Design Specification

for

WAVED

Version 2.0

Prepared By:

Sean Bluestein, Kristian Calhoun, Keith Horrocks, Steven Nguyen, Hannah Pinkos

Advisor:

Kurt Schmidt

Stakeholder:

Climate Central

Table of Contents

1. Document History

Name	Date	Reason	Version
Kristian Calhoun, Sean Bluestein, Hannah Pinkos, Steve Nguyen, Keith Horrocks	February 18, 2014	Initial Draft	1.0
Kristian Calhoun, Sean Bluestein, Hannah Pinkos, Steve Nguyen, Keith Horrocks	April 29, 2014	Updated for changes made to SRS	2.0

2. Introduction

2.1 Purpose

This document specifies the software architecture and design specifications for Web App for Visualizing Environmental Data (WAVED). The design decisions outlined in this document were made in compliance with the software constraints and functionality requirements outlined in the WAVED Software Requirements Specification document.

2.2 Scope

This document describes the software architecture for the initial release of WAVED. The implementation of WAVED adheres to the models and design decisions outlined in this report. The intended audience of this document includes the designers, developers and testers of WAVED.

2.3 Design Goals

- **Modular** - The architecture of the application separates functionality into logical modules that interact with one another. This aids in the creation and modification of the application by separating concerns into the appropriate modules and creating minimal code redundancy.

- **Extensible** - The application is extensible and facilitates the creation of new widgets. When developers add a new widget to the application as a plugin, they do not have to define all of its associated views. To add a new widget, the developer extends existing view model classes that supply common functionality. The application automatically generates the necessary Properties Panel view for each widget by inspecting the widget's properties. Each property type has an associated template used to create the view. Development work is reduced and consistency between the appearance for each widget's properties panel is improved by not requiring the widgets to define property views.

- **Safe Interface** - There are several measures taken to avoid the loss of work from accidental user operations. The interface allows users to undo and redo project changes to revert accidental actions. Removing data sources from the WAVED interface does not immediately delete the files from the server. Instead, the files are marked for deletion and removed from the server the next time the project is loaded. This allows the user to undo the deletion of the data source and return to a state where the data source is used by other components while the application is open.

2.4 Definitions

Action - A process that changes the properties of a widget or data set via an event

Asynchronous JavaScript and XML (AJAX) - A process that allows for exchanging information between a web page and a server and updating content on the web page without having to reload it.

Binding Data - An explicit association between a data set and a widget, property, or event

Cascading Style Sheets (CSS) - A style sheet language that defines the look of HTML documents.

Component - A widget or workspace

Data Set - A data source or data subset

Data Source - A collection of data, such as the contents of a CSV or JSON file

Data Subset - The resulting portion of a data source that meets some specified criteria.

Document Object Model (DOM) - A description of HTML document including elements, attributes and text

Event - A user interaction (click, hover, etc.) that triggers an action

Glyph - A marker indicated by some shape or icon

Hypertext Markup Language (HTML) - HTML is the primary markup language used to display web pages in web browsers.

JavaScript - A dynamic programming language used in web browsers to interact with clients.

JavaScript Object Notation (JSON) - A language-independent format used to transmit data

Knockout - A library that executes the MVVM design pattern via subscriptions.

PHP Hypertext Preprocessor (PHP) - A server-side scripting language

Model-View-View Model (MVVM) - An architecture design pattern that allows for the separation of data representation, data visualization and data manipulation

Project - The context in which a user works to create a single visualization using the WAVED application

Property - Defines a value used for a widget attribute, such as width, height, color, name, etc.

Widget - A top level component that has properties and bound data

Workspace - The interactive area of the WAVED user interface that displays the visualization as it is assembled from widgets.

2.5 References

This document references the requirements in the *Software Requirements Specification for WAVED* document. Additionally, information about validation tests for these requirements are found in the *Acceptance Test Plan for WAVED*.

3. System Overview

3.1 Context Diagram

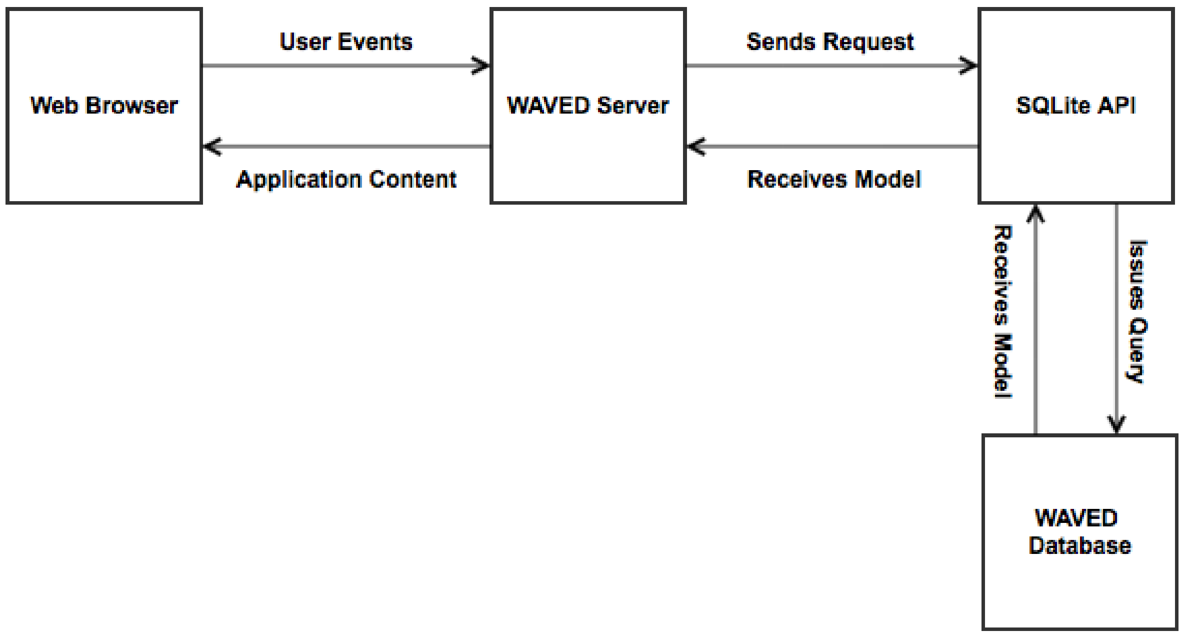


Figure 1. WAVED Context Diagram

Figure 1 shows the four main components of the WAVED system. Hosted on a web server, the WAVED application interacts with clients' web browsers via HTTP requests and a database via the SQLite API. The WAVED server generates content and the web browser displays it. The server receives user events from the browser, which updates the view model of the application. When the user saves the project, the server updates the model saved in the database accordingly. When the user loads a saved project, the server gets the model of the saved project from the database.

Projects exported from WAVED are a collection of CSS, HTML, and JavaScript files and are hosted on their own separate server. They do not require any interaction with the WAVED database.

3.2 Technologies Used

WAVED is implemented in JavaScript with HTML and CSS for styling the graphical user interface (GUI) and PHP for client-server communication. The application leverages the third-party JavaScript libraries jQuery, D3, and Knockout to simplify development. jQuery manipulates HTML DOM elements, handles user events, and make Asynchronous JavaScript and XML (AJAX) calls to execute server side PHP functions. Additionally, jQuery-UI is used to control the look and feel of the user interface. D3 creates the data-driven widgets in WAVED such as the U.S. Map and Line Graph widgets. Knockout assists with the implementation of a Model-View-View Model (MVVM) design pattern for the system by dynamically updating the view when changes are made to the associated view model.

On the server-side, an Apache HTTP Server serves the WAVED system content to clients. The server also communicates with a SQLite database, which stores saved projects and user accounts. WAVED is compatible with Google Chrome version 34.0+ and Mozilla Firefox version 26.0+ web browsers.

3.3 Deployment Diagram

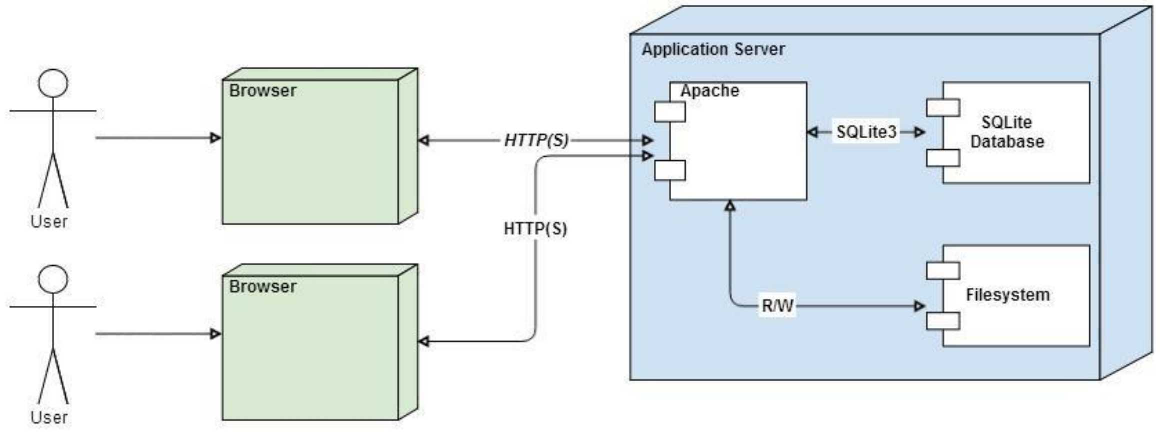


Figure 2. WAVED Deployment Diagram

Figure 2 shows a high level overview of the deployment of the WAVED application. Two components are required for using the application: a browser and a server. Users access the application using a web browser. The server stores the content needed to run the WAVED system. The browser communicates with an Apache server instance on the application server using HTTP(S) requests. To serve these requests, Apache reads or writes to the SQLite database or local filesystem as needed.

3.4 Architecture Diagram

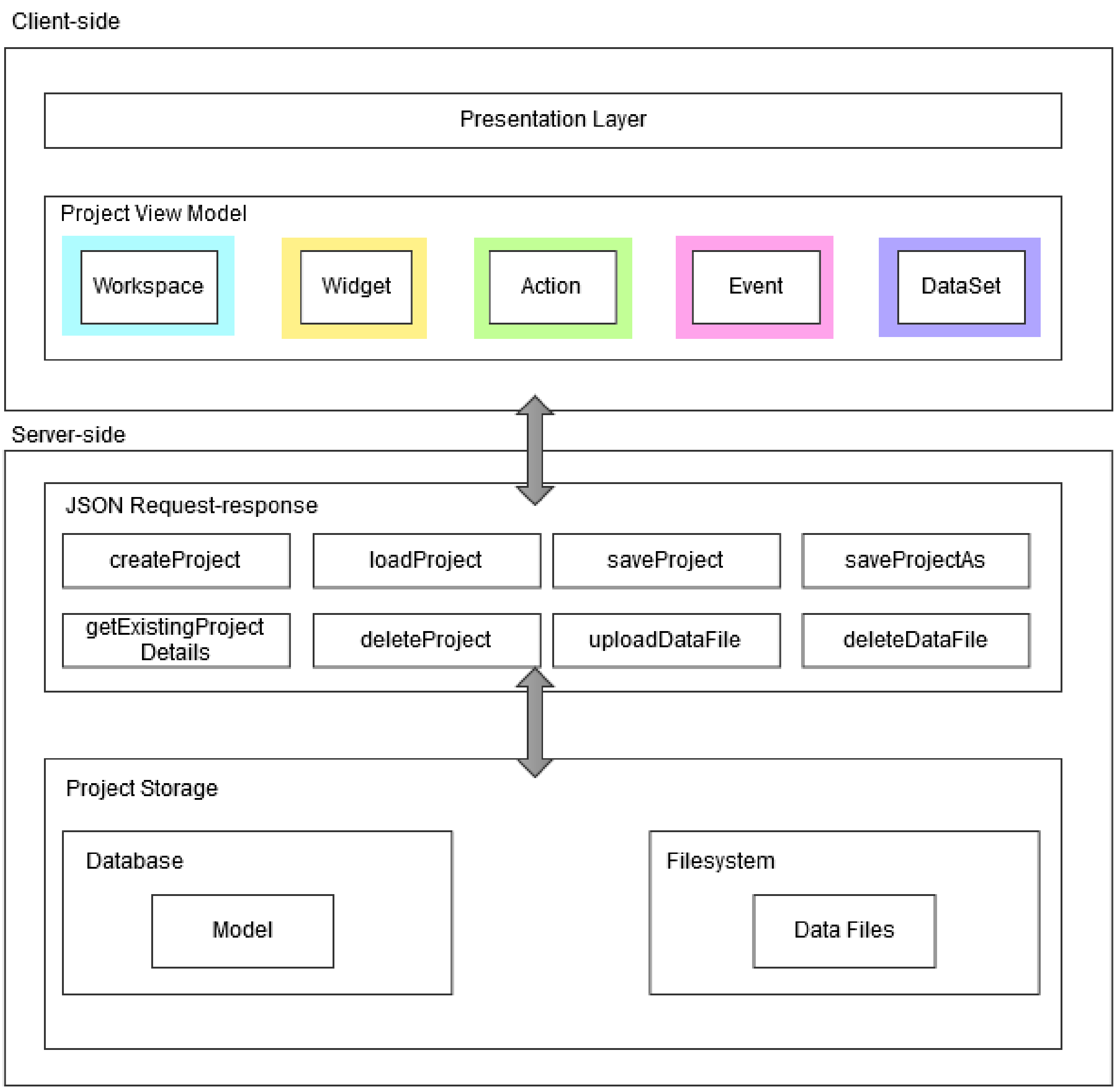


Figure 3. WAVED Architecture Diagram

Figure 3 shows a general architecture overview of the WAVED application, split between client-side and server-side components. Most work is done on the client-side, with the exception of persisting data, which is done server-side so that data can be accessed from any client. Communication between the client and server is done through JSON requests and responses.

The client-side components include the presentation layer and the project view model. The presentation layer allows users to interact with the system to view and modify the current state of a project being developed. The presentation layer interacts with the underlying project view model. The view model encapsulates the workspace, widgets, actions, events and data associated with that project.

The server-side components handle the persistent storage functions needed for saving, loading, and exporting projects. The project view model (excluding data source files) from the client-side is serialized and stored in the database as a project model. The data files associated with the project are stored on the server's file system.

3.5 Model-View-View Model Diagram

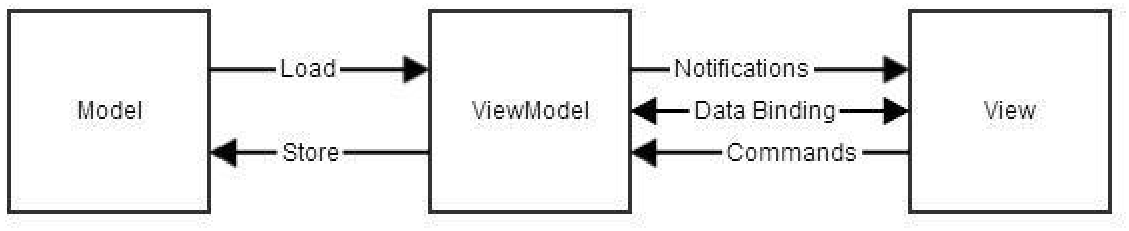


Figure 4. MVVM Diagram

Figure 4 shows the three main components of the Model-View-View Model (MVVM) design pattern and their interactions. MVVM separates the concerns of storing, operating on and viewing data into model, view model and view components. A model is the stored data that interacts with the view model by saving and loading the data as needed. A view model is a code representation of the stored data along with methods for operating on that data. A view describes how to present the data from the view model to the user, and includes UI elements to expose the methods of the view model. Data binding and notifications keep the view in sync with the data in the view model. The view also sends commands to the view model to operate on the data using the exposed methods.

3.6 Component Diagram

Figure 5 shows the interactions between the different components in the architectural and context diagrams. The Browser component sends commands to the view model component, which then updates the underlying data model and views. The models are saved to the State component, which resides within the Database component. Datasets are composed of Data Files stored within the File System component.

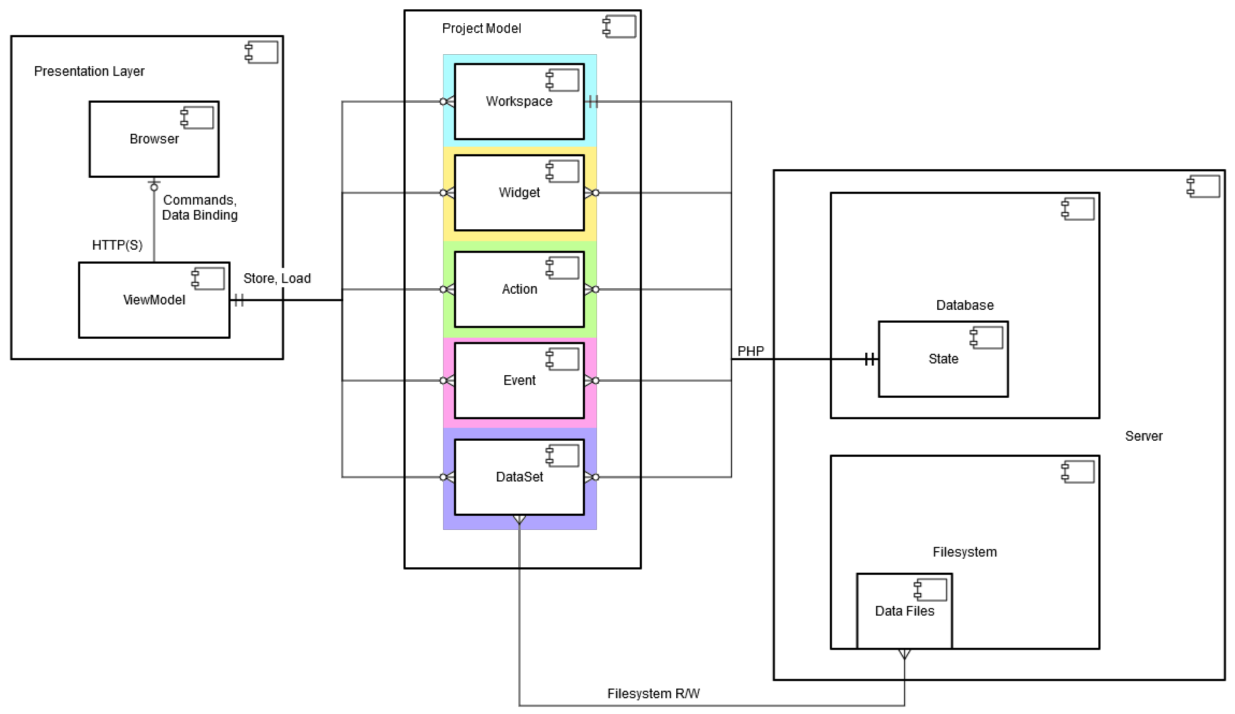


Figure 5. Component Diagram

3.7 Sequence Diagrams

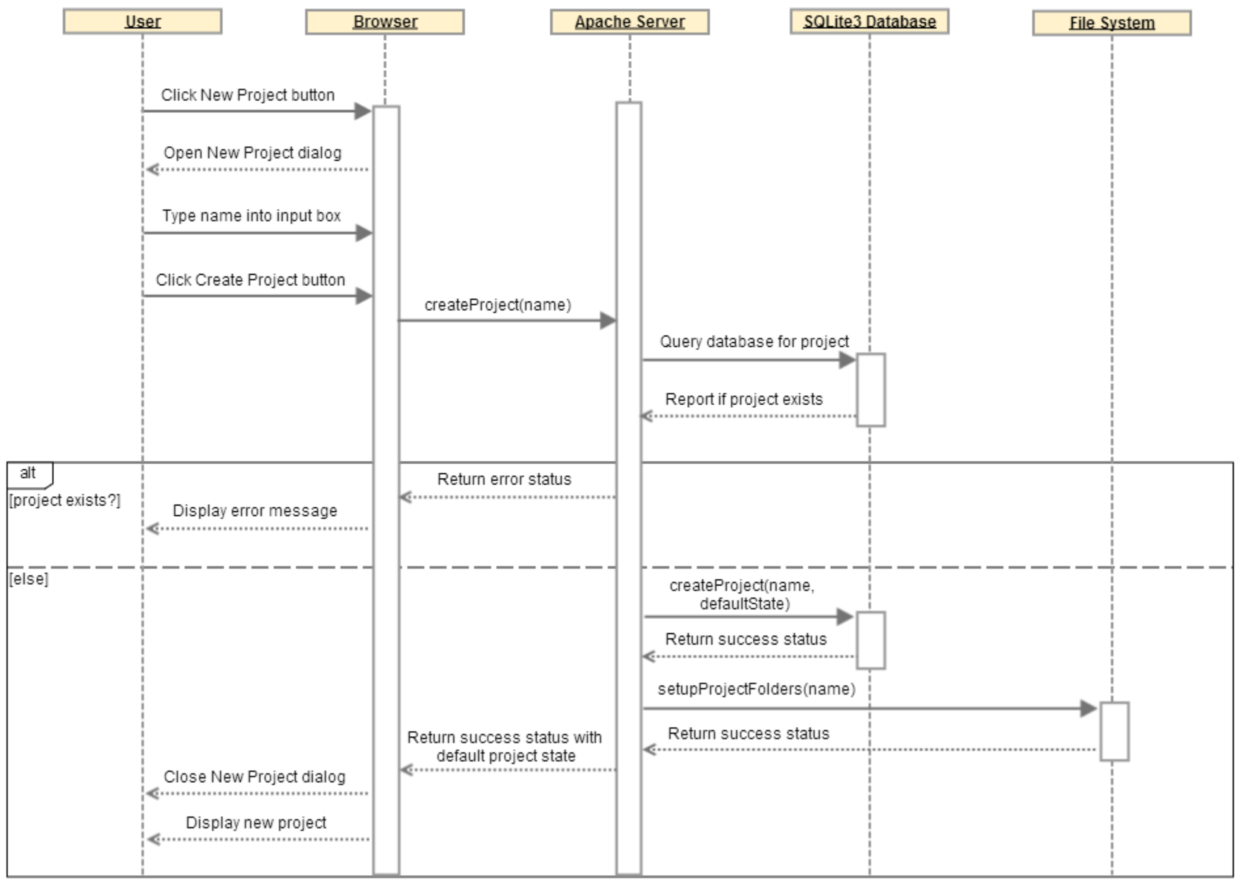


Figure 6. Sequence diagram for creating a new project

Figure 6 shows the process of creating a new project. First, the user clicks the New Project button which causes the browser to display the associated dialog. Then the user types a name for the new project and clicks the Create Project button. The server queries the database to check if the project exists. If the project already exists, the browser displays an error message to the user. If the project does not exist, the server saves the default project state to the database and creates the necessary project folders in the file system. The server returns the state to the browser, which then closes the New Project dialog and displays the new project.

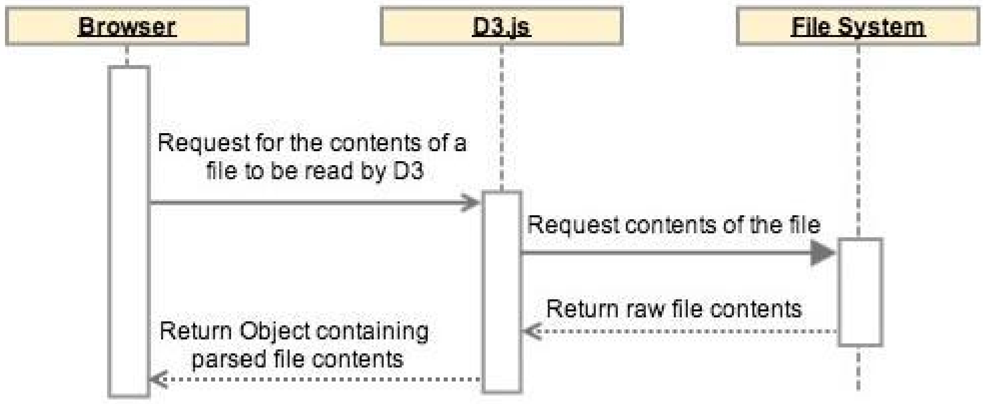

Browser
D3.js
File System
Request for the contents of a file to be read by D3
Request contents of the file
Return raw file contents
Return Object containing parsed file contents

Figure 7. Sequence diagram for reading file contents using D3

Figure 7 shows the process of creating data-driven objects (such as the U.S. Map and map glyphs) from data source files. The JavaScript D3 library requests the raw file contents from the server’s file system and then parses the file into an Object.

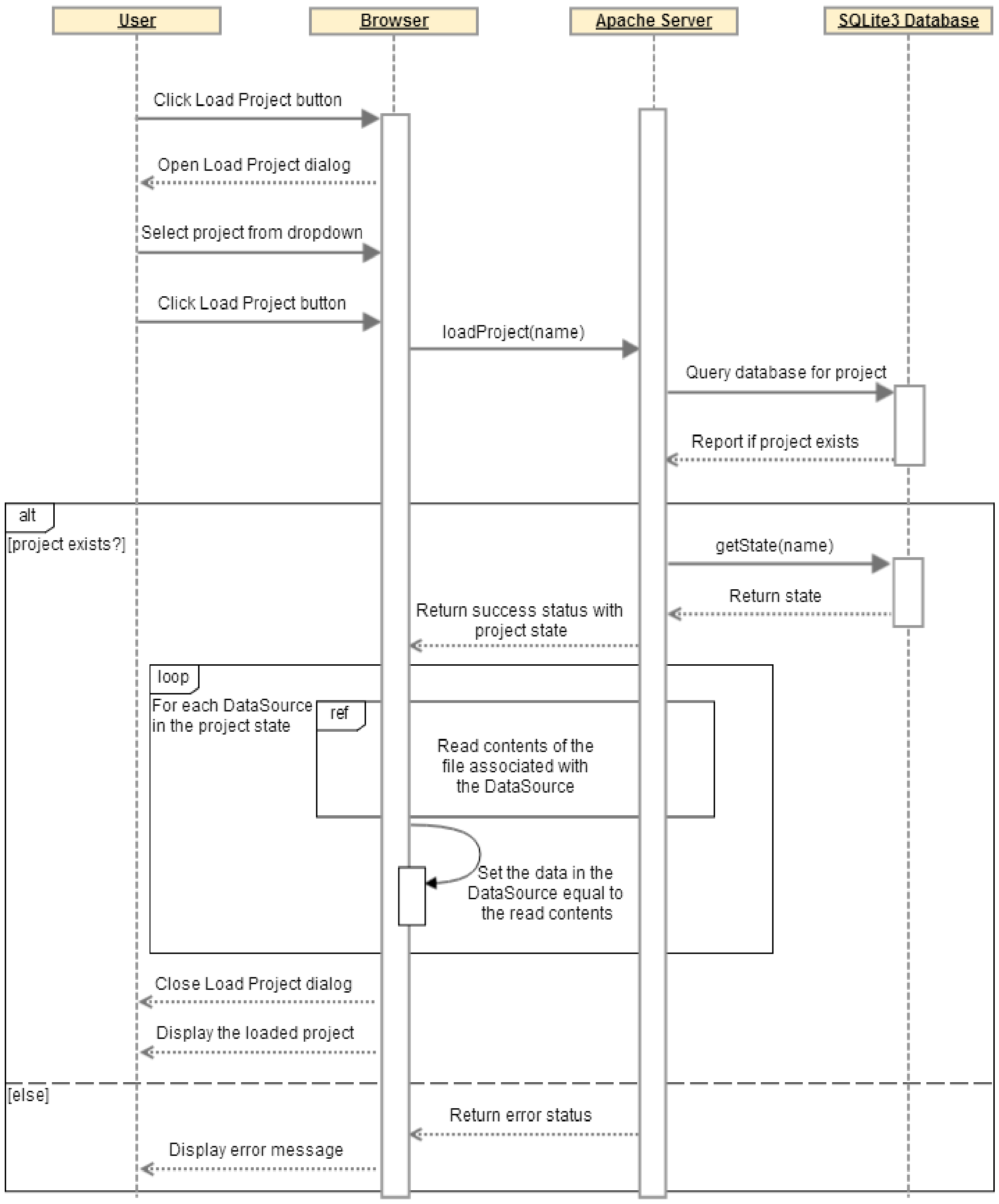

User
Browser
Apache Server
SQLite3 Database
Click Load Project button
Open Load Project dialog
Select project from dropdown
Click Load Project button
loadProject(name)
Query database for project
Report if project exists
alt
[project exists?]
getState(name)
Return state
Return success status with project state
loop
For each DataSource in the project state
ref
Read contents of the file associated with the DataSource
Set the data in the DataSource equal to the read contents
Close Load Project dialog
Display the loaded project
[else]
Return error status
Display error message

Figure 8. Sequence diagram for loading an existing project

Figure 8 shows the process of loading an existing project. First, the user clicks the Load Project button which causes the browser to display the associated dialog. Then the user selects a project from the list of available projects and clicks the Load Project button. The server queries the database to check if the project exists, which it may not if the project was just deleted. If the project does not exist, the browser displays an error message to the user. If the project exists, the server retrieves the project state from the database. The server returns the state to the browser, which closes the Load Project dialog and displays the loaded project. This diagram references Figure 7, the sequence diagram for reading file contents.

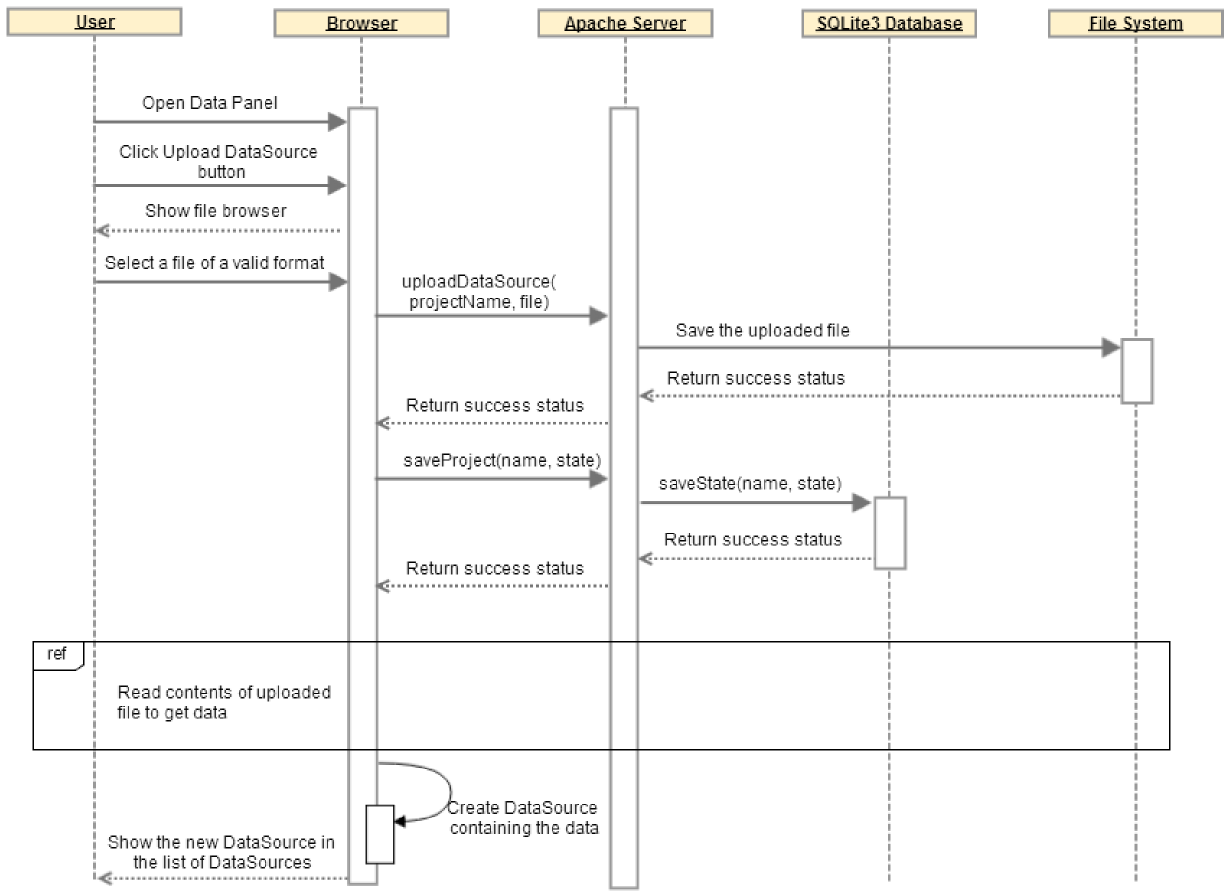


Figure 9. Sequence diagram for uploading a file to the server

Figure 9 shows the process of uploading a file to the server. The user first opens the Data Panel and clicks the button to upload a new DataSource. Then the browser displays a file browser, from which the user selects a file. The project automatically saves after the file is uploaded to the file system to avoid the possibility of unreferenced files. Finally, the JavaScript library D3 parses the new file, creates a new DataSource (an instance of the DataSet class), and displays the new DataSource in the list of available DataSources. This diagram references Figure 7, the sequence diagram for reading file contents.

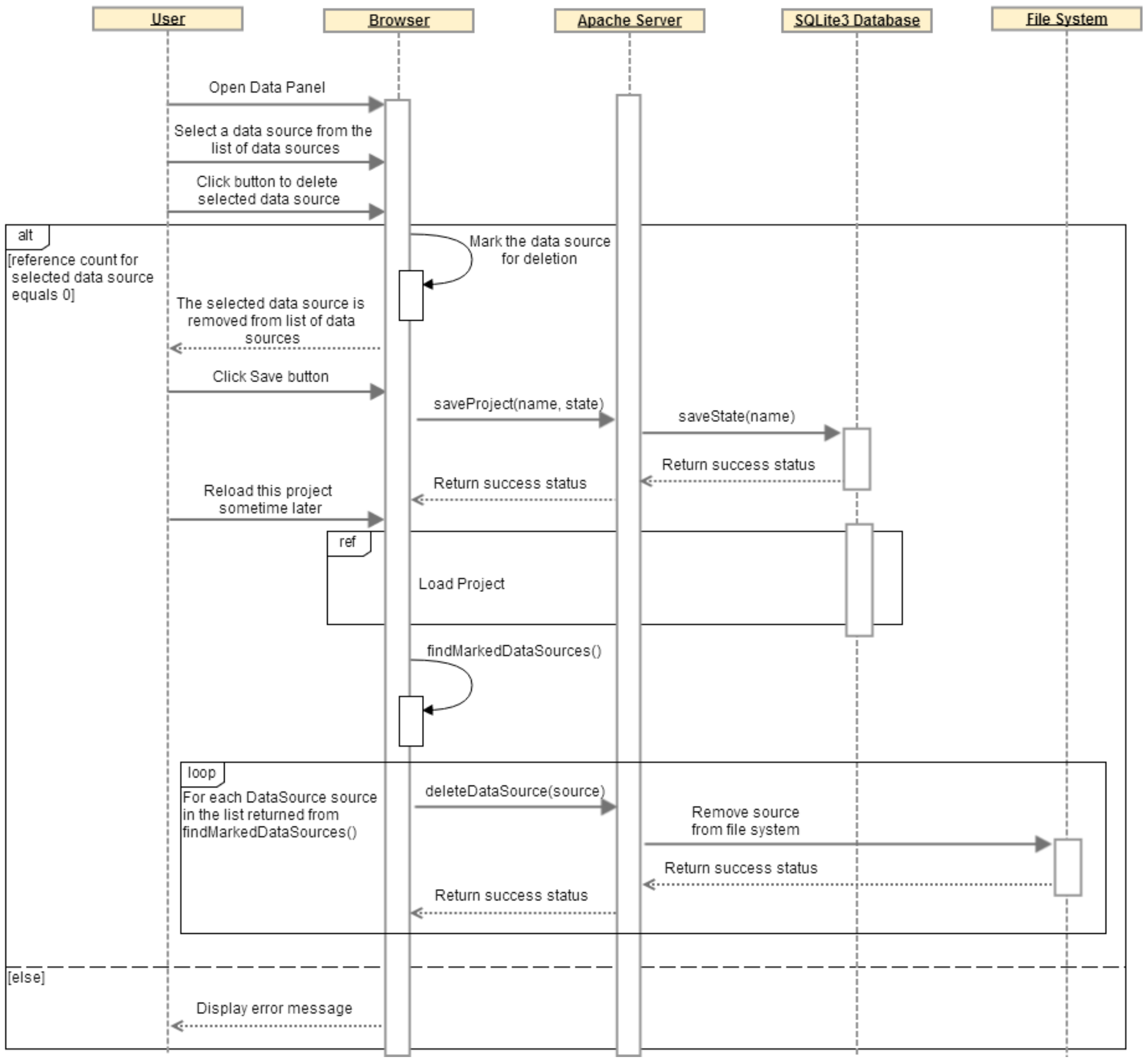

User
Browser
Apache Server
SQLite3 Database
File System
Open Data Panel
Select a data source from the list of data sources
Click button to delete selected data source
alt
[reference count for selected data source equals 0]
Mark the data source for deletion
The selected data source is removed from list of data sources
Click Save button
saveProject(name, state)
saveState(name)
Return success status
Return success status
Reload this project sometime later
ref
Load Project
findMarkedDataSources()
loop
For each DataSource source in the list returned from findMarkedDataSources()
deleteDataSource(source)
Remove source from file system
Return success status
Return success status
[else]
Display error message

Figure 10. Sequence diagram for deleting a DataSource (file) from the server

Figure 10 shows the process of deleting a DataSource from the current project. A DataSource cannot be deleted immediately after the user makes the request because then the user would not be able to undo that action. For that reason, a DataSource is first marked for deletion, and the actual file corresponding with the DataSource is deleted the next time the project is loaded. This diagram references Figure 8, the sequence diagram for loading a project.

