Instructor: Rakeshkumar Shukla
Course: CSCI 5335-01E
31 March 2021

Project Design Report
Better Notes

Team 01E 5: Accidental Consequences
Shane Casiano - Email: sc10196@georgiasouthern.edu; Eagle ID: 901014912
Jade Spahr - Email: Js21914@georgiasouthern.edu; Eagle ID: 901007178
Jacob French - Email: jf07042@georgiasouthern.edu; Eagle ID: 900965033
Alexis Jones - Email: aj06542@georgiasouthern.edu; Eagle ID: 900963115
Seth Aitcheson - Email: sa11637@georgiasouthern.edu; Eagle ID: 901194817

Table of Contents

1. Introduction

a. Purpose of the system

Memory is a large problem for both elderly and mentally challenged peoples. As such our team plans to create a program that could in some way improve the daily lives of this particular group of people. While there are many different implementations of similar ideas, much behind memory issues tend to stem from under-stimulation. The base idea we plan to tackle is to give people some method of storing the information they need in an intuitive way that could also give them the opportunity to remember what was written using images, rather than simply using words that may become jumbled or lost in the mental process. Thus the purpose of the program is to improve the capability of mentally challenged individuals specifically on memory of certain important upcoming events or tasks.

b. Design goals

Performance: Since Better Notes is being built to be working on Windows 10, the program will be designed so that it runs smoothly while not interfering with other programs that are running on the user computers. Better notes will also be able to clean the work areas such as extracted notes and the temp directories, so to allow the computer to run the program better.

Reliability: For our system Better Notes, it first should be bug-free and should not crash easily. If the system does crash the program will have a backup recovery option for any unsaved. The program will also offer a consistent recent item view for every note that it is open and saved. It will also function on minimum Windows hardware requirements for the target environment.

User-Friendliness: Since Better Notes is being designed specifically for people with disabilities,the system should be simple for these people to understand. The user-interface will be user-friendly so that the user won't be overly confused with the program. There will be several functions that will help the user such as recent notes, and backup recovery.

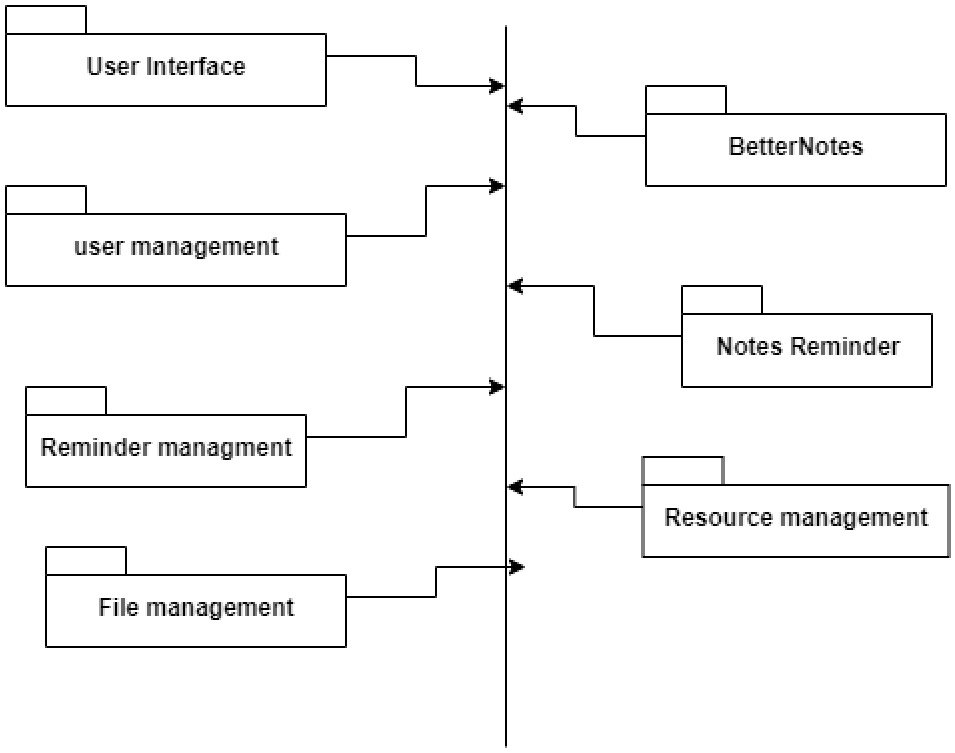
Integration: Better notes will have the ability to allow users to set reminders for the notes that they created. So the program will allow for reminders to be sent ,to the selected notification options, in already existing formats on either Windows or System.net.

