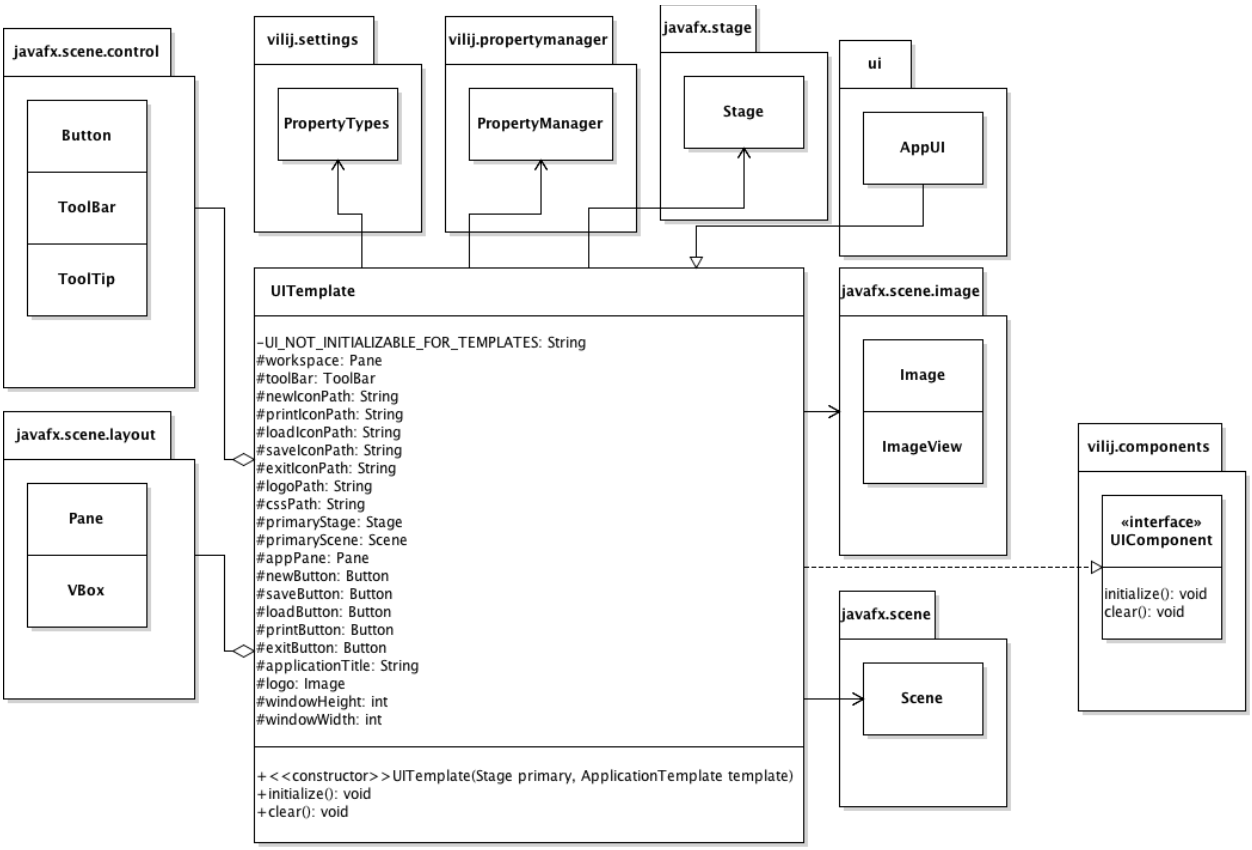


Figure 3.5: UITemplate Class Design

UITemplate provides a basic UI for generic data visualization applications. AppUI inherits from this class.

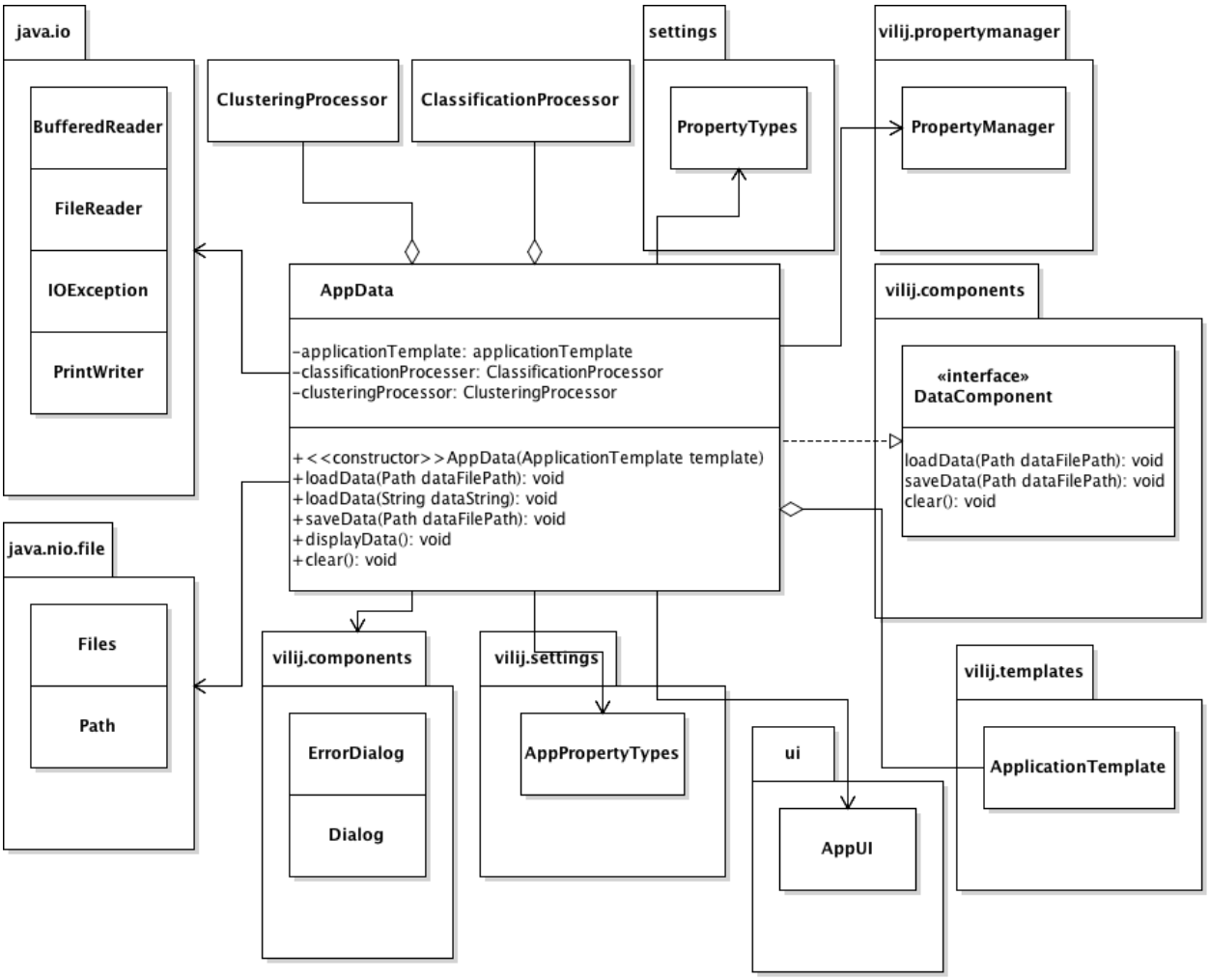


Figure 3.6: AppData Class Design

This figure depicts detailed versions of both the AppData class and the DataComponent interface.

