**CIS 422**

**Software Design Specification (SDS) for UNAMED\_CALENDAR**

The document in this file has been adapted from the IEEE Guide to Software Design Specifications (Std. 1016-2009)

**CIS 422**

Team Number 6

499ms

Software Design Specifications

Version 1 Date: (01/14/2019)

Table of Contents

[**1. Introduction**](#_g2qe5nbs1r4s) **4**

[1.1. Purpose of this Document](#_tnjbr32g0abe) 4

[1.2. Scope of the Development of this Project](#_kqzo9keudjk2) 4

[1.3. Definitions, Acronyms, and Abbreviations](#_p17z4o5zs0tt) 4

[1.4. References](#_teng9map97gr) 5

[1.5. Overview of Document](#_9llocpivio0l) 5

[**2. System Architecture Description**](#_failhon9mbmy) **6**

[2.1. Overview of Modules/Components](#_2e281fe3qbjx) 6

[2.2. Structure and Relationships](#_e67lkjdq1bim) 6

[2.3. User Interface Issues](#_7p51o7yghj0n) 6

[**3. Detailed Description of Components**](#_1jagi6gw4q81) **7**

[3.1. Component Template Description](#_6pqd2vrqkino) 7

[3.2. Driver](#_7up9wfjio0pp) 7

[3.3. Display](#_sl2lfz33m10t) 7

[3.3.1. Month View](#_g7nmd65zd9e) 7

[3.3.2. Day View](#_7y4cvjgnpabv) 8

[3.3.3. Event View](#_o1txc8bo6g1s) 8

[3.3.4. Event Date Editor](#_g2dqvi8cqrem) 8

[3.3.5. Event Time Editor](#_ilk7xofe6d7g) 9

[3.4. Data Structures](#_v2ki4rypdupr) 9

[3.4.1. Calendar](#_1edwmxkqxy6n) 9

[3.4.2. Calendar Year](#_mna3pr6liyx4) 9

[3.4.3. Calendar Month](#_m9ote9cmhye0) 9

[3.4.4. Calendar Day](#_rrtygkupcany) 10

[3.4.5. Calendar Event](#_fbrzkl268nkz) 10

[3.4.6. FileIO Status](#_2obtr1bmkqdb) 11

[3.4.5. Status Constants](#_8sbj67us4u3k) 12

[3.5. File IO](#_yd77folrr1h8) 12

[3.5.1. Read from File](#_2jgimkw992hu) 12

[3.5.2. Write to File](#_vetd3230r10z) 13

[**4. Reuse and Relationship to Other Products**](#_91dstimq033t) **14**

[**5. Design Decisions and Tradeoffs**](#_1gaujwxho7b) **14**

[**6. Pseudocode and Layout for Components**](#_vzx91pwwd8va) **15**

[6.1. Pseudocode and Layout Description](#_arzny0q9026r) 15

[6.2. Driver](#_1dbndebkom2q) 15

[6.3. Display](#_eed5nixm0aqe) 15

[6.3.1. Month View](#_7tsrggrf7hvy) 15

[6.3.2. Day View](#_b1i2l91fkrnk) 15

[6.3.3. Event View](#_cymvzwrgd5i0) 15

[6.3.4. Event Date Editor](#_saa0vy80ha3o) 16

[6.3.5. Event Time Editor](#_st61z6g6s71j) 16

[6.4. Data Structures](#_lu5t4oe2lhm4) 16

[6.4.1. Calendar](#_iqcck5x8jqv7) 16

[6.4.2. Calendar Year](#_eer7g4tbwifq) 16

[6.4.3. Calendar Month](#_f5sqgff0eqo6) 17

[6.4.4. Calendar Day](#_ltp7xh4h4rh8) 17

[6.4.5. Calendar Event](#_no6izgwl8ho6) 18

[6.4.6. FileIO Status](#_8qpjieo9e2ke) 19

[6.5. File I/O](#_hlgies2vbaot) 19

[6.5.1. Sample File Structure](#_n6vpdc2sw1xr) 19

[6.5.2. Read From File](#_5qw03g5qjsb7) 20

[6.5.3. Write to File](#_3lvv876472ox) 20

[6.6. Constant Classes](#_ppo0h1c1vcn5) 21

[6.6.1. Status Constants](#_iab84p9ljsy7) 21

[**7. Appendices**](#_a7huv77pw1t2) **22**

[7.1. Appendix 1: Home Screen (Drawing #1)](#_p0x9t4ed87z) 22

[7.2. Appendix 2: Blanked Day Clicked (Drawing #2)](#_66de1fbwochz) 23

[7.3. Appendix 3: Filled Day Clicked (Drawing #3)](#_z3dsypwwaemg) 24

[7.4. Appendix 4: Add Event (Drawing #4)](#_jkql1c2di6cl) 25

[7.5. Appendix 5: Edit Event (Drawing #5)](#_jcs5ogca5oku) 26

[7.6. Appendix 6: Time Editor (Drawing #6)](#_2xuua9fzbazo) 27

# **1. Introduction**

## 1.1. Purpose of this Document

This software design specification is made with the purpose of outlining the architecture and design of UNAMED\_CALENDAR in detail. This document will provide developers an insight into meeting an end users needs. This document will provide developers an understanding of UNAMED\_CALENDAR inner workings to facilitate an efficient and accurate production.