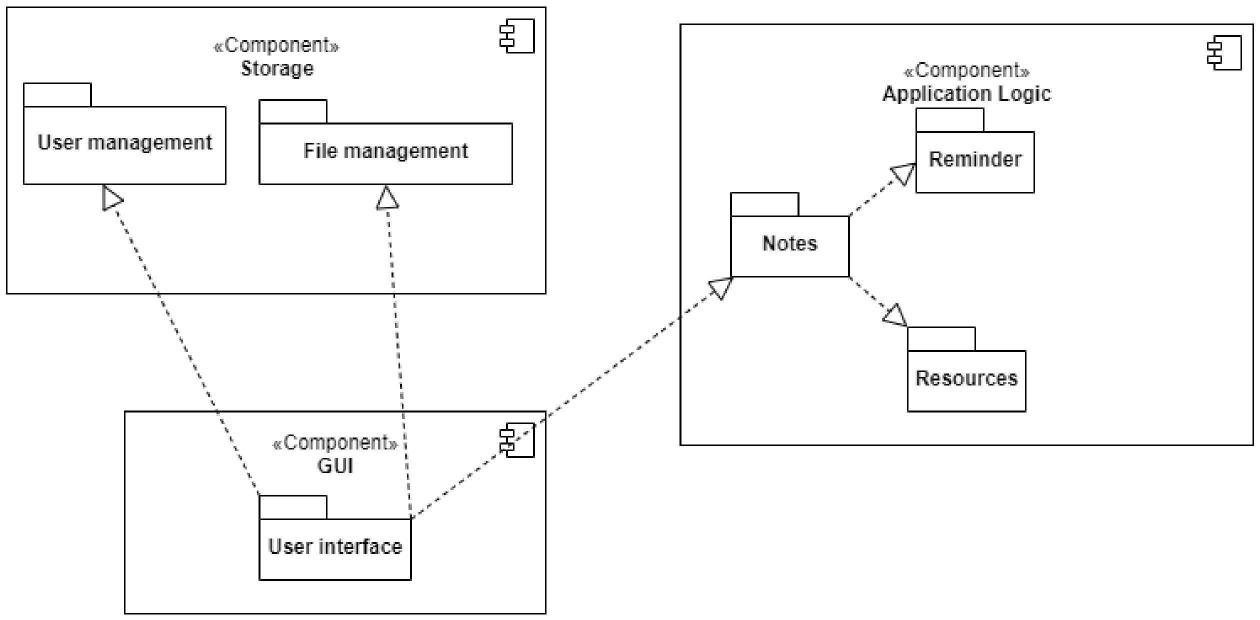
Implementation: Since Better Notes is being programmed with the target environment being the windows. Better Notes will be available to work on any working PC with any version of Windows 10 version 20h and any following updates will also work with the program.

Usability: The program is designed for users to record information in a notepad setting with additional features. Since it is most likely that the user will most likely have multiple notes the program will display short summaries of each note that the user makes. It will display the creation date and other metadata values.

2. System Architecture

a. Subsystem decomposition





b. Hardware/software mapping

Our software is developed for Windows 10 based operating systems running the Windows anniversary build. As such, it will be capable of running on each version post this one. Other requirements are the .NET Framework, specifically version 4.7.2 or later depending on specific machine situations, though this framework should be auto-installed as part of our installer. All other software dependencies are directly included in the software. Hardware requirements will be the minimum requirements of Windows 10 Anniversary.

- 50 GB Hard Drive
- 4GB of RAM
- Display that has at least a resolution of 1280x720p.
- Input devices will also be required, though these are already required by the target operating system.
- Intel UHD 600 series graphics or Vega Integrated Graphics or above

Are all recommended, though BetterNotes itself requires only:

- 200MB of Disk space
- 2GB of RAM (general usage is only 30-40 MB)
- 480p Panel
- If your computer has display output, the program will work.

c. Persistent data management

Persistent data management is handled mostly by CSV files. These files contain a list of Users, which are put in memory upon start of the application. A list of recent files, which is only loaded on open of the Homepage Window to populate the recent notes list. Finally, we find a list of reminders. This CSV file is primarily used to create Timer objects on startup to send notification on elapsed time. The CSV is also editable by the user using the NotificationManagement Window. This Window allows a user to see a visual representation of notifications from the CSV and allows the user to delete said entry in the case that they have removed the file, but have not removed the reminder.