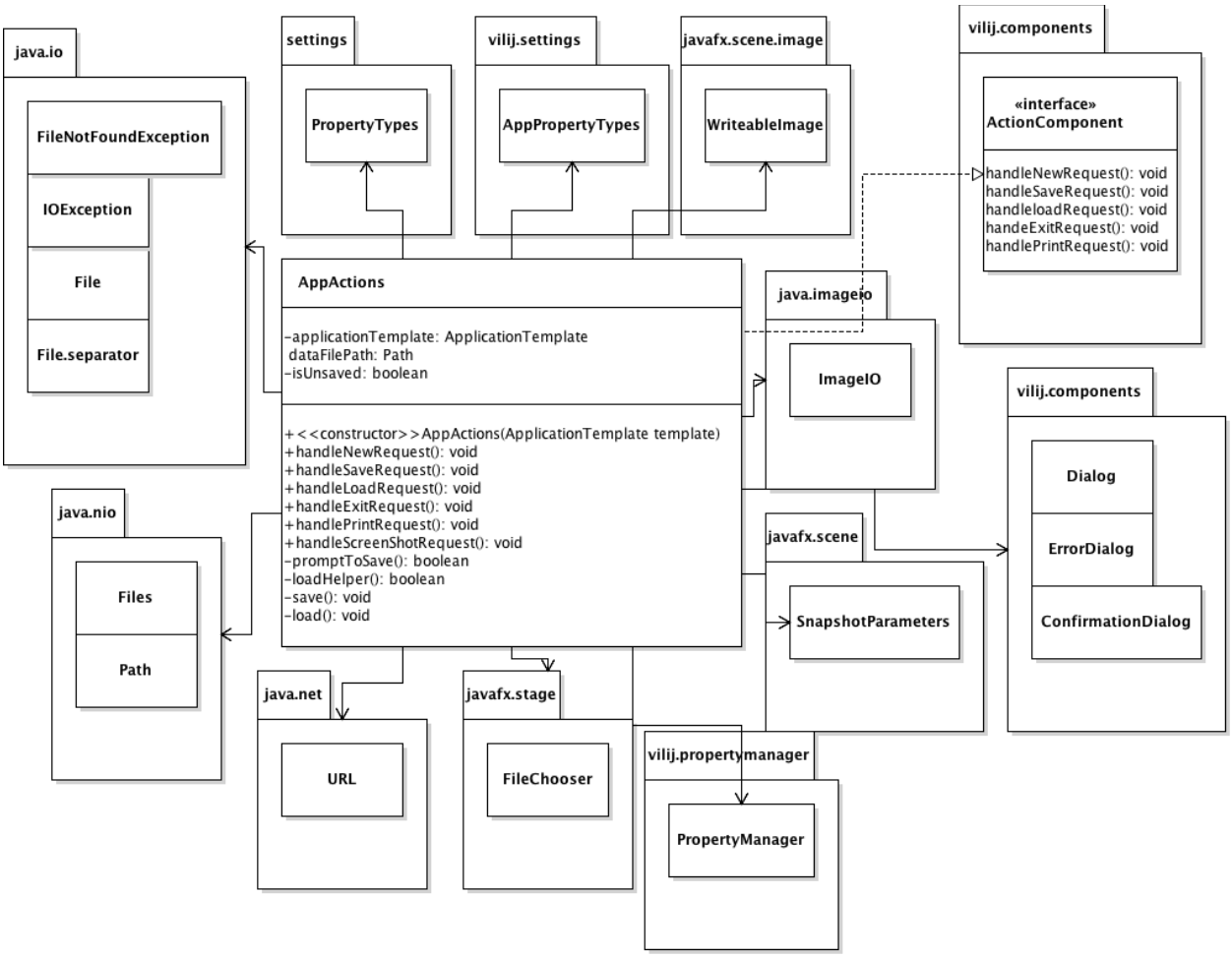


Figure 3.7: AppActions Class Diagram

This figure describes both the AppActions class and the ActionComponent interface.

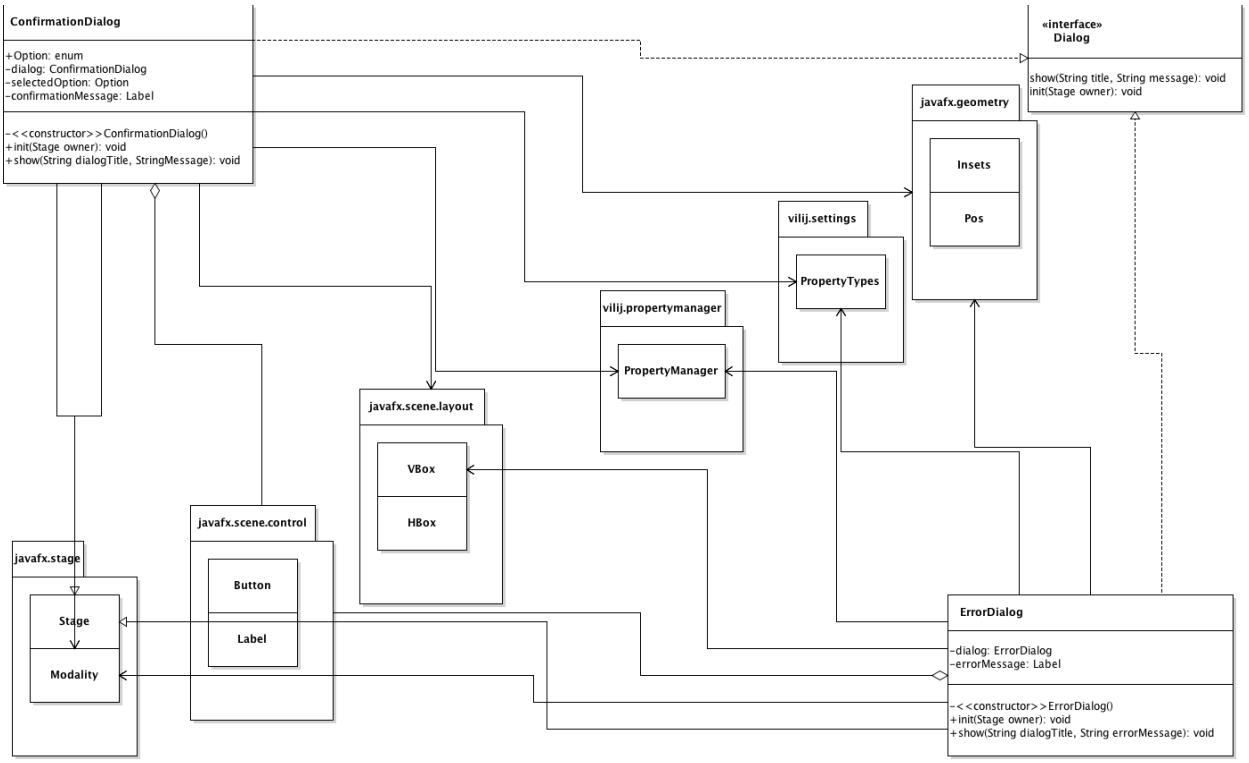


Figure 3.8: Dialog Class Design

This diagram shows the implementation of classes ConfirmationDialog and ErrorDialog, along with the Dialog interface.

4 Method-level Viewpoint

This section will contain UML Sequence Diagrams to describe class interactions at runtime.