## 1.2. Scope of the Development of this Project

This software design specification will demonstrate how UNAMED\_CALENDAR will meet the functional and nonfunctional requirement specified listed in the software requirement specification.

This document will provide a framework for all developers describing the high level components, application programming interface, graphical user interface, subsystems, and systems, and algorithms used.

## 1.3. Definitions, Acronyms, and Abbreviations

**Software Design Specifications** (SDS) : A document describing the built product. The SDS describes external visible behavior as precise as possible.

**Initial Project Plan** ( IPP ) : A document outlining the process from beginning to completion of UNAMED\_CALENDAR. This document will include : a management plan, multiple milestones including dates and items required, an outline of scheduled meetings, the responsibilities of each member, the coding style requirements for the development cycle, and a rationale behind each these choices.

**Application Programming Interface** ( API ) : A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service

**Functional Requirement** ( FR ) : A specification for the required behaviour of a system or component.

**Non-functional Requirement** ( NFR ) : A specification for the judgement of the operation of a system or component.

**User Experience** ( UX ) : The overall experience of the end consumer using UNAMED\_CALENDAR.

**Coding Style Document** ( CSD ) : A document outlining the expected coding style for all developers of UNAMED\_CALENDAR to follow. This will include: parentheses spacing, function naming, variable naming, file naming, comments, and more.

**Citation Expectations Document** ( CED ) : A document outlining the expected style of how a developer of UNAMED\_CALENDAR is to cite references used.

## 1.4. References

*References related to the internal development*

1. **IPP** : [https://docs.google.com/document/d/1ZszO4mlShik\_dEfztpab-m4zGZj5UHmMHrU0SFNLGkg/](https://docs.google.com/document/d/1ZszO4mlShik_dEfztpab-m4zGZj5UHmMHrU0SFNLGkg/edit)
2. **SRS**: [https://docs.google.com/document/d/1sNOXpG85zZVb8jMq4xzn\_6ZcFwR9cahqVJ5A0LunbMs](https://https//docs.google.com/document/d/1sNOXpG85zZVb8jMq4xzn_6ZcFwR9cahqVJ5A0LunbMs)
3. **CSD**: <https://docs.google.com/document/d/1KL8Q_9lhUDCnlcPMZtaPfaDTiaKeyBGpLgd7bN3cE5s/>
4. **CED:**
5. **Project Requirements**
   1. <https://classes.cs.uoregon.edu/19W/cis422/P1.html>
   2. <https://classes.cs.uoregon.edu/19W/cis422/P1_Grading.html>

*References used to create this document*

1. **SDS Template:** <https://web.cs.dal.ca/~arc/teaching/CS3130/Templates/Design%20Templates/SDS.doc>

## 1.5. Overview of Document

Section 2 of this document describe the architectural design of UNAMED\_CALENDAR. The high level components, application programming interface, graphical user interface, subsystems, and systems, and algorithms used.

Section 3 of this document describes component design from a high level. This allows every developer to efficiently produce and test code for UNAMED\_CALENDAR.

# **2. System Architecture Description**

## 2.1. Overview of Modules/Components

There are 4 primary components that define the system of the UNAMED\_CALENDAR. There is the driver, a display, data structures, and a file input/output system. Under the display, there is several components including the Month View, Day View, Event View, Date Editor, and Time Editor. Under the Data Structures, there are also several components including Calendar, CalendarYear, CalendarMonth, CalendarDay, CalendarEvent, FileIOStatus, and Status Constants. Under FileIO, we also have several components including reading from and writing to a file.