4. Detailed System Architecture

4.1 Class Diagram

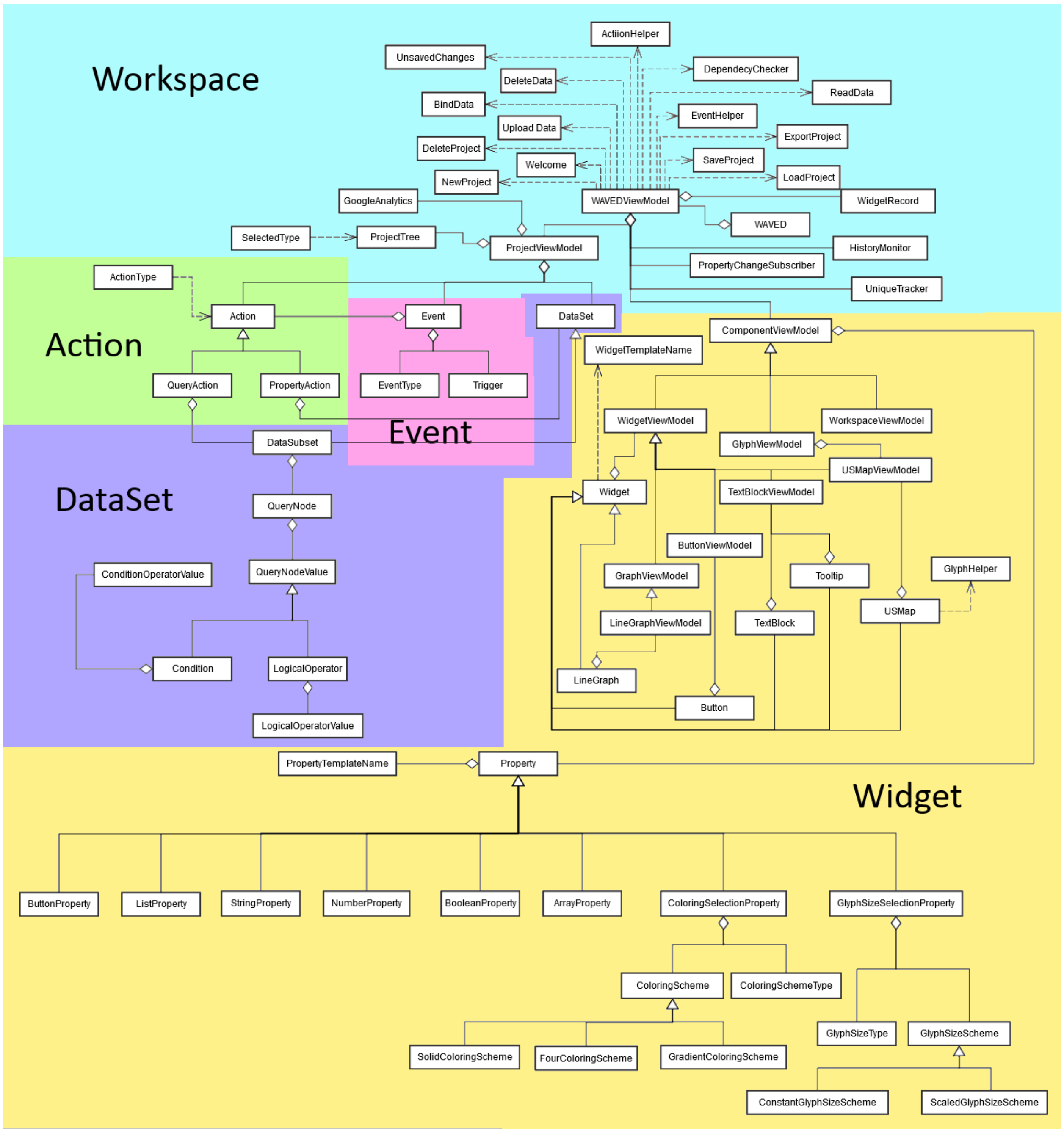
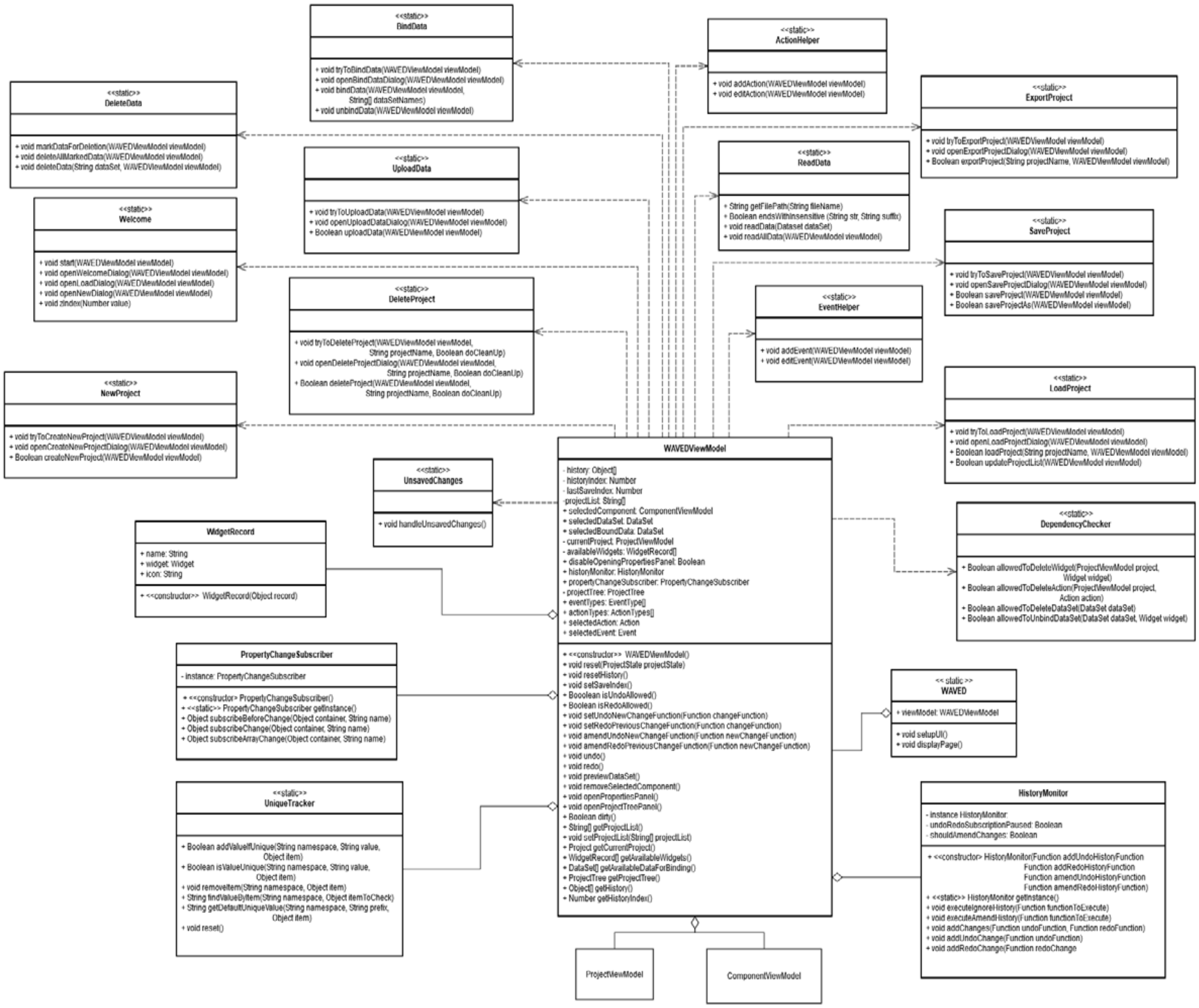


Figure 11. Simplified class relationship diagram for the entire WAVED application.



<<static>> BindData
+ void tryToBindData(WAVEDViewModel viewModel)
+ void openBindDataDialog(WAVEDViewModel viewModel)
+ void bindData(WAVEDViewModel viewModel, String[] dataSetNames)
+ void unbindData(WAVEDViewModel viewModel)
<<static>> ActionHelper
+ void addAction(WAVEDViewModel viewModel)
+ void editAction(WAVEDViewModel viewModel)
<<static>> DeleteData
+ void markDataForDeletion(WAVEDViewModel viewModel)
+ void deleteAllMarkedData(WAVEDViewModel viewModel)
+ void deleteData(String dataSet, WAVEDViewModel viewModel)
<<static>> ExportProject
+ void tryToExportProject(WAVEDViewModel viewModel)
+ void openExportProjectDialog(WAVEDViewModel viewModel)
+ Boolean exportProject(String projectName, WAVEDViewModel viewModel)
<<static>> UploadData
+ void tryToUploadData(WAVEDViewModel viewModel)
+ void openUploadDataDialog(WAVEDViewModel viewModel)
+ Boolean uploadData(WAVEDViewModel viewModel)
<<static>> ReadData
+ String getFilePath(String fileName)
+ Boolean endsWithInsensitive (String str, String suffix)
+ void readData(Dataset dataSet)
+ void readAllData(WAVEDViewModel viewModel)
<<static>> Welcome
+ void start(WAVEDViewModel viewModel)
+ void openWelcomeDialog(WAVEDViewModel viewModel)
+ void openLoadDialog(WAVEDViewModel viewModel)
+ void openNewDialog(WAVEDViewModel viewModel)
+ void zIndex(Number value)
<<static>> SaveProject
+ void tryToSaveProject(WAVEDViewModel viewModel)
+ void openSaveProjectDialog(WAVEDViewModel viewModel)
+ Boolean saveProject(WAVEDViewModel viewModel)
+ Boolean saveProjectAs(WAVEDViewModel viewModel)
<<static>> DeleteProject
+ void tryToDeleteProject(WAVEDViewModel viewModel, String projectName, Boolean doCleanUp)
+ void openDeleteProjectDialog(WAVEDViewModel viewModel, String projectName, Boolean doCleanUp)
+ Boolean deleteProject(WAVEDViewModel viewModel, String projectName, Boolean doCleanUp)
<<static>> EventHelper
+ void addEvent(WAVEDViewModel viewModel)
+ void editEvent(WAVEDViewModel viewModel)
<<static>> NewProject
+ void tryToCreateNewProject(WAVEDViewModel viewModel)
+ void openCreateNewProjectDialog(WAVEDViewModel viewModel)
+ Boolean createNewProject(WAVEDViewModel viewModel)
<<static>> LoadProject
+ void tryToLoadProject(WAVEDViewModel viewModel)
+ void openLoadProjectDialog(WAVEDViewModel viewModel)
+ Boolean loadProject(String projectName, WAVEDViewModel viewModel)
+ Boolean updateProjectList(WAVEDViewModel viewModel)
<<static>> UnsavedChanges
+ void handleUnsavedChanges()
WAVEDViewModel
- history: Object[]
- historyIndex: Number
- lastSaveIndex: Number
- projectList: String[]
+ selectedComponent: ComponentViewModel
+ selectedDataSet: DataSet
+ selectedBoundData: DataSet
- currentProject: ProjectViewModel
- availableWidgets: WidgetRecord[]
+ disableOpeningPropertiesPanel: Boolean
+ historyMonitor: HistoryMonitor
+ propertyChangeSubscriber: PropertyChangeSubscriber
- projectTree: ProjectTree
+ eventTypes: EventType[]
+ actionTypes: ActionTypes[]
+ selectedAction: Action
+ selectedEvent: Event
+ <<constructor>> WAVEDViewModel()
+ void reset(ProjectState projectState)
+ void resetHistory()
+ void setSaveIndex()
+ Booolean isUndoAllowed()
+ Boolean isRedoAllowed()
+ void setUndoNewChangeFunction(Function changeFunction)
+ void setRedoPreviousChangeFunction(Function changeFunction)
+ void amendUndoNewChangeFunction(Function newChangeFunction)
+ void amendRedoPreviousChangeFunction(Function newChangeFunction)
+ void undo()
+ void redo()
+ void previewDataSet()
+ void removeSelectedComponent()
+ void openPropertiesPanel()
+ void openProjectTreePanel()
+ Boolean dirty()
+ String[] getProjectList()
+ void setProjectList(String[] projectList)
+ Project getCurrentProject()
+ WidgetRecord[] getAvailableWidgets()
+ DataSet[] getAvailableDataForBinding()
+ ProjectTree getProjectTree()
+ Object[] getHistory()
+ Number getHistoryIndex()
WidgetRecord
+ name: String
+ widget: Widget
+ icon: String
+ <<constructor>> WidgetRecord(Object record)
<<static>> DependencyChecker
+ Boolean allowedToDeleteWidget(ProjectViewModel project, Widget widget)
+ Boolean allowedToDeleteAction(ProjectViewModel project, Action action)
+ Boolean allowedToDeleteDataSet(DataSet dataSet)
+ Boolean allowedToUnbindDataSet(DataSet dataSet, Widget widget)
PropertyChangeSubscriber
- instance: PropertyChangeSubscriber
+ <<constructor> PropertyChangeSubscriber()
+ <<static>> PropertyChangeSubscriber getInstance()
+ Object subscribeBeforeChange(Object container, String name)
+ Object subscribeChange(Object container, String name)
+ Object subscribeArrayChange(Object container, String name)
<< static >> WAVED
+ viewModel: WAVEDViewModel
+ void setupUI()
+ void displayPage()
<<static>> UniqueTracker
+ Boolean addValueIfUnique(String namespace, String value, Object item)
+ Boolean isValueUnique(String namespace, String value, Object item)
+ void removeItem(String namespace, Object item)
+ String findValueByItem(String namespace, Object itemToCheck)
+ String getDefaultUniqueValue(String namespace, String prefix, Object item)
+ void reset()
HistoryMonitor
- instance HistoryMonitor
- undoRedoSubscriptionPaused: Boolean
- shouldAmendChanges: Boolean
+ <<constructor> HistoryMonitor(Function addUndoHistoryFunction Function addRedoHistoryFunction Function amendUndoHistoryFunction Function amendRedoHistoryFunction)
+ <<static>> HistoryMonitor getInstance()
+ void executeIgnoreHistory(Function functionToExecute)
+ void executeAmendHistory(Function functionToExecute)
+ void addChanges(Function undoFunction, Function redoFunction)
+ void addUndoChange(Function undoFunction)
+ void addRedoChange(Function redoChange
ProjectViewModel
ComponentViewModel

Figure 11a. The main WAVED view model class

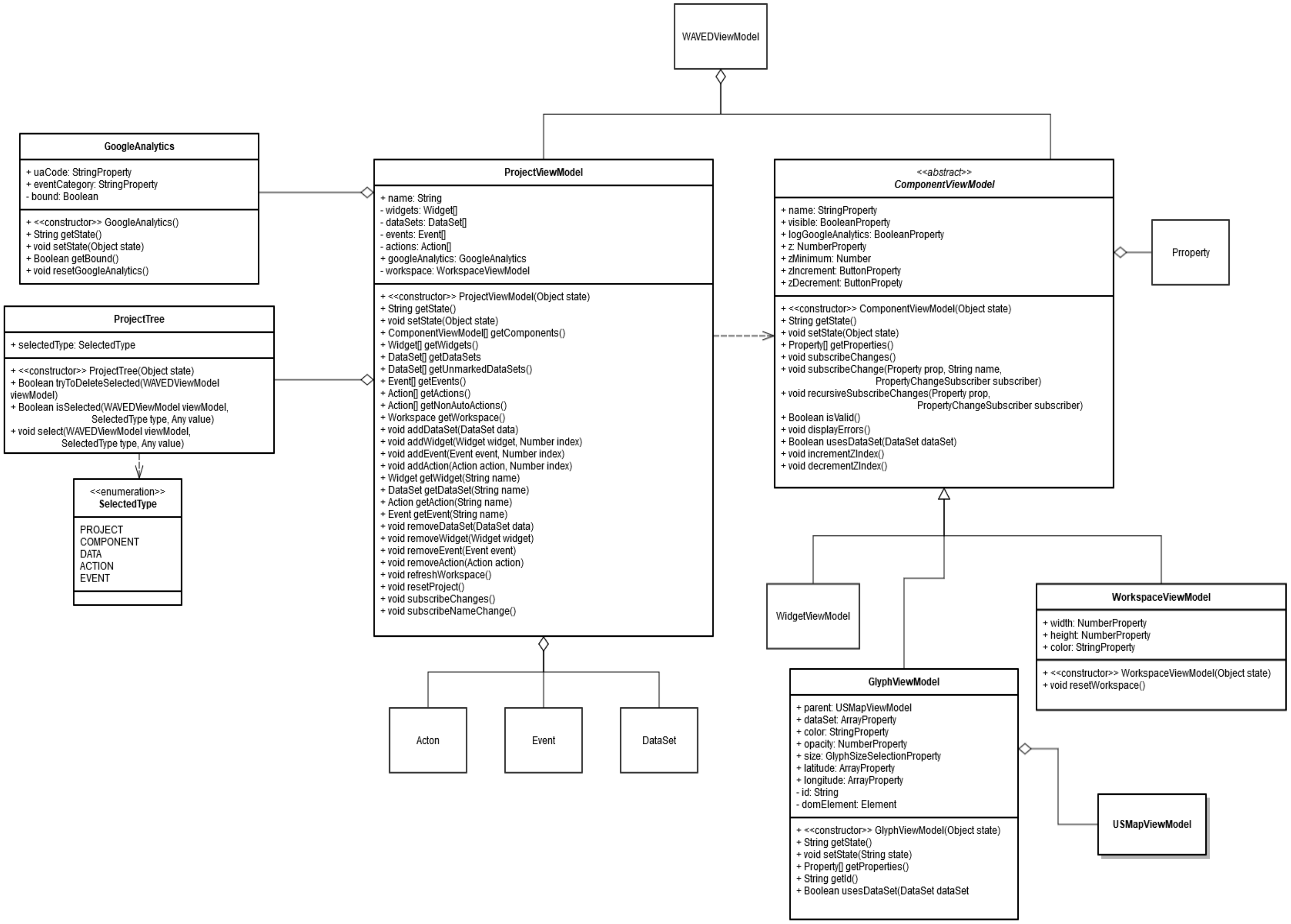

WAVEDViewModel
GoogleAnalytics
+ uaCode: StringProperty
+ eventCategory: StringProperty
- bound: Boolean
+ <<constructor>> GoogleAnalytics()
+ String getState()
+ void setState(Object state)
+ Boolean getBound()
+ void resetGoogleAnalytics()
ProjectTree
+ selectedType: SelectedType
+ <<constructor>> ProjectTree(Object state)
+ Boolean tryToDeleteSelected(WAVEDViewModel viewModel)
+ Boolean isSelected(WAVEDViewModel viewModel, SelectedType type, Any value)
+ void select(WAVEDViewModel viewModel, SelectedType type, Any value)
<<enumeration>>
SelectedType
PROJECT
COMPONENT
DATA
ACTION
EVENT
ProjectViewModel
+ name: String
- widgets: Widget[]
- dataSets: DataSet[]
- events: Event[]
- actions: Action[]
+ googleAnalytics: GoogleAnalytics
- workspace: WorkspaceViewModel
+ <<constructor>> ProjectViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ ComponentViewModel[] getComponents()
+ Widget[] getWidgets()
+ DataSet[] getDataSets
+ DataSet[] getUnmarkedDataSets()
+ Event[] getEvents()
+ Action[] getActions()
+ Action[] getNonAutoActions()
+ Workspace getWorkspace()
+ void addDataSet(DataSet data)
+ void addWidget(Widget widget, Number index)
+ void addEvent(Event event, Number index)
+ void addAction(Action action, Number index)
+ Widget getWidget(String name)
+ DataSet getDataSet(String name)
+ Action getAction(String name)
+ Event getEvent(String name)
+ void removeDataSet(DataSet data)
+ void removeWidget(Widget widget)
+ void removeEvent(Event event)
+ void removeAction(Action action)
+ void refreshWorkspace()
+ void resetProject()
+ void subscribeChanges()
+ void subscribeNameChange()
Acton
Event
DataSet
<<abstract>>
ComponentViewModel
+ name: StringProperty
+ visible: BooleanProperty
+ logGoogleAnalytics: BooleanProperty
+ z: NumberProperty
+ zMinimum: Number
+ zIncrement: ButtonProperty
+ zDecrement: ButtonPropety
+ <<constructor>> ComponentViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()
+ void subscribeChanges()
+ void subscribeChange(Property prop, String name, PropertyChangeSubscriber subscriber)
+ void recursiveSubscribeChanges(Property prop, PropertyChangeSubscriber subscriber)
+ Boolean isValid()
+ void displayErrors()
+ Boolean usesDataSet(DataSet dataSet)
+ void incrementZIndex()
+ void decrementZIndex()
Prroperty
WidgetViewModel
WorkspaceViewModel
+ width: NumberProperty
+ height: NumberProperty
+ color: StringProperty
+ <<constructor>> WorkspaceViewModel(Object state)
+ void resetWorkspace()
GlyphViewModel
+ parent: USMapViewModel
+ dataSet: ArrayProperty
+ color: StringProperty
+ opacity: NumberProperty
+ size: GlyphSizeSelectionProperty
+ latitude: ArrayProperty
+ longitude: ArrayProperty
- id: String
- domElement: Element
+ <<constructor>> GlyphViewModel(Object state)
+ String getState()
+ void setState(String state)
+ Property[] getProperties()
+ String getId()
+ Boolean usesDataSet(DataSet dataSet
USMapViewModel

Figure 11b. The project and component view model classes

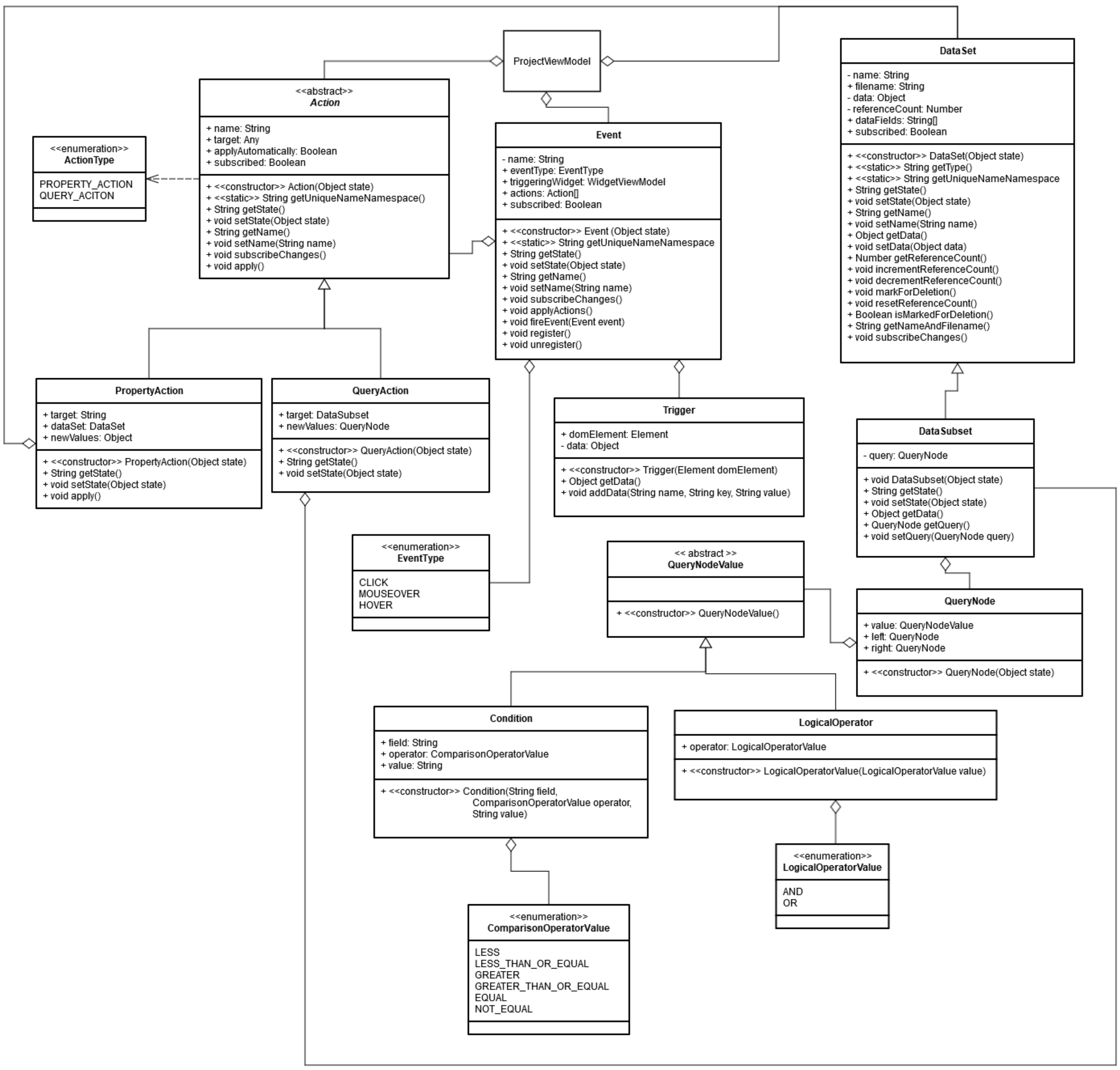

ProjectViewModel
<<abstract>>
Action
+ name: String
+ target: Any
+ applyAutomatically: Boolean
+ subscribed: Boolean
+ <<constructor>> Action(Object state)
+ <<static>> String getUniqueNameNamespace()
+ String getState()
+ void setState(Object state)
+ String getName()
+ void setName(String name)
+ void subscribeChanges()
+ void apply()
<<enumeration>>
ActionType
PROPERTY_ACTION
QUERY_ACITON
Event
- name: String
+ eventType: EventType
+ triggeringWidget: WidgetViewModel
+ actions: Action[]
+ subscribed: Boolean
+ <<constructor>> Event (Object state)
+ <<static>> String getUniqueNameNamespace
+ String getState()
+ void setState(Object state)
+ String getName()
+ void setName(String name)
+ void subscribeChanges()
+ void applyActions()
+ void fireEvent(Event event)
+ void register()
+ void unregister()
DataSet
- name: String
+ filename: String
- data: Object
- referenceCount: Number
+ dataFields: String[]
+ subscribed: Boolean
+ <<constructor>> DataSet(Object state)
+ <<static>> String getType()
+ <<static>> String getUniqueNameNamespace
+ String getState()
+ void setState(Object state)
+ String getName()
+ void setName(String name)
+ Object getData()
+ void setData(Object data)
+ Number getReferenceCount()
+ void incrementReferenceCount()
+ void decrementReferenceCount()
+ void markForDeletion()
+ void resetReferenceCount()
+ Boolean isMarkedForDeletion()
+ String getNameAndFilename()
+ void subscribeChanges()
PropertyAction
+ target: String
+ dataSet: DataSet
+ newValues: Object
+ <<constructor>> PropertyAction(Object state)
+ String getState()
+ void setState(Object state)
+ void apply()
QueryAction
+ target: DataSubset
+ newValues: QueryNode
+ <<constructor>> QueryAction(Object state)
+ String getState()
+ void setState(Object state)
Trigger
+ domElement: Element
- data: Object
+ <<constructor>> Trigger(Element domElement)
+ Object getData()
+ void addData(String name, String key, String value)
DataSubset
- query: QueryNode
+ void DataSubset(Object state)
+ String getState()
+ void setState(Object state)
+ Object getData()
+ QueryNode getQuery()
+ void setQuery(QueryNode query)
<<enumeration>>
EventType
CLICK
MOUSEOVER
HOVER
<< abstract >>
QueryNodeValue
+ <<constructor>> QueryNodeValue()
QueryNode
+ value: QueryNodeValue
+ left: QueryNode
+ right: QueryNode
+ <<constructor>> QueryNode(Object state)
Condition
+ field: String
+ operator: ComparisonOperatorValue
+ value: String
+ <<constructor>> Condition(String field, ComparisonOperatorValue operator, String value)
LogicalOperator
+ operator: LogicalOperatorValue
+ <<constructor>> LogicalOperatorValue(LogicalOperatorValue value)
<<enumeration>>
LogicalOperatorValue
AND
OR
<<enumeration>>
ComparisonOperatorValue
LESS
LESS_THAN_OR_EQUAL
GREATER
GREATER_THAN_OR_EQUAL
EQUAL
NOT_EQUAL

Figure 11c. The event, action, and data set classes

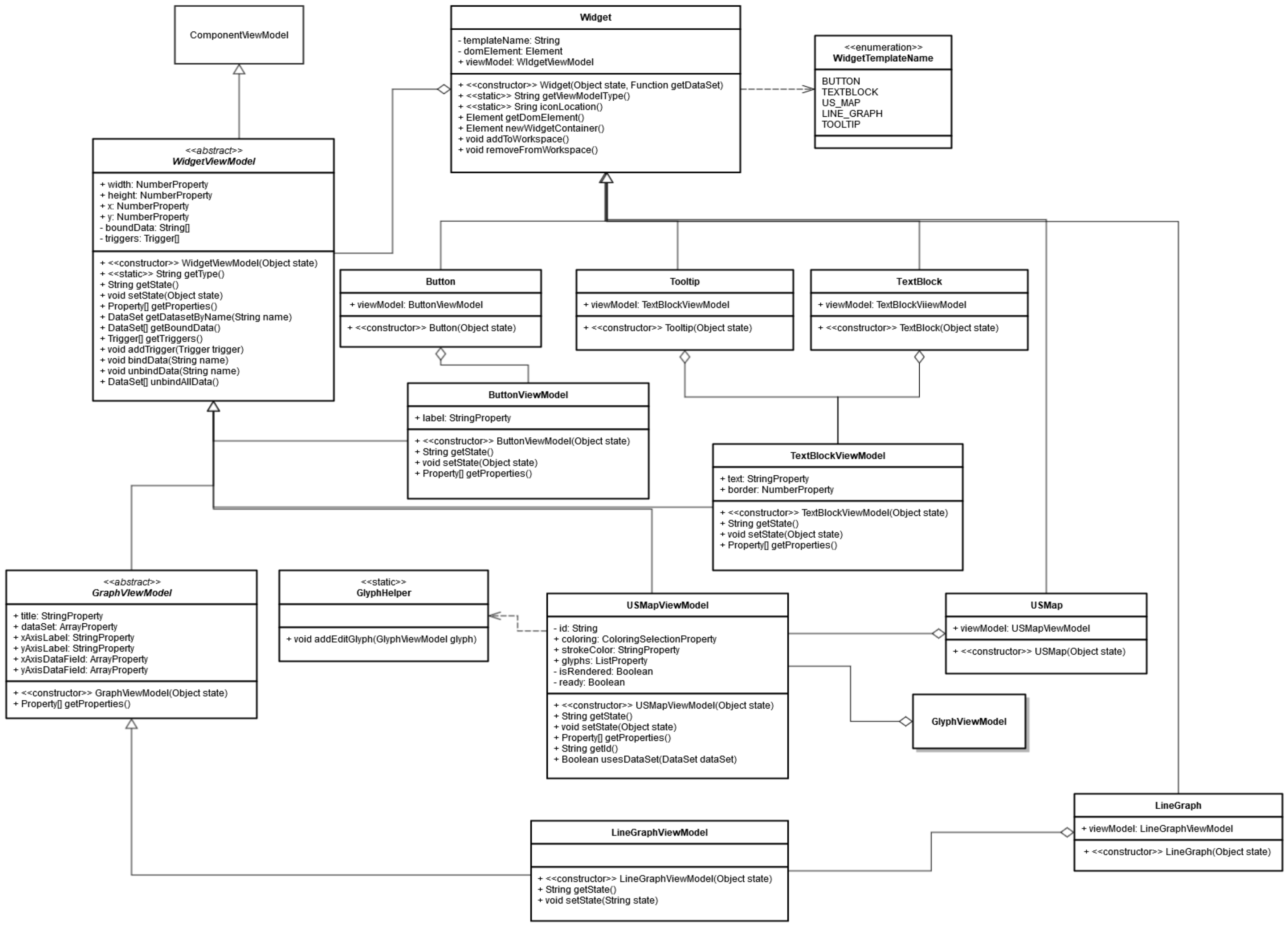

ComponentViewModel
<<abstract>>
WidgetViewModel
+ width: NumberProperty
+ height: NumberProperty
+ x: NumberProperty
+ y: NumberProperty
- boundData: String[]
- triggers: Trigger[]
+ <<constructor>> WidgetViewModel(Object state)
+ <<static>> String getType()
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()
+ DataSet getDatasetByName(String name)
+ DataSet[] getBoundData()
+ Trigger[] getTriggers()
+ void addTrigger(Trigger trigger)
+ void bindData(String name)
+ void unbindData(String name)
+ DataSet[] unbindAllData()
Widget
- templateName: String
- domElement: Element
+ viewModel: WidgetViewModel
+ <<constructor>> Widget(Object state, Function getDataSet)
+ <<static>> String getViewModelType()
+ <<static>> Sring iconLocation()
+ Element getDomElement()
+ Element newWidgetContainer()
+ void addToWorkspace()
+ void removeFromWorkspace()
<<enumeration>>
WidgetTemplateName
BUTTON
TEXTBLOCK
US_MAP
LINE_GRAPH
TOOLTIP
Button
+ viewModel: ButtonViewModel
+ <<constructor>> Button(Object state)
Tooltip
+ viewModel: TextBlockViewModel
+ <<constructor>> Tooltip(Object state)
TextBlock
+ viewModel: TextBlockViiewModel
+ <<constructor>> TextBlock(Object state)
ButtonViewModel
+ label: StringProperty
+ <<constructor>> ButtonViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()
TextBlockViewModel
+ text: StringProperty
+ border: NumberProperty
+ <<constructor>> TextBlockViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()
<<abstract>>
GraphViewModel
+ title: StringProperty
+ dataSet: ArrayProperty
+ xAxisLabel: StringProperty
+ yAxisLabel: StringProperty
+ xAxisDataField: ArrayProperty
+ yAxisDataField: ArrayProperty
+ <<constructor>> GraphViewModel(Object state)
+ Property[] getProperties()
<<static>>
GlyphHelper
+ void addEditGlyph(GlyphViewModel glyph)
USMapViewModel
- id: String
+ coloring: ColoringSelectionProperty
+ strokeColor: StringProperty
+ glyphs: ListProperty
- isRendered: Boolean
- ready: Boolean
+ <<constructor>> USMapViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()
+ String getId()
+ Boolean usesDataSet(DataSet dataSet)
USMap
+ viewModel: USMapViewModel
+ <<constructor>> USMap(Object state)
GlyphViewModel
LineGraph
+ viewModel: LineGraphViewModel
+ <<constructor>> LineGraph(Object state)
LineGraphViewModel
+ <<constructor>> LineGraphViewModel(Object state)
+ String getState()
+ void setState(String state)