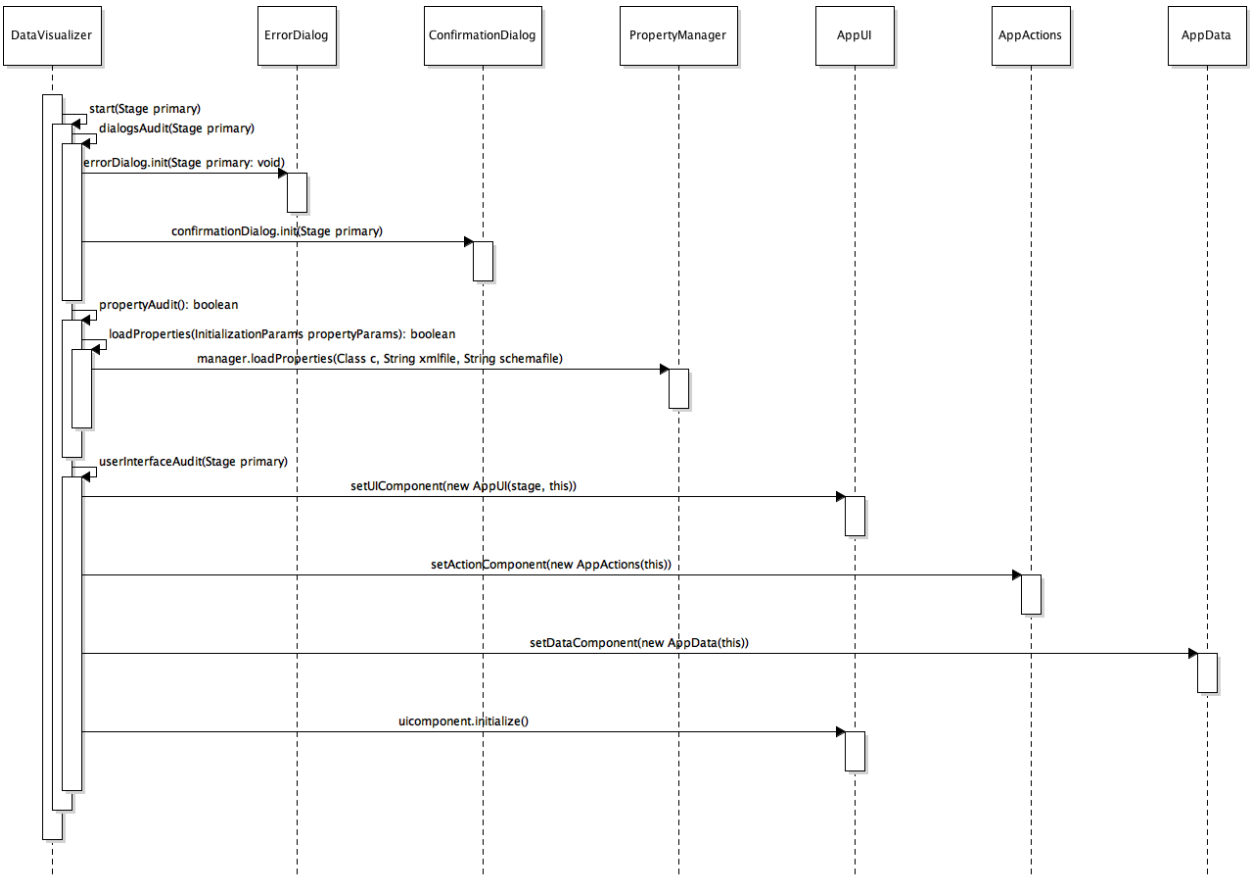


Figure 4.1: Start Application Sequence Diagram

This happens when the application is started for the first time.

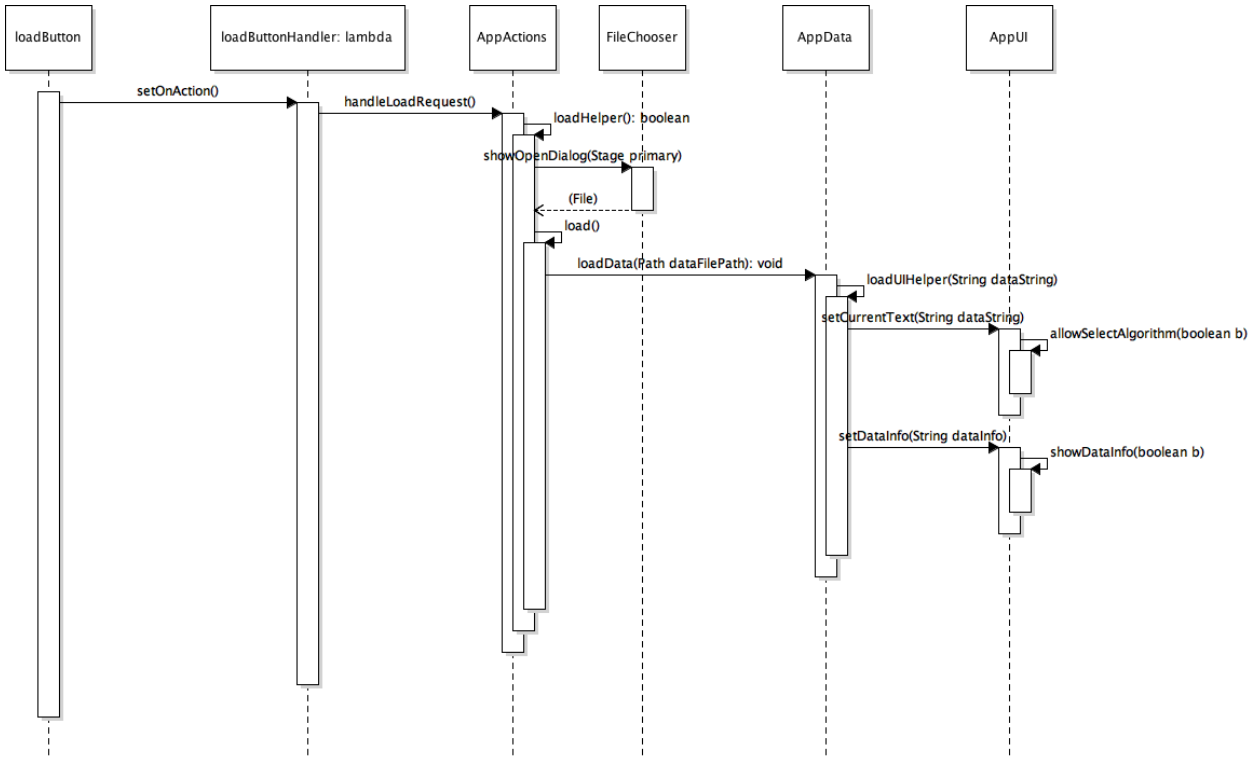


Figure 4.2: Load Data Sequence Diagram

After clicking the load button, the program will prompt the user to select a file to open, then open it (if it is in .tsd format).