d. Access control and security

In general, since our program lacks network based features, security and access control is a non-issue. Our program relies on the security already provided by Windows as a host operating system, however, there is a level of user control based on the assumption that many people may or may not have multiple users utilizing the same Windows User Account. Based on this assumption, users have been created to indicate specifically which user created a note object, and as a simplistic way for the user to select their default phone number and email address for settings. Using the users file system rather than a database has a few benefits and drawbacks, firstly, a user, or us as the developers need not concern themselves with the complexities of hosting a database without the necessity of one. Furthermore, the security of a database that may be hosted on the same device as the program is questionable mainly due to attacks that may target such a database more easily than a file system. This means that overall, our program has less entry points and the users data is stored only on their own device. While asynchronous programming became much less viable due to this decision, security is overall improved. While the security of the program is still questionable, since files are stored in plaintext, it is assumed that a user will perform their due diligence in protecting their own device. Quite simply, our program makes no changes to the protections a user should otherwise be taking to protect the system, and will not leave a user more vulnerable then they may have been before.

e. Boundary conditions

Initialization

A setup program will set up the program and install all required dependencies. On the first launch of the program, the user will be prompted to set up certain metadata entries, such as users. Subsequently the program will be launched by opening the .exe file, or by opening a

.bnot file from file explorer. The program will then either open a Homepage Window, or a BetterNotesMainView window, depending on entry type to the program. If this is the first launch since killing the program, the program will also initialize MinimizedView to allow the program to stay in memory.

Termination

Termination of the program should generally not occur, as it presents itself as a background application to support the reminder features. Regardless of this, an exit button is provided on the taskbar menu to allow the user to exit the program if they so desire. Ending an open instance of an open note causes the program to ask if a user would like to save a file (if they haven't manually done this yet). The program will can then save the file by writing the open folder to the save location using the Archive class. Alternatively, if the user wishes to discard changes, this will simply cause the current open folder to be deleted.

Failure

On failure of the program, as with any other program that saves files to memory, the program will simply lose all of the data written since last save. While it may be possible to recover the document, if the program crashes, there is a chance of corruption due to possible active streamwriters. Due to this, we decided against offering a document recovery for the time being, until such time where the corruption issue can be avoided, either by intermittently pausing the streamwriters, or auto saving the documents.

3. Subsystem Services

Our Better Notes Application has three subsystems, the User Interface system, the system for the Application Logic, and the Storage system.

The User Interface(UI) contains all the views or containers that allow the user to create their notes and reminders. The UI Interface will start on the Home Page which will show the recent notes and have options to manage users, modify notifications and open new or existing notes. Selecting any of these will bring the user to the related views. Selecting New Note will bring the user to a view that will ask for the name of the note, the user creating the note, and if the note is just a note or if it is also going to have a reminder. Upon the user selecting the option to have a reminder a new section of this view will pop up, asking for all the information required to create a reminder to attach to the note.The actual creation of the reminder notification is actually a function of the Application logic, that gets information from the users input into the UI. Upon the user completing all the required fields on this view and hitting create a new UI view will pop up to actually start creating a note. From the Homepage if Open Existing Note is selected a file explorer will pop up and let the user navigate to their existing .bnot file and open it in the app upon selection. When a note is opened the user can update the reminder/notification created, and add information to their note with text or by finding additional resources with the resource panel that will provide functions like text to speech, speech to text and the insertion of images or videos. The UI will note let the user exit this note without making sure that the user has a chance to save by popping up a new view that will prompt the user to save.

Selecting the User Management from the Home Page will bring the user to a view that presents a table of current users and their related information. This is where the users will put in their name, phone carrier, phone number and email to be added to the User Management Table. This will also be the view that the user will be brought to upon first opening the application after installation. The reason for this is no note can be made without an author so the first thing every user has to do before creating their first note is create a user to have as an author of a note. There is also a way for the user to interact with the interface to update and delete users by clicking on the buttons next to the users corresponding to the action that the user desires to change about the user data. There is also a way to access this view through an open note’s menu bar option of User. When the user selects Notification Management the GUI will take the user to a view that shows all the current note notifications that have been created with the date and time the notification will be sent out and the type of notification that will be sent for each note with a notification.

The Storage subsystem that exists on the user’s computer because our system doesn’t use a database that depends on a server. This subsystem has two parts: storage of the user data and the storage of the note file. The storage of the user data is stored in metadata related to our application. This means that if the application was uninstalled, the metadata will be deleted. The storage of the note is chosen by the user, but is saved in a .bnot format, which is a format that we created for our application. This note storage is separate from the metadata of the note’s notification, but this is still something that is handled with the Storage subsystem.