Figure 11d. The widget classes

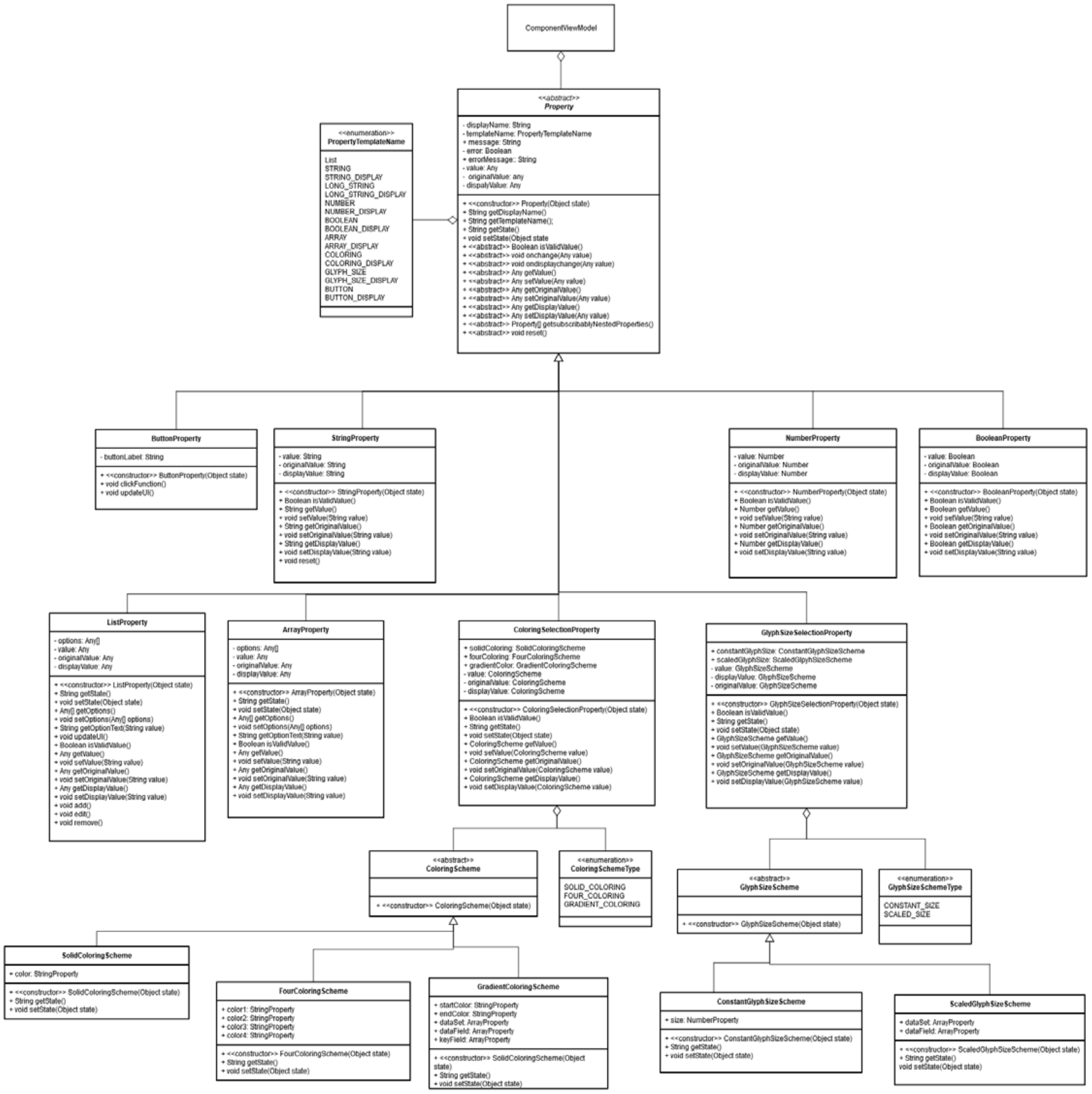

ComponentViewModel
<<abstract>>
Property
- displayName: String
- templateName: PropertyTemplateName
+ message: String
- error: Boolean
+ errorMessage:: String
- value: Any
- originalValue: any
- dispalyValue: Any
+ <<constructor>> Property(Object state)
+ String getDisplayName()
+ String getTemplateName();
+ String getState()
+ void setState(Object state
+ <<abstract>> Boolean isValidValue()
+ <<abstract>> void onchange(Any value)
+ <<abstract>> void ondisplaychange(Any value)
+ <<abstract>> Any getValue()
+ <<abstract>> Any setValue(Any value)
+ <<abstract>> Any getOriginalValue()
+ <<abstract>> Any setOriginalValue(Any value)
+ <<abstract>> Any getDisplayValue()
+ <<abstract>> Any setDisplayValue(Any value)
+ <<abstract>> Property[] getsubscribablyNestedProperties()
+ <<abstract>> void reset()
<<enumeration>>
PropertyTemplateName
List
STRING
STRING_DISPLAY
LONG_STRING
LONG_STRING_DISPLAY
NUMBER
NUMBER_DISPLAY
BOOLEAN
BOOLEAN_DISPLAY
ARRAY
ARRAY_DISPLAY
COLORING
COLORING_DISPLAY
GLYPH_SIZE
GLYPH_SIZE_DISPLAY
BUTTON
BUTTON_DISPLAY
ButtonProperty
- buttonLabel: String
+ <<constructor>> ButtonProperty(Object state)
+ void clickFunction()
+ void updateUI()
StringProperty
- value: String
- originalValue: String
- displayValue: String
+ <<constructor>> StringProperty(Object state)
+ Boolean isValidValue()
+ String getValue()
+ void setValue(String value)
+ String getOriginalValue()
+ void setOriginalValue(String value)
+ String getDisplayValue()
+ void setDisplayValue(String value)
+ void reset()
NumberProperty
- value: Number
- originalValue: Number
- displayValue: Number
+ <<constructor>> NumberProperty(Object state)
+ Boolean isValidValue()
+ Number getValue()
+ void setValue(String value)
+ Number getOriginalValue()
+ void setOriginalValue(String value)
+ Number getDisplayValue()
+ void setDisplayValue(String value)
BooleanProperty
- value: Boolean
- originalValue: Boolean
- displayValue: Boolean
+ <<constructor>> BooleanProperty(Object state)
+ Boolean isValidValue()
+ Boolean getValue()
+ void setValue(String value)
+ Boolean getOriginalValue()
+ void setOriginalValue(String value)
+ Boolean getDisplayValue()
+ void setDisplayValue(String value)
ListProperty
- options: Any[]
- value: Any
- originalValue: Any
- displayValue: Any
+ <<constructor>> ListProperty(Object state)
+ String getState()
+ void setState(Object state)
+ Any[] getOptions()
+ void setOptions(Any[] options)
+ String getOptionText(String value)
+ void updateUI()
+ Boolean isValidValue()
+ Any getValue()
+ void setValue(String value)
+ Any getOriginalValue()
+ void setOriginalValue(String value)
+ Any getDisplayValue()
+ void setDisplayValue(String value)
+ void add()
+ void edit()
+ void remove()
ArrayProperty
- options: Any[]
- value: Any
- originalValue: Any
- displayValue: Any
+ <<constructor>> ArrayProperty(Object state)
+ String getState()
+ void setState(Object state)
+ Any[] getOptions()
+ void setOptions(Any[] options)
+ String getOptionText(String value)
+ Boolean isValidValue()
+ Any getValue()
+ void setValue(String value)
+ Any getOriginalValue()
+ void setOriginalValue(String value)
+ Any getDisplayValue()
+ void setDisplayValue(String value)
ColoringSelectionProperty
+ solidColoring: SolidColoringScheme
+ fourColoring: FourColoringScheme
+ gradientColor: GradientColoringScheme
- value: ColoringScheme
- originalValue: ColoringScheme
- displayValue: ColoringScheme
+ <<constructor>> ColoringSelectionProperty(Object state)
+ Boolean isValidValue()
+ String getState()
+ void setState(Object state)
+ ColoringScheme getValue()
+ void setValue(ColoringScheme value)
+ ColoringScheme getOriginalValue()
+ void setOriginalValue(ColoringScheme value)
+ ColoringScheme getDisplayValue()
+ void setDisplayValue(ColoringScheme value)
GlyphSizeSelectionProperty
+ constantGlyphSize: ConstantGlyphSizeScheme
+ scaledGlyphSize: ScaledGlpyhSizeScheme
- value: GlyphSizeScheme
- displayValue: GlyphSizeScheme
- originalValue: GlyphSizeScheme
+ <<constructor>> GlyphSizeSelectionProperty(Object state)
+ Boolean isValidValue()
+ String getState()
+ void setState(Object state)
+ GlyphSizeScheme getValue()
+ void setValue(GlyphSizeScheme value)
+ GlyphSizeScheme getOriginalValue()
+ void setOriginalValue(GlyphSizeScheme value)
+ GlyphSizeScheme getDisplayValue()
+ void setDisplayValue(GlyphSizeScheme value)
<<abstract>>
ColoringScheme
+ <<constructor>> ColoringScheme(Object state)
<<enumeration>>
ColoringSchemeType
SOLID_COLORING
FOUR_COLORING
GRADIENT_COLORING
<<abstract>>
GlyphSizeScheme
+ <<constructor>> GlyphSizeScheme(Object state)
<<enumeration>>
GlyphSizeSchemeType
CONSTANT_SIZE
SCALED_SIZE
SolidColoringScheme
+ color: StringProperty
+ <<constructor>> SolidColoringScheme(Object state)
+ String getState()
+ void setState(Object state)
FourColoringScheme
+ color1: StringProperty
+ color2: StringProperty
+ color3: StringProperty
+ color4: StringProperty
+ <<constructor>> FourColoringScheme(Object state)
+ String getState()
+ void setState(Object state)
GradientColoringScheme
+ startColor: StringProperty
+ endColor: StringProperty
+ dataSet: ArrayProperty
+ dataField: ArrayProperty
+ keyField: ArrayProperty
+ <<constructor>> SolidColoringScheme(Object state)
+ String getState()
+ void setState(Object state)
ConstantGlyphSizeScheme
+ size: NumberProperty
+ <<constructor>> ConstantGlyphSizeScheme(Object state)
+ String getState()
+ void setState(Object state)
ScaledGlyphSizeScheme
+ dataSet: ArrayProperty
+ dataField: ArrayProperty
+ <<constructor>> ScaledGlyphSizeScheme(Object state)
+ String getState()
void setState(Object state)

Figure 11e. The property classes

4.2 Detailed Client-side Class Descriptions

4.2.1 JavaScript Primitive Data Types

The following data types are built in and do not require implementation:

- **String:** A character or string of characters
- **Number:** An integer or floating point number
- **Boolean:** A logical type (true or false)
- **Array:** An ordered list of elements
- **Element:** An HTML DOM Object
- **Object:** A dictionary with key-value pairs.

JavaScript is a dynamically typed language, so some class attributes can be any one of the types listed in 4.2.1. In this case, the attribute type is denoted as Any. If the attribute is an object that must be an instance of a particular class, the attribute type is listed as the name of the class.

4.2.2 Workspace

4.2.2.1 WAVED

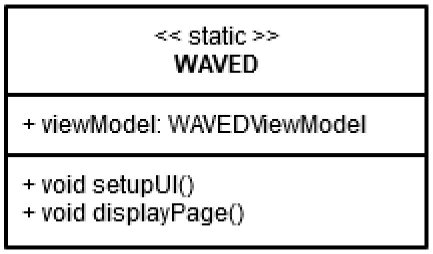


Figure 12. WAVED UML class description

WAVED is the top-level class that creates the view model for the application and initiates application procedures.

4.2.2.1.1 Attributes

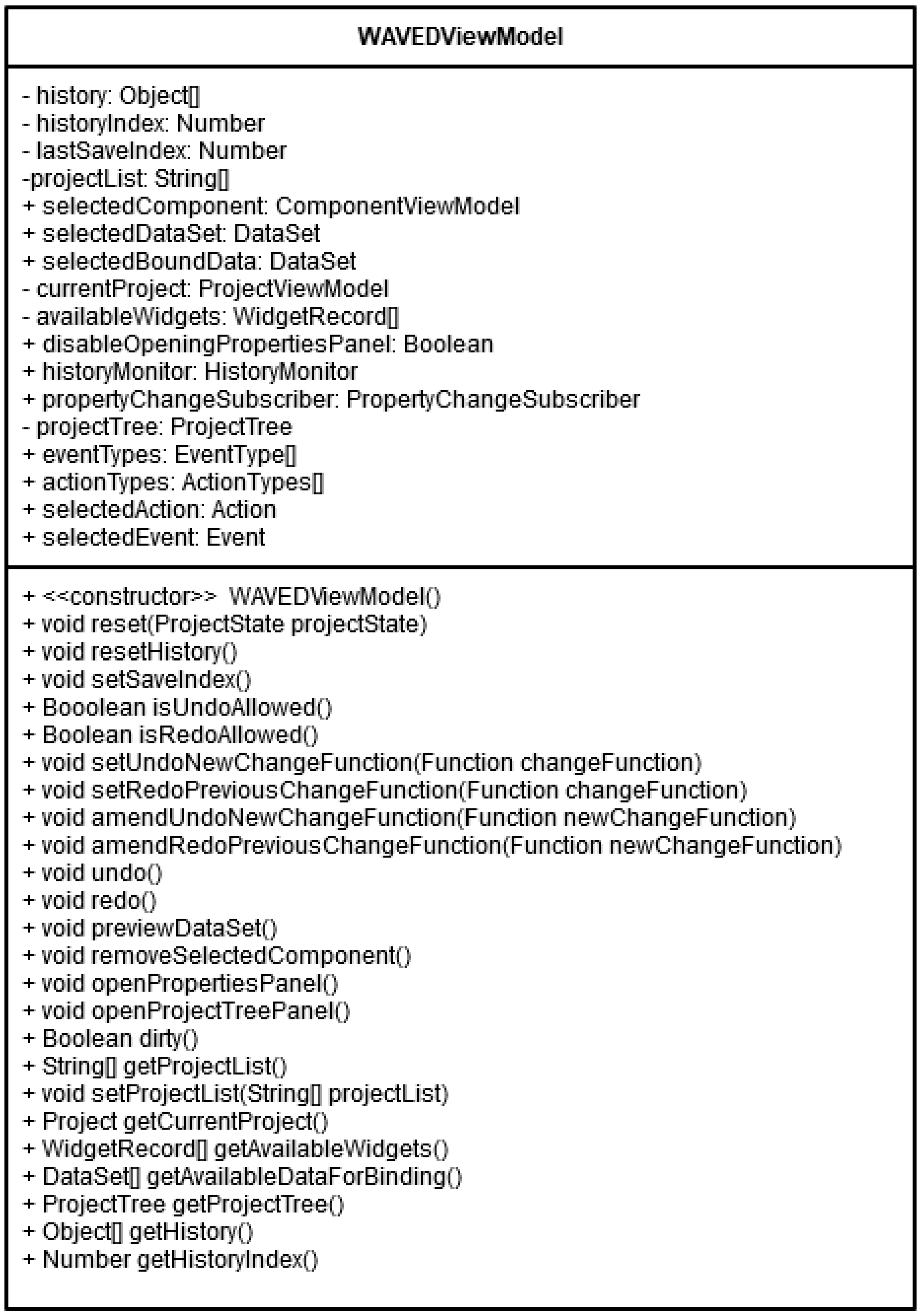
Name	Access	Type	Description
viewModel	public	WAVEDViewModel	The view model for the application

4.2.2.1.2 Methods

Name:	setupUI
Input:	void
Output:	void
Description:	Creates and styles user interface components

Name:	displayPage
Input:	void
Output:	void
Description:	Makes the user interface components visible

4.2.2.2 WAVEDViewModel



WAVEDViewModel

```
- history: Object[]
- historyIndex: Number
- lastSaveIndex: Number
-projectList: String[]
+ selectedComponent: ComponentViewModel
+ selectedDataSet: DataSet
+ selectedBoundData: DataSet
- currentProject: ProjectViewModel
- availableWidgets: WidgetRecord[]
+ disableOpeningPropertiesPanel: Boolean
+ historyMonitor: HistoryMonitor
+ propertyChangeSubscriber: PropertyChangeSubscriber
- projectTree: ProjectTree
+ eventTypes: EventType[]
+ actionTypes: ActionTypes[]
+ selectedAction: Action
+ selectedEvent: Event
```

```
+ <<constructor>>  WAVEDViewModel()
+ void reset(ProjectState projectState)
+ void resetHistory()
+ void setSaveIndex()
+ Booolean isUndoAllowed()
+ Boolean isRedoAllowed()
+ void setUndoNewChangeFunction(Function changeFunction)
+ void setRedoPreviousChangeFunction(Function changeFunction)
+ void amendUndoNewChangeFunction(Function newChangeFunction)
+ void amendRedoPreviousChangeFunction(Function newChangeFunction)
+ void undo()
+ void redo()
+ void previewDataSet()
+ void removeSelectedComponent()
+ void openPropertiesPanel()
+ void openProjectTreePanel()
+ Boolean dirty()
+ String[] getProjectList()
+ void setProjectList(String[] projectList)
+ Project getCurrentProject()
+ WidgetRecord[] getAvailableWidgets()
+ DataSet[] getAvailableDataForBinding()
+ ProjectTree getProjectTree()
+ Object[] getHistory()
+ Number getHistoryIndex()
```

Figure 13. WAVEDViewModel UML class description

WAVEDViewModel is the top-level class that maintains the data and operations used by the application’s user interface.

4.2.2.2.1 Attributes

Name	Access	Type	Description
history	private	Object[]	A collection of objections containing functions to undo and redo recent changes
historyIndex	private	Number	Indexes into history to note the current position
lastSaveIndex	private	Number	The history index at which the last save occurred
projectList	private	String[]	A list of the names of projects saved in the database
selectedComponent	public	ComponentViewModel	The widget that the user is interacting with
selectedDataSet	public	DataSet	The DataSet that is currently selected in the ProjectTree or Data Panel.
selectedBoundData	public	DataSet	The bound DataSet that is currently selected in the Properties Panel.
currentProject	private	ProjectViewModel	The project the user has open in the application
availableWidgets	private	WidgetRecord[]	A list of the types of widgets a user is able to add to the project
disableOpeningPropertiesPanel	public	Boolean	When true, the Properties Panel will not automatically open when a widget it added
historyMonitor	public	HistoryMonitor	An instance of HistoryMonitor
propertyChangeSubscriber	public	PropertyChangeSubscriber	An instance of PropertyChangeSubscriber
projectTree	private	ProjectTree	An instance of ProjectTree, which represents the Project Tree Panel
eventTypes	public	EventType[]	A list of available Event

			types (e.g. “click”)
actionTypes	public	ActionType[]	A list of available Action types (e.g. “Property Action”)
selectedAction	public	Action	The Action that is currently selected in the ProjectTree or Action Panel.
selectedEvent	public	Event	The Event that is currently selected in the ProjectTree or Event Panel.

4.2.2.2.2 Methods

Name:	<<constructor>> WAVEDViewModel
Input:	void
Output:	void
Description:	Sets the initial state of the view model attributes.

Name:	reset
Input:	projectState : the state of the current project
Output:	void
Description:	Resets the application and sets the current project to the given state.

Name:	resetHistory
Input:	void
Output:	void
Description:	Clears the history.

Name:	setSaveIndex
Input:	void
Output:	void
Description:	Sets the lastSaveIndex equal to the historyIndex.

Name:	isUndoAllowed
Input:	void
Output:	Boolean : True if the user is allowed to undo, false otherwise.
Description:	Returns true if the user is allowed to undo, otherwise returns false.

Name:	isRedoAllowed
Input:	void
Output:	Boolean : True if the user is allowed to redo.
Description:	Returns true if the user is allowed to redo, otherwise returns false.

Name:	setUndoNewChangeFunction
Input:	Function changeFunction : the undo function
Output:	void
Description:	Pushes a new undo change onto the history array.

Name:	setRedoPreviousChangeFunction
Input:	Function changeFunction : the redo function
Output:	void
Description:	Pushes a new redo change onto the history array.

Name:	amendUndoNewChangeFunction
Input:	Function newChangeFunction : the new undo function
Output:	void
Description:	Amends the last undo function so that it also calls newChangeFunction

Name:	amendRedoPreviousChangeFunction
Input:	Function : newChangeFunction : the new redo function
Output:	void
Description:	Amends the last redo function so that it also calls newChangeFunction

Name:	undo
Input:	void
Output:	void
Description:	If undo is enabled, executes the current undo function in history, and decrements historyIndex.

Name:	redo
Input:	void
Output:	void
Description:	If redo is enabled, executes the current redo function in history, and increments historyIndex.

Name:	previewDataSet
Input:	void
Output:	void
Description:	Displays fields from the selected data set.

Name:	removeSelectedComponent
Input:	void
Output:	void
Description:	Removes the selectedComponent from the currentProject.

Name:	openPropertiesPanel
Input:	void
Output:	void
Description:	Expands the properties panel.

Name:	openProjectTreePanel
Input:	void
Output:	void
Description:	Expands the project tree panel panel.

Name:	dirty
Input:	void
Output:	Boolean : True if there are unsaved changes, otherwise false.
Description:	Returns true if there are unsaved changes in the current project.

Name:	getProjectList
Input:	void
Output:	String[] : The value of the projectList attribute
Description:	Returns a list containing the names of all projects saved in the database.

Name:	setProjectList
Input:	String[] projectList : A list of project names saved in the database.
Output:	void
Description:	Sets the projectList attribute to the specified list of project names.

Name:	getCurrentProject
Input:	void
Output:	Project : The value of the currentProject attribute
Description:	Returns the currently open project.

Name:	getAvailableWidgets
Input:	void
Output:	WidgetRecord[] : The value of the availableWidget attribute
Description:	Returns a list of widgets available to the user to add to the project.

Name:	getAvailableDataForBinding
Input:	void
Output:	DataSet[] : A list of data sets.
Description:	Returns the list of data sets not bound to the selected widget.

Name:	getProjectTree
Input:	void
Output:	ProjectTree : The project tree.
Description:	Returns the project tree.

Name:	getHistory
Input:	void
Output:	Object[] : The value of the history attribute.
Description:	Returns the history for the current project.

Name:	getHistoryIndex
Input:	void
Output:	Number : The value of the historyIndex attribute.
Description:	Returns the project tree.

4.2.2.3 WidgetRecord

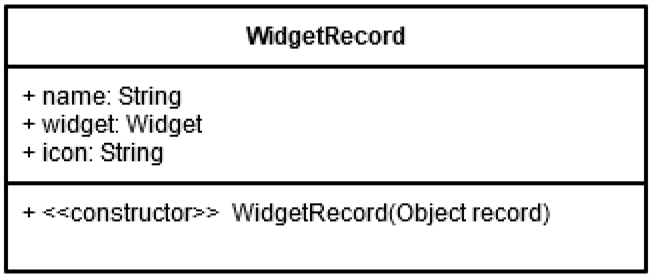


Figure 14. WidgetRecord UML class description

The WidgetRecord class represents a widget that is available to the user to add to the project.

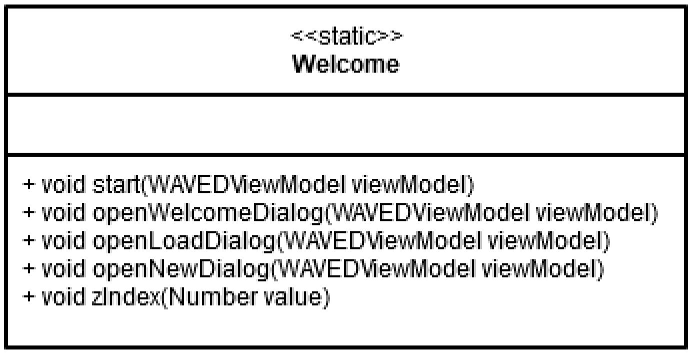
4.2.2.3.1 Attributes

Name	Access	Type	Description
name	public	String	The name of the widget
widget	public	Widget	The class responsible for creating the widget indicated by this WidgetRecord's name.
icon	public	String	The path to the widget icon seen in the user interface

4.2.2.3.2 Methods

Name:	<<constructor>> WidgetRecord
Input:	Object record : An object containing the display name and component object.
Output:	void
Description:	Sets the initial state of the record attributes

4.2.2.4 Welcome



<<static>>
Welcome
+ void start(WAVEDViewModel viewModel)
+ void openWelcomeDialog(WAVEDViewModel viewModel)
+ void openLoadDialog(WAVEDViewModel viewModel)
+ void openNewDialog(WAVEDViewModel viewModel)
+ void zIndex(Number value)

Figure 15. Welcome UML class description

The Welcome class contains helper functions for creating the application dialog displayed when the application first loads. The WAVED class references this module.

4.2.2.4.1 Attributes

Name	Access	Type	Description

4.2.2.4.2 Methods

Name:	start
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Starts the application and opens the welcome dialog.

Name:	openWelcomeDialog
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Creates a dialog for the user to decide to create a new project or load an existing project. Handles interaction with the dialog

Name:	openLoadDialog
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Opens the load project dialog.

Name:	openNewDialog
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Opens the new project dialog.

Name:	zIndex
Input:	Number value : the zIndex for the welcome dialog
Output:	void
Description:	Moves the welcome dialog forward or backward to the given z index value.

4.2.2.5 NewProject

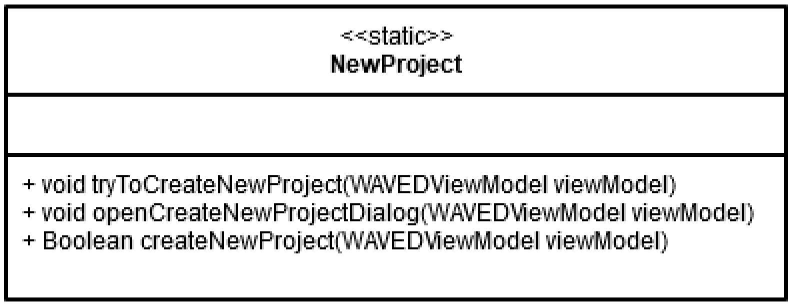

<<static>>
NewProject
+ void tryToCreateNewProject(WAVEDViewModel viewModel)
+ void openCreateNewProjectDialog(WAVEDViewModel viewModel)
+ Boolean createNewProject(WAVEDViewModel viewModel)

Figure 16. NewProject UML class description

The NewProject class contains helper functions for creating the dialog and handling the creation of a new project. The WAVEDViewModel class references this module.

4.2.2.5.1 Attributes

Name	Access	Type	Description

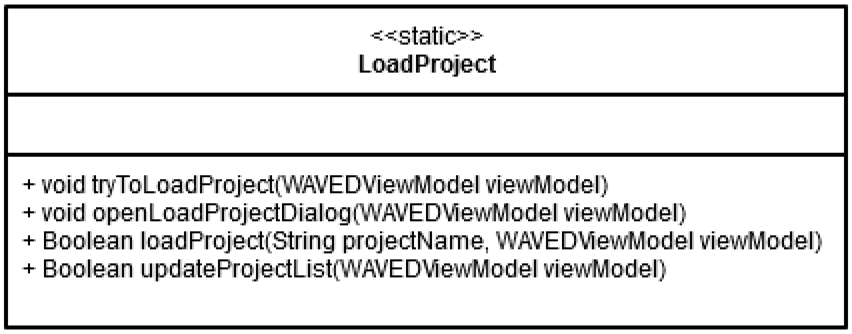
4.2.2.5.2 Methods

Name:	tryToCreateNewProject
Input:	WAVEDViewModel viewModel: The application view model
Output:	void
Description:	Checks for unsaved changes, then initiates the user interface for creating a new project

Name:	openCreateNewProjectDialog
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Creates a dialog for the user to input a name for the new project and handles interaction with the dialog

Name:	createNewProject
Input:	WAVEDViewModel viewModel : The application view model
Output:	Boolean : True if the project was successfully created
Description:	Adds an entry in the database for the new project name. Sets the currentProject on the application view model.

4.2.2.6 LoadProject



<<static>>
LoadProject
+ void tryToLoadProject(WAVEDViewModel viewModel)
+ void openLoadProjectDialog(WAVEDViewModel viewModel)
+ Boolean loadProject(String projectName, WAVEDViewModel viewModel)
+ Boolean updateProjectList(WAVEDViewModel viewModel)

Figure 17. LoadProject UML class description

The LoadProject class contains helper functions for creating the dialog and handling the project loading. The WAVEDViewModel class references this module.

4.2.2.6.1 Attributes

Name	Access	Type	Description

4.2.2.6.2 Methods

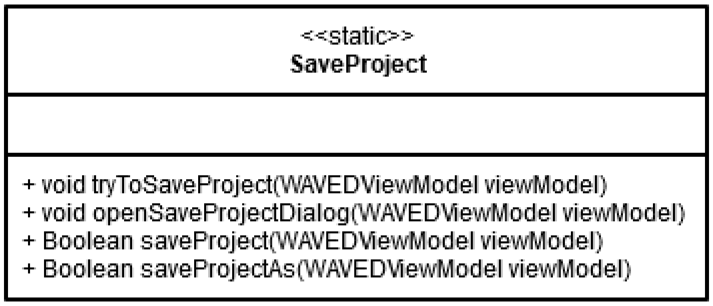
Name:	tryToLoadProject
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Checks for unsaved changes, then initiates the user interface for loading a saved project.

Name:	openLoadProjectDialog
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Creates a dialog for the user to select the project to load and handles interaction with the dialog

Name:	loadProject
Input:	String projectName : The name of the project to load WAVEDViewModel viewModel : The application view model
Output:	Boolean : True if the project was successfully created
Description:	Retrieves project data from the database. Sets the currentProject on the application view model to the loaded project.

Name:	updateProjectList
Input:	WAVEDViewModel viewModel : The application view model
Output:	Boolean : True if the project list was successfully loaded.
Description:	Retrieves a list of project names data from the database and sets the project list in the load project dialog.

4.2.2.7 SaveProject



<<static>>
SaveProject
+ void tryToSaveProject(WAVEDViewModel viewModel)
+ void openSaveProjectDialog(WAVEDViewModel viewModel)
+ Boolean saveProject(WAVEDViewModel viewModel)
+ Boolean saveProjectAs(WAVEDViewModel viewModel)

Figure 18. SaveProject UML class description

The SaveProject class contains helper functions for creating the save dialog and handling the project saving. The WAVEDViewModel class references this module.

4.2.2.7.1 Attributes

Name	Access	Type	Description

4.2.2.7.2 Methods

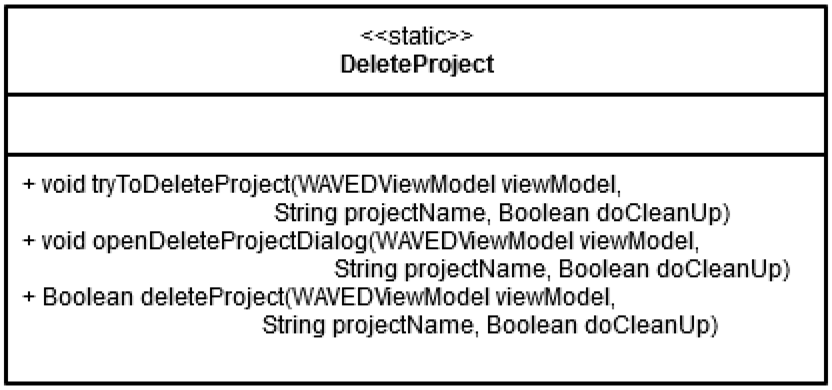
Name:	tryToSaveProject
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Initiates the user interface for saving a project.

Name:	openSaveProjectDialog
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Creates a dialog for the user to enter a name to save the project as and handles interaction with the dialog

Name:	saveProject
Input:	WAVEDViewModel viewModel : The application view model
Output:	Boolean : True if the project was successfully saved
Description:	Saves the project in the database under the current name.

Name:	saveProjectAs
Input:	WAVEDViewModel viewModel : The application view model
Output:	Boolean : True if the project was successfully saved
Description:	Saves the project in the database under a new name.

4.2.2.8 DeleteProject



```
+ void tryToDeleteProject(WAVEDViewModel viewModel,
                          String projectName, Boolean doCleanUp)
+ void openDeleteProjectDialog(WAVEDViewModel viewModel,
                               String projectName, Boolean doCleanUp)
+ Boolean deleteProject(WAVEDViewModel viewModel,
                        String projectName, Boolean doCleanUp)
```

Figure 19. DeleteProject UML class description

The DeleteProject class contains helper functions for creating the Delete Project dialog and handling project deletion. The WAVEDViewModel class references this module.

4.2.2.8.1 Attributes

Name	Access	Type	Description

4.2.2.8.2 Methods

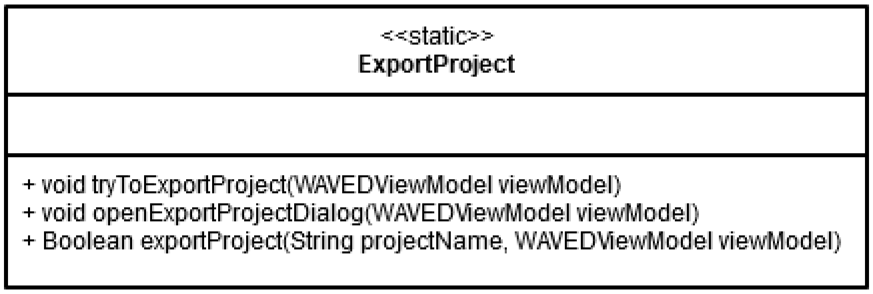
Name:	tryToDeleteProject
Input:	WAVEDViewModel viewModel : The application view model String projectName : The name of the project to attempt to delete Boolean doCleanUp: Flag to reset application on project deletion
Output:	void
Description:	Initiates the user interface for deleting a project.

Name:	openDeleteProjectDialog
Input:	WAVEDViewModel viewModel : The application view model String projectName : The name of the project to attempt to delete

	Boolean doCleanUp: Flag to reset application on project deletion
Output:	void
Description:	Creates a dialog for the user to confirm deletion and handles interaction with the dialog.

Name:	deleteProject
Input:	WAVEDViewModel viewModel : The application view model String projectName : The name of the project to delete Boolean doCleanUp: Flag to reset application on project deletion
Output:	Boolean : True if the project was successfully saved
Description:	Deletes the project in the database and filesystem. Resets the application and brings up the welcome screen if specified.

4.2.2.9 ExportProject



<<static>>
ExportProject
+ void tryToExportProject(WAVEDViewModel viewModel)
+ void openExportProjectDialog(WAVEDViewModel viewModel)
+ Boolean exportProject(String projectName, WAVEDViewModel viewModel)

Figure 20. ExportProject UML class description

The ExportProject class contains helper functions for creating the Export dialog and handling project exporting. The WAVEDViewModel class references this module.

4.2.2.9.1 Attributes

Name	Access	Type	Description

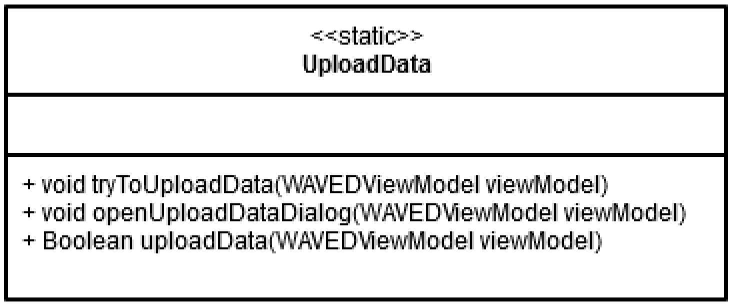
4.2.2.9.2 Methods

Name:	tryToExportProject
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Initiates the user interface for exporting a project.

Name:	openExportProjectDialog
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Creates a dialog for the user pick an export location and handles interaction with the dialog.

Name:	exportProject
Input:	WAVEDViewModel viewModel : The application view model
Output:	Boolean : True if the project was successfully saved
Description:	Zips and exports the project to the filesystem.

4.2.2.10 UploadData



<<static>>
UploadData
+ void tryToUploadData(WAVEDViewModel viewModel)
+ void openUploadDataDialog(WAVEDViewModel viewModel)
+ Boolean uploadData(WAVEDViewModel viewModel)

Figure 21. UploadData UML class description

The LoadData class contains helper functions for uploading a data source to the project. The WAVEDViewModel class references this module.

4.2.2.10.1 Attributes

Name	Access	Type	Description

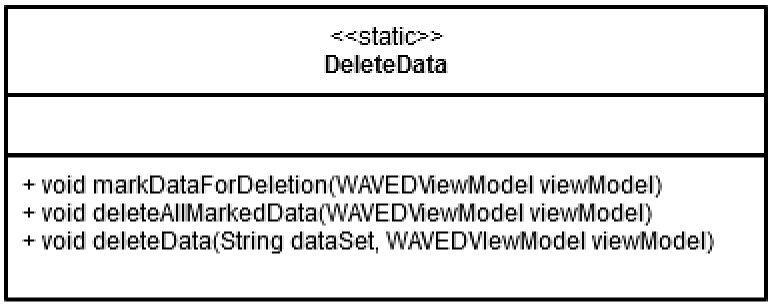
4.2.2.10.2 Methods

Name:	tryToUploadData
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Initiates the user interface for uploading a data source.

Name:	openUploadDataDialog
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Creates a dialog for the user to select a data source to upload and handles interaction with the dialog

Name:	uploadData
Input:	WAVEDViewModel viewModel : The application view model
Output:	Boolean : True if the data was successfully uploaded.
Description:	Adds the uploaded data set to the project.

4.2.2.11 DeleteData



<<static>>
DeleteData
+ void markDataForDeletion(WAVEDViewModel viewModel)
+ void deleteAllMarkedData(WAVEDViewModel viewModel)
+ void deleteData(String dataSet, WAVEDVIewModel viewModel)

Figure 22. DeleteData UML class description

The DeleteData class contains helper functions for creating the dialog and handling the deletion of data. The WAVEDViewModel class references this module.

4.2.2.11.1 Attributes

Name	Access	Type	Description

4.2.2.11.2 Methods

Name:	markDataForDeletion
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Puts a flag on data that needs to be deleted later.

Name:	deleteAllMarkedData
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Deletes all data that have been marked for deletion.

Name:	deleteData
Input:	String dataSet : The data set to be deleted WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Deletes the specified data set from the server.

4.2.2.12 BindData

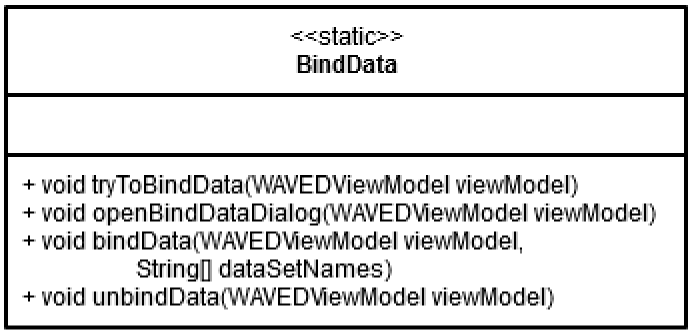

<<static>>
BindData
+ void tryToBindData(WAVEDViewModel viewModel)
+ void openBindDataDialog(WAVEDViewModel viewModel)
+ void bindData(WAVEDViewModel viewModel, String[] dataSetNames)
+ void unbindData(WAVEDViewModel viewModel)

Figure 23. BindData UML class description

The BindData class contains helper functions for creating the dialog and handling the binding of data to a widget. The WAVEDViewModel class references this module.