## 2.2. Structure and Relationships

The driver is the main component that runs the program. It starts with contacting the FileIO system that will search for a file of stored events. If it has not found a file, then the program loads normally, however, if it did find a file, then it uses this file to load the data structures. These data structures can be treated as an organizational set of arrays. The Calendar object contains an array of CalendarYear objects. The CalendarYear object contains an array of CalendarMonth objects. The CalendarMonth object contains an array of CalendarDay objects. The CalendarDay object contains an array of CalendarEvent objects. The CalendarEvent object contains all the information pertaining to that specific event.

While the program is running, a display is continually running that defaultedly shows the Month View (refer to Appendix 1). However, if you click on a specific day, you go into Day View (refer to Appendix 2/3) Within the Day view, you may see several events. If you click on an event, you go into Event View (refer to Appendix 4/5). From the Event View, you can see the event information including the time. Clicking on the time button will open the Time Editor (refer to Appendix 6).

When the program concludes, it refers back to the FileIO system to close up operations by making sure that any newly added or deleted events are taken care of. The FileIOStatus and Status Constants objects are used to assist in saving the files correctly and easily through a means of assigning enumerated constants to different months.

## 2.3. User Interface Issues

Refer to Appendices 1-6 for a detailed explanation on how the user interface works. Soon to be added to the diagrams is a delete button for events.

# **3. Detailed Description of Components**

## 3.1. Component Template Description

The following descriptions are following the IEEE std. 1016-2009 for describing components of the program. It contains many attributes including Identification, Type, Purpose, Function, Subordinates, Dependencies, Interfaces, Resources, Processing, and Data. According to the standard, the value of an attribute will be marked as N/A if it does not pertain rather than fully removing it from the document.

## 3.2. Driver

|  |  |
| --- | --- |
| Identification | Main |
| Type | A function |
| Purpose | This is the function that drives the program, it listens for events, contacts the display to make changes to the GUI, and contacts the File I/O to read and write to the file |
| Function | N/A |
| Subordinates | N/A |
| Dependencies | N/A |
| Interfaces | This function will interface with event listeners |
| Resources | It will use the Display Features, Data Structures, and FileIO to operate. |
| Processing | *See 6.2 for Pseudocode* |
| Data | Calendar: A reference to a Calendar object. |

## 3.3. Display

### *3.3.1. Month View*

|  |  |
| --- | --- |
| Identification | MonthView |
| Type | Module / User Interface |
| Purpose | This interface displays the corresponding month for the events. |
| Function | User able to read the date corresponding day. |
| Subordinates | N/A |
| Dependencies | Requires the driver to tell which month should be currently showing |
| Interfaces | Days of not current month show up, but are in light blue color boxes.  Days of current month are in white boxes. |
| Resources | N/A |
| Processing | *See Appendix 1 for Diagram* |
| Data | N/A |

### *3.3.2. Day View*

|  |  |
| --- | --- |
| Identification | DayView |
| Type | Module / User Interface |
| Purpose | This user interface is for allowing user’s to interact with a single day, specifically viewing the events of a day. |
| Function | The user has the ability check out events of the day, go to the next or previous day, or exit day view. |
| Subordinates | N/A |
| Dependencies | Requires the month to determine which days are available to see |
| Interfaces | Go to previous or next day when clicking on left/right button.  Popup goes over calendar to display the single day view. |
| Resources | N/A |
| Processing | *See Appendix 2/3 for Diagram* |
| Data | N/A |

### *3.3.3. Event View*

|  |  |
| --- | --- |
| Identification | EventView |
| Type | Module / User Interface |
| Purpose | This interface allows user to check event name, time, notes, location. |
| Function | The user can edit the current event |
| Subordinates | N/A |
| Dependencies | Requires the day to determine which events will be visible |
| Interfaces | Go to previous or next day when clicking on left/right button.  Popup goes over calendar to display the single day view. |
| Resources | N/A |
| Processing | *See Appendix 4/5 for Diagram* |
| Data | N/A |

### *3.3.4. Event Date Editor*

|  |  |
| --- | --- |
| Identification | Date Editor |
| Type | Module / User Interface |
| Purpose | Allows user to change the date of the event |
| Function | The user can change the date to a date that it is not currently and the data will update accordingly |
| Subordinates | N/A |
| Dependencies | N/A |
| Interfaces | N/A |
| Resources | N/A |
| Processing | *Appendix Diagram Pending* |
| Data | N/A |