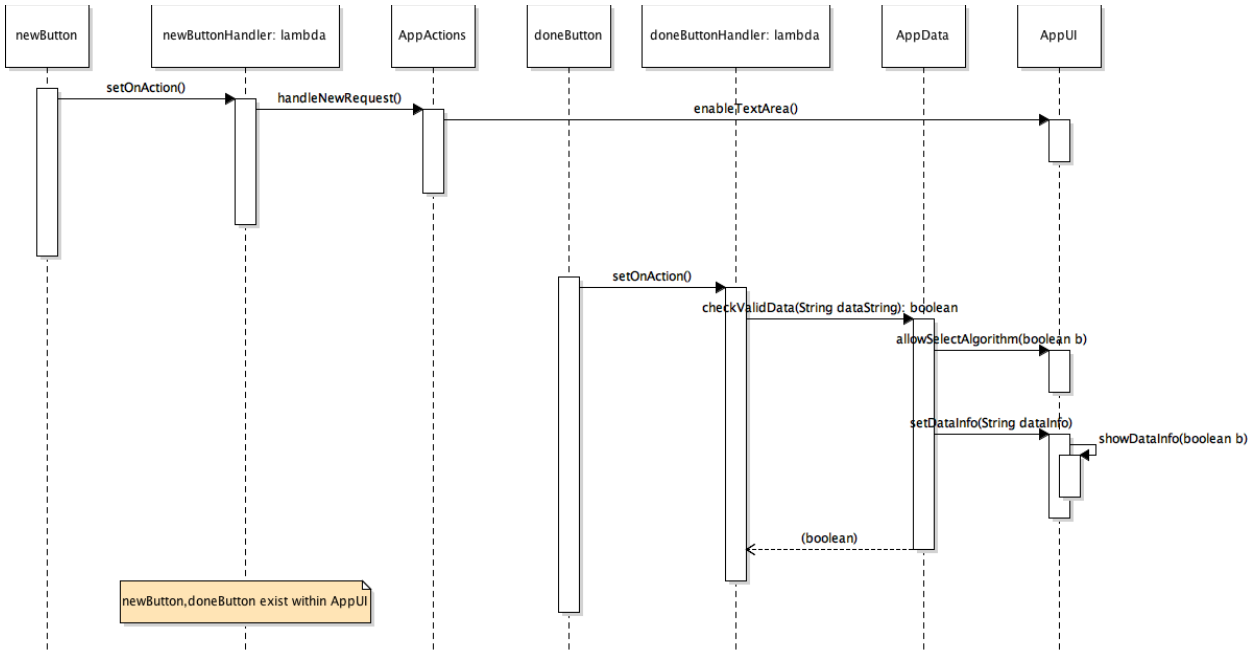


Figure 4.3: New Data Sequence Diagram

This occurs when the user clicks the new button. A text area will pop up, allowing the user to enter their own data. When they click the done button, the data will be checked for validity and its info will be shown.

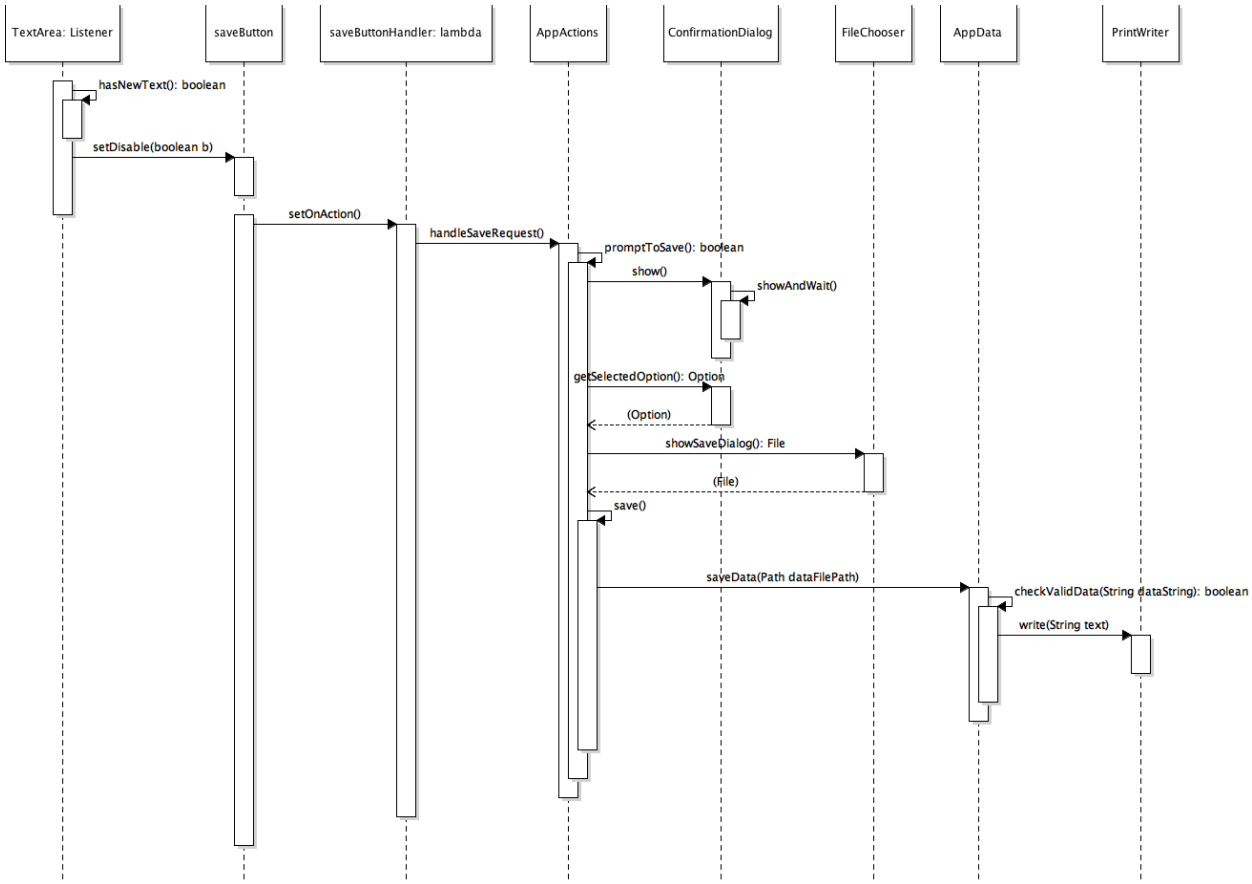


Figure 4.4: Save Data Sequence Diagram

When the user clicks the save button (and the text area has data to be saved), the user will be prompted to save their data in a tsd format. Their data will also be checked for validity before saving.