4.2.2.12.1 Attributes

Name	**Access**	**Type**	**Description**

4.2.2.12.2 Methods

Name:	tryToBindData
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Initiates the user interface for binding data to a widget.

Name:	openBindDataDialog
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Creates a dialog for the user to bind data to a widget and handles interaction with the dialog

Name:	bindData
Input:	WAVEDViewModel viewModel : The application view model String[] dataSetNames : The names of the datasets to bind
Output:	void
Description:	Binds the datasets to the selected widget

Name:	unbindData
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Unbinds the selected datasets from the selected widget

4.2.2.13 ReadData

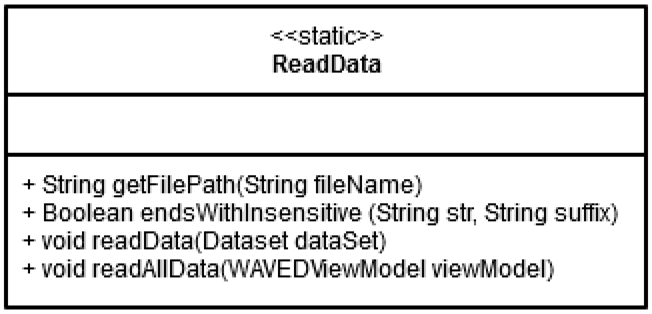

<<static>>
ReadData
+ String getFilePath(String fileName)
+ Boolean endsWithInsensitive (String str, String suffix)
+ void readData(Dataset dataSet)
+ void readAllData(WAVEDViewModel viewModel)

Figure 24. ReadData UML class description

The ReadData class contains helper functions for reading data files. The WAVEDViewModel class references this module.

4.2.2.13.1 Attributes

Name	**Access**	**Type**	**Description**
dataFolderPath	public	String	The path to the data folder

4.2.2.13.2 Methods

Name:	getFilePath
Input:	String filename : the name of the file
Output:	String : the path, based on dataFolderPath and filename
Description:	Returns the path to the given filename.

Name:	endsWithInsensitive
Input:	String str : the string to check String suffix : the suffix to check
Output:	Boolean : True if str ends with suffix, case insensitively, otherwise False
Description:	Checks if str ends with suffix.

Name:	readData
Input:	DataSet dataSet : The DataSet to read

Output:	void
Description:	Reads the contents of dataSet's file.

Name:	readAllData
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Reads the contents of all DataSet files in the current project.

4.2.2.14 EventHelper

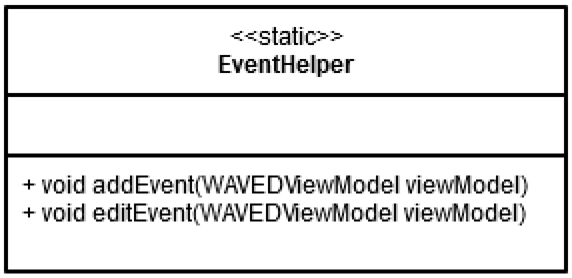

<<static>>
EventHelper
+ void addEvent(WAVEDViewModel viewModel)
+ void editEvent(WAVEDViewModel viewModel)

Figure 25. EventHelper UML class description

The EventHelper class contains helper functions for creating and editing events. The WAVEDViewModel class references this module.

4.2.2.14.1 Attributes

Name	Access	Type	Description

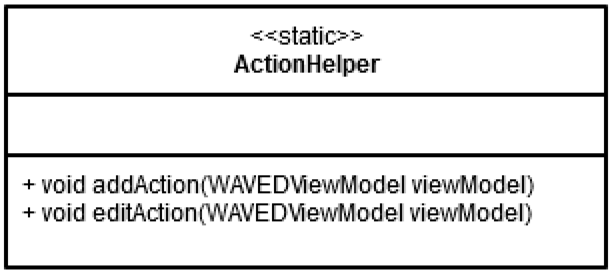
4.2.2.14.2 Methods

Name:	addEvent
Input:	WAVEDViewModel : The application view model
Output:	void
Description:	Initializes the event dialog, and adds an event to the project

Name:	editEvent
Input:	WAVEDViewModel viewModel : The application view model

Output:	void
Description:	Initializes the event dialog, and edits an event

4.2.2.15 ActionHelper



<<static>>
ActionHelper
+ void addAction(WAVEDViewModel viewModel)
+ void editAction(WAVEDViewModel viewModel)

Figure 26. ActionHelper UML class description

The ActionHelper class contains helper functions for creating and editing action. The WAVEDViewModel class references this module.

4.2.2.15.1 Attributes

Name	Access	Type	Description

4.2.2.15.2 Methods

Name:	addAction
Input:	WAVEDViewModel viewModel : The application view model
Output:	void
Description:	Initializes the dialog and adds an action to the project

Name:	editAction
Input:	WAVEDViewModel viewModel
Output:	void
Description:	Initializes the dialog to edit an existing action

4.2.2.16 DependencyChecker

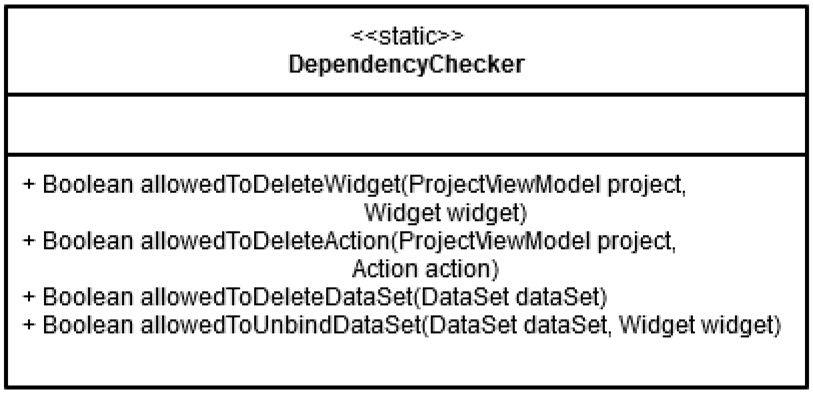

<<static>>
DependencyChecker
+ Boolean allowedToDeleteWidget(ProjectViewModel project, Widget widget)
+ Boolean allowedToDeleteAction(ProjectViewModel project, Action action)
+ Boolean allowedToDeleteDataSet(DataSet dataSet)
+ Boolean allowedToUnbindDataSet(DataSet dataSet, Widget widget)

Figure 27. DependencyChecker UML class description

The DependencyChecker class contains helper functions used to determine if widgets, datasets, actions, and events can safely be deleted. The WAVEDViewModel class references this module.

4.2.2.16.1 Attributes

Name	Access	Type	Description

4.2.2.16.2 Methods

Name:	allowedToDeleteWidget
Input:	Widget widget : the Widget to delete ProjectViewModel project : the current project
Output:	Boolean : True, if widget can safely be deleted; otherwise, False.
Description:	Checks if widget is used by any Actions or Events.

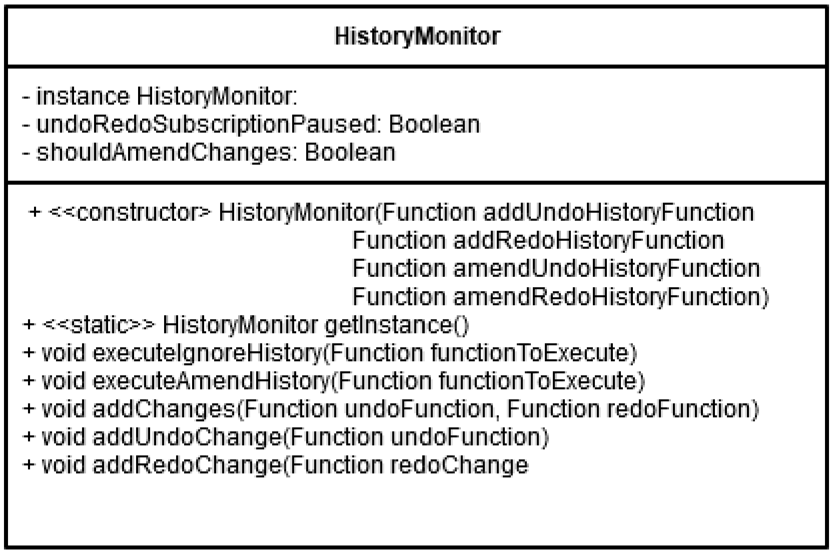
Name:	allowedToDeleteAction
Input:	Action action : the Action to delete ProjectViewModel project : the current project
Output:	Boolean : True, if action can safely be deleted; otherwise, False.
Description:	Checks if action is used by any Events.

Name:	allowedToDeleteDataSet

Input:	DataSet dataSet : the DataSet to delete
Output:	Boolean : True, if dataSet can safely be deleted; otherwise, False.
Description:	Checks if dataSet is bound to a widget.

Name:	allowedToUnbindDataSet
Input:	DataSet dataSet : the dataSet to unbind Widget widget : the Widget to unbind from
Output:	Boolean : True, if dataSet can safely be unbound from widget; otherwise, False.
Description:	Checks if dataSet is used by widget.

4.2.2.17 HistoryMonitor



```
HistoryMonitor

- instance HistoryMonitor:
- undoRedoSubscriptionPaused: Boolean
- shouldAmendChanges: Boolean

 + <<constructor> HistoryMonitor(Function addUndoHistoryFunction
                                 Function addRedoHistoryFunction
                                 Function amendUndoHistoryFunction
                                 Function amendRedoHistoryFunction)
+ <<static>> HistoryMonitor getInstance()
+ void executeIgnoreHistory(Function functionToExecute)
+ void executeAmendHistory(Function functionToExecute)
+ void addChanges(Function undoFunction, Function redoFunction)
+ void addUndoChange(Function undoFunction)
+ void addRedoChange(Function redoChange
```

Figure 28. HistoryMonitor UML class description

The HistoryMonitor singleton class contains helper functions used to add to history. These functions execute the functions in WAVEDViewModel. All additions to history must be routed through this class so that WAVEDViewModel is not used directly.

4.2.2.17.1 Attributes

Name	Access	Type	Description
instance	private	HistoryMonitor	Private instance of HistoryMonitor
undoRedoSubscriptionPaused	private	Boolean	When true, don't add to history
shouldAmendChanges	private	Boolean	When true, amend instead of add to history

4.2.2.17.2 Methods

Name:	<<constructor>> HistoryMonitor
Input:	Function addUndoHistoryFunction : reference to function from WAVEDViewModel Function addRedoHistoryFunction : reference to function from WAVEDViewModel Function amendUndoHistoryFunction : reference to function from WAVEDViewModel Function amendRedoHistoryFunction : reference to function from WAVEDViewModel
Output:	void
Description:	The constructor must be called once to pass in the required functions. After this point, getInstance should be used.

Name:	<<static>> getInstance
Input:	void
Output:	HistoryMonitor : current instance
Description:	Returns instance of HistoryMonitor

Name:	executeIgnoreHistory
Input:	Function functionToExecute : the function to execute
Output:	void
Description:	Executes functionToExecute, while suspending history by setting undoRedoSubscriptionPaused to true.

Name:	executeAmendHistory
Input:	Function functionToExecute : the function to execute
Output:	void
Description:	Executes functionToExecute, while amending instead of adding history by setting shouldAmendChanges to true.

Name:	addChanges
Input:	Function undoFunction: the function to add to undo history Function redoFunction: the function to add to redo history
Output:	void
Description:	Adds or amends undoFunction and redoFunction to history as specified by shouldAmendChanges.

Name:	addUndoChange
Input:	Function undoFunction: the function to add to undo history
Output:	void
Description:	If undoRedoSubscriptionPaused is false, adds or amends undoFunction to history as specified by shouldAmendChanges.

Name:	addRedoChange
Input:	Function redoFunction: the function to add to redo history
Output:	void
Description:	If undoRedoSubscriptionPaused is false, adds or amends redoFunction to history as specified by shouldAmendChanges.

4.2.2.18 PropertyChangeSubscriber

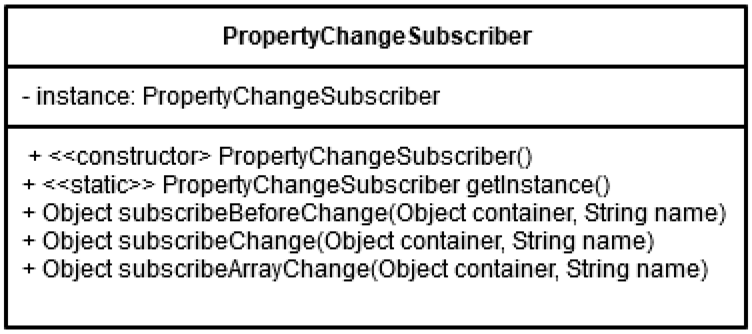

PropertyChangeSubscriber
- instance: PropertyChangeSubscriber
+ <<constructor> PropertyChangeSubscriber()
+ <<static>> PropertyChangeSubscriber getInstance()
+ Object subscribeBeforeChange(Object container, String name)
+ Object subscribeChange(Object container, String name)
+ Object subscribeArrayChange(Object container, String name)

Figure 29. PropertyChangeSubscriber UML class description

The PropertyChangeSubscriber singleton class contains helper functions used to add Property changes to history.

4.2.2.18.1 Attributes

Name	Access	Type	Description
instance	private	PropertyChangeSubscriber	Instance of PropertyChangeSubscriber

4.2.2.18.2 Methods

Name:	<<constructor>> PropertyChangeSubscriber
Input:	void
Output:	PropertyChangeSubscriber : instance of PropertyChangeSubscriber
Description:	Returns instance if defined, or creates a new instance.

Name:	<<static>> getInstance
Input:	void
Output:	PropertyChangeSubscriber : current instance
Description:	Returns instance of PropertyChangeSubscriber.

Name:	subscribeBeforeChange
Input:	Object container : the object that contains name

	String name : The name of the attribute in container
Output:	Object : A representation of the subscription
Description:	Subscribes to the beforeChange event, adding the undo change to the history.

Name:	subscribeChange
Input:	Object container : the object that contains name String name : The name of the attribute in container
Output:	Object : A representation of the subscription
Description:	Subscribes to the change event, adding the redo change to the history.

Name:	subscribeArrayChange
Input:	Object container : the object that contains name String name : The name of the attribute in container
Output:	Object : A representation of the subscription
Description:	Subscribes to the arrayChange event, adding the undo and redo changes to the history.

4.2.2.19 UniqueTracker

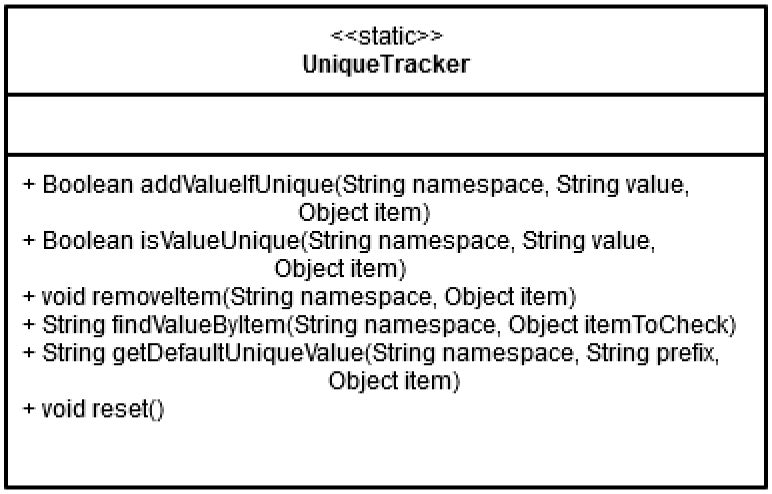

<<static>>
UniqueTracker
+ Boolean addValueIfUnique(String namespace, String value, Object item)
+ Boolean isValueUnique(String namespace, String value, Object item)
+ void removeItem(String namespace, Object item)
+ String findValueByItem(String namespace, Object itemToCheck)
+ String getDefaultUniqueValue(String namespace, String prefix, Object item)
+ void reset()

Figure 30. UniqueTracker UML class description

The UniqueTracker class contains helper functions that keeps track of properties that must have a unique value.

4.2.2.19.1 Attributes

Name	Access	Type	Description

4.2.2.19.2 Methods

Name:	addValueIfUnique
Input:	String namespace : the namespace to which to add the unique value String value : the unique value Object item : the item that contains value
Output:	Boolean : True, if the value is unique; otherwise False
Description:	Adds the unique value to namespaceValueItemMap if it is unique within the given namespace.

Name:	isValueUnique
Input:	String namespace : the namespace in which to check if value is unique String value : the value to check Object item : the item that contains value
Output:	Boolean : True, if the value is unique; otherwise False
Description:	Checks if value is unique within the given namespace.

Name:	removeItem
Input:	String namespace : the namespace that contains item Object item : the item to remove
Output:	void
Description:	Removes item from namespace.

Name:	findValueByItem
Input:	String namespace : the namespace in which to look for itemToCheck Object itemToCheck : the item look for in namespace
Output:	String : the value of itemToCheck if it is contained within namespace; otherwise returns null
Description:	Retrieves the value of itemToCheck within namespace.

Name:	getDefaultUniqueValue
Input:	String namespace : the namespace used to ensure that the retured value is unique String prefix : the prefix of the unique value Object item : the item that will be given the unique value
Output:	String : unique value based on the given prefix
Description:	Checks if the String prefix + prefixCounter[prefix] is unique. If not, prefixCounter[prefix] is incremented until a unique value is found and returned.

Name:	reset
Input:	void
Output:	void
Description:	Resets namespaceValueItemMap and prefixCounter.

4.2.2.20 UnsavedChanges

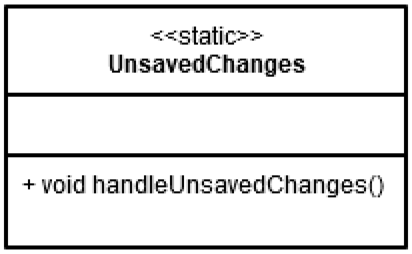


Figure 31. UnsavedChanges UML class description

The UnsavedChanges class contains helper functions that handle unsaved changes. The NewProject and LoadProject modules reference this class.

4.2.2.20.1 Attributes

Name	Access	Type	Description

4.2.2.20.2 Methods

Name:	handleUnsavedChanges
Input:	void
Output:	void
Description:	Handles when the user tries to create or load a project when there are unsaved changes.

4.2.2.21 GoogleAnalytics

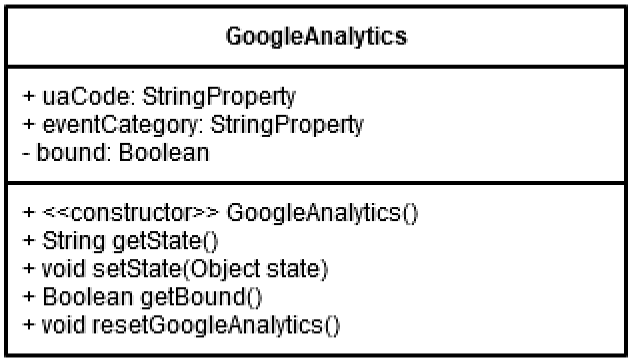

GoogleAnalytics
+ uaCode: StringProperty
+ eventCategory: StringProperty
- bound: Boolean
+ <<constructor>> GoogleAnalytics()
+ String getState()
+ void setState(Object state)
+ Boolean getBound()
+ void resetGoogleAnalytics()

Figure 32. GoogleAnalytics UML class description

The GoogleAnalytics class stores information needed by the Google Analytics logging system.

4.2.2.21.1 Attributes

Name	**Access**	**Type**	**Description**
uaCode	public	StringProperty	The GoogleAnalytics UA code
eventCategory	public	StringPropety	The event category
bound	private	Boolean	True of the attributes are submiitted

4.2.2.21.2 Methods

Name:	<<constructor>> GoogleAnalytics
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the Google Analytics attributes.

Name:	getState
Input:	void
Output:	String : A string representing the google analytics
Description:	Returns a JSON stringified representation of the state of the google analytics and its attributes

Name:	setState
Input:	Object state : The new state for the google analytics
Output:	void
Description:	Sets the attributes of the google analtyics based on the given state

Name:	getBound
Input:	void
Output:	Boolean : True if it is bound
Description:	Returns true if the google analytics is bound, otherwise false.

Name:	resetGoogleAnalytics
Input:	void
Output:	void
Description:	Clears the values of the attributes.

4.2.2.22 ProjectTree

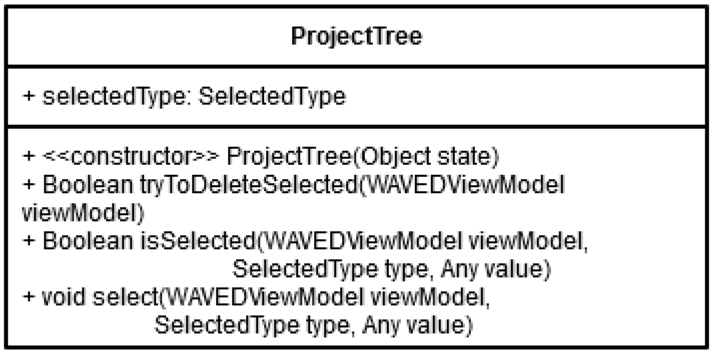

ProjectTree
+ selectedType: SelectedType
+ <<constructor>> ProjectTree(Object state)
+ Boolean tryToDeleteSelected(WAVEDViewModel viewModel)
+ Boolean isSelected(WAVEDViewModel viewModel, SelectedType type, Any value)
+ void select(WAVEDViewModel viewModel, SelectedType type, Any value)

Figure 33. ProjectTree UML class description

The ProjectTree class defines the view of the project tree.

4.2.2.22.1 Attributes

Name	Access	Type	Description
selectedType	public	SelectedType	The type of the selected item in the tree

4.2.2.22.2 Methods

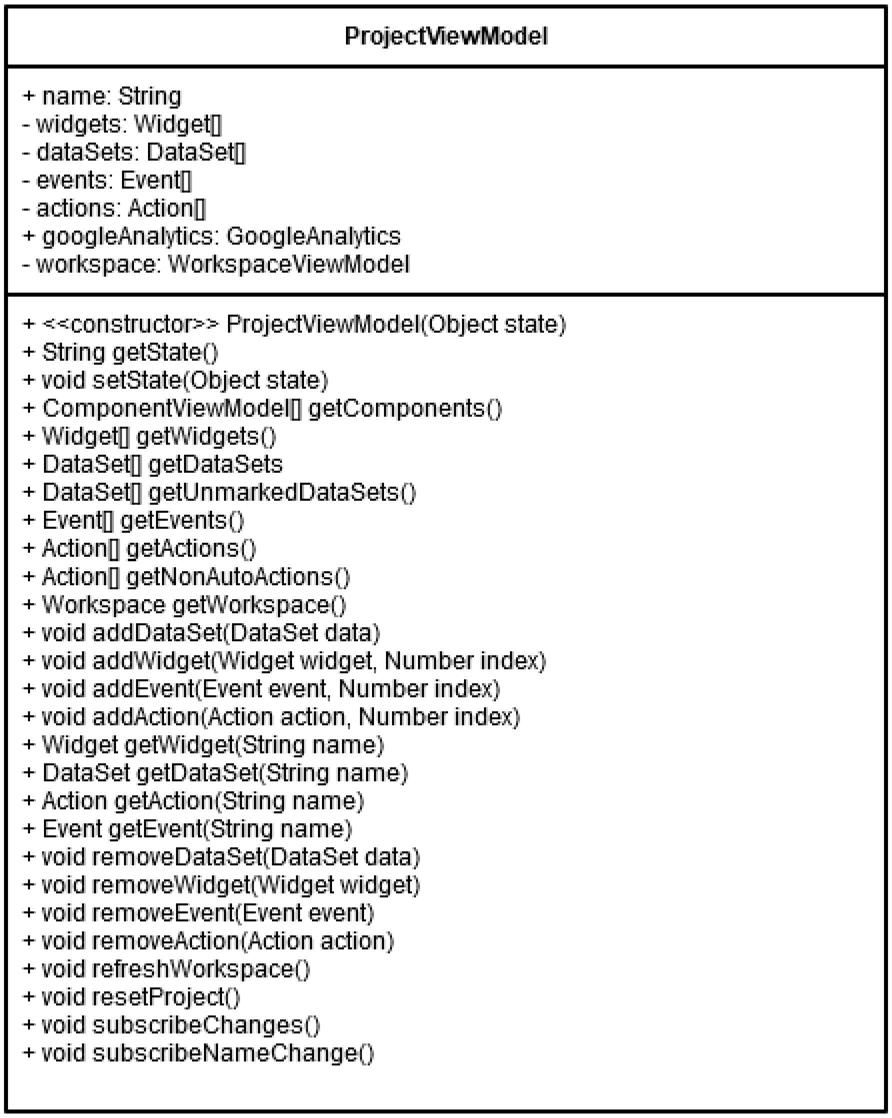
Name:	<<constructor>> ProjectTree
Input:	Object state : The state of the attributes
Output:	void
Description:	Creates the ProjectTree DOM elements

Name:	tryToDeleteSelected
Input:	WAVEDViewModel viewModel : the application view model
Output:	Boolean : True if delete was successful
Description:	Deletes the selected item in the tree

Name:	isSelected
Input:	WAVEDViewmodel viewModel : the application view model SelectedType type : the selected type Any value : the value of the selection
Output:	Boolean : True if the value is selected
Description:	Returns true if the value is selected, otherwise false

Name:	select
Input:	WAVEDVieModel viewModel : the application view model SelectedType type : the selected type Any value : the value to select
Output:	void
Description:	Selects the given value

4.2.2.23 ProjectViewModel



ProjectViewModel

```
+ name: String
- widgets: Widget[]
- dataSets: DataSet[]
- events: Event[]
- actions: Action[]
+ googleAnalytics: GoogleAnalytics
- workspace: WorkspaceViewModel
```

```
+ <<constructor>> ProjectViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ ComponentViewModel[] getComponents()
+ Widget[] getWidgets()
+ DataSet[] getDataSets
+ DataSet[] getUnmarkedDataSets()
+ Event[] getEvents()
+ Action[] getActions()
+ Action[] getNonAutoActions()
+ Workspace getWorkspace()
+ void addDataSet(DataSet data)
+ void addWidget(Widget widget, Number index)
+ void addEvent(Event event, Number index)
+ void addAction(Action action, Number index)
+ Widget getWidget(String name)
+ DataSet getDataSet(String name)
+ Action getAction(String name)
+ Event getEvent(String name)
+ void removeDataSet(DataSet data)
+ void removeWidget(Widget widget)
+ void removeEvent(Event event)
+ void removeAction(Action action)
+ void refreshWorkspace()
+ void resetProject()
+ void subscribeChanges()
+ void subscribeNameChange()
```

Figure 34. ProjectViewModel UML class description

The ProjectViewModel class is a view model that contains data related to a project and methods for making changes to the project.

4.2.2.23.1 Attributes

Name	**Access**	**Type**	**Description**
name	public	String	The name of the project
widgets	private	Widget[]	A collection of widgts added to the project
dataSets	private	DataSet[]	A collection of data sources and data subsets added to the project
events	private	Event[]	A collection of events added to the project
actions	private	Action[]	A collection of actions added to the project
googleAnalytics	public	GoogleAnalytics	The Google Analytics tracking codes for the project
workspace	private	WorkspaceViewModel	The project workspace

4.2.2.23.2 Methods

Name:	<<constructor>> ProjectViewModel
Input:	Object state : The state of the project view model attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the project
Description:	Returns a JSON stringified representation of the state of the project and its attributes

Name:	setState
Input:	Object state : The new state for the project
Output:	void
Description:	Sets the attributes of the project based on the given state JSON representation

Name:	getComponents
Input:	void
Output:	ComponentViewModel[] : A list of components
Description:	Returns the list of components (widgets and the workspace)

Name:	getWidgets
Input:	void
Output:	Widget[] : A list of widgets
Description:	Returns the list of widgets

Name:	getDataSets
Input:	void
Output:	DataSet[] : The value of the dataSets attribute
Description:	Returns the list of data sets added to the project

Name:	getUnmarkedDataSets
Input:	void
Output:	DataSet[] : A list of DataSets
Description:	Returns all data sets not marked for deletion.

Name:	getEvents
Input:	void
Output:	Event[] : The value of the events attribute
Description:	Returns the list of events added to the project

Name:	getActions
Input:	void
Output:	Action[] : The value of the actions attribute
Description:	Returns the list of actions added to the project

Name:	getNonAutoActions
Input:	void
Output:	Action[] : A list of actions.
Description:	Returns all actions that are not applied automatically.

Name:	getWorkspace
Input:	void
Output:	WorkspaceViewModel : The value of the workspace attribute
Description:	Returns the project workspace

Name:	addDataSet
Input:	DataSet data : The DataSet to add
Output:	void
Description:	Adds a data set to the list of data on the project.

Name:	addWidget
Input:	Widget widget : The widget to add Number index : An optional index to insert the widget.
Output:	void
Description:	Adds a widget to the list of widgets in the project.

Name:	addEvent
Input:	Event event : The Event to add Number index : An optional index to insert the event.
Output:	void
Description:	Adds an event to the list of events in the project.

Name:	addAction
Input:	Action action : The Action to add Number index : An optional index to insert the action.
Output:	void
Description:	Adds an action to the list of actions in the project.

Name:	getWidget
Input:	String name : the name of the widget to return
Output:	Widget : the Widget with the given name, or null if none is found
Description:	Returns the widget added to the project with the given name.

Name:	getDataSet
Input:	String name : the name of the data set to return
Output:	DataSet : the DataSet with the given name, or null if none is found
Description:	Returns the data set added to the project with the given name.

Name:	getAction
Input:	String name : the name of the action to return
Output:	Action : the Action with the given name, or null if none is found
Description:	Returns the action added to the project with the given name.

Name:	getEvent
Input:	String name : the name of the event to return
Output:	Event : the Event with the given name, or null if none is found
Description:	Returns the event added to the project with the given name.

Name:	removeDataSet
Input:	DataSet data : The DataSet to remove
Output:	void
Description:	Removes the specified data set from the list of data sets

Name:	removeWidget
Input:	Widget widget : The widget to remove
Output:	void
Description:	Removes the specified widget from the list of widgets

Name:	removeEvent
Input:	Event event : The event to remove
Output:	void
Description:	Removes the specified event from the list of events

Name:	removeAction
Input:	Action action : The action to remove
Output:	void
Description:	Removes the specified action from the list of actions

Name:	refreshWorkspace
Input:	void
Output:	void
Description:	Resets the workspace to a state where no events have triggered.

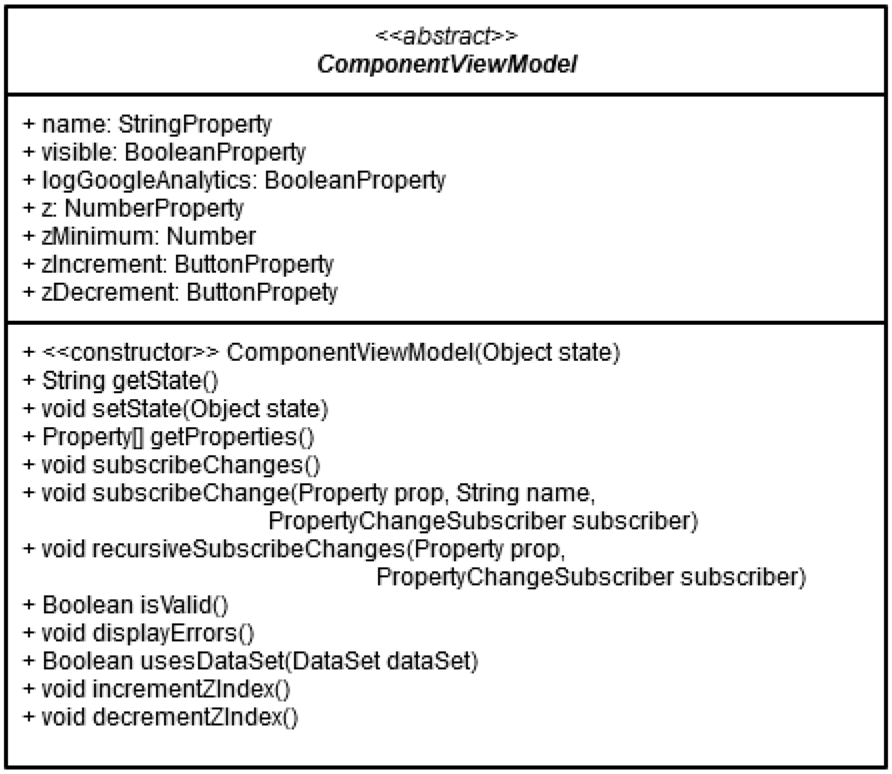
Name:	resetProject
Input:	void
Output:	void
Description:	Resets the project to the initial state, as if it were a new project

Name:	subscribeChanges
Input:	void
Output:	void
Description:	Creates knockout subscriptions so that history of properties can be tracked.

Name:	subscribeNameChange
Input:	void
Output:	void
Description:	Tracks when the project name changes so that the URL parameter can be updated.

4.2.3 Widget Classes

4.2.3.1 ComponentViewModel



```
<<abstract>>
ComponentViewModel

+ name: StringProperty
+ visible: BooleanProperty
+ logGoogleAnalytics: BooleanProperty
+ z: NumberProperty
+ zMinimum: Number
+ zIncrement: ButtonProperty
+ zDecrement: ButtonPropety

+ <<constructor>> ComponentViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()
+ void subscribeChanges()
+ void subscribeChange(Property prop, String name,
                        PropertyChangeSubscriber subscriber)
+ void recursiveSubscribeChanges(Property prop,
                                  PropertyChangeSubscriber subscriber)
+ Boolean isValid()
+ void displayErrors()
+ Boolean usesDataSet(DataSet dataSet)
+ void incrementZIndex()
+ void decrementZIndex()
```

Figure 35. ComponentViewModel UML class description

The ComponentViewModel class is an abstract class that encapsulates the widget and workspace classes. ComponentViewModel objects can trigger events and be the target of actions.

4.2.3.1.1 Attributes

Name	**Access**	**Type**	**Description**
name	public	StringProperty	The name of the component
visible	public	BooleanProperty	True if the widget is visible
logGoogleAnalytics	public	BooleanProperty	True if the widget should log googleAnalytics
z	public	NumberProperty	The z-index of the component
zMinimum	public	Number	The minimum value of z
zIncrement	public	ButtonProperty	Creates a button that increments the value of z using the incrementZIndex method
zDecrement	public	ButtonProperty	Creates a button that decrements the value of z using the decrementZIndex method

4.2.3.1.2 Methods

Name:	<<constructor>> ComponentViewModel
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the componentt
Description:	Returns a JSON stringified representation of the state of the component and its attributes

Name:	setState
Input:	Object state : The new state for the component
Output:	void
Description:	Sets the attributes of the component based on the given state JSON representation

Name:	getProperties
Input:	void
Output:	Property[] : A list of properties the user can modify
Description:	Returns a list of componnt properties the user can modify on the properties panel

Name:	subscribeChanges
Input:	void
Output:	void
Description:	Creates knockout subscriptions so that history of properties can be tracked.

Name:	subscribeChange
Input:	Property prop : the object that contains the name to subscribe String name : the name of the attribute in prop to subscribe PropertyChangeSubscriber subscriber : helper class used to subscribe properties
Output:	void
Description:	Creates knockout subscriptions for a single property so that history of properties can be tracked.

Name:	recursiveSubscribeChanges
Input:	Property prop : the Property to recursively subscribe changes PropertyChangeSubscriber propertyChangeSubscriber : helper class used to subscribe properties
Output:	void
Description:	Traverse a Property recursively for when a Property contains other Properties.

Name:	isValid
void	void
Output:	Boolean : true of the component has no errors
Description:	Returns true if no properties of the component have errors, otherwise false.

Name:	displayErrors
Input:	void
Output:	void
Description:	Displays error messages for all properties that have an error.

Name:	usesDataSet
Input:	DataSet dataSet : the dataSet to check
Output:	Boolean : true if the given data set is used by the component.
Description:	Returns true if the given data set is used by the component, otherwise false.

Name:	incrementZIndex
Input:	void
Output:	void
Description:	Increment the Component’s z value by 1

Name:	decrementZIndex
Input:	void
Output:	void
Description:	Decrement the Component’s z value by 1 with a lower bound of zMinimum

4.2.3.2 WorkspaceViewModel

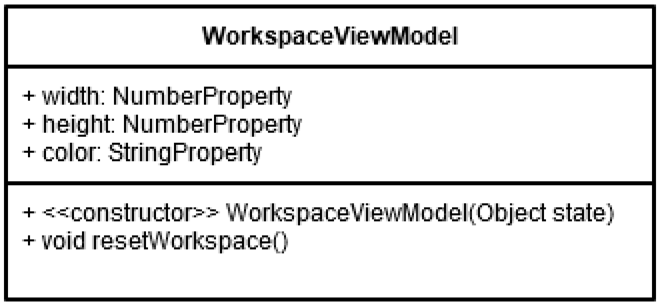


Figure 36. WorkspaceViewModel UML class description

The WorkspaceViewModel class is the view model for the project workspace. This extends the ComponentViewModel class.

4.2.3.2.1 Attributes

Name	**Access**	**Type**	**Description**
width	public	NumberProperty	The width of the workspace
height	public	NumberProperty	The height of the workspace
color	public	StringProperty	The background color of the workspace

4.2.3.2.2 Methods

Name:	<<constructor>> WorkspaceViewModel
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	resetWorkspace
Input:	void
Output:	void
Description:	Sets the workspace properties to the default values.