### *3.3.5. Event Time Editor*

|  |  |
| --- | --- |
| Identification | Time Editor |
| Type | Module / User Interface |
| Purpose | Allows user to change the date of the event |
| Function | The user can change the time of the event even after initial set up |
| Subordinates | N/A |
| Dependencies | N/A |
| Interfaces | N/A |
| Resources | N/A |
| Processing | *See Appendix 6 for Diagram* |
| Data | N/A |

## 3.4. Data Structures

### 3.4.1. Calendar

|  |  |
| --- | --- |
| Identification | Calendar |
| Type | Class |
| Purpose | This class is used to store a hashmap of CalendarYear objects. |
| Function | N/A |
| Subordinates | N/A |
| Dependencies | This class depends on the array of CalendarYear objects within it |
| Interfaces | N/A |
| Resources | N/A |
| Processing | *See 6.4.1 for pseudo-code.* |
| Data | Array of CalendarYear Objects |

### *3.4.2. Calendar Year*

|  |  |
| --- | --- |
| Identification | CalendarYear |
| Type | Class |
| Purpose | This class is used to store an array of CalendarMonth objects. |
| Function | N/A |
| Subordinates | N/A |
| Dependencies | This class depends on the array of CalendarMonth objects within it |
| Interfaces | N/A |
| Resources | N/A |
| Processing | *See 6.4.2 for pseudo-code.* |
| Data | Array of CalendarMonth Objects |

### 3.4.3. Calendar Month

|  |  |
| --- | --- |
| Identification | CalendarMonth |
| Type | Class |
| Purpose | This class is used to store an array of CalendarDay objects. |
| Function | The CalendarMonth will be passed to the display module so that the display module can read data from individual days of the month and display accordingly. |
| Subordinates | N/A |
| Dependencies | CalendarDay: This class holds an array of CalendarDays  Display Module: Utilizes CalendarMonth to display the information on the GUI. |
| Interfaces | N/A |
| Resources | N/A |
| Processing | *See 6.4.3 for pseudo-code.* |
| Data | Month: An unsigned integer for identifying the month. Initialized to 0 (Invalid Month). Valid values are 1-12.  Year: An unsigned integer for identifying the year. Initialized to 0 (Invalid Year). Valid values are 1 - 150. This is to be offset by 1969 to obtain the current year of the month.  CalendarDays: A hashmap with unsigned integers as keys, the values will be a linear array of CalendarDay objects with a nonzero amount of CalendarEvents. Initialized to an empty hashmap. |

### *3.4.4. Calendar Day*

|  |  |
| --- | --- |
| Identification | CalendarDay |
| Type | Class |
| Purpose | This class is used to store an array of events |
| Function | The CalendarDay will display an array of CalendarEvents if they exist at the instance. |
| Subordinates | N/A |
| Dependencies | This class is dependant on the array of events held within the class |
| Interfaces | N/A |
| Resources | N/A |
| Processing | *See 6.4.4 for pseudo-code.* |
| Data | Day: An unsigned integer for identifying the day. Initialized to 0. (Invalid day). Valid values are 1 - 31.  CalendarEvents: A sorted linear array of CalendarEvent objects, sorted by the CalendarEvent’s time followed by title. Initialized to empty linear array.  Month: A CalendarMonth object that contains this CalendarDay. |

### *3.4.5. Calendar Event*

|  |  |
| --- | --- |
| Identification | CalendarEvent |
| Type | Class |
| Purpose | This class is used to store events created by the user to be passed to Event View, Day View and Month View. |
| Function | N/A |
| Subordinates | N/A |
| Dependencies | This class is dependant of the data within the event |
| Interfaces | N/A |
| Resources | N/A |
| Processing | *See 6.4.5 for pseudo-code.* |
| Data | Day: A CalendarDay object that contains this CalendarEvent.  Start Time: A Date object to store the time of the event. Initialized to Unix Epoch. Only the time properties should be used for this object. An all day event shall have the value as 00:00:00.  End Time: A Date object to store the end time of the event. Initialized to 1/1/1970 23:59:59, representing all day. All other times shall have 0’s for seconds except for all day events, which should have a time of 23:59:59.  Title: A string storing the title of the event. Initialized to an empty string. |