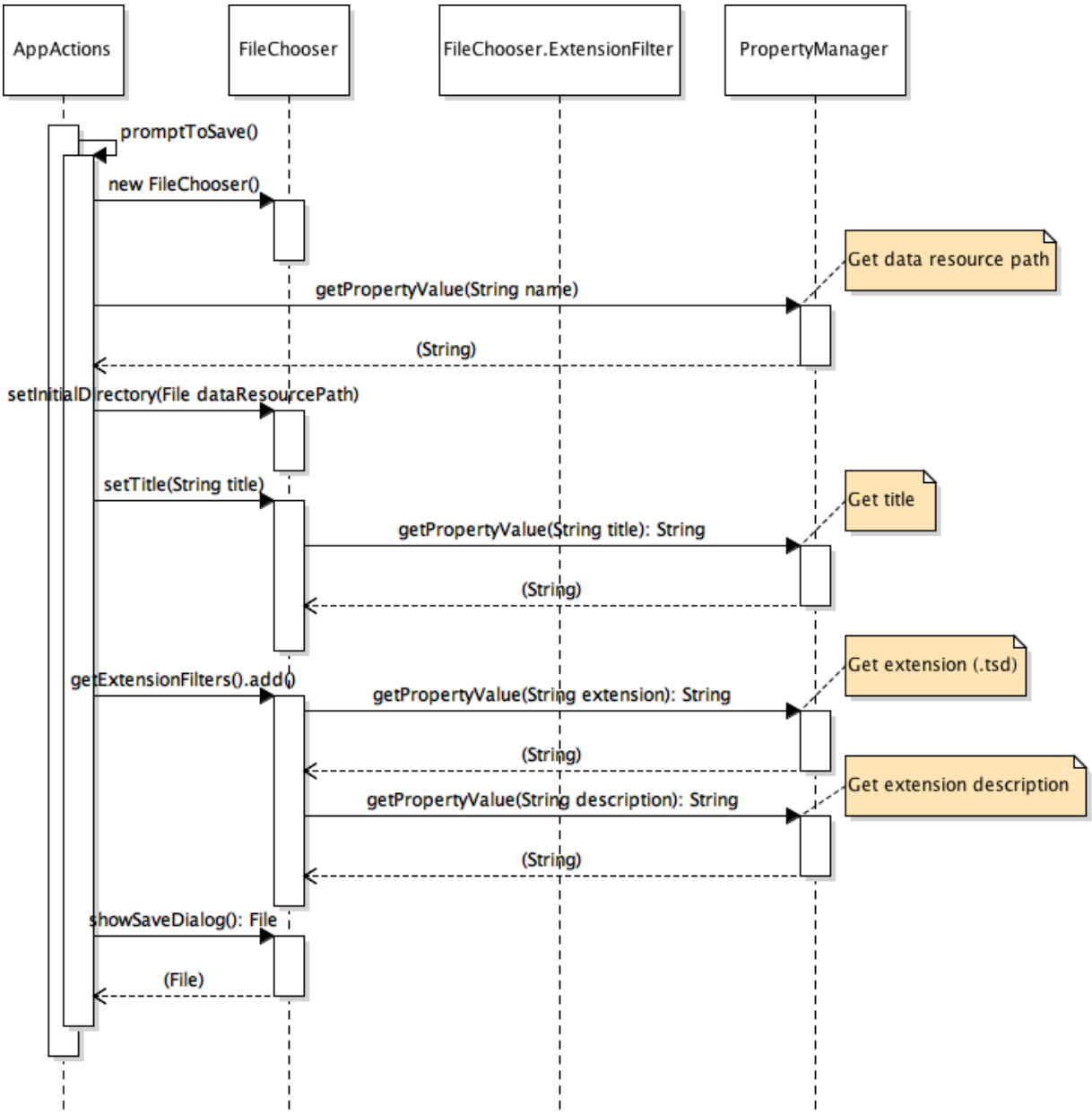


Figure 4.5: FileChooser Sequence Diagram

Figure 4.5 shows what happens when the filechooser is initialized. In this case, the filechooser will prompt the user to save their data.