The Application logic subsystem contains the functions that create content and actions for the note being created. The content of the note is created within a rich textbox that would be saved with the Storage subsystem. There are also resources that include

Text-to-Speech, Speech-to-Text, Image Insert and possibly Video Insert to add additional content to the rich text box that is the created note. Text-to-Speech, Speech-to-Text depend more on the Windows device Speech-to-Text and Text to Speech function. Our apps use the Windows functions to then save the speech portion of both functions under the same directory as the note when that audio is either recorded from the user or text is translated into audio. The functions Video Insert will allow the user to search a video and insert it into the note and likewise Image Insert will search images from user input and insert that into the note. Both Video Insert and Image Insert require connection to the internet.Another function is a way to convert the note to a pdf format upon completion. The saving of this PDF file is handled by the Storage subsystem. The function for creating the reminder for the note with the given user information is handled here by setting up a timer after storing the metadata for the reminder.

4. Object Design

a. Object design trade-offs

Understandability vs Functionality:

Overall, our program focuses on understandability instead of functionality. Striking a balance between the two is obviously a goal, but leaning to one instead of another makes it easier to target a specific audience. Regardless of the fact that our program tends to focus on understandability by means of placing functions in multiple places and correcting for overall user errors and simplifying features as much as possible, the program still contains feature rich selecting in a light-weight packages that will hopefully not interfere with any systems our target audience may already be used to.

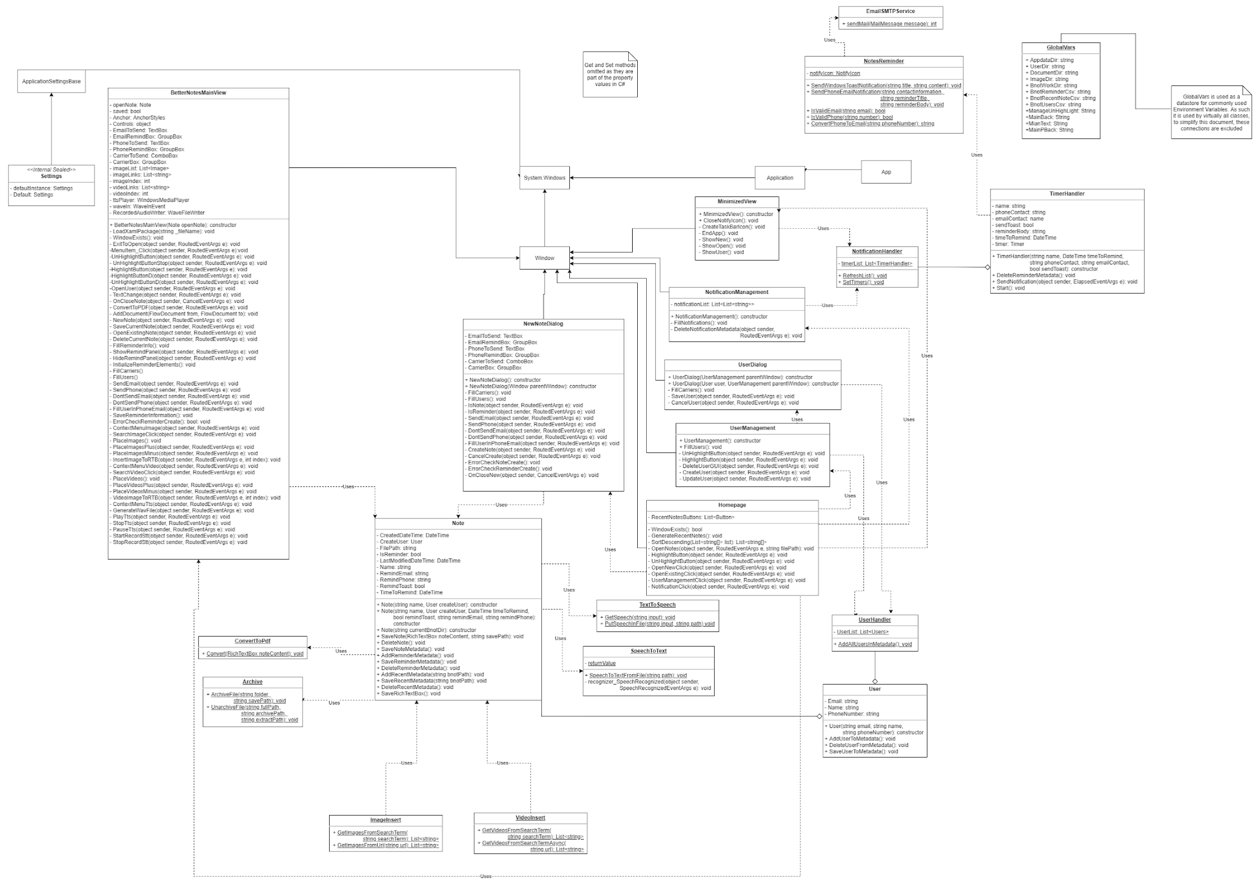
Memory vs Maintainability:

Memory was a huge priority for this project. In general, it is why objects are loaded into memory only sporadically. Overall, only one design choice led directly to more memory usage despite the lack of need. This was the User class. The User class could have its functionality simplified to simply being a polled CSV, much like the notification reminder list. Instead, to display the different options when designing such a program, we chose Users to be loaded into memory as well. Unlike users, the recent notes and notification lists both have no direct objects associated with them in memory. Instead, the files are read and modified in-place when needed and are only loaded to memory when they are directly being referenced. These files are later unloaded from memory by way of .NETs garbage collector. Maintainability is obviously seen as an afterthought due to the high priority to minimal memory consumption (we want this program to be capable of running smoothly on Windows 10's minimum requirements) we chose to directly implement classes with little to no abstraction.

Development Time vs User Experience:

Time is obviously an issue with such a large project over a mere three months. This is compounded by the fact that we decided early on to prioritize user experience. While we could have chosen to code this program in any language, C# with the .NET Framework ended up being ideal, this is because it would allow us to more precisely control UI elements as they would display on Windows, and simplifies the release process. Overall, we feel that while we could have lessened our development time by using more familiar tools, such as JavaFX, it would also create more stringent system requirements and complicate the UI elements as they would mainly be dynamically created. Overall, we still feel that we have hit all major development schedule goals and have remained on track for this program throughout the semester. Looking back, this decision could have been detrimental, however, in our specific case, our choice to prioritize user experience, we believe, has yielded positive results.

b. Final object design



c. Namespaces

System

System was primarily used when identifying basic data types such as Double.NaN and the DateTime element.

System.ComponentModel

ComponentModel is the namespace of which CancelEventArgs is a child. This allows us to interrupt normal app closing procedures to perform necessary cleanup to prevent any issues in persistent storage.

System.Collections.Generic

Collections, and specifically, Collections.Generic contains the List<> data type, which is used throughout BetterNotes as a method of storing information in memory.

System.Globalization

Globalization is used for its IdnMapping class to get chars in confirming that an email is correctly formatted

System.IO

IO is used here for its Directory and File classes, this is the primary means of interaction between the information in memory, and the information placed in persistent storage.

System.IO.Compression

Compression is the class that allows us to save the information from persistent metadata into a filetype that the user can place anywhere on their device, or even, another device. Each note has its own folder in metadata which is zipped by a save function and sent to a .bnot file, the reverse is true when a user opens a file.

System.Linq.IEnumerable

Linq.IEnumerable is used only for the Select function, this allows me to iterate an IEnumerable more efficiently before sending it to a list.

System.Net

System.Net is used in sending phone (which is translated to an email address) and email notifications, as well as accessing other webutilities with ImageInsert and VideoInsert.

System.Net.Mail

Mail is specifically used in SMTPService to send an email through an SMTP server to send a user notification.

System.Printing

Printing is used as an intermediary step between PDF output of a given note file.

System.Speech

Speech is used for its ability to convert text to speech and the reverse in either memory or as an input/output file.

System.Text.RegularExpressions

RegularExpressions are mainly used in input validation.

System.Threading.Tasks

Tasks are objects returned from asynchronous methods. YoutubeExplode is written almost entirely asynchronous and as such, Tasks are needed to synchronize each function with its invoker.

System.Timers

Timers are used to allow the program to time when to send a notification about a note, be it email, phone, or toast.

System.Windows

Windows is used throughout the program as the primary gateway to Windows operations. It allows us to identify common directories, but is more commonly used in GUI design as it constraints a majority of the functions needed. Elements such as buttons, panels, and the Window class are found here.

System.Windows.Media

Also used in GUI, Media contains items such as a SolidColorBrush, or other drawing elements not contained in System.Drawing.

System.Windows.Controls

As in System.Windows, GUI elements used throughout BetterNotes can be found in the Controls namespace.

System.Windows.Documents

Documents is used mainly in controlling the XamlPackage used to store a user's note. It is also used in the creation of ToastNotifications whose format is XML

Microsoft.Win32

Used for its SaveFileDialog and OpenFileDialog classes, this allows a user the consistent experience of saving files as is used in all other major Windows programs.

HtmlAgilityPack

HtmlAgilityPack is used in scraping Google Search results for a specific image.