### *3.4.6. FileIO Status*

|  |  |
| --- | --- |
| Identification | FileIOStatus |
| Type | Class |
| Purpose | This class is used to store the status of a call to write to the file directory. |
| Function | All properties of this class may be modified by any routine at any time. |
| Subordinates | This class contains only five (5) variables which are all public and described in the data section of 3.4.6. |
| Dependencies | This class is used by both the LoadEvents module and the SaveEvents module to provide one way message passing to a calling function to denote the status of performing file I/O. This class does not contain any functions, however it does contain a lock that any routine may use to perform thread safe access and modification of values of this class. |
| Interfaces | This class will provide a one way message passing interface to denote the status of a File I/O request. No errors will be passed directly to this class, however java [IOExceptions](https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html) shall have a unique ErrorCode associated with them. IOExceptions JavaDocs: <https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html> |
| Resources | This class requires:  The ability to contain locks to be used by other routines |
| Processing | N/A. |
| Data | CurrentStatus: An Status enum object to denote the status of the save request. Updated asynchronously. Requires acquiring a thread lock to access.  ErrorCode: An integer denoting the error code ID.  StoredValue: A Calendar object assuming there was not an error loading from a file.  IsFromSaving: A boolean denoting if this FileIOStatus object was spawned from a read request or a write request.  Lock: A lock object to prevent multiple threads from reading and writing data at the same time. |

### *3.4.5. Status Constants*

|  |  |
| --- | --- |
| Identification | Status |
| Type | Constants |
| Purpose | To contain all constants to denote the status of saving CalendarEvents. |
| Function | N/A. |
| Subordinates | An java enum object. |
| Dependencies | This component stores the possible statuses for the success of calling SaveEvents. This status will be used in FileIOStatus. |
| Interfaces | N/A. |
| Resources | N/A. |
| Processing | N/A. |
| Data | Enum values:  Waiting: The SaveEvent asynchronous function has yet to return.  Success: The SaveEvent asynchronous function has successfully saved the data.  Failed: The SaveEvent asynchronous function has failed and the error code ErrorCode in the WritingStatus object has been set. |

## 3.5. File IO

### *3.5.1. Read from File*

|  |  |
| --- | --- |
| Identification | LoadEvents |
| Type | Module |
| Purpose | To load CalendarEvents from the user’s file system to populate the calendar application. |
| Function | The LoadEvents module will take a file name to retrieve a from the user’s file system along with a target years to load from and will process the contents of the file into CalendarEvents, which will be stored in a Calendar object (creating all CalendarYear, CalendarMonth and CalendarDay objects as required). The process is asynchronous, so a FileIOStatus object is returned to the caller immediately. Once this process is done, with or without errors, the process will update the FileIOStatus object. The FileIOStatus object should have isFromSaving false. |
| Subordinates | This module is broken into three parts, a main function that is public to other modules, a private function that is meant to be executed in a coroutine and a helper function.  The main function is responsible for starting the coroutine and passing back a shared FileIOStatus object to the caller.  The coroutine function is responsible for loading a file from the user’s file system asynchronously and updating the FileIOStatus once complete.  The helper function is responsible for interpreting CSV strings to create and return a CalendarYear object. |
| Dependencies | This module is responsible for managing all File I/O relating to saving to the user’s file system. This will create and manage a coroutine that will maintain access to a FileIOStatus object that will be passed to the caller. Once the file has been saved successfully or an error has occurred, this coroutine will be destroyed. |
| Interfaces | All standard java [IOExceptions](https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html) shall be caught and handled.  The FileIOStatus object shall pass the status of the coroutine to the caller. IOExceptions JavaDocs: <https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html> |
| Resources | This module will require:  The ability to create coroutines  The ability to access the user’s file system  The ability to use locks to prevent clashing routines |
| Processing | *See 6.5.2 for pseudo-code.* |
| Data | N/A. |