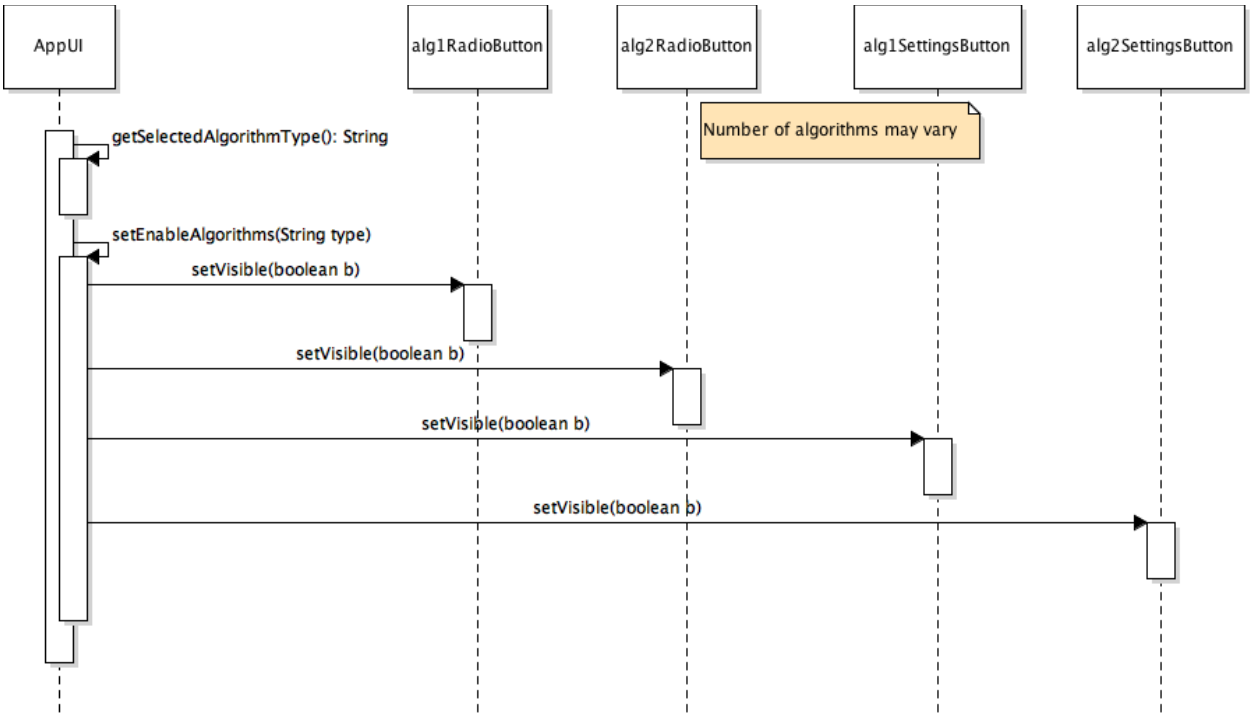


Figure 4.6: Select Algorithm Type Sequence Diagram

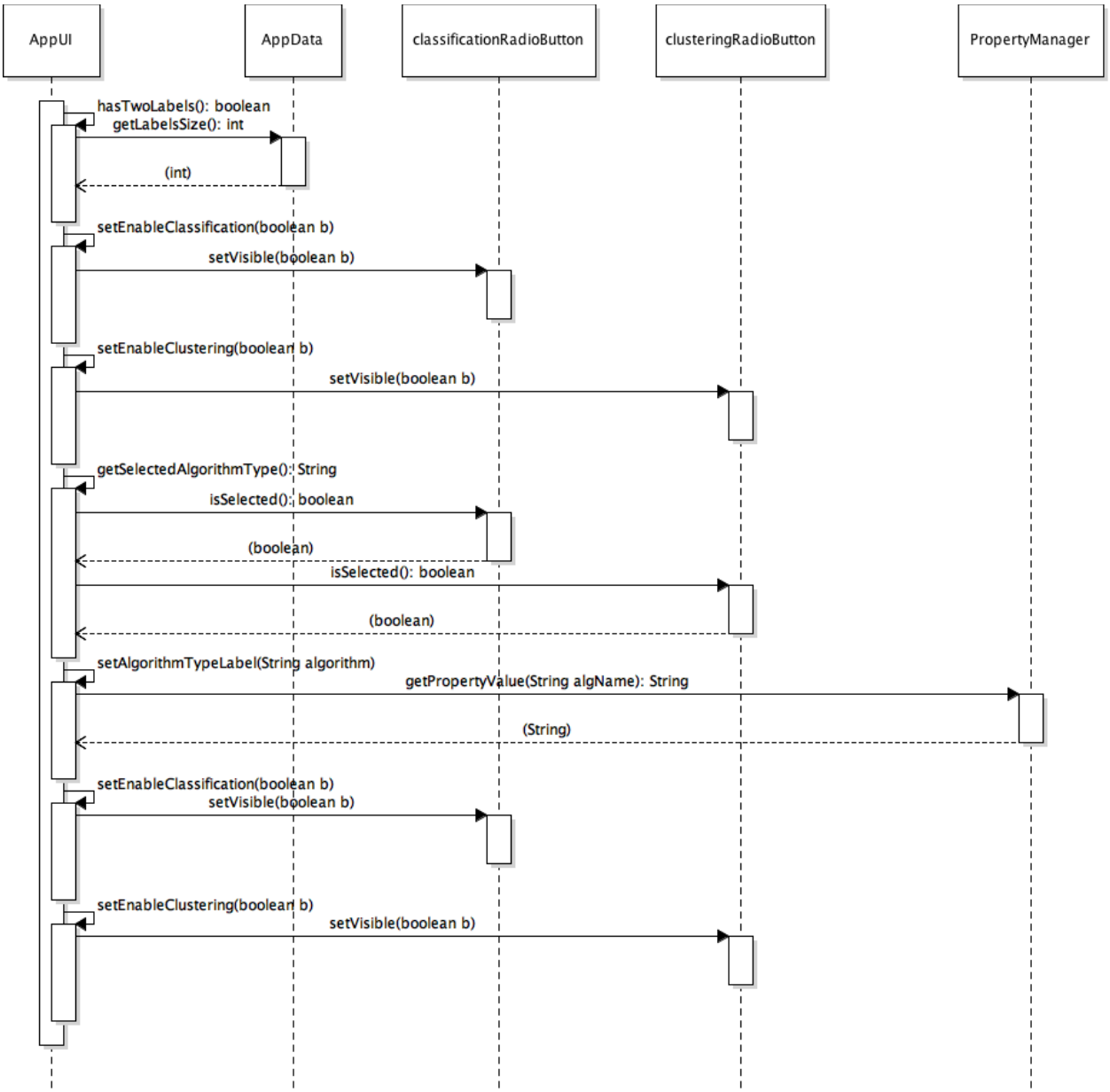


Figure 4.7: Select Algorithm Sequence Diagram

After the user has selected an algorithm type, they will be able to select from a series of algorithms by selecting the corresponding radio button.

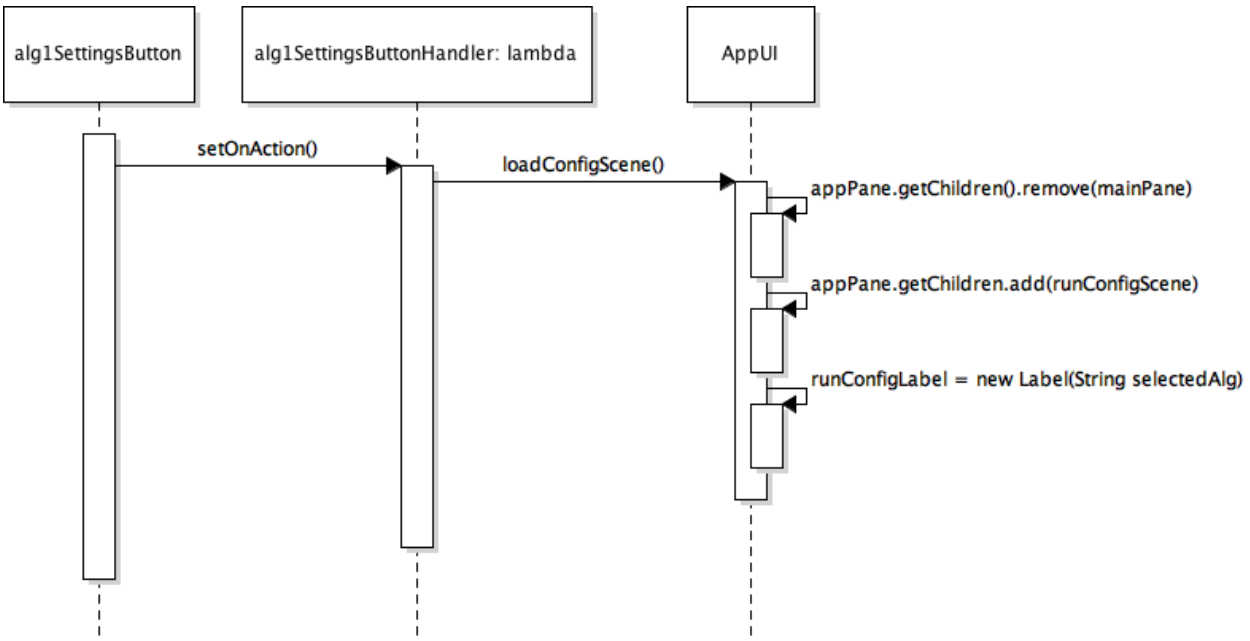


Figure 4.8: Select Algorithm Running Configuration

When the user clicks a settings button for a corresponding algorithm, the mainPane is replaced with the runConfigScene to allow the user to make changes to the algorithm settings.

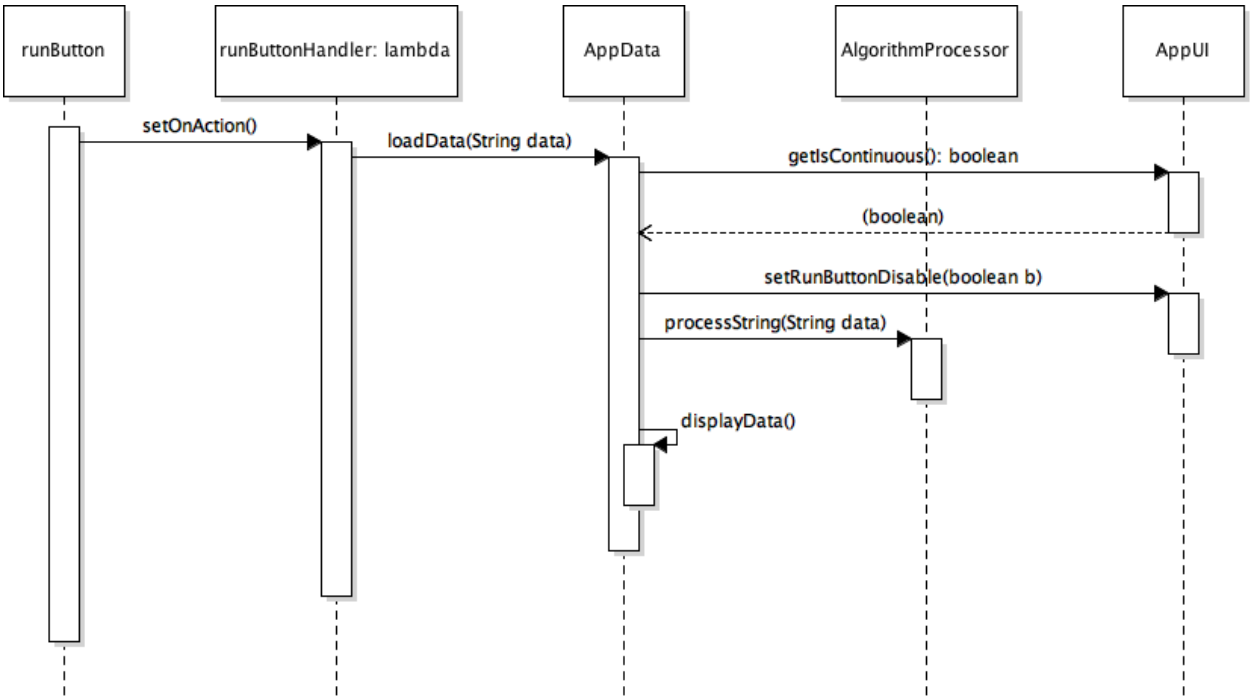


Figure 4.9: Run Algorithm Sequence Diagram

Once the user has configured the algorithm's settings, they are able to click the run button. The app will load one data point at a time. If the continuous run checkbox was not selected, the user will have to hit run after each iteration.