4.2.3.3 Widget

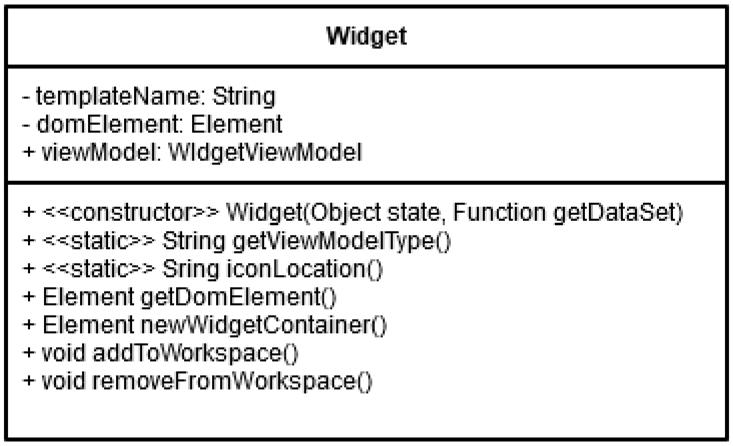

Widget
- templateName: String
- domElement: Element
+ viewModel: WIdgetViewModel
+ <<constructor>> Widget(Object state, Function getDataSet)
+ <<static>> String getViewModelType()
+ <<static>> Sring iconLocation()
+ Element getDomElement()
+ Element newWidgetContainer()
+ void addToWorkspace()
+ void removeFromWorkspace()

Figure 37. Widget UML class description

The Widget class is an abstract class that defines the attributes and methods required by all widgets.

4.2.3.3.1 Attributes

Name	Access	Type	Description
templateName	private	String	The name of the template for the UI element
domElement	private	Element	The widget DOM element
viewModel	public	WidgetViewModel	The view model for the widget

4.2.3.3.2 Method

Name:	<<constructor>> Widget
Input:	Object state : The state of the attributes Function getDataSet : A function that returns available data sets
Output:	void
Description:	Sets the values of the attributes

Name:	<<static>> getViewModelType
Input:	void
Output:	String : the type of the widget view model
Description:	Returns the type of the widget view model.

Name:	<<static>> iconLocation
Input:	void
Output:	String : The relative path to the widget icon
Description:	Returns the relative path to the widget icon for displaying the widget panel.

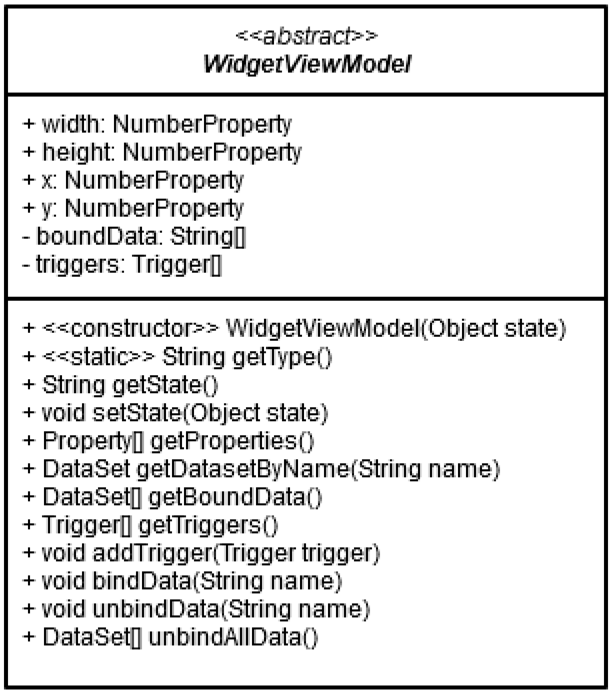
Name:	getDomElement
Input:	void
Output:	Element : The DOM element
Description:	Returns the widget's DOM element

Name:	newWidgetContainer
Input:	void
Output:	Element : A DOM Element to contain the widget
Description:	Returns a newly created container for the widget.

Name:	addToWorkspace
Input:	void
Output:	void
Description:	Adds the widget DOM element to the workspace.

Name:	removeFromWorkspace
Input:	void
Output:	void
Description:	Removes the DOM element from the workspace.

4.2.3.4 WidgetViewModel



<<abstract>>
WidgetViewModel
+ width: NumberProperty
+ height: NumberProperty
+ x: NumberProperty
+ y: NumberProperty
- boundData: String[]
- triggers: Trigger[]
+ <<constructor>> WidgetViewModel(Object state)
+ <<static>> String getType()
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()
+ DataSet getDatasetByName(String name)
+ DataSet[] getBoundData()
+ Trigger[] getTriggers()
+ void addTrigger(Trigger trigger)
+ void bindData(String name)
+ void unbindData(String name)
+ DataSet[] unbindAllData()

Figure 38. WidgetViewModel UML class description

The WidgetViewModel class is an abstract class that defines the attributes and methods required by all widget view models. WidgetViewModel extends the ComponentViewModel class.

4.2.3.4.1 Attributes

Name	Access	Type	Description
width	public	NumberProperty	The width of the widget
height	public	NumberProperty	The height of the widget
x	public	NumberProperty	The horizontal offset of the widget as a percentage from the left of the workspace
y	public	NumberProperty	The vertical offset of the widget as a percentage from the top of the workspace
boundData	private	String[]	A list of the names of data sets bound to the widget
triggers	private	Trigger[]	A list of triggers for the widget

4.2.3.4.2 Methods

Name:	<<constructor>> WidgetViewModel
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	<<static>> getType
Input:	void
Output:	String : the type of the widget view model
Description:	Returns the type of the widget view model.

Name:	getState
Input:	void
Output:	String : A string representing the widget
Description:	Returns a JSON stringified representation of the state of the widget and its attributes

Name:	setState
Input:	Object state : The new state for the widget
Output:	void
Description:	Sets the attributes of the widget based on the given state JSON representation

Name:	getProperties
Input:	void
Output:	Property[] : A list of properties the user can modify
Description:	Returns a list of widget properties the user can modify on the properties panel

Name:	getDatasetByName
Input:	String name : the data set name
Output:	DataSet : the data set with the given name
Description:	Returns the data set bound to the widget with the given name.

Name:	getBoundData
Input:	void
Output:	DataSet[] : The list of bound data
Description:	Returns the list of data sets bound to the widget.

Name:	getTriggers
Input:	void
Output:	Trigger[] : A list of triggers
Description:	Returns the list of triggers for the widget.

Name:	addTrigger
Input:	Trigger trigger : the trigger to add
Output:	void
Description:	Adds a trigger to the list of triggers.

Name:	bindData
Input:	String name : The name of the data set
Output:	void
Description:	Binds a loaded data set to the widget.

Name:	unbindData
Input:	String name : The name of the data set to remove
Output:	void
Description:	Unbinds the data set with the specified name from the list of bound data.

Name:	unbindAllData
Input:	void
Output:	DataSet[] : The data bound to the widget
Description:	Unbinds all data from the widget, then returns a list of the data sets that were bound.

4.2.3.5 USMap

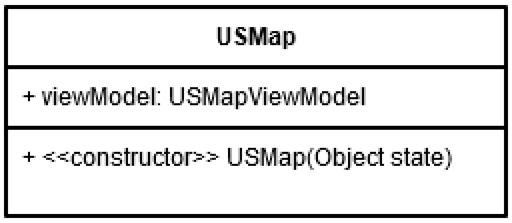

USMap
+ viewModel: USMapViewModel
+ <<constructor>> USMap(Object state)

Figure 39. USMap UML class description

The USMap class creates the user interface components and the view model for a U.S. map widget. This class extends the Widget class.

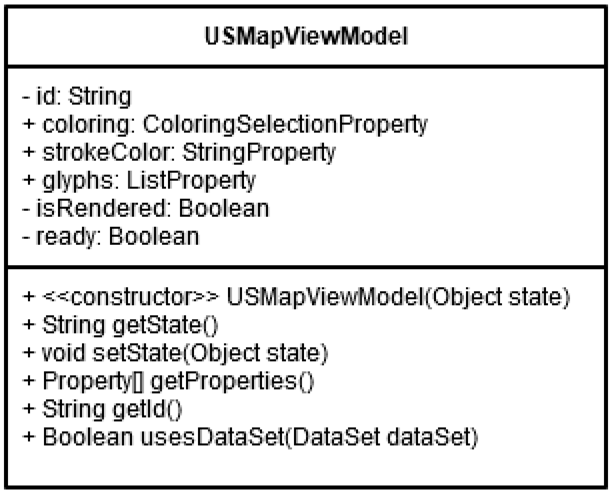
4.2.3.5.1 Attributes

Name	Access	Type	Description
viewModel	public	USMapViewModel	The map view model

4.2.3.5.2 Methods

Name:	<<constructor>> USMap
Input:	Object state : The state of the view model attributes
Output:	void
Description:	Creates the view model and the component in the user interface

4.2.3.6 USMapViewModel



USMapViewModel

- id: String
+ coloring: ColoringSelectionProperty
+ strokeColor: StringProperty
+ glyphs: ListProperty
- isRendered: Boolean
- ready: Boolean

+ <<constructor>> USMapViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()
+ String getId()
+ Boolean usesDataSet(DataSet dataSet)

Figure 40. USMapViewModel UML class description

The USMapViewModel class inherits from the WidgetViewModel class. It contains the attributes and methods necessary to represent a map of the United States.

4.2.3.6.1 Attributes

Name	**Access**	**Type**	**Description**
id	private	String	A unique ID
coloring	public	ColoringSelectionProperty	The coloring of the map states
strokeColor	public	StringProperty	The color of the state outlines
glyphs	public	ListProperty	The list of map glyphs
isRendered	private	Boolean	True if the map has been rendered
ready	private	Boolean	True if the DOM element is ready

4.2.3.6.2 Methods

Name:	<<constructor>> USMapViewModel
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the widget
Description:	Returns a JSON stringified representation of the state of the widget and its attributes

Name:	setState
Input:	Object state : The new state for the widget
Output:	void
Description:	Sets the attributes of the widget based on the given state

Name:	getProperties
Input:	void
Output:	Property[] : A list of properties the user can modify
Description:	Returns a list of widget properties the user can modify on the properties panel

Name:	getId
Input:	void
Output:	String : the id of the map
Description:	Returns the unique ID of the us map.

Name:	usesDataSet
Input:	DataSet dataSet : the dataSet to check
Output:	Boolean : true if the given data set is used by the us map
Description:	Returns true if the given data set is used by the us map or its glyphs, otherwise false.

4.2.3.7 GlyphHelper

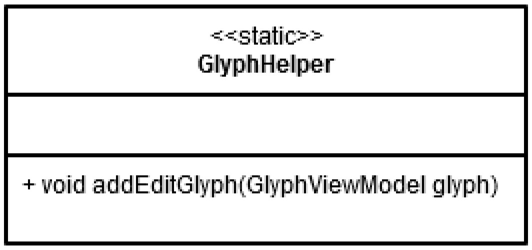


Figure 41. GlyphHelper UML class description

The GlyphHelper class contains helper functions for creating and editing glyphs. The USMapViewModel class references this module.

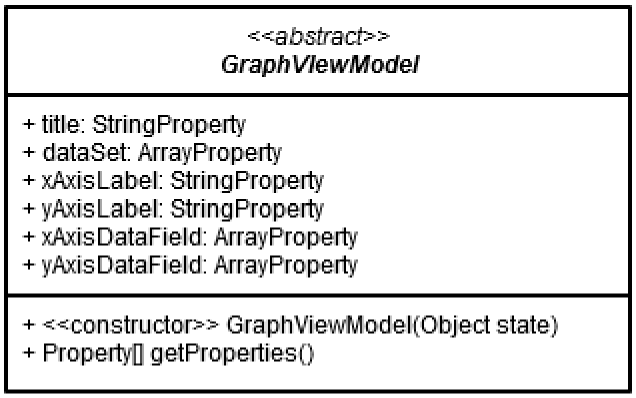
4.2.3.7.1 Attributes

Name	**Access**	**Type**	**Description**

4.2.3.7.2 Methods

Name:	addEditGlyph
Input:	GlyphViewModel glyph
Output:	void
Description:	Initializes the dialog to add or edit the given glyph

4.2.3.8 GraphViewModel



<<abstract>>
GraphViewModel
+ title: StringProperty
+ dataSet: ArrayProperty
+ xAxisLabel: StringProperty
+ yAxisLabel: StringProperty
+ xAxisDataField: ArrayProperty
+ yAxisDataField: ArrayProperty
+ <<constructor>> GraphViewModel(Object state)
+ Property[] getProperties()

Figure 42. GraphViewModel UML class description

The GraphViewModel class is an abstract class that inherits from the WidgetViewModel class. It contains the attributes and methods necessary for any type of graph widget.

4.2.3.8.1 Attributes

Name	Access	Type	Description
title	public	StringProperty	The title of the graph
dataSet	public	ArrayProperty	The data associated with the graph
xAxisLabel	public	StringProperty	The horizontal axis label of the graph
yAxisLabel	public	StringProperty	The vertical axis label of the graph
xAxisDataField	public	ArrayProperty	The data field associated with the horizontal axis
yAxisDataField	public	ArrayProperty	The data field associated with the vertical axis

4.2.3.8.2 Methods

Name:	<<constructor>> GraphViewModel
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getProperties
Input:	void
Output:	Property[] : A list of properties the user can modify
Description:	Returns a list of widget properties the user can modify on the properties panel

4.2.3.9 LineGraph

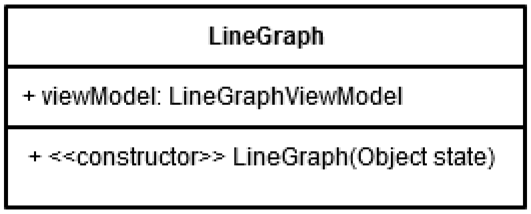

LineGraph
+ viewModel: LineGraphViewModel
+ <<constructor>> LineGraph(Object state)

Figure 43. LineGraph UML class description

The LineGraph class creates the user interface components and the view model for a line graph.

4.2.3.9.1 Attributes

Name	Access	Type	Description
viewModel	public	LineGraphViewModel	The view model

4.2.3.9.2 Methods

Name:	<<constructor>> LineGraph
Input:	Object state : The state of the view model attributes
Output:	void
Description:	Creates the view model and the component in the user interface

4.2.3.10 LineGraphViewModel

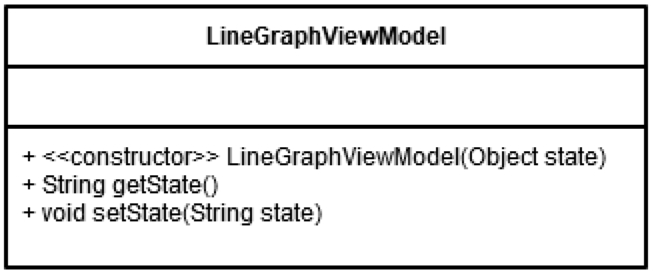

LineGraphViewModel
+ <<constructor>> LineGraphViewModel(Object state)
+ String getState()
+ void setState(String state)

Figure 44. LineGraphViewModel UML class description

The LineGraphViewModel class inherits from the GraphViewModel class. It contains the attributes and methods necessary to represent a line graph.

4.2.3.10.1 Attributes

Name	Access	Type	Description

4.2.3.10.2 Methods

Name:	<<constructor>> LineGraphViewModel
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the widget
Description:	Returns a JSON stringified representation of the state of the widget and its attributes

Name:	setState
Input:	Object state : The new state for the widget
Output:	void
Description:	Sets the attributes of the widget based on the given state

4.2.3.11 Button

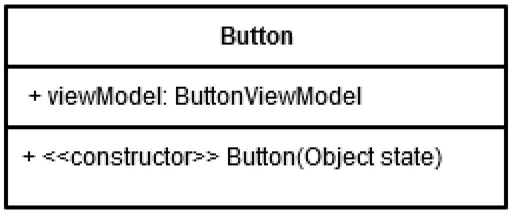

Button
+ viewModel: ButtonViewModel
+ <<constructor>> Button(Object state)

Figure 45. Button UML class description

The Button class creates the user interface components and the view model for a button widget.

4.2.3.11.1 Attributes

Name	Access	Type	Description
viewModel	public	ButtonViewModel	The view model

4.2.3.11.2 Methods

Name:	<<constructor>> Button
Input:	Object state : The state of the view model attributes
Output:	void
Description:	Creates the view model and the component in the user interface

4.2.3.12 ButtonViewModel

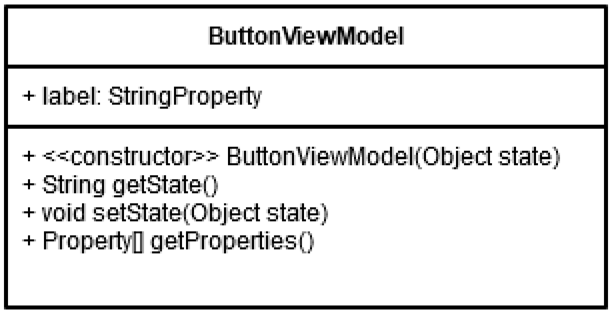

ButtonViewModel
+ label: StringProperty
+ <<constructor>> ButtonViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()

Figure 46. ButtonViewModel UML class description

The ButtonViewModel class inherits from the WidgetViewModel class. It contains the attributes and methods necessary to represent a standard HTML button.

4.2.3.12.1 Attributes

Name	Access	Type	Description
label	public	StringProperty	The text on the button

4.2.3.12.2 Methods

Name:	<<constructor>> ButtonViewModel
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the widget
Description:	Returns a JSON stringified representation of the state of the widget and its attributes

Name:	setState
Input:	Object state : The new state for the widget
Output:	void
Description:	Sets the attributes of the widget based on the given state JSON representation

Name:	getProperties
Input:	void
Output:	Property[] : A list of properties the user can modify
Description:	Returns a list of widget properties the user can modify on the properties panel

4.2.3.13 TextBlock

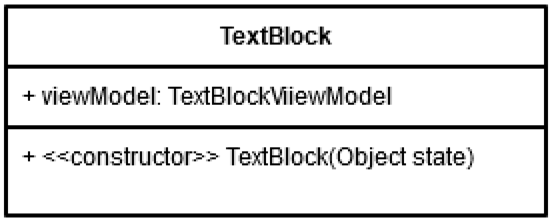

TextBlock
+ viewModel: TextBlockViewModel
+ <<constructor>> TextBlock(Object state)

Figure 47. TextBlock UML class description

The TextBlock class creates the user interface components and the view model for a text block.

4.2.3.13.1 Attributes

Name	Access	Type	Description
viewModel	public	TextBlockViewModel	The view model

4.2.3.13.2 Methods

Name:	<<constructor>> TextBlock
Input:	Object state : The state of the view model attributes
Output:	void
Description:	Creates the view model and the component in the user interface

4.2.3.14 Tooltip

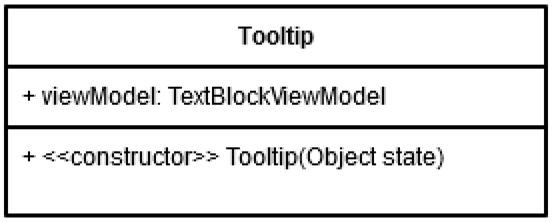

Tooltip
+ viewModel: TextBlockViewModel
+ <<constructor>> Tooltip(Object state)

Figure 48. Tooltip UML class description

The Tooltip class creates the user interface components and the view model for a tooltip.

4.2.3.14.1 Attributes

Name	Access	Type	Description
viewModel	public	TextBlockViewModel	The view model

4.2.3.14.2 Methods

Name:	<<constructor>> Tooltip
Input:	Object state : The state of the view model attributes
Output:	void
Description:	Creates the view model and the component in the user interface

4.2.3.15 TextBlockViewModel

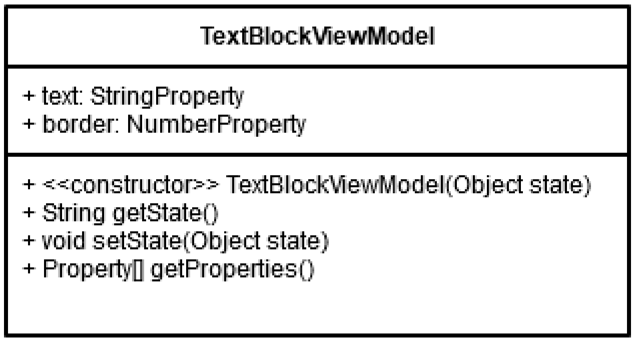

TextBlockViewModel
+ text: StringProperty
+ border: NumberProperty
+ <<constructor>> TextBlockViewModel(Object state)
+ String getState()
+ void setState(Object state)
+ Property[] getProperties()

Figure 49. TextBlockViewModel UML class description

The TextBlock class inherits from the WidgetViewModel class. It contains the attributes and methods necessary to represent a standard HTML text area.

4.2.3.15.1 Attributes

Name	Access	Type	Description
text	public	StringProperty	The text in the text block
border	public	NumberProperty	The border width

4.2.3.15.2 Methods

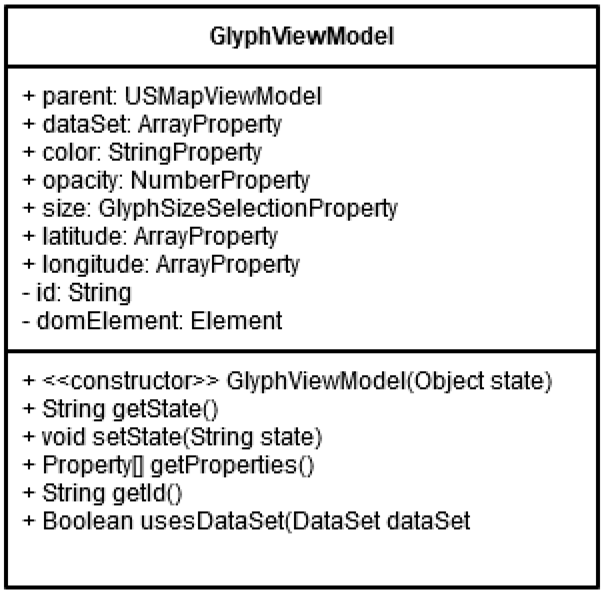
Name:	<<constructor>> TextBlockViewModel
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the widget
Description:	Returns a JSON stringified representation of the state of the widget and its attributes

Name:	setState
Input:	Object state : The new state for the widget
Output:	void
Description:	Sets the attributes of the widget based on the given state

Name:	getProperties
Input:	void
Output:	Property[] : A list of properties the user can modify
Description:	Returns a list of widget properties the user can modify on the properties panel

4.2.3.16 GlyphViewModel



GlyphViewModel

+ parent: USMapViewModel
+ dataSet: ArrayProperty
+ color: StringProperty
+ opacity: NumberProperty
+ size: GlyphSizeSelectionProperty
+ latitude: ArrayProperty
+ longitude: ArrayProperty
- id: String
- domElement: Element

+ <<constructor>> GlyphViewModel(Object state)
+ String getState()
+ void setState(String state)
+ Property[] getProperties()
+ String getId()
+ Boolean usesDataSet(DataSet dataSet

Figure 50. GlyphViewModel UML class description

The GlyphViewModel class inherits from the ComponentViewModel and defines the attributes and methods required by the glyph. The attributes are used by D3 to create the view.

4.2.3.16.1 Attributes

Name	Access	Type	Description
parent	public	USMapViewMdoel	The map containing the glyphs
dataSet	public	ArrayProperty	The data set used to draw the glyphs
color	public	StringProperty	The color of the glyphs
opacity	public	NumberProperty	The opacity of the glyphs
size	public	GlyphSizeSelectionProperty	The size of the glyphs
latitude	public	ArrayProperty	The latitude field of the data set
longitude	public	ArrayProperty	The longitude field of the data set
id	private	String	A unique glyph id
dom	private	Element	The DOM element containing the glyph

4.2.3.16.2 Methods

Name:	<<constructor>> GlyphViewModel
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the widget
Description:	Returns a JSON stringified representation of the state of the widget and its attributes

Name:	setState
Input:	Object state : The new state for the widget
Output:	void
Description:	Sets the attributes of the widget based on the given state

Name:	getProperties
Input:	void
Output:	Property[] : A list of properties the user can modify
Description:	Returns a list of widget properties the user can modify on the properties panel

Name:	getId
Input:	void
Output:	String : The glyph id
Description:	Returns the unique id for the glyph.

Name:	usesDataSet
Input:	DataSet dataSet : the dataSet to check
Output:	Boolean : true if the given data set is used by the glyph
Description:	Returns true if the given data set is used by the glyph, otherwise false.

4.2.3.17 Property

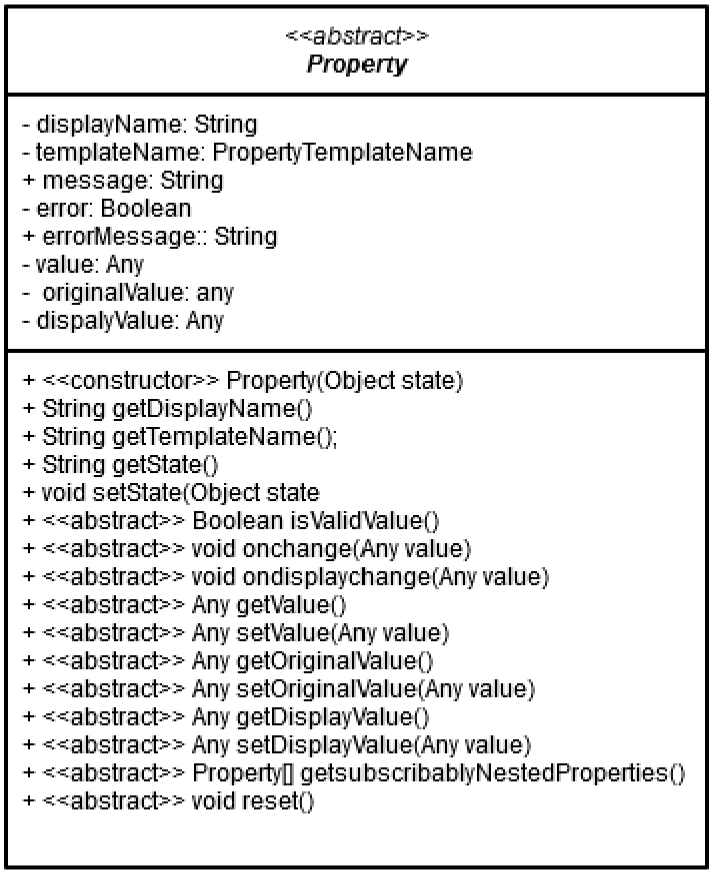

<<abstract>>
Property
- displayName: String
- templateName: PropertyTemplateName
+ message: String
- error: Boolean
+ errorMessage:: String
- value: Any
- originalValue: any
- dispalyValue: Any
+ <<constructor>> Property(Object state)
+ String getDisplayName()
+ String getTemplateName();
+ String getState()
+ void setState(Object state
+ <<abstract>> Boolean isValidValue()
+ <<abstract>> void onchange(Any value)
+ <<abstract>> void ondisplaychange(Any value)
+ <<abstract>> Any getValue()
+ <<abstract>> Any setValue(Any value)
+ <<abstract>> Any getOriginalValue()
+ <<abstract>> Any setOriginalValue(Any value)
+ <<abstract>> Any getDisplayValue()
+ <<abstract>> Any setDisplayValue(Any value)
+ <<abstract>> Property[] getsubscribablyNestedProperties()
+ <<abstract>> void reset()

Figure 51. Property UML class description

The Property class is an abstract class to describe properties used in objects that inherit from the ComponentViewModel class.

4.2.3.17.1 Attribute

Name	Access	Type	Description
displayName	private	String	The name of the property
templateName	private	PropertyTemplateName	The name of the template to display the property
message	public	String	The message displayed in the user interface
error	private	Boolean	True when the value is not valid
errorMessage	public	String	The error message to display when the property has an error
value	private	Any	The current value of the property, as reflected in the workspace view.
originalValue	private	Any	The user-set value of the property from the Properties Panel, unaffected by user events.
displayValue	private	Any	A value for the property shown in the user interface input fields, but which does not automatically update the originalValue or value properties.

4.2.3.17.2 Methods

Name:	<<constructor>> Property
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	<<abstract>> isValidValue
Input:	Any value: The value to test
Output:	Boolean : True if the value is valid
Description:	Checks whether the value meets the validation requirements.

Name:	<<abstract>> onchange
Input:	Any value: The new value
Output:	void
Description:	A function that is executed when the value is changed.

Name:	<<abstract>> ondisplaychange
Input:	Any value: The new value
Output:	void
Description:	A function that is executed when the display value is changed.

Name:	getDisplayName
Input:	void
Output:	String : The name of the property
Description:	Returns the name of the property to display in the user interface

Name:	getTemplateName
Input:	void
Output:	PropertyTemplateName: The name of the template
Description:	Returns the name of the template to use to display the property on the properties panel

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

Name:	<<abstract>> getValue
Input:	void
Output:	Any : The value
Description:	Returns the value of the property

Name:	<<abstract>> setValue
Input:	Any : The new value of the property
Output:	void
Description:	Sets the value of the property

Name:	<<abstract>> getOriginalValue
Input:	void
Output:	Any : The original value
Description:	Returns the original value of the property

Name:	<<abstract>> setOriginalValue
Input:	Any : The new original value of the property
Output:	void
Description:	Sets the original value of the property

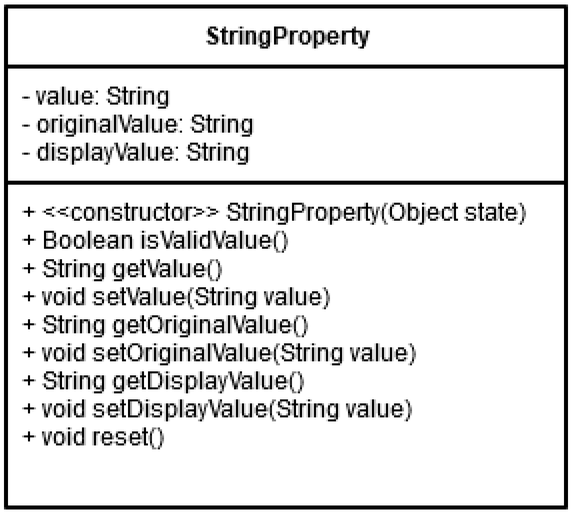
Name:	<<abstract>> getDisplayValue
Input:	void
Output:	Any : The display value
Description:	Returns the display value of the property

Name:	<<abstract>> setDisplayValue
Input:	Any : The new display value of the property
Output:	void
Description:	Sets the display value of the property

Name:	<<abstract>> getSubscribablyNestedProperties
Input:	void
Output:	Property[] : A list of sub-properties
Description:	Returns a list of properties nested in this property.

Name:	<<abstract>> reset
Input:	void
Output:	void
Description:	Resets the property values to be empty.

4.2.3.18 StringProperty



```
StringProperty
- value: String
- originalValue: String
- displayValue: String
+ <<constructor>> StringProperty(Object state)
+ Boolean isValidValue()
+ String getValue()
+ void setValue(String value)
+ String getOriginalValue()
+ void setOriginalValue(String value)
+ String getDisplayValue()
+ void setDisplayValue(String value)
+ void reset()
```

Figure 52. StringProperty UML class description

The StringProperty class extends the Property class and has a String value.

4.2.3.18.1 Attributes

Name	**Access**	**Type**	**Description**
value	private	String	The current value of the property, as reflected in the workspace view.
originalValue	private	String	The user-set value of the property from the Properties Panel, unaffected by user events.
displayValue	private	String	A value for the property shown in the user interface input fields, but which does not automatically update the originalValue or value properties.

4.2.3.18.2 Methods

Name:	<<constructor>> StringProperty
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	isValidValue
Input:	value
Output:	Boolean : True if the value is valid
Description:	Checks whether the value meets the validation requirements

Name:	getValue
Input:	void
Output:	String : The value
Description:	Returns the value of the property

Name:	setValue
Input:	String : The new value of the property
Output:	void
Description:	Sets the value of the property

Name:	getOriginalValue
Input:	void
Output:	String : The original value
Description:	Returns the original value of the property

Name:	setOriginalValue
Input:	String : The new original value of the property
Output:	void
Description:	Sets the original value of the property

Name:	getDisplayValue

Input:	void
Output:	String : The display value
Description:	Returns the display value of the property

Name:	setDisplayValue
Input:	String : The new display value of the property
Output:	void
Description:	Sets the display value of the property

Name:	reset
Input:	void
Output:	void
Description:	Resets the property values to be empty.

4.2.3.19 NumberProperty

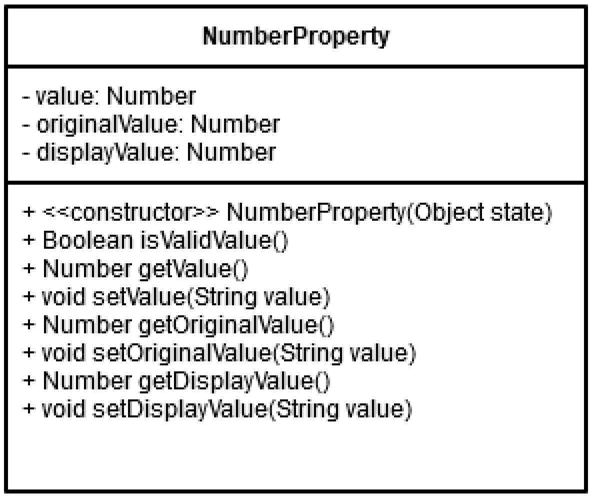

NumberProperty
- value: Number
- originalValue: Number
- displayValue: Number
+ <<constructor>> NumberProperty(Object state)
+ Boolean isValidValue()
+ Number getValue()
+ void setValue(String value)
+ Number getOriginalValue()
+ void setOriginalValue(String value)
+ Number getDisplayValue()
+ void setDisplayValue(String value)

Figure 53. NumberProperty UML class description

The NumberProperty class extends the Property class and has a Number value.

4.2.3.19.1 Attributes

Name	Access	Type	Description
value	private	Number	The current value of the property, as reflected in the workspace view.
originalValue	private	Number	The user-set value of the property from the Properties Panel, unaffected by user events.
displayValue	private	Number	A value for the property shown in the user interface input fields, but which does not automatically update the originalValue or value properties.

4.2.3.19.2 Methods

Name:	<<constructor>> NumberProperty
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	isValidValue
Input:	value
Output:	Boolean : True if the value is valid
Description:	Checks whether the value meets the validation requirements

Name:	getValue
Input:	void
Output:	Number : The value
Description:	Returns the value of the property

Name:	setValue
Input:	Number : The new value of the property
Output:	void
Description:	Sets the value of the property

Name:	getOriginalValue

Input:	void
Output:	Number : The original value
Description:	Returns the original value of the property

Name:	setOriginalValue
Input:	Number : The new original value of the property
Output:	void
Description:	Sets the original value of the property

Name:	getDisplayValue
Input:	void
Output:	Number : The display value
Description:	Returns the display value of the property

Name:	setDisplayValue
Input:	Number : The new display value of the property
Output:	void
Description:	Sets the display value of the property

4.2.3.20 BooleanProperty

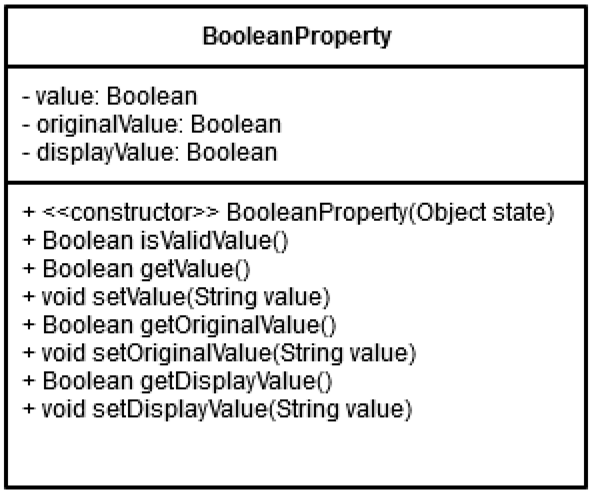

BooleanProperty
- value: Boolean
- originalValue: Boolean
- displayValue: Boolean
+ <<constructor>> BooleanProperty(Object state)
+ Boolean isValidValue()
+ Boolean getValue()
+ void setValue(String value)
+ Boolean getOriginalValue()
+ void setOriginalValue(String value)
+ Boolean getDisplayValue()
+ void setDisplayValue(String value)

Figure 54. BooleanProperty UML class description

The BooleanProperty class extends the Property class and has a Boolean value.

4.2.3.20.1 Attributes

Name	Access	Type	Description
value	private	Boolean	The current value of the property, as reflected in the workspace view.
originalValue	private	Boolean	The user-set value of the property from the Properties Panel, unaffected by user events.
displayValue	private	Boolean	A value for the property shown in the user interface input fields, but which does not automatically update the originalValue or value properties.