### *3.5.2. Write to File*

|  |  |
| --- | --- |
| Identification | SaveEvents |
| Type | Module |
| Purpose | To save CalendarEvents from the calendar to the user’s file system for future use of the application. |
| Function | The SaveEvents module will take a Calendar object containing all stored CalendarYears, which contains all CalendarMonths, containing all CalendarDays, which contain all CalendarEvents. This module will write all events to the output file specified, replacing the current file if present, returning a FileIOStatus object to the caller immediately. The FileIOStatus object should have isFromSaving true. This module shall execute asynchronously.  Because it may be the case that there are years in the previous output file specified, the SaveEvents shall have a slightly different behaviour. The system shall read in the output file, while reading, it will be directly writing out the lines in the file unless they match a year in the Calendar object passed. If this is the case, the line is instead replaced by writing out the CalendarYear in question like above. |
| Subordinates | This module is broken into three parts, a main function that is public to other modules, a private function that is meant to be executed in a coroutine and a helper function.  The main function is responsible for starting the coroutine and passing back a shared FileIOStatus object to the caller.  The coroutine function is responsible for saving a file to the user’s file system asynchronously and updating the FileIOStatus once complete.  The helper function is responsible for creating strings to be saved in CSV format from CalendarYear objects. |
| Dependencies | This module is responsible for managing all File I/O relating to saving to the user’s file system. This will create and manage a coroutine that will maintain access to a FileIOStatus object that will be passed to the caller. Once the file has been saved successfully or an error has occurred, this coroutine will be destroyed. |
| Interfaces | All standard java [IOExceptions](https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html) shall be caught and handled.  The FileIOStatus object shall pass the status of the coroutine to the caller. IOExceptions JavaDocs: <https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html> |
| Resources | This module will require:  The ability to create coroutines  The ability to access the user’s file system  The ability to write to the user’s file system  The ability to use locks to prevent clashing routines |
| Processing | *See 6.5.3 for pseudo-code.* |
| Data | N/A. |

# **4. Reuse and Relationship to Other Products**

The UNAMED\_CALENDAR application isn’t being derived off of any previous code, however, it was inspired by the layout of calendars found on Android, Ubuntu, and MacOS X. The ability to switch between months in month view and days in day view is common on all of the above operating systems, making it a key feature that will be implemented into the design.

# **5. Design Decisions and Tradeoffs**

The design structure of the calendar follows a basic calendar design that most people are already familiar with. Our structure was designed with simplicity in mind in order to make it easier for new users to adapt quickly to the interface. The first design choice that was agreed by all team members was having Monday being the first day rather than Sunday. The main reason for this design choice is due to the fact that most people normally start the week on Monday and perceive Sunday as part of the weekend. This feature allows users to have a clear and organized representation of their week days since it separates the weekdays from the weekend days.

The second design choice was the pop out day-menu. The pop out menu was inspired by the calendar application on Mac OS X. The benefits of the pop out window design is that it is minimal and only block a small portion of the screen. It allows for a user friendly environment that is not going to overwhelm the user. Each pop out menu allows the user to either view an event or add a new event for the selected day. In addition, users can use the arrows to move to the next or previous day and close the window by simply clicking on the ‘X’ symbol on the top left side of the window. These button were placed in the top portion of the window and designed to be big enough in order for users to be able to find and click on easily while not taking too much of the available space. Similarly, users can find the same navigation buttons in the top right corner of the calendar. These buttons have similar functionality to move to previous or next month. Lastly in the monthly view of the calendar, users can clearly see which days have one or more events in them and which days are available.

# **6. Pseudocode and Layout for Components**

## 6.1. Pseudocode and Layout Description

The following is either pseudocode or a reference to a diagram explaining the major components of the program. The Driver, Data Structure, and FileIO have basic pseudo explaining the basic goals for implementing the project. The Display has reference to diagrams that are planning to get our design to match as we build the program.

## 6.2. Driver

|  |
| --- |
| int main(){  if file exists, read file into data structures  while notExited:  draw the gui  listen for a user click  if user click, react accordingly  save data to file } |

## 6.3. Display

### *6.3.1. Month View*

View Appendix 1 for an idea of the layout of the Month View

### *6.3.2. Day View*

View Appendix 2 and 3 for an idea of the layout of the Day View

### *6.3.3. Event View*

View Appendix 4 and 5 for an idea of the layout of the Event View

### *6.3.4. Event Date Editor*

Diagram Pending...

### *6.3.5. Event Time Editor*

View Appendix 6 for an idea of layout of the Time Editor

## 6.4. Data Structures

### *6.4.1. Calendar*

|  |
| --- |
| Class CalendarKeeper{  //variables  CalendarYear[] years   //constructors  CalendarKeeper(){  years = new array of CalendarYear objects  for years 2018 to 2020  initialize CalendarYear cy  append cy to years  }   //methods  CalendarYear getCalendarYear(int year){  //returns CalendarYear object  } } |

### *6.4.2. Calendar Year*

|  |
| --- |
| Class CalendarYear{  //variables  CalendarMonth[] months  Int year  Calendar calendar  //constructors  CalendarYear(){  months = new array of CalendarMonth objects  for months in year  initialize CalendarMonth cm  append cm to months  }   //methods  CalendarMonth getCalendarMonth(int month){  //returns CalendarMonth object  } } |