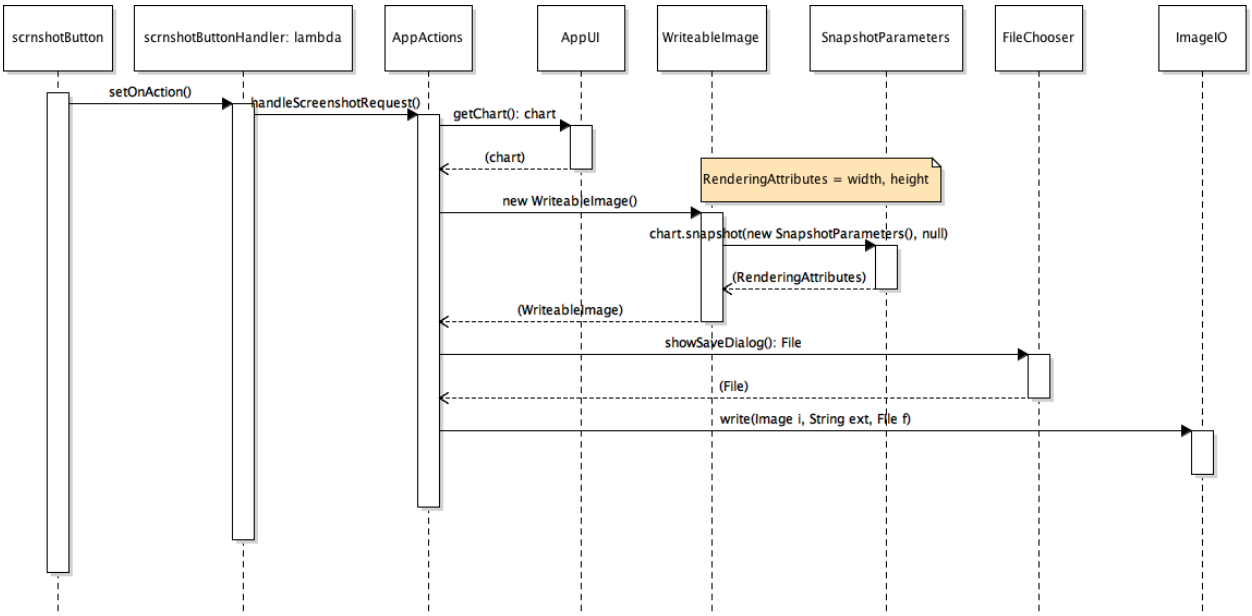


Figure 4.10: Screenshot Sequence Diagram

When the user clicks on the screenshot button, they will be prompted to select a file name and location, then the chart will be saved.

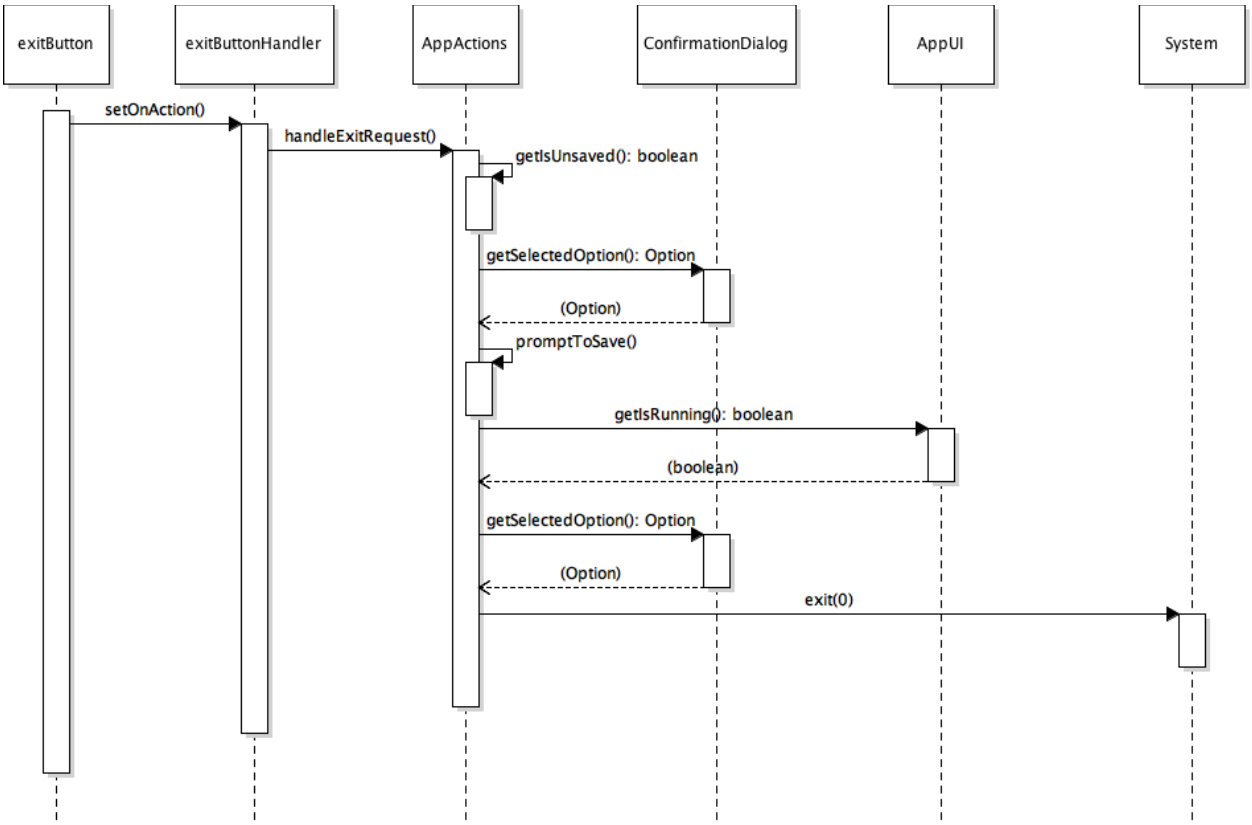


Figure 4.11: Exit Sequence Diagram

This happens whenever the user clicks the exit button. If there is unsaved data, they will be prompted to save the data before exiting. If an algorithm is running, they will be asked if they would like to wait for the algorithm to finish or exit anyway. If the user chooses to exit anyway, the program will end.

5 Files/Data Structures and Formats

This section will provide an example of what the project files will look like for this application. It will also describe the proper format for xml documents and TSD files.