4.2.3.20.2 Methods

Name:	<<constructor>> BooleanProperty
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	isValidValue
Input:	value
Output:	Boolean : True if the value is valid
Description:	Checks whether the value meets the validation requirements

Name:	getValue
Input:	void
Output:	Boolean : The value
Description:	Returns the value of the property

Name:	setValue
Input:	Boolean : The new value of the property
Output:	void
Description:	Sets the value of the property

Name:	getOriginalValue
Input:	void
Output:	Boolean : The original value
Description:	Returns the original value of the property

Name:	setOriginalValue
Input:	Boolean : The new original value of the property
Output:	void
Description:	Sets the original value of the property

Name:	getDisplayValue
Input:	void
Output:	Boolean : The display value
Description:	Returns the display value of the property

Name:	setDisplayValue
Input:	Boolean : The new display value of the property
Output:	void
Description:	Sets the display value of the property

4.2.3.21 ArrayProperty

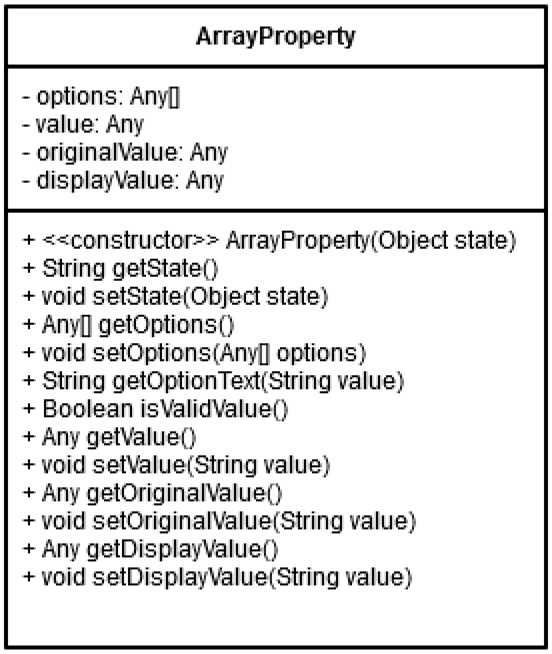

ArrayProperty
- options: Any[]
- value: Any
- originalValue: Any
- displayValue: Any
+ <<constructor>> ArrayProperty(Object state)
+ String getState()
+ void setState(Object state)
+ Any[] getOptions()
+ void setOptions(Any[] options)
+ String getOptionText(String value)
+ Boolean isValidValue()
+ Any getValue()
+ void setValue(String value)
+ Any getOriginalValue()
+ void setOriginalValue(String value)
+ Any getDisplayValue()
+ void setDisplayValue(String value)

Figure 55. ArrayProperty UML class description

The ArrayProperty class extends the Property class and has an array of values.

4.2.3.21.1 Attributes

Name	**Access**	**Type**	**Description**
options	private	Any[]	The possible values to pick from
value	private	Any	The current value of the property, as reflected in the workspace view.
originalValue	private	Any	The user-set value of the property from the Properties Panel, unaffected by user events.
displayValue	private	Any	A value for the property shown in the user interface input fields, but which does not automatically update the originalValue or value properties.

4.2.3.21.2 Methods

Name:	<<constructor>> ArrayProperty
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

Name:	isValidValue
Input:	value
Output:	Boolean : True if the value is valid
Description:	Checks whether the value meets the validation requirements

Name:	getValue
Input:	void
Output:	Any : The value
Description:	Returns the value of the property

Name:	setValue
Input:	Any : The new value of the property
Output:	void
Description:	Sets the value of the property

Name:	getOriginalValue
Input:	void
Output:	Any : The original value
Description:	Returns the original value of the property

Name:	setOriginalValue
Input:	Any : The new original value of the property
Output:	void
Description:	Sets the original value of the property

Name:	getDisplayValue
Input:	void
Output:	Any : The display value
Description:	Returns the display value of the property

Name:	setDisplayValue
Input:	Any : The new display value of the property
Output:	void
Description:	Sets the display value of the property

Name:	getOptions
Input:	void
Output:	Any[]: The values
Description:	Returns the list of possible values of the property

Name:	setOptions
Input:	Any[] : The new options of the property
Output:	void
Description:	Sets the values of the property

Name:	getOptionText
Input:	String value : the selected option
Output:	String : Formatted text for the value
Description:	Returns a string formatted to display in the user interface

4.2.3.22 ListProperty

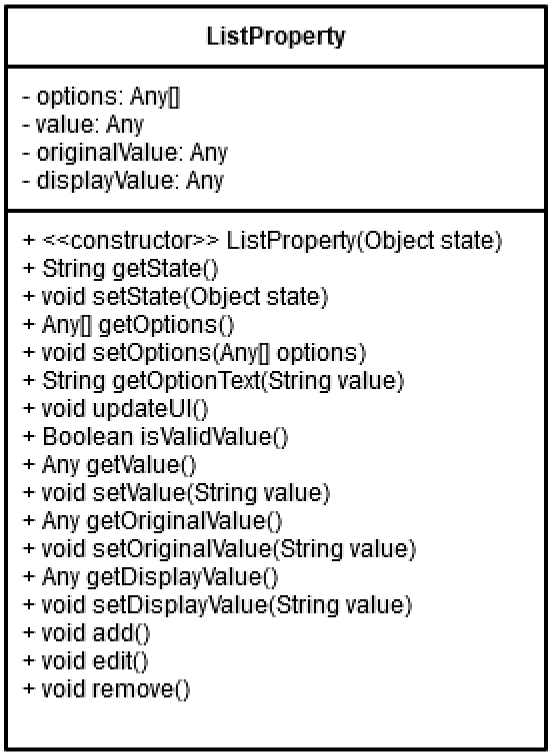


Figure 56. ListProperty UML class description

The ListProperty class extends the Property class and has an array of values. ListProperty also has hooks for methods to control adding, editing and removing values in the list.

4.2.3.22.1 Attributes

Name	Access	Type	Description
options	private	Any[]	The possible values to pick from
value	private	Any	The current value of the property, as reflected in the workspace view.
originalValue	private	Any	The user-set value of the property from the Properties Panel, unaffected by user events.
displayValue	private	Any	A value for the property shown in the user interface input fields, but which does not automatically update the originalValue or value properties.

4.2.3.22.2 Methods

Name:	<<constructor>> ListProperty
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	isValidValue
Input:	value
Output:	Boolean : True if the value is valid
Description:	Checks whether the value meets the validation requirements

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

Name:	getValue
Input:	void
Output:	Any : The value
Description:	Returns the value of the property

Name:	setValue
Input:	Any : The new value of the property
Output:	void
Description:	Sets the value of the property

Name:	getOriginalValue
Input:	void
Output:	Any : The original value
Description:	Returns the original value of the property

Name:	setOriginalValue
Input:	Any : The new original value of the property
Output:	void
Description:	Sets the original value of the property

Name:	getDisplayValue
Input:	void
Output:	Any : The display value
Description:	Returns the display value of the property

Name:	setDisplayValue
Input:	Any : The new display value of the property
Output:	void
Description:	Sets the display value of the property

Name:	getOptions
Input:	void
Output:	Any[]: The values
Description:	Returns the list of possible values of the property

Name:	setOptions
Input:	Any[] : The new options of the property
Output:	void
Description:	Sets the values of the property

Name:	updateUI
Input:	void
Output:	void
Description:	Adds styling to the list property UI elements

Name:	getOptionText
Input:	String value : the selected option
Output:	String : Formatted text for the value
Description:	Returns a string formatted to display in the user interface

Name:	add
Input:	void
Output:	void
Description:	A function that executes when the add button is clicked

Name:	edit
Input:	void
Output:	void
Description:	A function that executes when the edit button is clicked

Name:	remove
Input:	void
Output:	void
Description:	A function that executes when the remove button is clicked

4.2.3.23 ButtonProperty

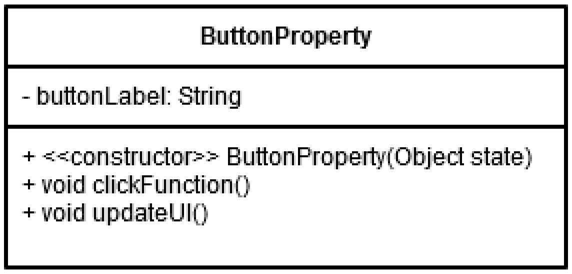


Figure 57: ButtonProperty UML class description

The ButtonProperty class extends the Property class and executes a function.

4.2.3.23.1 Attributes

Name	Access	Type	Description
buttonLabel	public	String	The label to display on the button

4.2.3.23.2 Methods

Name:	<<constructor>> ButtonProperty
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	clickFunction
Input:	void
Output:	void
Description:	A function that executes when the button is clicked.

Name:	updateUI
Input:	void
Output:	void
Description:	Adds styling to the list property UI elements

4.2.3.24 ColoringSelectionProperty

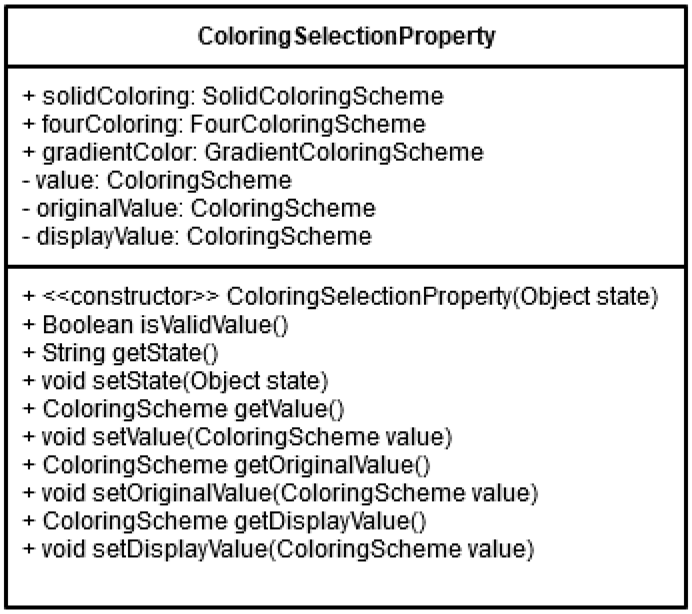

ColoringSelectionProperty
+ solidColoring: SolidColoringScheme
+ fourColoring: FourColoringScheme
+ gradientColor: GradientColoringScheme
- value: ColoringScheme
- originalValue: ColoringScheme
- displayValue: ColoringScheme
+ <<constructor>> ColoringSelectionProperty(Object state)
+ Boolean isValidValue()
+ String getState()
+ void setState(Object state)
+ ColoringScheme getValue()
+ void setValue(ColoringScheme value)
+ ColoringScheme getOriginalValue()
+ void setOriginalValue(ColoringScheme value)
+ ColoringScheme getDisplayValue()
+ void setDisplayValue(ColoringScheme value)

Figure 58. ColoringSelectionProperty UML class description

The ColoringProperty class extends the Property class. It defines the options available for coloring and the specific properties depending on which option is selected.

4.2.3.24.1 Attributes

Name	Access	Type	Description
solidColoring	public	SolidColoringScheme	The solid coloring scheme option
fourColoring	public	FourColoringScheme	The four coloring scheme option
gradientColor	public	GradientColoringScheme	The gradient coloring scheme option
value	private	ColoringScheme	The current value of the property, as reflected in the workspace view.
originalValue	private	ColoringScheme	The user-set value of the property from the Properties Panel, unaffected by user events.
displayValue	private	ColoringScheme	A value for the property shown in the user interface input fields, but which does not automatically update the originalValue or value properties.

4.2.3.24.2 Methods

Name:	<<constructor>> ColoringProperty
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	isValidValue
Input:	value
Output:	Boolean : True if the value is valid
Description:	Checks whether the value meets the validation requirements

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

Name:	getValue
Input:	void
Output:	ColoringScheme : The value
Description:	Returns the value of the property

Name:	setValue
Input:	ColoringScheme : The new value of the property
Output:	void
Description:	Sets the value of the property

Name:	getOriginalValue
Input:	void
Output:	ColoringScheme : The original value
Description:	Returns the original value of the property

Name:	setOriginalValue
Input:	ColoringScheme : The new original value of the property
Output:	void
Description:	Sets the original value of the property

Name:	getDisplayValue
Input:	void
Output:	ColoringScheme : The display value
Description:	Returns the display value of the property

Name:	setDisplayValue
Input:	ColoringScheme : The new display value of the property
Output:	void
Description:	Sets the display value of the property

4.2.3.25 ColoringScheme

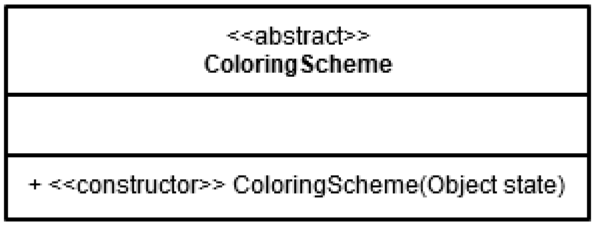


Figure 59. ColoringScheme UML class description

The ColoringScheme class is an abstract class that defines specific coloring implementations.

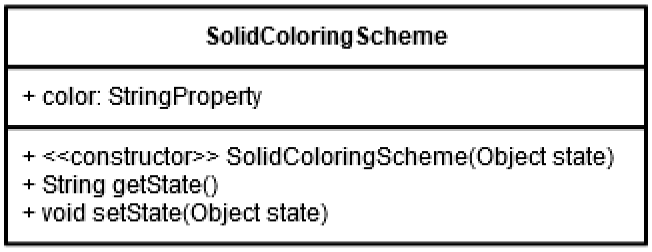
4.2.3.25.1 Attributes

Name	Access	Type	Description

4.2.3.25.2 Methods

Name:	<<constructor>> ColoringScheme
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

4.2.3.26 SolidColoringScheme



SolidColoringScheme
+ color: StringProperty
+ <<constructor>> SolidColoringScheme(Object state)
+ String getState()
+ void setState(Object state)

Figure 60. SolidColoringScheme UML class description

The SolidColoringScheme class extends the ColoringScheme class. It defines the coloring of a map that uses a single color.

4.2.3.26.1 Attributes

Name	Access	Type	Description
color	public	StringProperty	The property defining the color

4.2.3.26.2 Methods

Name:	<<constructor>> SolidColoringScheme
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

4.2.3.27 FourColoringScheme

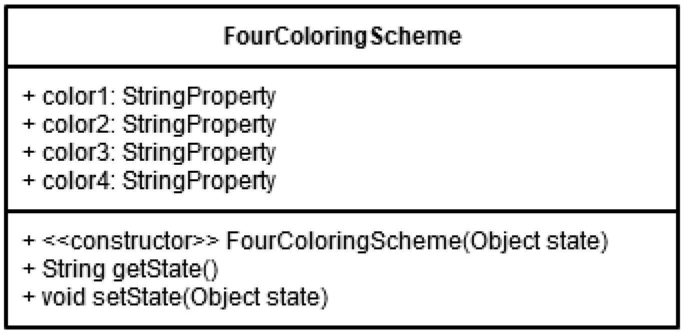


Figure 61. FourColoringScheme UML class description

The FourColoringScheme class extends the ColoringScheme class. It defines coloring of a map that uses four colors.

4.2.3.27.1 Attributes

Name	Access	Type	Description
color1	public	StringProperty	The property defining the first color
color2	public	StringProperty	The property defining the second color
color3	public	StringProperty	The property defining the third color
color4	public	StringProperty	The property defining the fourth color

4.2.3.27.2 Methods

Name:	<<constructor>> FourColoringScheme
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

4.2.3.28 GradientColoringScheme

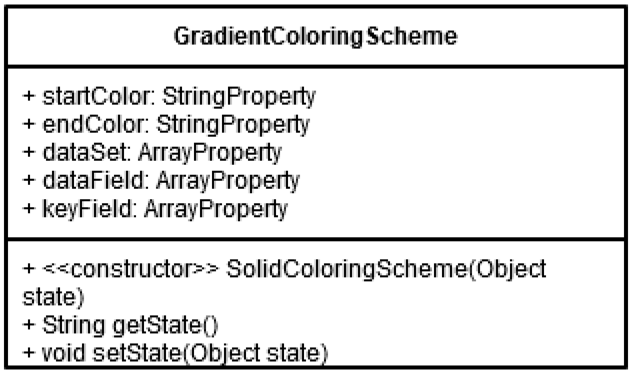

GradientColoringScheme
+ startColor: StringProperty
+ endColor: StringProperty
+ dataSet: ArrayProperty
+ dataField: ArrayProperty
+ keyField: ArrayProperty
+ <<constructor>> SolidColoringScheme(Object state)
+ String getState()
+ void setState(Object state)

Figure 62. GradientColoringScheme UML class description

The GradientColoringScheme class extends from the ColoringScheme class. It defines coloring of a map that uses two colors and a field from a DataSet to gradiently scale the coloring.

4.2.3.28.1 Attributes

Name	Access	Type	Description
startColor	public	StringProperty	The color for the lowest data value
endColor	public	StringProperty	The color for the highest data value
dataSet	public	ArrayProperty	The DataSet which contains the field for gradient ramp scaling
dataField	public	ArrayProperty	The field from dataSet which determines how to color the individual states
keyField	public	ArrayProperty	The key to decide how object are colored

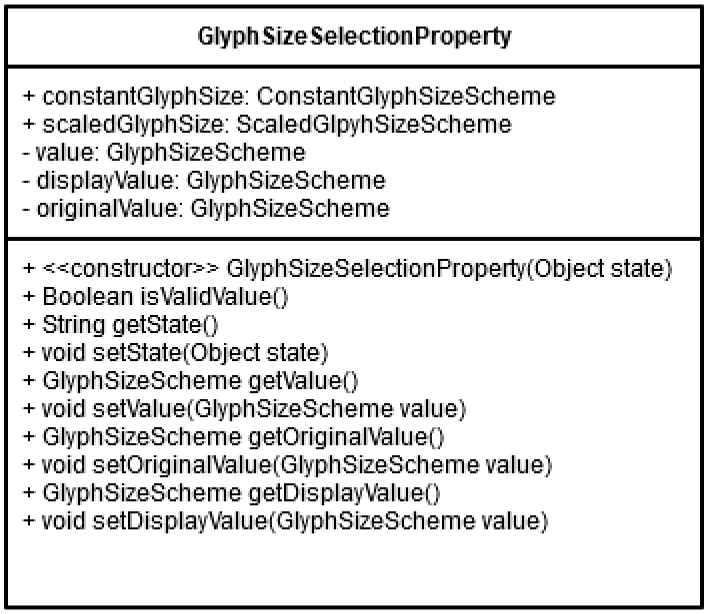
4.2.3.28.2 Methods

Name:	<<constructor>> GradientColoringScheme
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

4.2.3.29 GlyphSizeSelectionProperty



```
GlyphSizeSelectionProperty

+ constantGlyphSize: ConstantGlyphSizeScheme
+ scaledGlyphSize: ScaledGlpyhSizeScheme
- value: GlyphSizeScheme
- displayValue: GlyphSizeScheme
- originalValue: GlyphSizeScheme

+ <<constructor>> GlyphSizeSelectionProperty(Object state)
+ Boolean isValidValue()
+ String getState()
+ void setState(Object state)
+ GlyphSizeScheme getValue()
+ void setValue(GlyphSizeScheme value)
+ GlyphSizeScheme getOriginalValue()
+ void setOriginalValue(GlyphSizeScheme value)
+ GlyphSizeScheme getDisplayValue()
+ void setDisplayValue(GlyphSizeScheme value)
```

Figure 63. GlyphSizeSelectionProperty UML class description

The GlyphSizeProperty class extends the Property class. It defines the options available for glyph sizes and the specific properties depending on which option is selected.

4.2.3.29.1 Attributes

Name	Access	Type	Description
constantGlyphSize	public	ConstantGlyphSizeScheme	The constant glyph size option
scaledGlyphSize	public	ScaledGlyphSizeScheme	The scaled glyph size option
value	private	GlyphSizeScheme	The current value of the property, as reflected in the workspace view.
displayValue	private	GlyphSizeScheme	A value for the property shown in the user interface input fields, but which does not automatically update the originalValue or value properties.
originalValue	private	GlphSizeScheme	The user-set value of the property from the Properties Panel, unaffected by user events.

4.2.3.29.2 Methods

Name:	<<constructor>> GlyphSizeProperty
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	isValidValue
Input:	value
Output:	Boolean : True if the value is valid
Description:	Checks whether the value meets the validation requirements

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

Name:	getValue
Input:	void
Output:	ColoringScheme : The value
Description:	Returns the value of the property

Name:	setValue
Input:	ColoringScheme : The new value of the property
Output:	void
Description:	Sets the value of the property

Name:	getOriginalValue
Input:	void
Output:	ColoringScheme : The original value
Description:	Returns the original value of the property

Name:	setOriginalValue
Input:	ColoringScheme : The new original value of the property
Output:	void
Description:	Sets the original value of the property

Name:	getDisplayValue
Input:	void
Output:	ColoringScheme : The display value
Description:	Returns the display value of the property

Name:	setDisplayValue
Input:	ColoringScheme : The new display value of the property
Output:	void
Description:	Sets the display value of the property

4.2.3.30 GlyphSizeScheme

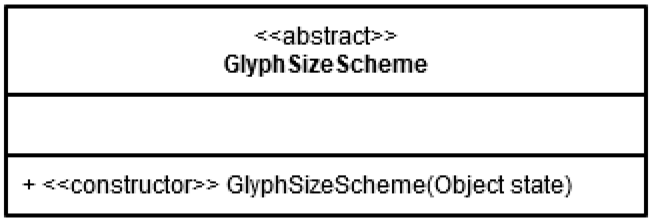


Figure 64. GlyphSizeScheme UML class description

The GlyphSizeScheme class is an abstract class that defines specific glyph size implementations.

4.2.3.30.1 Attributes

Name	Access	Type	Description

4.2.3.30.2 Methods

Name:	<<constructor>>
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

4.2.3.31 ConstantGlyphSizeScheme

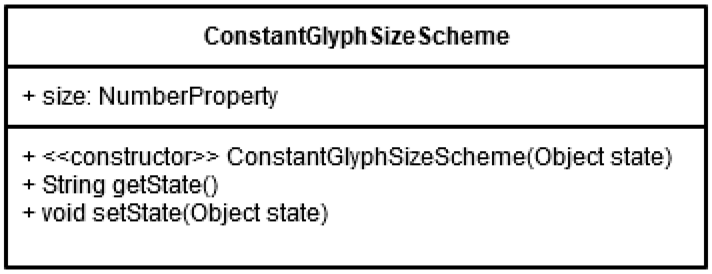

ConstantGlyphSizeScheme
+ size: NumberProperty
+ <<constructor>> ConstantGlyphSizeScheme(Object state)
+ String getState()
+ void setState(Object state)

Figure 65. ConstantGlyphSizeScheme UML class description

The ConstantGlyphSizeScheme class extends the GlyphSizeScheme class. It defines glyphs that are scaled using a constant value.

4.2.3.31.1 Attributes

Name	Access	Type	Description
size	public	NumberProperty	The property defining the color

4.2.3.31.2 Methods

Name:	<<constructor>> ConstantGlyphSizeScheme
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

4.2.3.32 ScaledGlyphSizeScheme

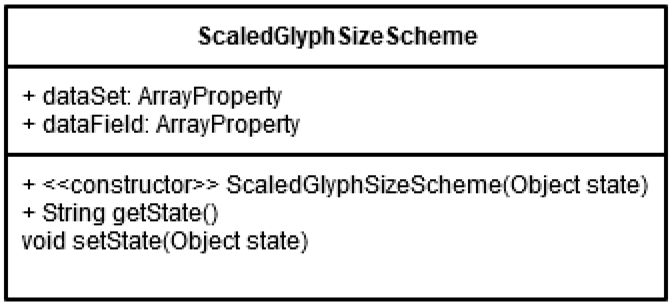

ScaledGlyph Size Scheme
+ dataSet: ArrayProperty
+ dataField: ArrayProperty
+ <<constructor>> ScaledGlyphSizeScheme(Object state)
+ String getState()
void setState(Object state)

Figure 66. ScaledGlyphSizeScheme UML class description

The ScaledGlyphSizeScheme class extends the GlyphSizeScheme class. It defines glyphs that are scaled based on values of a field in a DataSet.

4.2.3.32.1 Attributes

Name	Access	Type	Description
dataSet	public	ArrayProperty	The DataSet that contains the field for scaling
dataField	public	ArrayProperty	The field from the selected DataSet which determines how to scale the size of the glyphs

4.2.3.32.2 Methods

Name:	<<constructor>> ScaledGlyphSizeScheme
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the property
Description:	Returns a JSON stringified representation of the state of the property and its attributes

Name:	setState
Input:	Object state : The new state for the property
Output:	void
Description:	Sets the attributes of the property based on the given state

4.2.4 Data Set Classes

4.2.4.1 DataSet

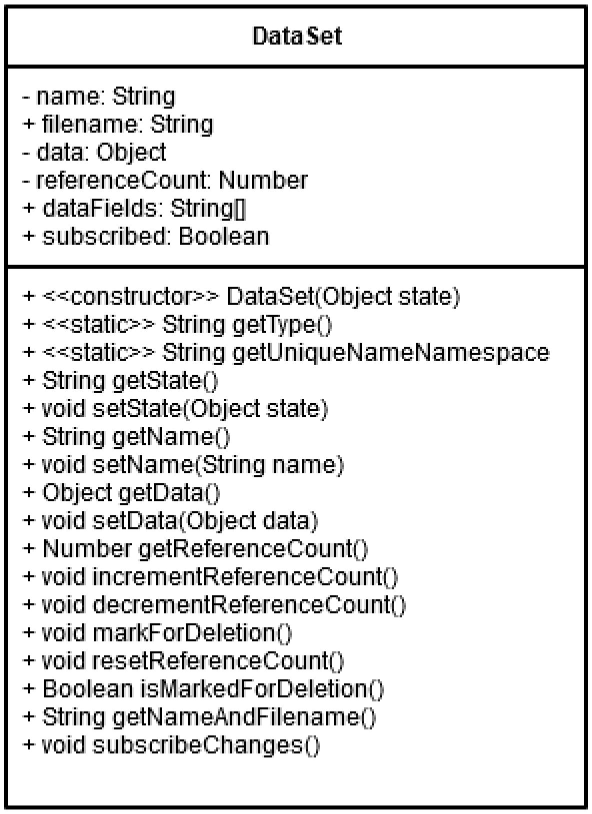

DataSet
- name: String
+ filename: String
- data: Object
- referenceCount: Number
+ dataFields: String[]
+ subscribed: Boolean
+ <<constructor>> DataSet(Object state)
+ <<static>> String getType()
+ <<static>> String getUniqueNameNamespace
+ String getState()
+ void setState(Object state)
+ String getName()
+ void setName(String name)
+ Object getData()
+ void setData(Object data)
+ Number getReferenceCount()
+ void incrementReferenceCount()
+ void decrementReferenceCount()
+ void markForDeletion()
+ void resetReferenceCount()
+ Boolean isMarkedForDeletion()
+ String getNameAndFilename()
+ void subscribeChanges()

Figure 67. DataSet UML class description

The DataSet class defines the attributes and methods necessary to maintain data from a file from the server. An instance of this class is referred to as a DataSource.

4.2.4.1.1 Attributes

Name	Access	Type	Description
name	private	String	The name of the data set
filename	public	String	The name of the file for the data set
data	private	Object	The data contained in the file
referenceCount	private	Number	The number of widgets that bind this data set
dataFields	public	String[]	The data fields in the data
subscribed	public	Boolean	Indicates if the DataSet has subscribed changes so it is not subscribed more than once

4.2.4.1.2 Methods

Name:	<<constructor>> DataSet
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	<<static>> getType
Input:	void
Output:	String : the type of the widget view model
Description:	Returns the type of the widget view model.

Name:	<<static>> getUniqueNameNamespace
Input:	void
Output:	String : the namespace
Description:	Returns the namespace in which all DataSet names must be unique

Name:	getState
Input:	void
Output:	String : A string representing the data set
Description:	Returns a JSON stringified representation of the state of the data set and its attributes

Name:	setState
Input:	Object state : The new state for the data set
Output:	void
Description:	Sets the attributes of the data set based on the given state

Name:	getName
Input:	void
Output:	String : The name of the DataSet
Description:	Returns a string containing the name of the DataSet

Name:	setName
Input:	String : The name
Output:	void
Description:	Sets the name of the DataSet

Name:	getData
Input:	void
Output:	Object : The data
Description:	Returns the data contained in the file

Name:	setData
Input:	Object data : The data
Output:	void
Description:	Sets the value of the data attribute

Name:	getReferenceCount
Input:	void
Output:	Number : An integer representing the reference count
Description:	Returns the reference count for the DataSet

Name:	incrementReferenceCount
Input:	void
Output:	void
Description:	Increments the reference count

Name:	decrementReferenceCount
Input:	void
Output:	void
Description:	Decrements the reference count

Name:	markForDeletion
Input:	void
Output:	void
Description:	Marks this data set for deletion

Name:	resetReferenceCount
Input:	void
Output:	void
Description:	Sets the reference count to zero.

Name:	isMarkedForDeletion
Input:	void
Output:	Boolean : True if the data set is marked for deletion
Description:	Returns true if the data set is marked for deletion, otherwise false.

Name:	getNameAndFilename
Input:	void
Output:	String : Contains the name and file name
Description:	Returns a string containing the name and filename of the data set.

Name:	subscribeChanges
Input:	void
Output:	void
Description:	Creates knockout subscriptions so that history of properties can be tracked.

4.2.4.2 DataSubset

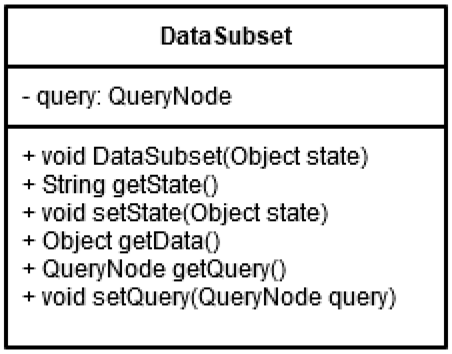

DataSubset
- query: QueryNode
+ void DataSubset(Object state)
+ String getState()
+ void setState(Object state)
+ Object getData()
+ QueryNode getQuery()
+ void setQuery(QueryNode query)

Figure 68. DataSet UML class description

The DataSubset class extends from the DataSet class. It takes the data from the DataSet and filters it using a query.

4.2.4.2.1 Attributes

Name	Access	Type	Description
query	private	QueryNode	The query to use to filter the original data

4.2.4.2.2 Methods

Name:	<<constructor>> DataSubset
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

Name:	getState
Input:	void
Output:	String : A string representing the data set
Description:	Returns a JSON stringified representation of the state of the data set and its attributes

Name:	setState
Input:	Object state : The new state for the data set
Output:	void
Description:	Sets the attributes of the data set based on the given state

Name:	getData
Input:	void
Output:	Object : The data
Description:	Returns the data contained in the file that matches the DataSubset's query.

Name:	getQuery
Input:	void
Output:	QueryNode : The query
Description:	Returns the query used to filter the original data from the file.

Name:	setQuery
Input:	QueryNode query : The query
Output:	void
Description:	Sets the query used to filter the original data from the file.

4.2.4.3 QueryNode

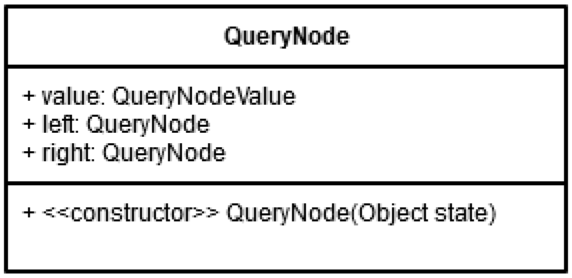


Figure 69. QueryNode UML class description

The QueryNode class is used to filter data in the DataSubset class.

4.2.4.3.1 Attributes

Name	Access	Type	Description
left	public	QueryNode	The left child of the node
right	public	QueryNode	The right child of the node
value	public	QueryNodeValue	The value of the query at this node

4.2.4.3.2 Methods

Name:	<<constructor>> QueryNode
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

4.2.4.4 QueryNodeValue

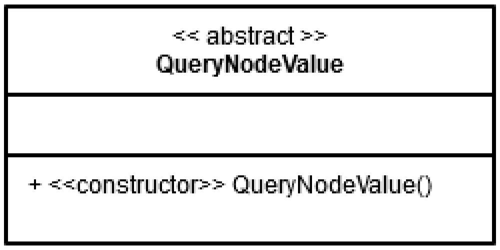


Figure 70. QueryNodeValue UML class description

The QueryNodeValue class is an abstract class to represent either a conditional statement or a logical operator that joins two conditional statements.

4.2.4.4.1 Attributes

Name	Access	Type	Description

4.2.4.4.2 Methods

Name:	<<constructor>> QueryNodeValue
Input:	Object state : The state of the attributes
Output:	void
Description:	Sets the values of the attributes

4.2.4.5 Condition

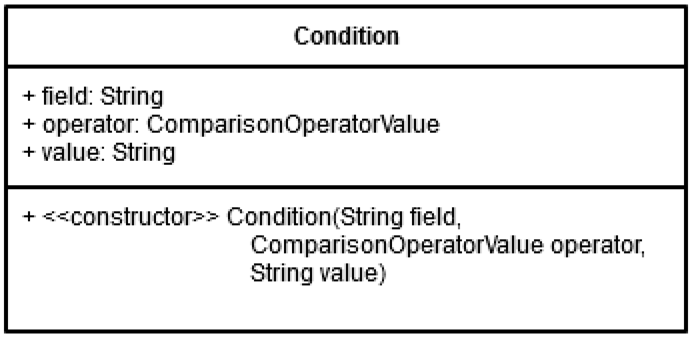

Condition
+ field: String
+ operator: ComparisonOperatorValue
+ value: String
+ <<constructor>> Condition(String field, ComparisonOperatorValue operator, String value)

Figure 71. Condition UML class description

The Condition class represents a conditional statement.

4.2.4.5.1 Attributes

Name	**Access**	**Type**	**Description**
field	public	String	The left hand side of the condition
operator	public	ComparisonOperatorValue	The comparison of the condition
value	public	String	The right hand side of the condition

4.2.4.5.2 Methods

Name:	<<constructor>> Condition
Input:	String field : The name of the data field ComparisonOperatorValue operator : The comparison String value : The user defined value
Output:	void
Description:	Sets the attributes of the condition.

4.2.4.6 LogicalOperator

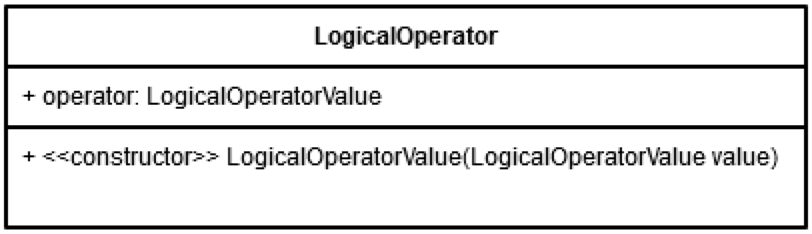

LogicalOperator
+ operator: LogicalOperatorValue
+ <<constructor>> LogicalOperatorValue(LogicalOperatorValue value)

Figure 72. LogicalOperator UML class description

The LogicalOperator class stores a logical operator value.

4.2.4.6.1 Attributes

Name	**Access**	**Type**	**Description**
operator	public	LogicalOperator	The operator value

4.2.4.6.2 Methods

Name:	<<constructor>> LogicalOperator
Input:	LogicalOperator : The value of the operator
Output:	void
Description:	Sets the attributes of the logical operator.

4.2.5 Action Classes

4.2.5.1 Action

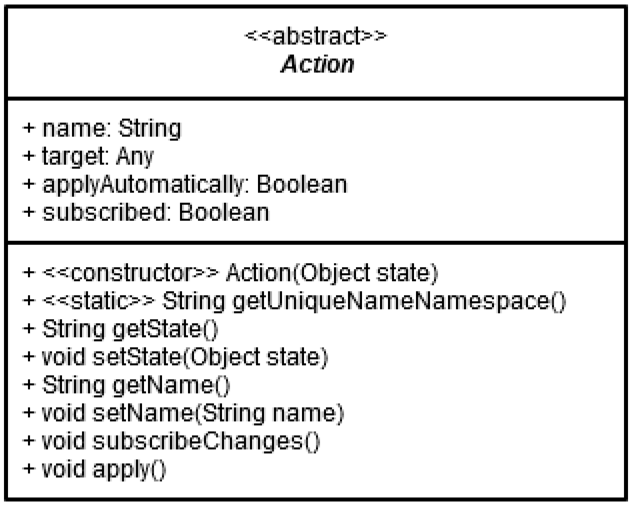

<<abstract>>
Action
+ name: String
+ target: Any
+ applyAutomatically: Boolean
+ subscribed: Boolean
+ <<constructor>> Action(Object state)
+ <<static>> String getUniqueNameNamespace()
+ String getState()
+ void setState(Object state)
+ String getName()
+ void setName(String name)
+ void subscribeChanges()
+ void apply()

Figure 73. Action UML class description

Action is an abstract class that defines a change affecting a target widget.

4.2.5.1.1 Attributes

Name	Access	Type	Description
name	private	String	The name of the Action
target	public	Any	The affected component or data subset
applyAutomatically	public	Boolean	True if the action is fired on load
subscribed	public	Boolean	Indicates if the Action has subscribed changes so it is not subscribed more than once

4.2.5.1.2 Methods

Name:	<<constructor>> Action
Input:	Object state
Output:	void
Description:	Sets the attributes of the action.