### *6.4.3. Calendar Month*

|  |
| --- |
| Class CalendarMonth{  //variables  CalendarDay[] days  Int month  CalendarYear year  //constructors  CalendarMonth(){  days = new array of CalendarDay objects  for days in month  Initialize CalendarDay cd  append cd to days  }  //methods  CalendarDay getCalendarDay(int day){  //returns a CalendarDay object  }  } |

### *6.4.4. Calendar Day*

|  |
| --- |
| Class CalendarDay{  //variables  CalendarEvent[] events  Int day  CalendarMonth month  //constructors  CalendarDay(){  events = new array of CalendarEvent objects  for events in day  initialize CalendarEvent ce  append ce to events  }  //methods  CalendarEvent[] getCalendarEvents(){  //returns an array of Calendar Events  }  } |

### *6.4.5. Calendar Event*

|  |
| --- |
| Class CalendarEvent{  //variables  string title  Time timeStart  Time timeEnd string note string location string label  CalendarDay day  //constructors  CalendarEvent(string \_title, Time \_time, string \_note, string \_location, string \_label){  set title, time, note, location, and label  }   //methods  string getTitle(){//returns title}   void setTitle(string \_title){//set title}   Time getTime(){//returns time}   void setTime(Time \_time){//sets time}   string getNote(){//returns note}   void setNote(string \_note){//set note}   string getLocation(){//returns location}   void setLocation(string \_location){//set location}   string getLabel(){//returns label}   void setLabel(string \_label){//sets label} } |

### *6.4.6. FileIO Status*

|  |
| --- |
| Class FileIOStatus{  //variables  Status currentStatus  Int errorCode  Calendar storedValue  Boolean isFromSaving  Lock lock  } |

## 6.5. File I/O

### *6.5.1. Sample File Structure*

|  |
| --- |
| V,1,0,0,2,\n Y,0,M,2,D,5,E,11:50:00,12:20:00,"Sample Title String",E,12:50:00,13:20:00,"Other Sample Title String",D,6,E,00:00:00,23:59:59,"Sample All Day Title",\n Y,20,M,5,D,1,E,01:00:00,01:50:00,"Sample Event in 5/1/1990 01:00 until 01:50",M,6,D,2,E,02:00:00,02:50,00,"Sample Event in 6/2/1990 02:00 until 02:50",\n |

### *6.5.2. Read From File*

|  |
| --- |
| Class LoadEvents{  //methods  void ThreadedLoad(string fileName, FileIOStatus status){  //reads file  }  CalendarYear ConvertFromString(string line){  //converts read in string to CalendarYear object  }  FileIOStatus LoadCalendar(string fileName){  FileIOStatus status = new FileIOStatus()  Create Coroutine ThreadedLoad(fileName, status)  Return status  }  } |

### *6.5.3. Write to File*

|  |
| --- |
| Class SaveEvents{  //methods  void ThreadedSave(Calendar cal, string fileName, FileIOStatus status){  //runs separate thread to save data  }  string ConvertToString(CalendarYear year){  string output = “Y,” + year.year  for CalendarMonth month in year  output = output + “,M,” + month.month  For CalendarDay day in month  output = output + “,D,” + day.day  For CalendarEvent e in day  output = output + “,E,” + e.timeStart +”,” + e.timeEnd +”,” + e.title  return output  }  FileIOStatus SaveCalendar(Calendar cal, string fileName){  FileIOStatus status = new FileIOStatus()  Create Coroutine ThreadedSave(cal, fileName, status)  Return status  }  } |

## 6.6. Constant Classes

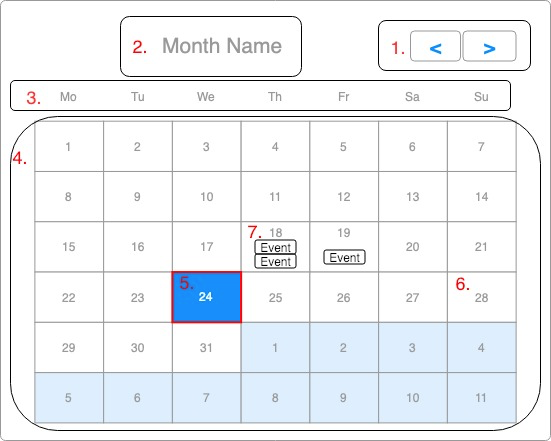
### *6.6.1. Status Constants*

|  |
| --- |
| Enum Status{  Waiting, Success, Failed //Possible enum values for Status  }; |