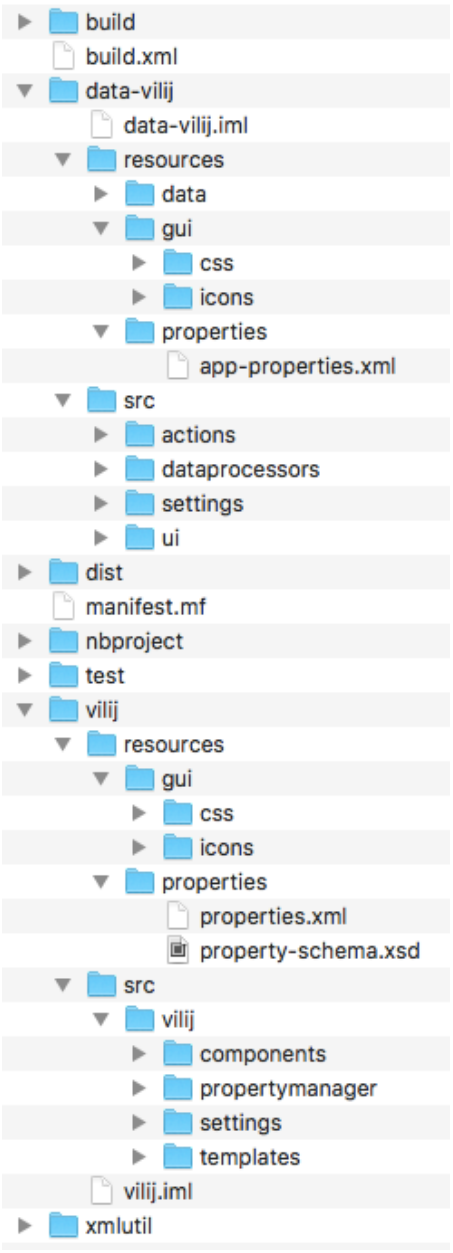
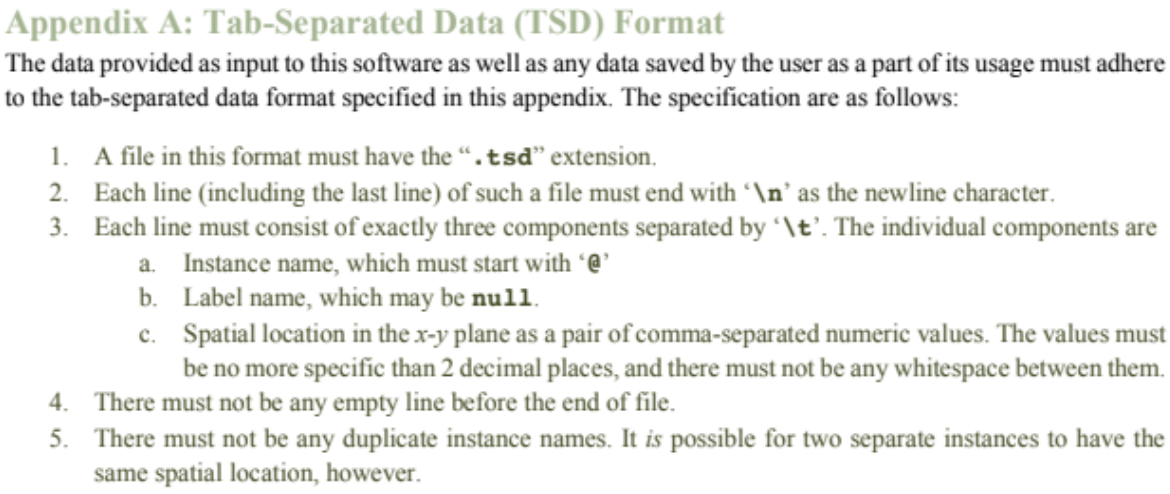


Figure 5.1 DataViliJ Program and File Structures

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This XML file allows us to easily customize the visual aesthetics of the user interface. Note that this properties
file is only for the standard high-level controls for an application, so it does not include properties meant for any
functionality within the main workspace. -->
<properties>
    <property_list>
        <!-- RESOURCE FILES AND FOLDERS -->
        <property name="DATA_RESOURCE_PATH" value="data"/>

        <property name="DATA_VILIJ_CSS_NAME" value="data-vilij.css"/>

        <!-- DATA-VILIJ DIRECTORY PATH -->
        <property name="DATA_VILIJ_RESOURCE_PATH" value="data-vilij"/>

        <!-- RESOURCE DIRECTORY PATH -->
        <property name="RESOURCE_PATH" value="resource"/>

        <!-- USER INTERFACE ICON FILES -->
        <property name="SCREENSHOT_ICON" value="screenshot.png"/>

        <!-- TOOLTIPS FOR BUTTONS -->
        <property name="SCREENSHOT_TOOLTIP" value="Screenshot"/> <!-- will print current view of image to a file -->

        <!-- WARNING MESSAGES -->
        <property name="EXIT_WHILE_RUNNING_WARNING"
                  value="An algorithm is running. If you exit now, all unsaved changes will be lost. Are you sure?"/>

        <!-- ERROR MESSAGES -->
        <property name="RESOURCE_SUBDIR_NOT_FOUND" value="Directory not found under resources."/>

        <!-- APPLICATION-SPECIFIC MESSAGE TITLES -->
        <property name="SAVE_UNSAVED_WORK_TITLE" value="Save Current Work"/>

        <!-- APPLICATION-SPECIFIC MESSAGES -->
        <property name="SAVE_UNSAVED_WORK" value="Would you like to save current work?"/>
        <property name="LOAD_WORK_TITLE" value="LOAD"/>

        <!-- APPLICATION-SPECIFIC PARAMETERS -->
        <property name="DATA_FILE_EXT" value=".tsd"/>
        <property name="DATA_FILE_EXT_DESC" value="Tab-Separated Data File"/>
        <property name="TEXT_AREA" value="text area"/>
        <property name="SPECIFIED_FILE" value=" specified file"/>
        <property name="CHART_TITLE" value="Data Visualization"/>
        <property name="DISPLAY_BUTTON_TEXT" value="Display Data"/>
        <property name="CHECKBOX_TEXT" value="Read Only"/>
        <property name="LOADING_10_OF" value="Loading 10 out of "/>
        <property name="PNG_FILE_EXT" value=".png"/>
        <property name="PNG_FILE_EXT_DESC" value="PortableNetworkGraphics"/>
    </property_list>
    <property_options_list/>
</properties>
```

Figure 5.2 app-properties.xml Format



Appendix A: Tab-Separated Data (TSD) Format

The data provided as input to this software as well as any data saved by the user as a part of its usage must adhere to the tab-separated data format specified in this appendix. The specification are as follows:

1. A file in this format must have the "**`.tsd`**" extension.
2. Each line (including the last line) of such a file must end with '**`\n`**' as the newline character.
3. Each line must consist of exactly three components separated by '**`\t`**'. The individual components are
    a. Instance name, which must start with '**`@`**'
    b. Label name, which may be **`null`**.
    c. Spatial location in the *x*-*y* plane as a pair of comma-separated numeric values. The values must be no more specific than 2 decimal places, and there must not be any whitespace between them.
4. There must not be any empty line before the end of file.
5. There must not be any duplicate instance names. It *is* possible for two separate instances to have the same spatial location, however.

Figure 5.3 TSD Format

Figure 5.3 is taken directly from the DataViliJ SRS.

6 Supporting Information

The understanding of the Clustering and Classification algorithms are not necessary to understand the SDD of DataViliJ, but please see the DataViliJ SRS for basic information about the algorithms.