Name:	<<static>> getUniqueNameNamespace
Input:	void
Output:	String : the namespace
Description:	Returns the namespace in which all Action names must be unique

Name:	getState
Input:	void
Output:	String : A string representing the action
Description:	Returns a JSON stringified representation of the state of the action and its attributes

Name:	setState
Input:	Object state : The new state for the action
Output:	void
Description:	Sets the attributes of the action based on the given state

Name:	getName
Input:	void
Output:	String : The action name
Description:	Returns the name of the action.

Name:	setName
Input:	String name : The new name for the action
Output:	void
Description:	Sets the name of the action to the supplied value.

Name:	subscribeChanges
Input:	void
Output:	void
Description:	Creates knockout subscriptions so that history of properties can be tracked.

Name:	<<abstract>> apply

Input:	void
Output:	void
Description:	Applies the action changes to the target.

4.2.5.2 PropertyAction

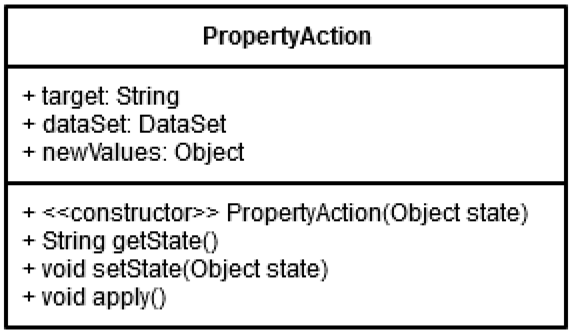


Figure 74. PropertyAction UML class description

PropertyAction is a class that extends from the Action class. It defines an action where the target is a Component and one or more properties of that Component are changed.

4.2.5.2.1 Attributes

Name	**Access**	**Type**	**Description**
target	public	String	The name of the component whose properties the action changes
dataSet	public	DataSet	The DataSet which the action uses
newValues	public	Object	A map from property names to values

4.2.5.2.2 Methods

Name:	<<constructor>>
Input:	Object state
Output:	void
Description:	Sets the attributes of the action.

Name:	getState
Input:	void
Output:	String : A string representing the aciton
Description:	Returns a JSON stringified representation of the state of the action and its attributes

Name:	setState
Input:	Object state : The new state for the action
Output:	void
Description:	Sets the attributes of the action based on the given state

Name:	apply
Input:	void
Output:	void
Description:	Applies the action changes to the target.

4.2.5.3 QueryAction

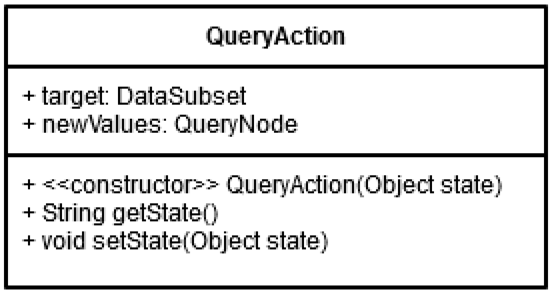

QueryAction
+ target: DataSubset
+ newValues: QueryNode
+ <<constructor>> QueryAction(Object state)
+ String getState()
+ void setState(Object state)

Figure 75. QueryAction UML class description

QueryAction is a class that extends from Action. It defines an action where the target is a DataSubset and the query of that DataSubset is changed.

4.2.5.3.1 Attributes

Name	**Access**	**Type**	**Description**
target	public	DataSubset	The DataSubset that the action modifies
newValue	public	QueryNode	The new Query to use

4.2.5.3.2 Methods

Name:	<<constructor>>
Input:	Object state
Output:	void
Description:	Sets the attributes of the action.

Name:	getState
Input:	void
Output:	String : A string representing the action
Description:	Returns a JSON stringified representation of the state of the action and its attributes

Name:	setState
Input:	Object state : The new state for the action
Output:	void
Description:	Sets the attributes of the action based on the given state

4.2.6 Event Classes

4.2.6.1 Event

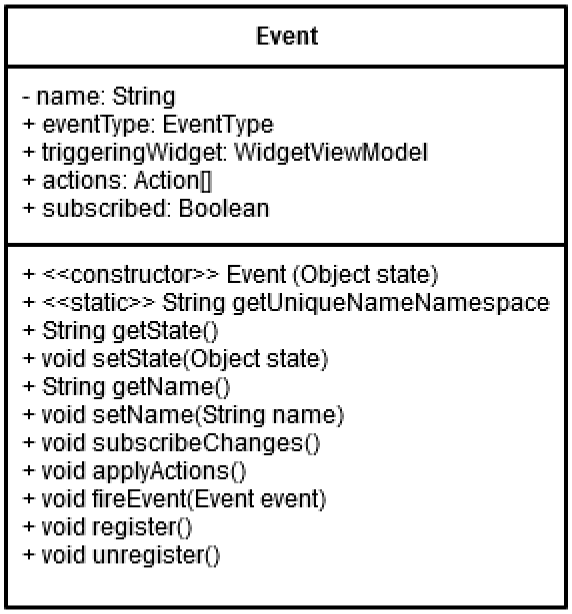

Event
- name: String
+ eventType: EventType
+ triggeringWidget: WidgetViewModel
+ actions: Action[]
+ subscribed: Boolean
+ <<constructor>> Event (Object state)
+ <<static>> String getUniqueNameNamespace
+ String getState()
+ void setState(Object state)
+ String getName()
+ void setName(String name)
+ void subscribeChanges()
+ void applyActions()
+ void fireEvent(Event event)
+ void register()
+ void unregister()

Figure 76. Event UML class description

The Event class associates a given Action object with a triggering widget and user event.

4.2.6.1.1 Attributes

Name	Access	Type	Description
name	private	String	The name of the Event
eventType	public	EventType	The type of event to trigger
triggeringWidget	public	WidgetViewModel	The Widget that triggers the event
actions	public	Action[]	The actions that the event triggers
subscribed	public	Boolean	Indicates if the Event has subscribed changes so it is not subscribed more than once

4.2.6.1.2 Methods

Name:	<<constructor>> Event
Input:	Object state : The state of the event attributes
Output:	void
Description:	Sets the event attributes.

Name:	<<static>> getUniqueNameNamespace
Input:	void
Output:	String : the namespace
Description:	Returns the namespace in which all Event names must be unique

Name:	getState
Input:	void
Output:	String : A string representing the event
Description:	Returns a JSON stringified representation of the state of the event and its attributes

Name:	setState
Input:	Object state : The new state for the event
Output:	void
Description:	Sets the attributes of the event based on the given state

Name:	getName
Input:	void
Output:	String : The event name
Description:	Returns the name of the event.

Name:	setName
Input:	String name : The new name for the event
Output:	void
Description:	Sets the name of the event to the supplied value.

Name:	subscribeChanges
Input:	void
Output:	void
Description:	Creates knockout subscriptions so that history of properties can be tracked.

Name:	applyActions
Input:	void
Output:	void
Description:	Executes the actions of the event

Name:	fireEvent
Input:	Event event
Output:	void
Description:	Processes mouse events.

Name:	register
Input:	void
Output:	void
Description:	Adds the DOM event to the target DOM element.

Name:	unregister
Input:	void
Output:	void
Description:	Removes the DOM event from the target DOM element.

4.2.6.2 Trigger

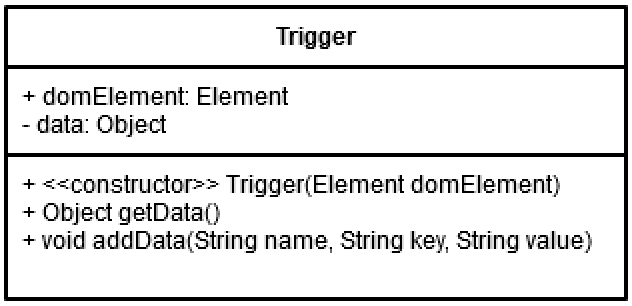

Trigger
+ domElement: Element
- data: Object
+ <<constructor>> Trigger(Element domElement)
+ Object getData()
+ void addData(String name, String key, String value)

Figure 77. Trigger UML class description

The Trigger class encapsulates the DOM element used for a particular widget and communicates widget-specific data with Action.

4.2.6.2.1 Attributes

Name	Access	Type	Description
domElement	public	Element	The DOM element associated with the trigger
data	private	Object	An object with properties containing information about where the user event was fired and any widget-specific information that needs to be sent to an Action.

4.2.6.2.2 Methods

Name:	<<constructor>> Trigger
Input:	Element domElement : the DOM element
Output:	void
Description:	Sets the event attributes.

Name:	getData
Input:	void
Output:	Object: The trigger data
Description:	Returns the trigger data

Name:	addData
Input:	String name: A name for the root level object attribute in which to store the supplied key-value pair. String key: The name of the property to store on the trigger's data object. String value: The value to associate with the supplied key.
Output:	void
Description:	Adds a data field to the data object.

4.3 Client-side Enumerations

4.3.1 WidgetTemplateName

BUTTON
TEXTBLOCK
US_MAP
LINE_GRAPH
TOOLTIP

4.3.2 PropertyTemplateName

LIST
STRING
STRING_DISPLAY
LONG_STRING
LONG_STRING_DISPLAY
NUMBER
NUMBER_DISPLAY
BOOLEAN
BOOLEAN_DISPLAY
ARRAY
ARRAY_DISPLAY
COLORING
COLORING_DISPLAY
GLYPH_SIZE
GLYPH_SIZE_DISPLAY
BUTTON
BUTTON_DISPLAY

4.3.3 ColoringSchemeType

SOLID_COLORING
FOUR_COLORING
GRADIENT_COLORING

4.3.4 GlyphSizeSchemeType

CONSTANT_SIZE
SCALED_SIZE

4.3.5 EventType

CLICK
MOUSEOVER
HOVER

4.3.5 ActionType

PROPERTY_ACTION
QUERY_ACTION

4.3.6 ComparisonOperatorValue

LESS_THAN
LESS_THAN_OR_EQUAL
GREATER_THAN
GREATER_THAN_OR_EQUAL
EQUAL
NOT_EQUAL

4.3.7 LogicalOperatorValue

AND
OR

4.3.5 SelectedType

PROJECT
COMPONENT
DATA
ACTION
EVENT

4.4 Detailed Server-side Class Descriptions

4.4.1 ISerializer

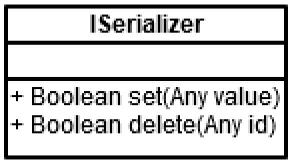


Figure 78. ISerializer UML class description

The ISerializer interface defines the method signatures for serializing an element into the database

4.4.1.1 Attributes

Name	Access	Type	Description

4.4.1.2 Methods

Name:	set
Input:	Any value: Value to store into the database
Output:	Boolean: TRUE on success, FALSE otherwise
Description:	Stores a value into the database

Name:	delete
Input:	Any id: Key of the entry to delete from the database
Output:	Boolean: TRUE on success, FALSE otherwise
Description:	Removes an entry from the database

4.4.2 IDeserializer

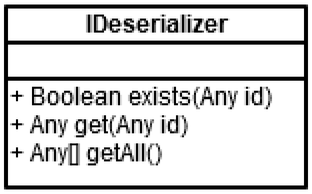


Figure 79. IDeseriaizer UML class description

The IDeserializer interface defines the method signatures for deserializing an element from the database..

4.4.2.1 Attributes

Name	Access	Type	Description

4.4.2.2 Methods

Name:	exists
Input:	Any id: Key of the entry to look for
Output:	Boolean: TRUE if project exists, FALSE otherwise
Description:	Returns whether an entry with the given key exists

Name:	get
Input:	Any id: Key of the entry to retrieve
Output:	Any: Deserialized entry from the database that matches the given key
Description:	Retrieves a deserialized entry from the database

Name:	getAll
Input:	void
Output:	Any[]: Deserialized entries from the database
Description:	Retrieves all deserialized entries from the database.

4.4.3 SQLiteProjectSerializer

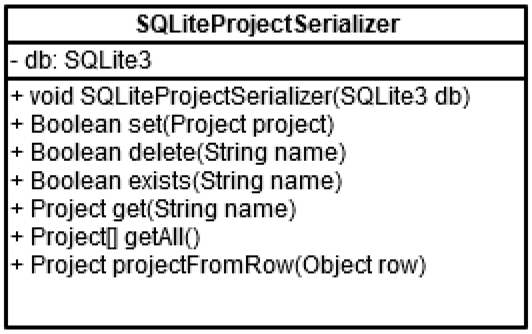

SQLiteProjectSerializer
- db: SQLite3
+ void SQLiteProjectSerializer(SQLite3 db)
+ Boolean set(Project project)
+ Boolean delete(String name)
+ Boolean exists(String name)
+ Project get(String name)
+ Project[] getAll()
+ Project projectFromRow(Object row)

Figure 80. SQLiteProjectSerializer UML class description

The SQLiteProjectSerializer class handles serialization and deserialization of projects to a SQLite3 database.

4.4.3.1 Attributes

Name	Access	Type	Description
db	private	SQLite3	The database connection with which to serialize..

4.4.3.2 Methods

Name:	SQLiteProjectSerializer
Input:	SQLite3 db: Database connection to use
Output:	void
Description:	Constructs a new SQLiteProjectSerializer.

Name:	set
Input:	Project project: Project to serialize
Output:	Boolean: TRUE on success, FALSE otherwise
Description:	Serializes the given project to the database.

Name:	delete
Input:	String name: Project name
Output:	Boolean: TRUE on success, FALSE otherwise
Description:	Deletes the given project from the database.

Name:	exists
Input:	String name: Project name
Output:	Boolean: TRUE if project exists, FALSE otherwise
Description:	Returns whether a project with the given name exists

Name:	get
Input:	String name: Project name
Output:	Project: Project retrieved from the database
Description:	Returns the project that has the given name.

Name:	getAll
Input:	void
Output:	Project[]: Projects retrieved from the database
Description:	Returns a list of all existing projects

Name:	projectFromRow
Input:	Object row: Associative array with values fetched from the database
Output:	Project: Project object created from values given
Description:	Helper function to construct a Project from the results of a database query.

4.4.4 Serializer

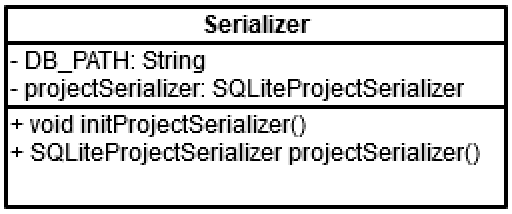

Serializer
- DB_PATH: String
- projectSerializer: SQLiteProjectSerializer
+ void initProjectSerializer()
+ SQLiteProjectSerializer projectSerializer()

Figure 81. Serializer UML class description

The Serializer class creates and gives access to the database serializer.

4.4.4.1 Attributes

Name	Access	Type	Description
DB_PATH	private	String	Path to the SQLite3 database file.
projectSerializer	private	SQLiteProjectSerializer	Object used to serialize to the database

4.4.4.2 Methods

Name:	initProjectSerializer
Input:	void
Output:	void
Description:	Initializes the database connection and serializer

Name:	projectSerializer
Input:	void
Output:	SQLiteProjectSerializer: Project serializer
Description:	Returns the lazily initialized project serializer.

4.4.5 Project

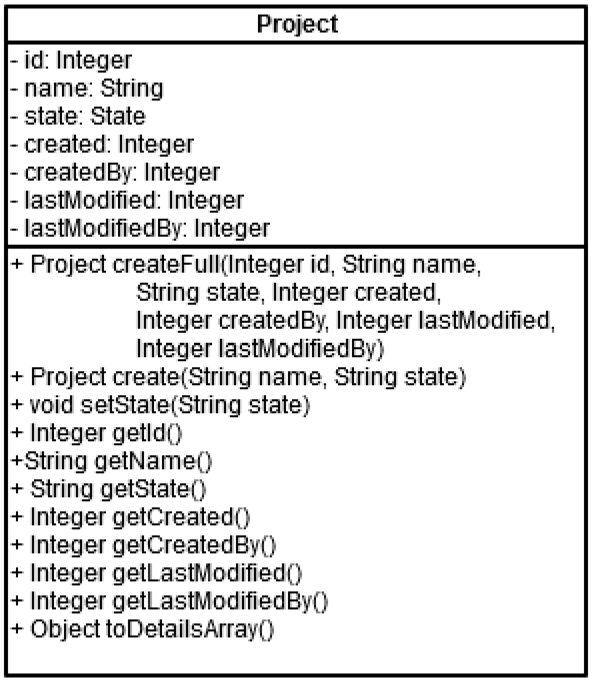

Project
- id: Integer
- name: String
- state: State
- created: Integer
- createdBy: Integer
- lastModified: Integer
- lastModifiedBy: Integer
+ Project createFull(Integer id, String name, String state, Integer created, Integer createdBy, Integer lastModified, Integer lastModifiedBy)
+ Project create(String name, String state)
+ void setState(String state)
+ Integer getId()
+String getName()
+ String getState()
+ Integer getCreated()
+ Integer getCreatedBy()
+ Integer getLastModified()
+ Integer getLastModifiedBy()
+ Object toDetailsArray()

Figure 82. Project UML class description

The Project class contains the information persisted for a project.

4.4.5.1 Attributes

Name	Access	Type	Description
id	private	Integer	Unique id of the project
name	private	String	Unique name of the project
state	private	State	The client side state of the project
created	private	Integer	Unix timestamp of when the project was created
createdBy	private	Integer	Unique id of the user who created the project
lastModified	private	Integer	Unix timestamp of when the project was last modified
lastModifiedBy	private	Integer	Unique id of the user who last modified the project

4.4.5.2 Methods

Name:	createFull
Input:	Integer id: Unique id of the project String name: Unique name of the project String state: The client side state of the project Integer created: Unix timestamp of when the project was created Integer createdBy: Unique id of the user who created the project Integer lastModified: Unix timestamp of when the project was last modified Integer lastModifiedBy: Unique id of the user who last modified the project
Output:	Project: Project object with all the information set
Description:	Creates a new project object

Name:	create
Input:	String name: Unique name of the project String state: The client side state of the project
Output:	Project: Project object with name and state set
Description:	Creates a new project object

Name:	setState
Input:	String state: The client side state of the project
Output:	void
Description:	Sets the state of the project

Name:	getId
Input:	void
Output:	Integer: Unique id of the project
Description:	Returns the id of the project.

Name:	getName
Input:	void
Output:	String: Unique String of the project
Description:	Returns the name of the project.

Name:	getState
Input:	void
Output:	String: The client side state of the project
Description:	Returns the state of the project.

Name:	getCreated
Input:	void
Output:	Integer: Unix timestamp of when the project was created
Description:	Returns the creation time of the project.

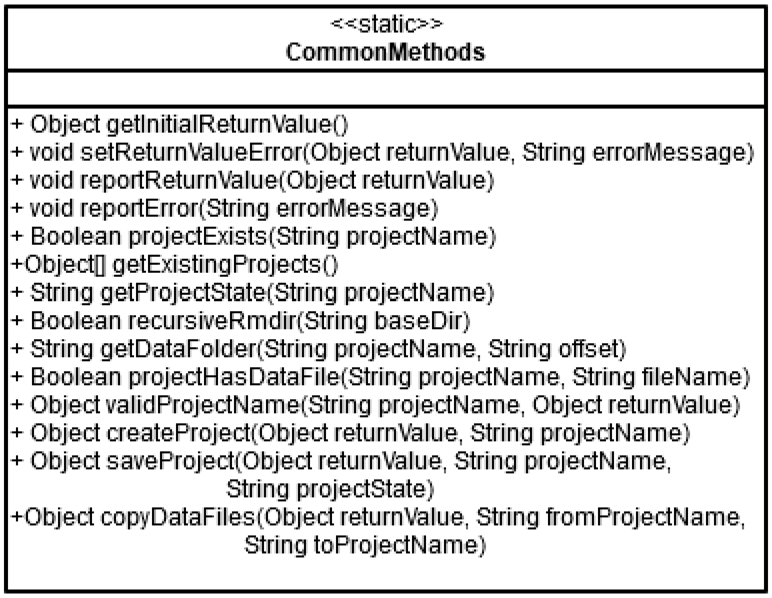
Name:	getCreatedBy
Input:	void
Output:	Integer: Unique id of the user who created the project
Description:	Returns the id of the user who created the project.

Name:	getLastModified
Input:	void
Output:	Integer: Unix timestamp of when the project was last modified
Description:	Returns the last modified time of the project.

Name:	getLastModifiedBy
Input:	void
Output:	Integer: Unique id of the user who last modified the project
Description:	Returns the id of the user who last modified the project.

Name:	toDetailsArray
Input:	void
Output:	Object: Associative array of details to return to the client on request
Description:	Returns associative array of details to return to the client on request.

4.4.6 CommonMethods



```
+ Object getInitialReturnValue()
+ void setReturnValueError(Object returnValue, String errorMessage)
+ void reportReturnValue(Object returnValue)
+ void reportError(String errorMessage)
+ Boolean projectExists(String projectName)
+Object[] getExistingProjects()
+ String getProjectState(String projectName)
+ Boolean recursiveRmdir(String baseDir)
+ String getDataFolder(String projectName, String offset)
+ Boolean projectHasDataFile(String projectName, String fileName)
+ Object validProjectName(String projectName, Object returnValue)
+ Object createProject(Object returnValue, String projectName)
+ Object saveProject(Object returnValue, String projectName,
                     String projectState)
+Object copyDataFiles(Object returnValue, String fromProjectName,
                      String toProjectName)
```

Figure 83. CommonMethods UML class description

The CommonMethods class contains helper functions for requests made to the server.

4.4.6.1 Attributes

Name	Access	Type	Description

4.4.6.2 Methods

Name:	getInitialReturnValue
Input:	void
Output:	Object: Return object with default values
Description:	Initializes the return object with default values.

Name:	setReturnValueError
Input:	Object returnValue: Return value to modify String errorMessage: Message describing the error encountered
Output:	void
Description:	Modifies return value to show an unsuccessful operation with the given error message.

Name:	reportReturnValue
Input:	Object returnValue: Return value to respond with
Output:	void
Description:	Responds to a server request with the given return value.

Name:	reportError
Input:	String errorMessage: Message describing the error encountered
Output:	void
Description:	Responds to a server request with a return value that shows an unsuccessful operation with the given error message.

Name:	projectExists
Input:	String projectName: Name of the project
Output:	Boolean: TRUE if the project exists, FALSE otherwise
Description:	Checks for the existence of the given project by name.

Name:	getExistingProjects
Input:	void
Output:	Object[]: Details for all of the existing projects
Description:	Returns the details for all of the existing projects.

Name:	getProjectState
Input:	String projectName: Name of the project
Output:	String: JSON state of the project
Description:	Returns the stored state of a project.

Name:	recursiveRmdir
Input:	String baseDir: Path to the directory to remove recursively
Output:	Boolean: TRUE if the directory did not exist or was successfully removed, FALSE otherwise
Description:	Recursively removes a directory.

Name:	getDataFolder
Input:	String projectName: Name of the project String offset: Prefix to add to the path
Output:	String: Path to the data folder for the given project.
Description:	Returns the data folder for the given project.

Name:	projectHasDataFile
Input:	String projectName: Name of the project String fileName: Name of data file
Output:	Boolean: TRUE if the given project has a file with the given name, FALSE otherwise
Description:	Determines if a given project has a file with the given name.

Name:	validProjectName
Input:	String projectName: Name of the project Object returnValue: Current return value
Output:	Object: Updated return value indicating whether the name already exists or not
Description:	Checks to see if the given project name is valid

Name:	createProject
Input:	Object returnValue: Current return value String projectName: Name of the project
Output:	Object: Updated return value
Description:	Attempts to create a new project with the given name

Name:	saveProject
Input:	Object returnValue: Current return value String projectName: Name of the project String projectState: State of the project
Output:	Object: Updated return value
Description:	Saves the given project with the given state.

Name:	copyDataFiles
Input:	Object returnValue: Current return value String fromProjectName: Name of the project to copy from String toProjectName: Name of the project to copy to
Output:	Object: Updated return value
Description:	Copies the data files from one project’s data folder to another’s.

4.5 JSON Request-response

4.5.1 createProject

The createProject request is responsible for the creation of new projects.

4.5.1.1 Request

Name	Type	Description
project	String	Name of the project to create

4.5.1.2 Response

Name	Type	Description
errorMessage	String	Message describing any errors
success	Boolean	TRUE if the operation was successful, FALSE otherwise
projectName	String	Name of the project created
projectState	String	State of the project created
dataFolder	String	Path of the server side data folder of the project created

4.5.2 loadProject

The loadProject request is responsible for retrieving the state and dataFolder path for the given project.

4.5.2.1 Request

Name	Type	Description
project	String	Name of the project to load

4.5.2.2 Response

Name	Type	Description
errorMessage	String	Message describing any errors
success	Boolean	TRUE if the operation was successful, FALSE otherwise
projectName	String	Name of the project loaded
projectState	String	State of the project loaded
dataFolder	String	Path of the server side data folder of the project loaded

4.5.3 saveProject

The saveProject request is responsible for saving new states of projects.

4.5.3.1 Request

Name	Type	Description
project	String	Name of the project to save
state	String	State of the project to save

4.5.3.2 Response

Name	Type	Description
errorMessage	String	Message describing any errors
success	Boolean	TRUE if the operation was successful, FALSE otherwise
projectName	String	Name of the project saved

4.5.4 saveProjectAs

The saveProjectAs request is responsible for saving the state of a project under a new name.

4.5.4.1 Request

Name	Type	Description
oldProject	String	Name of the project to copy from
project	String	Name of the project to save
state	String	State of the project to save

4.5.4.2 Response

Name	Type	Description
errorMessage	String	Message describing any errors
success	Boolean	TRUE if the operation was successful, FALSE otherwise
projectName	String	Name of the project saved
projectState	String	State of the project saved
dataFolder	String	Path of the server side data folder of the project saved

4.5.5 getExistingProjectDetails

The getExistingProjectDetails request is responsible for returning the details of all existing projects.

4.5.5.1 Request

Name	Type	Description

4.5.5.2 Response

Name	Type	Description
errorMessage	String	Message describing any errors
success	Boolean	TRUE if the operation was successful, FALSE otherwise
projects	Object[]	Array of details from *toDetailsArray()* for each project

4.5.6 deleteProject

The deleteProject request is responsible for deleting a given project.

4.5.6.1 Request

Name	Type	Description
project	String	Name of project to delete

4.5.6.2 Response

Name	Type	Description
errorMessage	String	Message describing any errors
success	Boolean	TRUE if the operation was successful, FALSE otherwise
projectName	String	Name of project deleted

4.5.7 uploadDataFile

The uploadDataFile request is responsible for uploading a data file to a given project.

4.5.7.1 Request

Name	Type	Description
project	String	Name of project to add a data file to
file	File	File to add to a project

4.5.7.2 Response

Name	Type	Description
errorMessage	String	Message describing any errors
success	Boolean	TRUE if the operation was successful, FALSE otherwise
filename	String	Name of the file uploaded to the project

4.5.8 deleteDataFile

The deleteDataFile request is responsible for removing a data file from a given project.

4.5.8.1 Request

Name	Type	Description
project	String	Name of project to remove a data file from
fileName	String	Name of data file to remove from project

4.5.8.2 Response

Name	Type	Description
errorMessage	String	Message describing any errors
success	Boolean	TRUE if the operation was successful, FALSE otherwise

5. Data Model

5.1. Database Schema

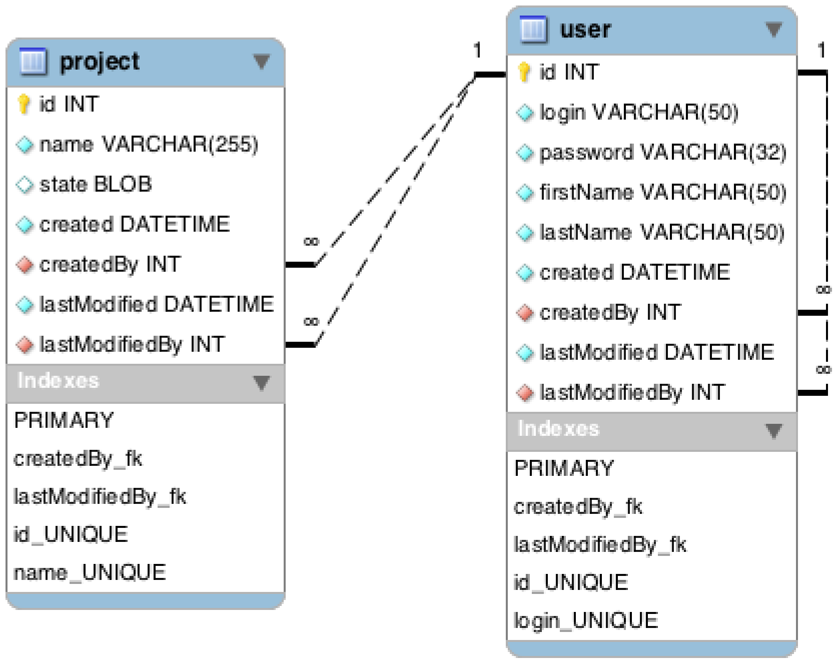


Figure 84. Relational database schema

The WAVED database contains two main tables: Project and User. The Project table stores relations for each project users create. Aside from creation metadata, a project relation contains a serialized data blob representing the state of the project. The User table stores relations for each of the user accounts in the WAVED system. Version 1.0 of WAVED does not support multiple user accounts, so the user table contains only the default user account information.

5.2 Saved State Schema

The WAVED system saves the projects in the database as JSON formatted text instead of creating data tables to represent the different objects. This section details the schema of the JSON object needed to save a project state.

5.2.1 ProjectState

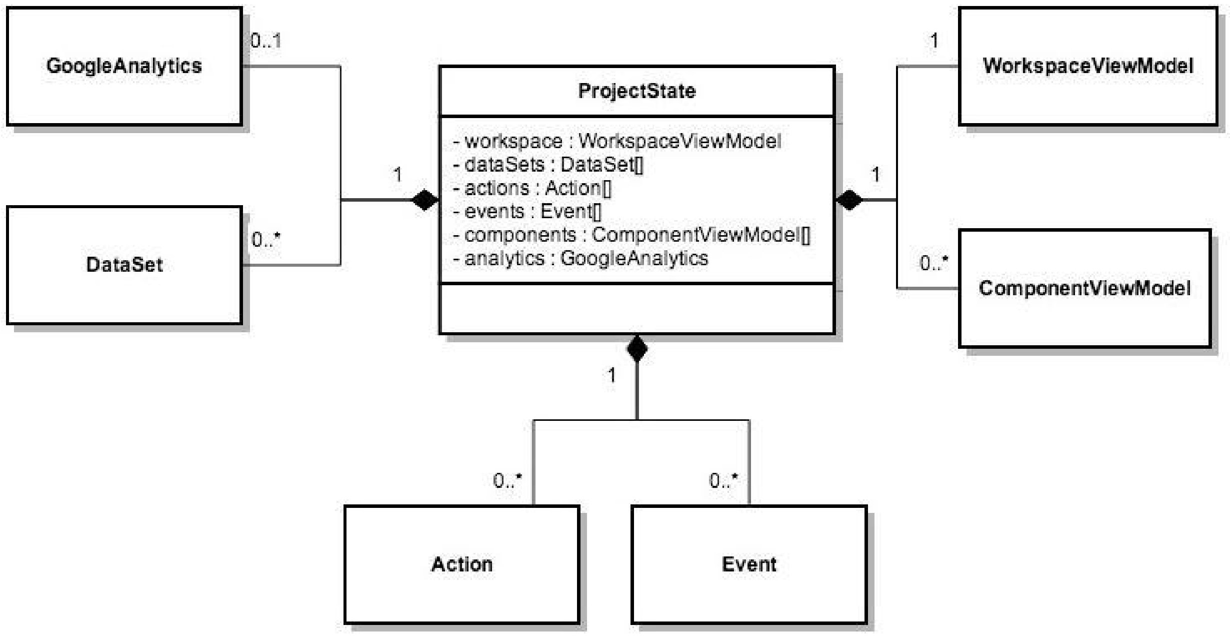


Figure 85. ProjectState saved state schema

Figure 85 represents the schema for saving the state of a project in the database. The ProjectState contains references to a WorkspaceViewModel, a GoogleAnalytics object, as well as all references to DataSets, ComponentViewModels, Actions, and Events.

5.2.2 ComponentViewModel

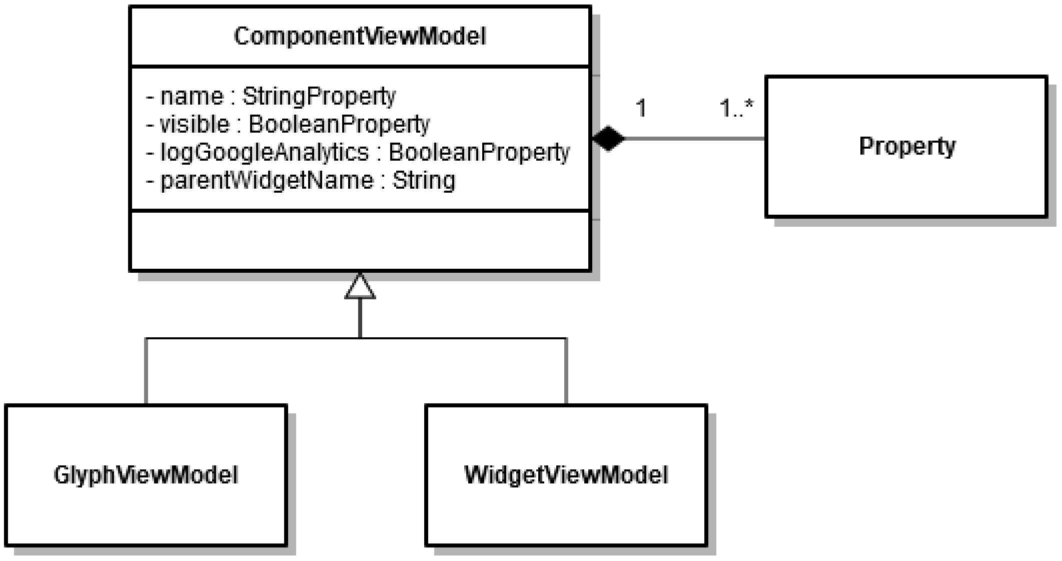

ComponentViewModel
- name : StringProperty
- visible : BooleanProperty
- logGoogleAnalytics : BooleanProperty
- parentWidgetName : String
1
1..*
Property
GlyphViewModel
WidgetViewModel

Figure 86. ComponentViewModel saved state schema

Figure 86 is the representation of the saved state schema for a ComponentViewModel. ComponentViewModel include instances of GlyphViewModels and WidgetViewModels as well as any Property objects the ViewModels use.

5.2.3 WidgetViewModel

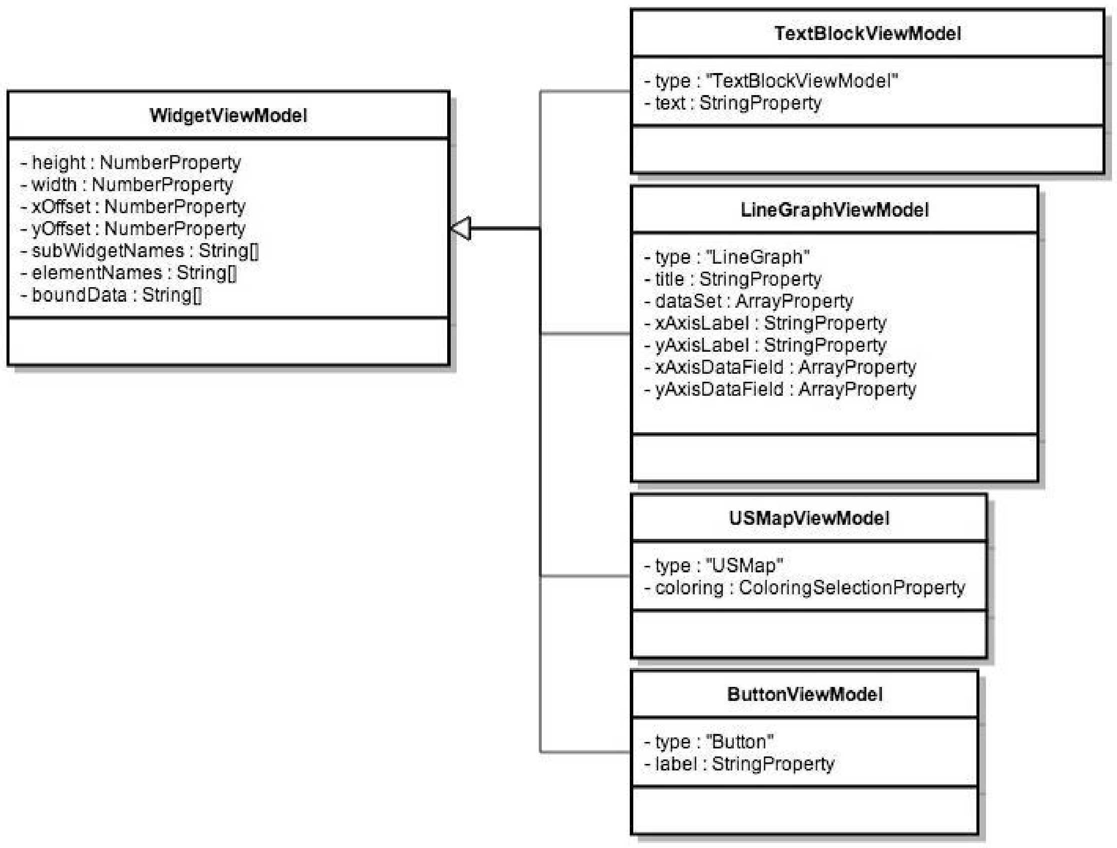

WidgetViewModel
- height : NumberProperty
- width : NumberProperty
- xOffset : NumberProperty
- yOffset : NumberProperty
- subWidgetNames : String[]
- elementNames : String[]
- boundData : String[]
TextBlockViewModel
- type : "TextBlockViewModel"
- text : StringProperty
LineGraphViewModel
- type : "LineGraph"
- title : StringProperty
- dataSet : ArrayProperty
- xAxisLabel : StringProperty
- yAxisLabel : StringProperty
- xAxisDataField : ArrayProperty
- yAxisDataField : ArrayProperty
USMapViewModel
- type : "USMap"
- coloring : ColoringSelectionProperty
ButtonViewModel
- type : "Button"
- label : StringProperty

Figure 87. WidgetViewModel saved state schema

Figure 87 shows the WidgetViewModel's saved state schema. WidgetViewModels represent instances of TextBlockViewModels, GraphViewModels (including LineGraphViewModels), USMapViewModels, and ButtonViewModels. All classes inheriting from WidgetViewModel have a type attribute represented by a static string unique to each ViewModel. This string is used to aid in the differentiation of WidgetViewModel types.