Xceed.Wpf.Toolkit

This namespace contains an extended WPF toolkit with items such as a DateTimePicker, and other useful elements used in creating the GUI of BetterNotes

YoutubeExplode

YoutubeExplode is used in scraping Youtube results for a specific video.

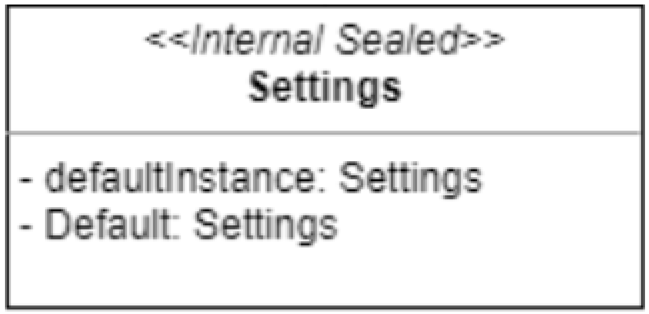
BetterNotes

The primary backend namespace. This is used as a reference in BetterNotesGUI to allow it to use items such as the Note Class. This means that our code remains modular, and it is not reliant on one or the other part to compile.

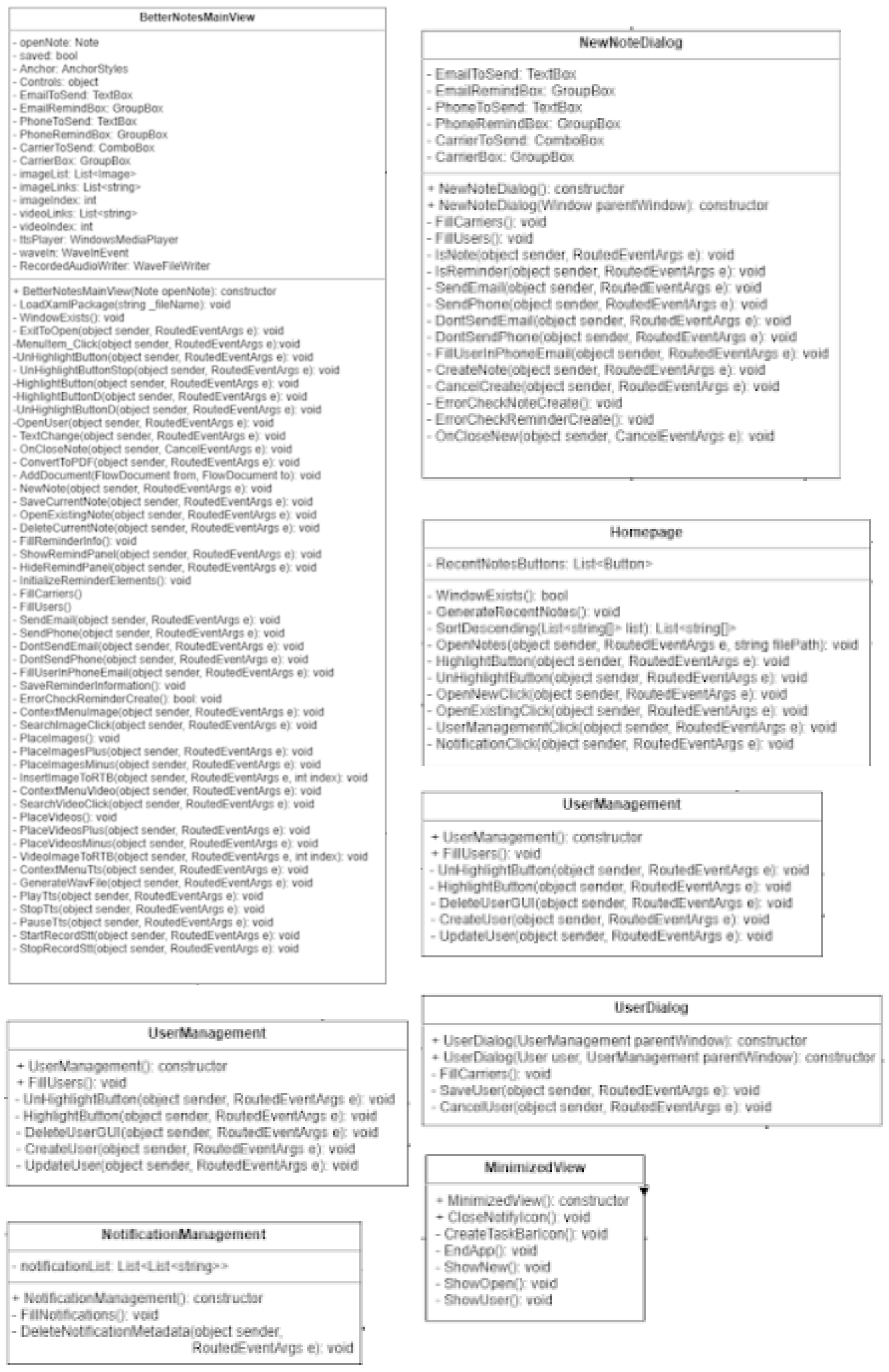
BetterNoteGUI

The namespace that contains all of our GUI elements.