# 

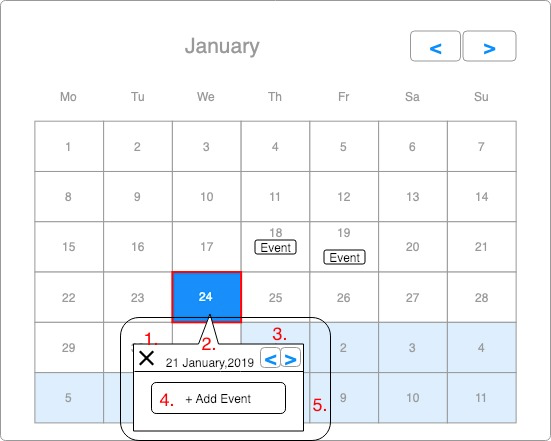
# **7. Appendices**

## 7.1. Appendix 1: Home Screen (Drawing #1)



1. Click the left button to go to previous month or click the right button to go to next
2. month.
3. Month name changes when clicking on left/right buttons, and opens up to current month.
4. Days of the week stay consistent (monday-sunday)
5. Days of not current month show up, but are in light blue color boxes.
6. Days of current month are in white boxes.
7. View drawing #2
8. View drawing #3

## 7.2. Appendix 2: Blanked Day Clicked (Drawing #2)



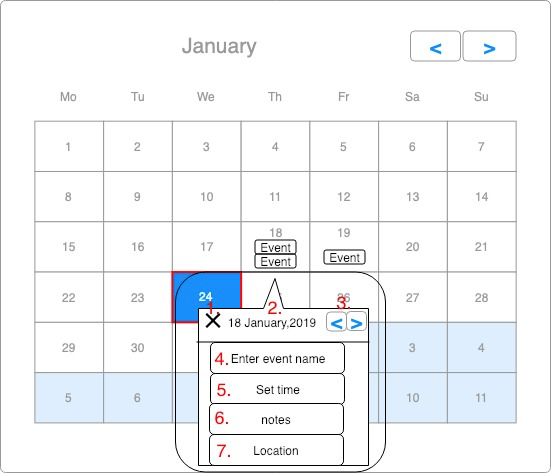
1. Click to exit single day view.
2. Displays selected date.
3. Go to previous or next day when clicking on left/right button.
4. Add a new event for the day.
5. Popup goes over calendar to display the single day view.

## 7.3. Appendix 3: Filled Day Clicked (Drawing #3)

1. Click to exit single day view.
2. Displays selected date.
3. Go to previous or next day when clicking on left/right button.
4. Add a new event for the day.
5. Popup goes over calendar to display the single day view.

6-7. Click to go into edit mode of event, go to drawing #4/5 for add/edit mode.

## 7.4. Appendix 4: Add Event (Drawing #4)

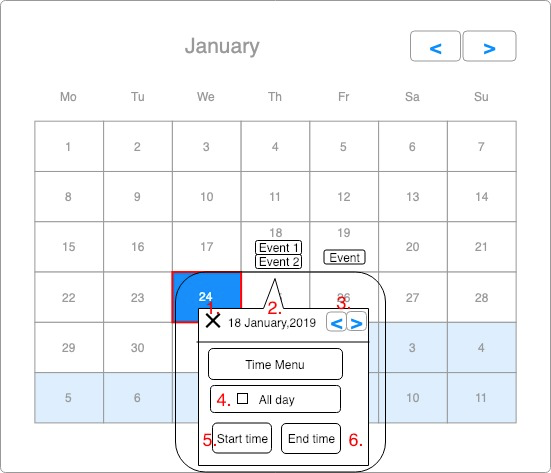


1. Click to exit single day view.
2. Displays selected date.
3. Go to previous or next day when clicking on left/right button.
4. Click to enter event name.
5. Click to set time (drawing #6)
6. Click to add notes about the event.
7. Click to add location.

## 7.5. Appendix 5: Edit Event (Drawing #5)

1. Click to exit single day view.
2. Displays selected date.
3. Go to previous or next day when clicking on left/right button.
4. Event name is visible, can click on it to edit.
5. Time is visible, can click on it to edit.
6. Note is visible, can click on it to edit.
7. Location is visible, can click on it to edit.

## 7.6. Appendix 6: Time Editor (Drawing #6)



1. Click to exit single day view.
2. Displays selected date.
3. Go to previous or next day when clicking on left/right button.
4. If all day box is checked, grey out start time and end time.
5. Edit start time (hh:mm:am/pm).
6. Edit end time (hh:mm:am/pm).