5.2.4 DataSet

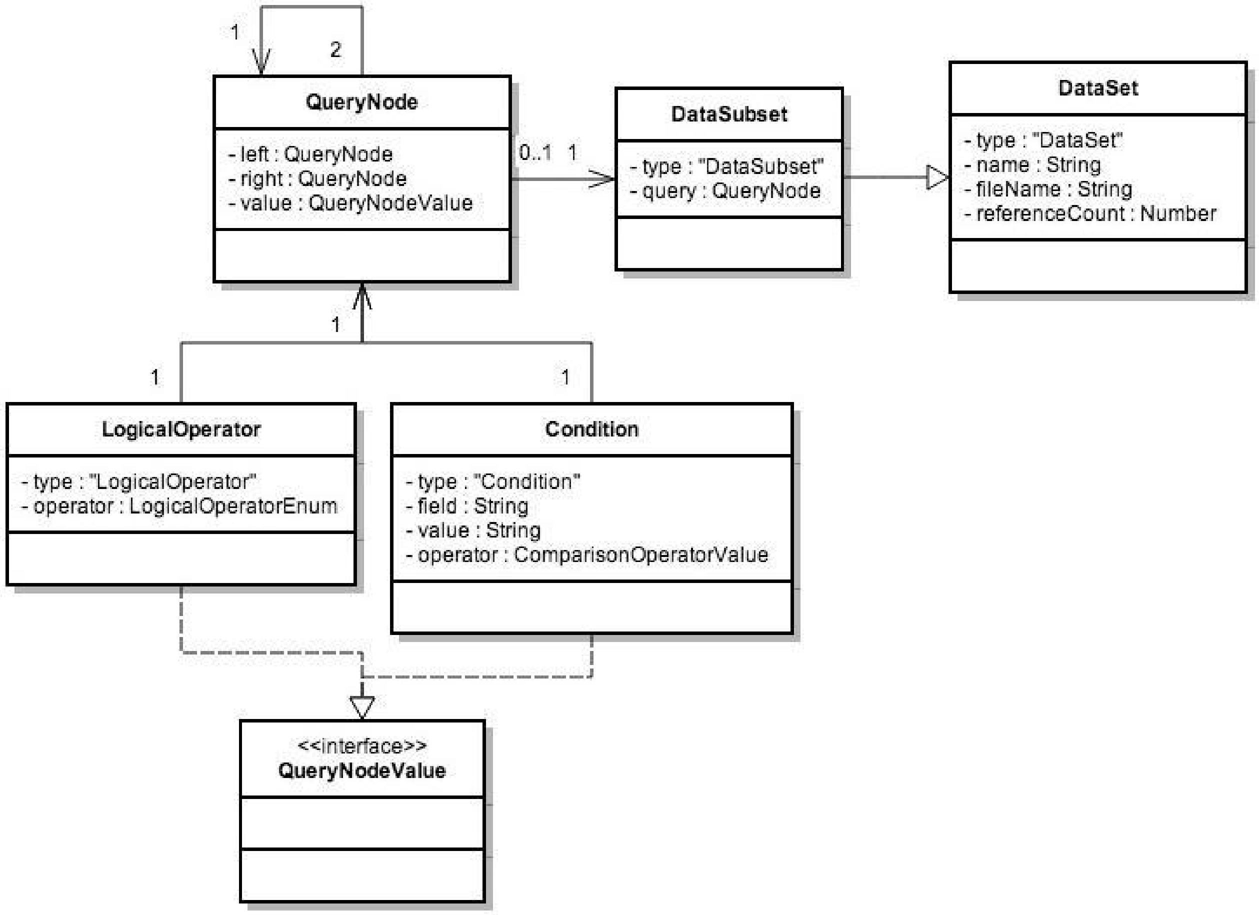


Figure 88. DataSet saved state schema

Figure 88 represents the saved state schema for DataSet, DataSubset, QueryNode, LogicalOperator, Condition, and QueryNodeValue. A QueryNode can recursively contain other QueryNodes that use LogicalOperators or Conditions. The LogicalOperator and Condition states have type strings that differentiate them from each other since they are both QueryNodeValue instances. DataSet and DataSubset also have type attributes to distinguish themselves from each other in the state schema.

5.2.5 Property

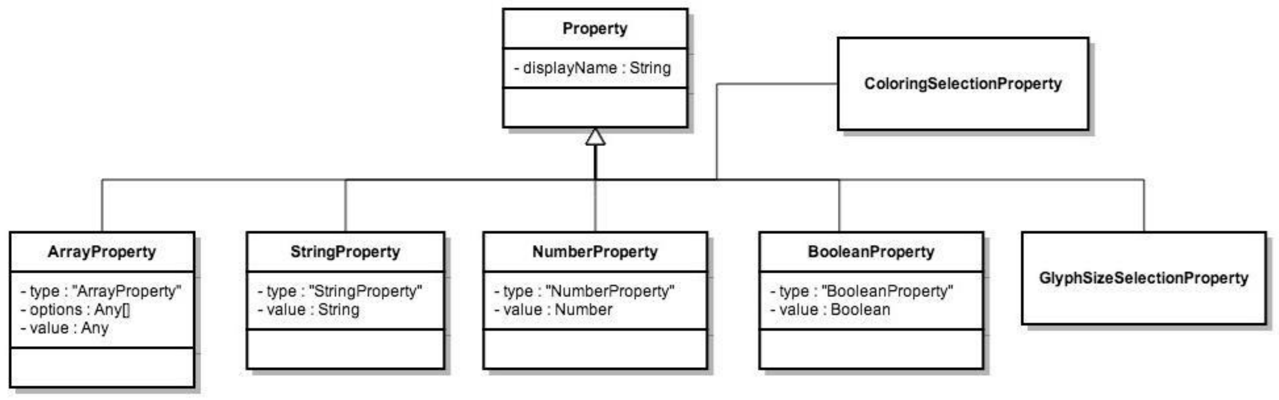


Figure 89. Property saved state schema

Figure 89 shows the state representations of all possible Property objects. Each Property object has a unique type attribute to differentiate different Property instances from each other. ColoringSelectionProperty and GlyphSizeSelectionProperty are more complex than the other Property types and are covered in 5.2.6 and 5.2.7.

5.2.6 ColoringSelectionProperty

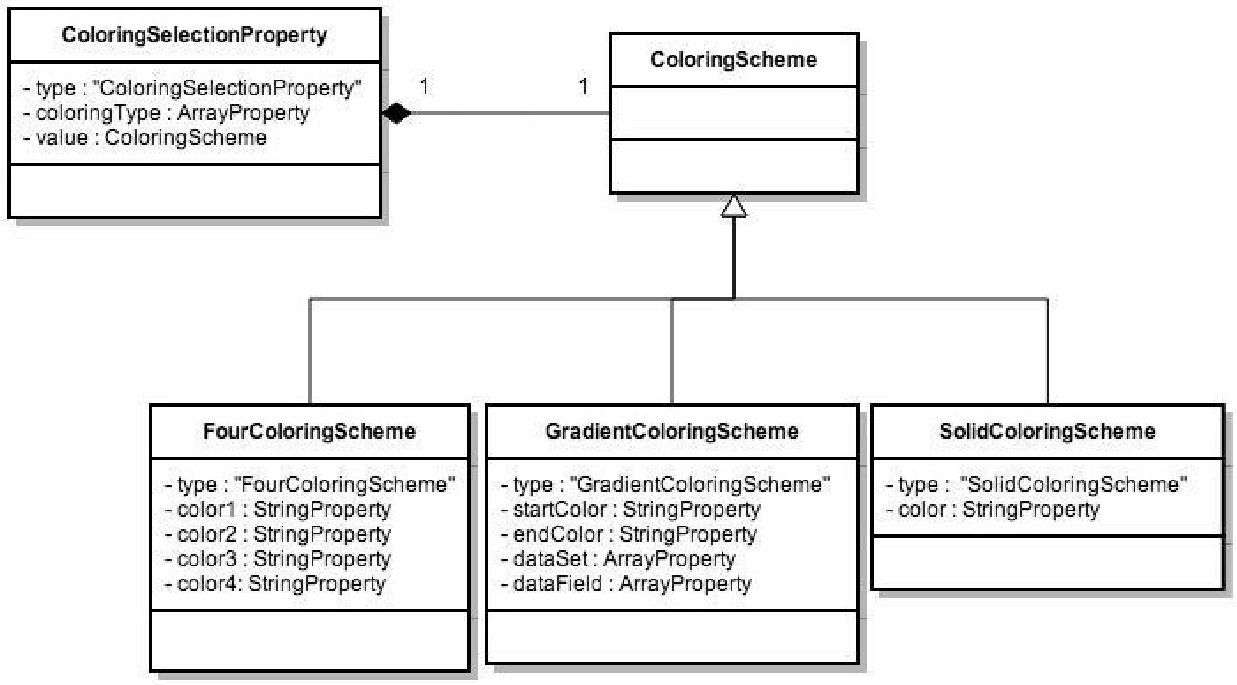

ColoringSelectionProperty
- type : "ColoringSelectionProperty"
- coloringType : ArrayProperty
- value : ColoringScheme
1
1
ColoringScheme
FourColoringScheme
- type : "FourColoringScheme"
- color1 : StringProperty
- color2 : StringProperty
- color3 : StringProperty
- color4: StringProperty
GradientColoringScheme
- type : "GradientColoringScheme"
- startColor : StringProperty
- endColor : StringProperty
- dataSet : ArrayProperty
- dataField : ArrayProperty
SolidColoringScheme
- type : "SolidColoringScheme"
- color : StringProperty

Figure 90. ColoringSelectionProperty saved state schema

Figure 90's ColoringSelectionProperty state represents any one of three possible ColoringScheme objects, each differentiated in the state representation by a unique type attribute.

5.2.7 GlyphSizeSelectionProperty

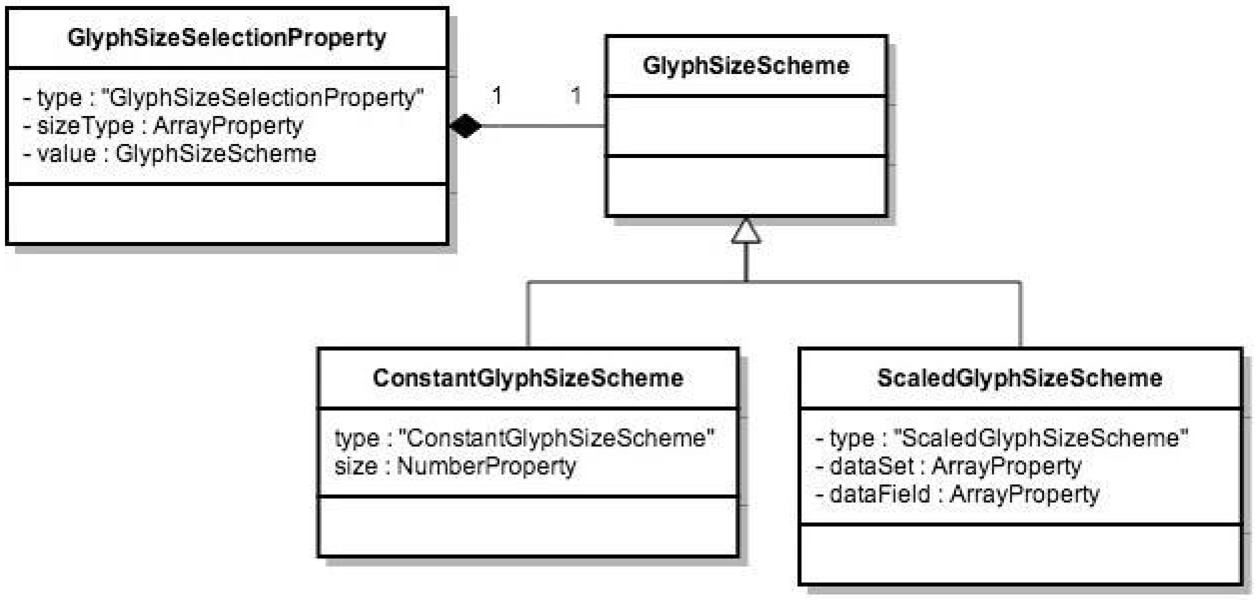

GlyphSizeSelectionProperty
- type : "GlyphSizeSelectionProperty"
- sizeType : ArrayProperty
- value : GlyphSizeScheme
1
1
GlyphSizeScheme
ConstantGlyphSizeScheme
type : "ConstantGlyphSizeScheme"
size : NumberProperty
ScaledGlyphSizeScheme
- type : "ScaledGlyphSizeScheme"
- dataSet : ArrayProperty
- dataField : ArrayProperty

Figure 91. GlyphSizeSelectionProperty saved state schema

Figure 91's GlyphSizeSelectionProperty state represents any one of three possible GlyphSizeScheme objects, each differentiated in the state representation by a unique type attribute.

5.2.8 WorkspaceViewModel

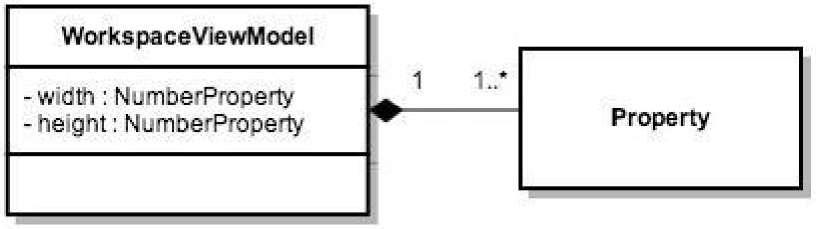


Figure 92. WorkspaceViewModel saved state schema

The WorkSpaceViewModel state is represented in Figure 92 and contains a width and height, both of which are NumberProperty types.

5.2.9 Google Analytics

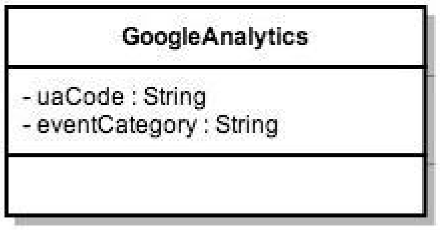


Figure 93. Google Analytics saved state schema

Figure 93 represents the schema for saving the state of the GoogleAnalytics objects within the project. It consists of the GoogleAnalytics class's uaCode and eventCategory attributes.

5.2.10 Event

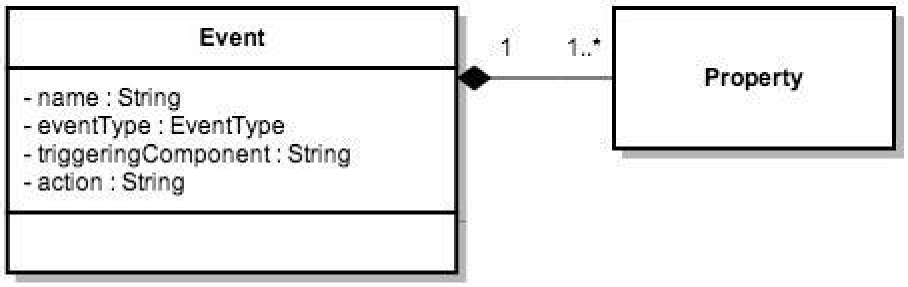


Figure 94. Event saved state schema

Figure 94 shows the saved state schema for an Event. It differs from the Event class's model by typing the triggeringComponent and action as String.

5.2.11 Action

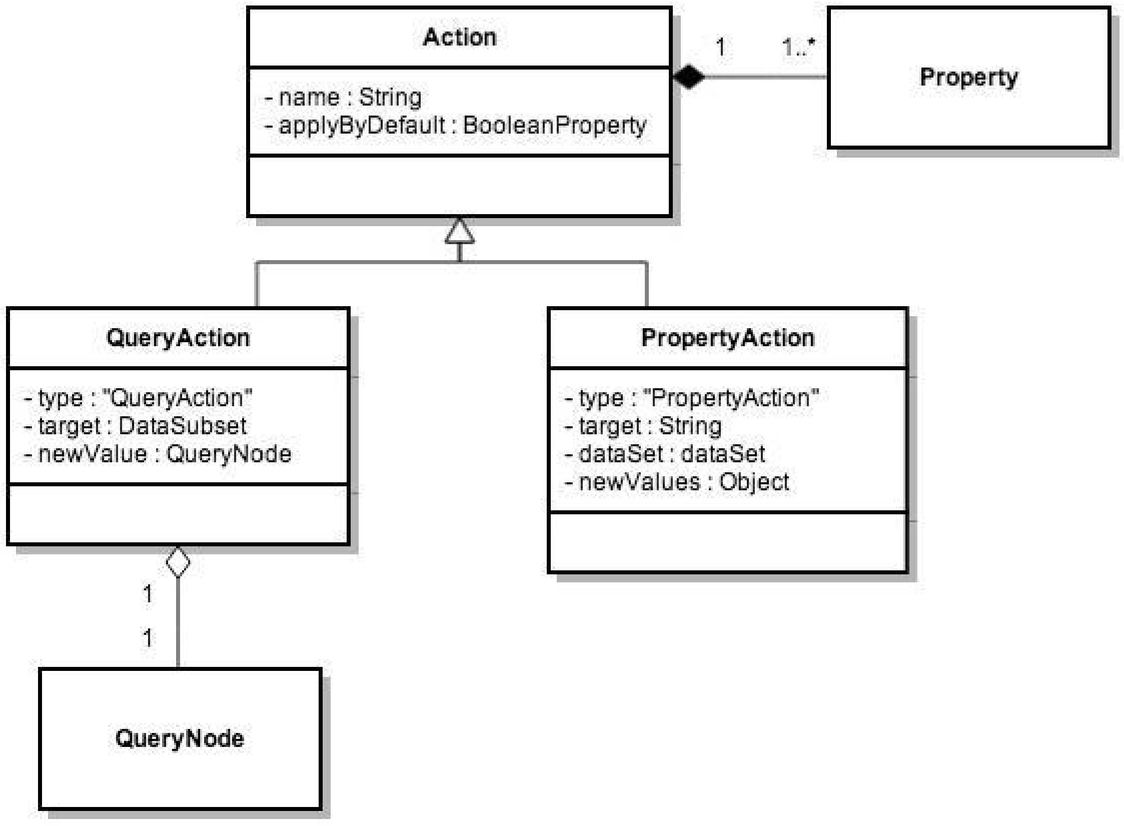


Figure 95. Action saved state schema

Figure 95 shows the saved state schema for an Action and the two subtypes for an Action, QueryAction and PropertyAction. QueryAction references QueryNode in its state representation and thus contains state data for any associated QueryNode objects. Unique type attributes differentiate PropertyAction and QueryAction from each other in the state representation.

6. Traceability Matrix

Requirement	Description	Class Number	Class Name
R3.1	**Welcome Screen Requirements**		
R3.1.1	On launch		
R3.1.1.1	Create a new project	4.2.2.1, 4.2.2.2	Welcome, WAVED
R3.1.1.2	Load an existing project	4.2.2.1, 4.2.2.2	Welcome, WAVED
R3.1.2	**Creating a Project**		
R3.1.2.1	Launching an application	4.2.2.1, 4.2.2.2	Welcome, WAVED
R3.1.2.2	Creating a new project	4.2.2.5	NewProject
R3.1.2.3	Abandoning changes	4.2.2.5	NewProject
R3.1.3	Project name	4.2.2.5	NewProject
R3.1.3.1	Project name uniqueness	4.2.2.5	NewProject
R3.1.3.2	Project name length	4.2.2.5	NewProject
R3.1.3.3	Project name character restriction	4.2.2.5	NewProject
R3.1.4	**Loading a Project**		
R3.1.4.1	Loading a saved project	4.2.2.6	LoadProject
R3.1.4.2	Selecting a saved project from a list	4.2.2.6	LoadProject
R3.1.2.3	Validating the project loaded correctly	4.2.2.6	LoadProject
R3.1.2.4	Abandoning changes	4.2.2.6	LoadProject
R3.2	**Main Interface Requirements**		
R3.2.1	Updating the workspace	4.2.3.2	WorkspaceViewModel
R3.2.2	Refreshing the workspace	4.2.3.2	WorkspaceViewModel
R3.2.3	Functionality of the workspace	4.2.3.2	WorkspaceViewModel
R3.3	**Data Requirements**		
R3.3.1	Valid Data File Formats		
R3.3.1.1	Acceptable Formats		
R3.3.1.1.1	CSV	4.2.4.1	DataSet
R3.3.2	Uploading a Data Sourcce		
R3.3.2.1	Uploading data	4.2.2.10	UploadData
R3.3.2.2	File type restrictions	4.2.2.10	UploadData
R3.3.2.3	Name Requirements		
R3.3.2.3.1	Name length	4.2.2.10	UploadData
R3.3.2.3.2	Valid characters	4.2.2.10	UploadData
R3.3.3	Viewing a Data Set		
R3.3.3.1	Viewing data	4.2.2.13	ReadData

R3.3.3.2	Preview row count	4.2.2.13	ReadData
R3.3.4	Creating a Data Subset		
R3.3.4.1	Data subset name		
R3.3.4.1.1	Name Length	4.2.4.2	DataSubset
R3.3.4.1.2	Character Restriction	4.2.4.2	DataSubset
R3.3.4.2	Data source to derive from	4.2.4.2	DataSubset
R3.3.4.3	Creating the conditional		
R3.3.4.3.1	A data field	4.2.4.3	QueryNode
4.10.4.3.2	Comparison Operators		
R3.3.4.3.2.1	Less than	4.3.6	ComparisonOperatorValue
R3.3.4.3.2.2	Less than or equal to	4.3.6	ComparisonOperatorValue
R3.3.4.3.2.3	Equal to	4.3.6	ComparisonOperatorValue
R3.3.4.3.2.4	Not equal to	4.3.6	ComparisonOperatorValue
R3.3.4.3.2.5	Greater than	4.3.6	ComparisonOperatorValue
R3.3.4.3.2.6	Greater than or equal to	4.3.6	ComparisonOperatorValue
R3.3.4.3.3	A user-supplied value	4.2.4.3	QueryNode
R3.3.4.4	Logical Operators		
R3.3.4.4.1	AND	4.3.7	LogicalOperatorValue
R3.3.4.4.2	OR	4.3.7	LogicalOperatorValue
R3.3.4.5	Logical Operator Precedence	4.2.4.2	DataSubset
R3.3.5	Removing A Data Set		
R3.3.5.1	Removing data	4.2.2.23	ProjectViewModel
R3.3.5.2	Prompt if children exist	4.2.2.23	ProjectViewModel
R3.3.5.3	Recursively delete	4.2.2.11	DeleteData
R3.3.5.4	Remove data from server	4.2.2.11	DeleteData
R3.3.5.5	Restrictions on removing data	4.2.2.23	ProjectViewModel
R3.3.5.6	Prompt if removing is restricted	4.2.2.23	ProjectViewModel
R3.4	**Widgets Panel Requirements**		
R3.4.1	Adding widgets to a project	4.2.2.23	ProjectViewModel
R3.4.2	Available Widgets		
R3.4.2.1	U.S. Map	4.2.3.5	USMap
R3.4.2.1.1	Map Display	4.2.3.5	USMap
R3.4.2.2	Line Graph	4.2.3.9	LineGraph
R3.4.2.2.1	Display data points	4.2.3.9	LineGarph
R3.4.2.2.2	Display lines	4.2.3.9	LineGraph
R3.4.2.3	Button	4.2.3.11	Button
R3.4.2.3.1	Button display	4.2.3.11	Button

R3.4.2.4	Text Block	4.2.3.13	TextBlock
R3.4.2.4.1	Text block display	4.2.3.13	TextBlock
R3.4.2.9	Tooltip	4.2.3.14	Tooltip
R3.4.2.9.1	Tooltip display	4.2.3.14	Tooltip
R3.5	**Properties Panel Requirements**		
R3.5.1	Modifying a Widget		
R3.5.1.1	Modifying defined properties	4.2.3.4	WidgetViewModel
R3.5.1.2	Selecting to change properties	4.2.2.23	ProjectViewModel
R3.5.2	Workspace Properties		
R3.5.2.1	Height		
R3.5.2.1.1	Valid height	4.2.3.1	ComponentViewModel
R3.5.2.1.2	Default height	4.2.3.1	ComponentViewModel
R3.5.2.2	Width		
R3.5.2.2.1	Valid width	4.2.3.1	ComponentViewModel
R3.5.2.2.2	Default width	4.2.3.1	ComponentViewModel
R3.5.2.3	Background Color		
R3.5.2.3.1	Valid color	4.2.3.2	WorkspaceViewModel
R3.5.2.3.2	Default color	4.2.3.2	WorkspaceViewModel
R3.5.3	Common Widget Properties		
R3.5.3.1.1	Name		
R3.5.3.1.1.1	Name length	4.2.3.1	ComponentViewModel
R3.5.3.1.1.2	Name uniqueness	4.2.3.1	ComponentViewModel
R3.5.3.1.1.3	Name character restriction	4.2.3.1	ComponentViewModel
R3.5.3.1.1.4	Default name	4.2.3.1	ComponentViewModel
R3.5.3.1.2	Visibility		
R3.5.3.1.2.1	Visibility values	4.2.3.1	ComponentViewModel
R3.5.3.1.2.2	Visibility default value	4.2.3.1	ComponentViewModel
R3.5.3.1.3	Log Google Analytics		
R3.5.3.1.3.1	Google Analytics	4.2.3.1	ComponentViewModel
R3.5.3.1.3.2	Google analytics default value	4.2.3.1	ComponentViewModel
R3.5.3.1.4	Horizontal Offset		
R3.5.3.1.4.1	Valid horizontal offset values	4.2.3.4	WidgetViewModel
R3.5.3.1.4.2	Measuring the offset	4.2.3.4	WidgetViewModel
R3.5.3.1.4.3	Default value	4.2.3.4	WidgetViewModel
R3.5.3.1.5	Vertical Offset		
R3.5.3.1.5.1	Valid vertical offset values	4.2.3.4	WidgetViewModel

R3.5.3.1.5.2	Measuring the offset	4.2.3.4	WidgetViewModel
R3.5.3.1.5.3	Default value	4.2.3.4	WidgetViewModel
R3.5.3.1.6	Height		
R3.5.3.1.6.1	Height value restriction	4.2.3.4	WidgetViewModel
R3.5.3.1.7	Width		
R3.5.3.1.7.1	Width value restriction	4.2.3.4	WidgetViewModel
R3.5.4	U.S. Map Properties		
R3.5.4.1	Coloring		
R3.5.4.1.1	Coloring Schemes		
R3.5.4.1.1.1	All states same color	4.2.3.26	SolidColoringScheme
R3.5.4.1.1.2	Four color scheme	4.2.3.27	FourColoringScheme
R3.5.4.1.2	Default coloring	4.2.3.6	USMapViewModel
R3.5.4.2	Glyphs		
R3.5.4.1.1	Name		
R3.5.4.1.1.1	Name length	4.2.3.1	ComponentViewModel
R3.5.4.1.1.2	Name uniqueness	4.2.3.1	ComponentViewModel
R3.5.4.1.1.3	Name character restriction	4.2.3.1	ComponentViewModel
R3.5.4.1.1.4	Default name	4.2.3.1	ComponentViewModel
R3.5.4.1.2	Visibility		
R3.5.4.1.2.1	Visibility values	4.2.3.1	ComponentViewModel
R3.5.4.1.2.2	Visibility default value	4.2.3.1	ComponentViewModel
R3.5.4.1.3	Log Google Analytics		
R3.5.4.1.3.1	Google Analytics	4.2.3.1	ComponentViewModel
R3.5.4.1.3.2	Google analytics default value	4.2.3.1	ComponentViewModel
R3.5.4.2.4	Data		
R3.5.4.2.4.1	Selecting data	4.2.3.16	GlyphViewModel
R3.5.4.2.5	Color		
R3.5.4.2.5.1	Valid color value	4.2.3.16	GlyphViewModel
R3.5.4.2.5.2	Default color	4.2.3.16	GlyphViewModel
R3.5.4.2.6	Shape		
R3.5.4.2.6.1	Available glyph shapes		
R3.5.4.2.6.1.1	Circle	4.2.3.16	GlyphViewModel
R3.5.4.2.6.2	Default shape	4.2.3.16	GlyphViewModel
R3.5.4.2.7	Size		
R3.5.4.2.7.1	Specifying dimensions		
R3.5.4.2.7.1.1	Constant size	4.2.3.31	ConstantGlyphSizeScheme

R3.5.4.2.7.1.2	Data dependent size	4.2.3.32	ScaledGlyphSizeScheme
R3.5.4.2.8	Latitude		
R3.5.4.2.8.1	Latitude source	4.2.3.16	GlyphViewModel
R3.5.4.2.8.2	Valid latitude	4.2.3.16	GlyphViewModel
R3.5.4.2.9	Longitude		
R3.5.4.2.9.1	Longitude source	4.2.3.16	GlyphViewModel
R3.5.4.2.9.2	Valid longitude	4.2.3.16	GlyphViewModel
R3.5.5	Line Graph Properties		
R3.5.5.1	Title		
R3.5.5.1.1	Character restriction	4.2.3.8	GraphViewModel
R3.5.5.2	X-Axis Label		
R3.5.5.2.1	Character restriction	4.2.3.8	GraphViewModel
R3.5.5.3	Y-Axis Label		
R3.5.5.3.1	Character restriction	4.2.3.8	GraphViewModel
R3.5.5.4	Graph Data		
R3.5.5.4.1	Graph data requirement	4.2.3.8	GraphViewModel
R3.5.5.5	X-Axis Data		
R3.5.5.5.1	Valid x-axis value	4.2.3.8	GraphViewModel
R3.5.5.6	Y-Axis Data		
R3.5.5.6.1	Valid y-axis value	4.2.3.8	GraphViewModel
R3.5.6	Button Properties		
R3.5.6.1	Label		
R3.5.6.1.1	Label length	4.2.3.12	ButtonViewModel
R3.5.8	Text Block Properties		
R3.5.8.1	Text Content		
R3.5.8.1.1	Text content length	4.2.3.15	TextBlockViewModel
R3.5.12	Tooltip Properties		
R3.5.12.1	Text Content		
R3.5.12.1.1	Text content length	4.2.3.15	TextBlockViewModel
R3.5.13	Binding Data To Widgets		
R3.5.13.1	Data must be bound to access the data	4.2.3.4	WidgetViewModel
R3.5.14	Unbinding Data from Widgets		
R3.5.14.1	Removing bound data	4.2.3.4	WidgetViewModel
R3.5.14.2	Only remove unused data	4.2.3.4	WidgetViewModel
R3.6	**Action Panel Requirements**		
R3.6.1	Creating an Action		

R3.6.1.1	Required information		
R3.6.1.1.1	Name	4.2.5.1	Action
R3.6.1.1.2	Action type	4.2.5.1	Action
R3.6.1.1.3	Affected widget	4.2.5.2	PropertyAction
R3.6.1.1.4	Data set	4.2.5.3	QueryAction
R3.6.1.1.5	New value	4.2.5.1	Action
R3.6.1.1.6	Apply immediately	4.2.5.1	Action
R3.6.2	Action Types		
R3.6.2.1	Valid action types		
R3.6.2.1.1	Change a property value	4.2.5.2	PropertyAction
R3.6.2.1.2	Change a data subset query	4.2.5.3	QueryAction
R3.6.3	Modifying an Action		
R3.6.3.1	Actions can be modified	4.2.5.1	Action
R3.6.4	Removing an Action		
R3.6.4.1	Removing an action	4.2.2.23	ProjectViewModel
R3.6.4.2	Restrictions on removing an action	4.2.2.23	ProjectViewModel
R3.6.5	Action Property Values		
R3.6.5.1	Specified Values		
R3.6.5.1.1	Valid value	4.2.5.1	Action
R3.6.5.1.2	Valid string form		
R3.6.5.1.2.1	X		
R3.6.5.1.2.1.1	Event x coordinate	4.2.5.1	Action
R3.6.5.1.2.2	Y		
R3.6.5.1.2.2.1	Event y coordinate	4.2.5.1	Action
R3.6.5.1.2.3	dataset.field		
R3.6.5.1.2.3.1	dataset	4.2.5.1	Action
R3.6.5.1.2.3.2	field	4.2.5.1	Action
R3.7	**Event Panel Requirements**		
R3.7.1	Event Types		
R3.7.1.1	Valid event types		
R3.7.1.1.1	A click event	4.3.5	EventType
R3.7.1.1.2	A hover event	4.3.5	EventType
R3.7.1.1.3	A move event	4.3.5	EventType
R3.7.2	Creating an Event		
R3.7.2.1	Required information		
R3.7.2.1.1	Event type	4.2.6.1	Event

R3.7.2.1.2	Name of widget	4.2.6.1	Event
R3.7.2.1.3	Actions	4.2.6.1	Event
R3.7.3	Modifying an Event		
R3.7.3.1	Events can be modified	4.2.6.1	Event
R3.7.4	Removing An Event		
R3.7.4.1	Events can be removed	4.2.2.23	ProjectViewModel
R3.8	**Google Analytics**		
R3.8.1	Google Analytics tracking	4.2.2.21	GoogleAnalytics
R3.8.2	Google Analytics requirements	4.2.2.21	GoogleAnalytics
R3.8.3	Character Restriction	4.2.2.21	GoogleAnalytics
R3.8.4	Google Analytics UA code form	4.2.2.21	GoogleAnalytics
R3.8.5	Event character restrictoin	4.2.2.21	GoogleAnalytics
R3.9	**Project Tree**		
R3.9.1	Listing Components		
R3.9.1.1	Displayed components	4.2.2.22	ProjectTree
R3.6.1.2	Select components	4.2.2.22	ProjectTree
R3.9.1.3	Deleteable	4.2.2.22	ProjectTree
R3.9.2	Deleting a Widget		
R3.9.2.1	Deleting a widget	4.2.2.22	ProjectTree
R3.9.2.2	Selected	4.2.2.22	ProjectTree
R3.9.2.3	Delete restrictions on events	4.2.2.22	ProjectTree
R3.9.2.4	Delete restrictions on actions	4.2.2.22	ProjectTree
R3.10	**Main Menu Requirements**		
R3.10.1	Saving a project		
R3.10.1.1	Saving current project	4.2.2.7	SaveProject
R3.10.1.2	Copying the project	4.2.2.7	SaveProject
R3.10.1.3	Resaving	4.2.2.7	SaveProject
R3.10.2	Deleting a project		
R3.10.2.1	Deletable	4.2.2.8	DeleteProject
R3.10.2.2	Open and deletable	4.2.2.8	DeleteProject
R3.10.2.3	Prompt before delete	4.2.2.8	DeleteProject
R3.10.2.4	Not visible after delete	4.2.2.8	DeleteProject
R3.10.2.5	Delete files	4.2.2.8	DeleteProject
R3.10.3	Export Visualization		
R3.10.3.1	Export File		
R3.10.3.1.1	Export a zip file	4.2.2.9	ExportProject

R3.10.3.1.2	Exported contents		
R3.10.3.1.2.1	HTML	4.2.2.9	ExportProject
R3.10.3.1.2.2	Javascript	4.2.2.9	ExportProject
R3.10.3.1.2.3	CSS	4.2.2.9	ExportProject
R3.10.3.1.2.4	Data Files	4.2.2.9	ExportProject
R3.10.4	Project Revision History		
R3.10.4.1	Undo		
R3.10.4.1.1	The user is able to undo	4.2.2.17	HistoryMonitor
R3.10.4.1.2	Undo disabled	4.2.2.17	HistoryMonitor
R3.10.4.2	Redo		
R3.10.4.2.1	Revert can be restored	4.2.2.17	HistoryMonitor
R3.10.4.2.2	After change, cannot be restored	4.2.2.17	HistoryMonitor
R3.10.4.2.3	Redo disabled	4.2.2.17	HistoryMonitor