d. Class Interfaces



Settings implements System.Windows.ApplicationSettingsBase and is used as a dependency for WPF.



```
BetterNotesMainView
- openNote: Note
- saved: bool
- Anchor: AnchorStyles
- Controls: object
- EmailToSend: TextBox
- EmailRemindBox: GroupBox
- PhoneToSend: TextBox
- PhoneRemindBox: GroupBox
- CarrierToSend: ComboBox
- CarrierBox: GroupBox
- imageList: List<Image>
- imageLinks: List<string>
- imageIndex: int
- videoLinks: List<string>
- videoIndex: int
- ttsPlayer: WindowsMediaPlayer
- waveIn: WaveInEvent
- RecordedAudioWriter: WaveFileWriter
+ BetterNotesMainView(Note openNote): constructor
- LoadXamlPackage(string _fileName): void
- WindowExists(): void
- ExitToOpen(object sender, RoutedEventArgs e): void
-MenuItem_Click(object sender, RoutedEventArgs e):void
-UnHighlightButton(object sender, RoutedEventArgs e): void
- UnHighlightButtonStop(object sender, RoutedEventArgs e): void
-HighlightButton(object sender, RoutedEventArgs e): void
-HighlightButtonD(object sender, RoutedEventArgs e): void
-UnHighlightButtonD(object sender, RoutedEventArgs e): void
-OpenUser(object sender, RoutedEventArgs e): void
- TextChange(object sender, RoutedEventArgs e): void
- OnCloseNote(object sender, CancelEventArgs e): void
- ConvertToPDF(object sender, RoutedEventArgs e): void
- AddDocument(FlowDocument from, FlowDocument to): void
- NewNote(object sender, RoutedEventArgs e): void
- SaveCurrentNote(object sender, RoutedEventArgs e): void
- OpenExistingNote(object sender, RoutedEventArgs e): void
- DeleteCurrentNote(object sender, RoutedEventArgs e): void
- FillReminderInfo(): void
- ShowRemindPanel(object sender, RoutedEventArgs e): void
- HideRemindPanel(object sender, RoutedEventArgs e): void
- InitializeReminderElements(): void
- FillCarriers()
- FillUsers()
- SendEmail(object sender, RoutedEventArgs e): void
- SendPhone(object sender, RoutedEventArgs e): void
- DontSendEmail(object sender, RoutedEventArgs e): void
- DontSendPhone(object sender, RoutedEventArgs e): void
- FillUserInPhoneEmail(object sender, RoutedEventArgs e): void
- SaveReminderInformation(): void
- ErrorCheckReminderCreate(): bool: void
- ContextMenuImage(object sender, RoutedEventArgs e): void
- SearchImageClick(object sender, RoutedEventArgs e): void
- PlaceImages(): void
- PlaceImagesPlus(object sender, RoutedEventArgs e): void
- PlaceImagesMinus(object sender, RoutedEventArgs e): void
- InsertImageToRTB(object sender, RoutedEventArgs e, int index): void
- ContextMenuVideo(object sender, RoutedEventArgs e): void
- SearchVideoClick(object sender, RoutedEventArgs e): void
- PlaceVideos(): void
- PlaceVideosPlus(object sender, RoutedEventArgs e): void
- PlaceVideosMinus(object sender, RoutedEventArgs e): void
- VideoImageToRTB(object sender, RoutedEventArgs e, int index): void
- ContextMenuTts(object sender, RoutedEventArgs e): void
- GenerateWavFile(object sender, RoutedEventArgs e): void
- PlayTts(object sender, RoutedEventArgs e): void
- StopTts(object sender, RoutedEventArgs e): void
- PauseTts(object sender, RoutedEventArgs e): void
- StartRecordStt(object sender, RoutedEventArgs e): void
- StopRecordStt(object sender, RoutedEventArgs e): void
```

```
NewNoteDialog
- EmailToSend: TextBox
- EmailRemindBox: GroupBox
- PhoneToSend: TextBox
- PhoneRemindBox: GroupBox
- CarrierToSend: ComboBox
- CarrierBox: GroupBox
+ NewNoteDialog(): constructor
+ NewNoteDialog(Window parentWindow): constructor
- FillCarriers(): void
- FillUsers(): void
- IsNote(object sender, RoutedEventArgs e): void
- IsReminder(object sender, RoutedEventArgs e): void
- SendEmail(object sender, RoutedEventArgs e): void
- SendPhone(object sender, RoutedEventArgs e): void
- DontSendEmail(object sender, RoutedEventArgs e): void
- DontSendPhone(object sender, RoutedEventArgs e): void
- FillUserInPhoneEmail(object sender, RoutedEventArgs e): void
- CreateNote(object sender, RoutedEventArgs e): void
- CancelCreate(object sender, RoutedEventArgs e): void
- ErrorCheckNoteCreate(): void
- ErrorCheckReminderCreate(): void
- OnCloseNew(object sender, CancelEventArgs e): void
```

```
Homepage
- RecentNotesButtons: List<Button>
- WindowExists(): bool
- GenerateRecentNotes(): void
- SortDescending(List<string[]> list): List<string[]>
- OpenNotes(object sender, RoutedEventArgs e, string filePath): void
- HighlightButton(object sender, RoutedEventArgs e): void
- UnHighlightButton(object sender, RoutedEventArgs e): void
- OpenNewClick(object sender, RoutedEventArgs e): void
- OpenExistingClick(object sender, RoutedEventArgs e): void
- UserManagementClick(object sender, RoutedEventArgs e): void
- NotificationClick(object sender, RoutedEventArgs e): void
```

```
UserManagement
+ UserManagement(): constructor
+ FillUsers(): void
- UnHighlightButton(object sender, RoutedEventArgs e): void
- HighlightButton(object sender, RoutedEventArgs e): void
- DeleteUserGUI(object sender, RoutedEventArgs e): void
- CreateUser(object sender, RoutedEventArgs e): void
- UpdateUser(object sender, RoutedEventArgs e): void
```

```
UserDialog
+ UserDialog(UserManagement parentWindow): constructor
+ UserDialog(User user, UserManagement parentWindow): constructor
- FillCarriers(): void
- SaveUser(object sender, RoutedEventArgs e): void
- CancelUser(object sender, RoutedEventArgs e): void
```

```
UserManagement
+ UserManagement(): constructor
+ FillUsers(): void
- UnHighlightButton(object sender, RoutedEventArgs e): void
- HighlightButton(object sender, RoutedEventArgs e): void
- DeleteUserGUI(object sender, RoutedEventArgs e): void
- CreateUser(object sender, RoutedEventArgs e): void
- UpdateUser(object sender, RoutedEventArgs e): void
```

```
MinimizedView
+ MinimizedView(): constructor
+ CloseNotifyIcon(): void
- CreateTaskBarIcon(): void
- EndApp(): void
- ShowNew(): void
- ShowOpen(): void
- ShowUser(): void
```

```
NotificationManagement
- notificationList: List<List<string>>
+ NotificationManagement(): constructor
- FillNotifications(): void
- DeleteNotificationMetadata(object sender,
                             RoutedEventArgs e): void
```

All of the previous classes implement System.Windows.Window. A class in the WPF library used to generate and control objects on a Window. Along with this, these Window objects also each extend their own control surface which is used to allow the Windows.Controls namespace objects to properly display within the windows.

App, which has no member functions or variables (it is only used for its included XAML file) generates application wide styles, fonts, and other UI elements. It inherits from System.Windows.Application

5. Conclusions

The overall objective of the BetterNotes project is to provide software in which users with memory difficulties can record information in an easier way and to provide a method to remember that information in a convenient fashion. We will make a software that stores information into a notepad application and can insert the appropriate images that associate with the information given. We also plan to provide a reminder setting for people to set to remind the person that they made the notes in each format as well as a way that will allow users to store these notes in other formats such as PDF. We will also offer a feature that will allow users to use speech to text options for those who have difficulties typing and a text to speech option for those that may struggle to read the included font types. The primary goal is to create an application conducive to simplifying the lives of those that specifically struggle with memory capacity, specifically, those with memory loss issues. The program could provide structure and a “home base” of sorts that can be checked regularly, the program will also be capable of specifically aiding this group by sending reminders (if set by the user) for those who may still be adjusting to checking the program regularly. Overall, implementing the program will require some further research into specific assistive technology, however, as a basis, this project is growing into a program that may actually make a difference in people’s lives.