**OiGO – Augmentative and Assistive Communication Application**

Documentation – by Assistive Communication Technologies™

Written by Andrew Luis Lopez

Version 1.0.0

Table of Contents

1. Summary of OiGO ………………………………………………………………………3
2. Xamarin Documentation ………………………………………………………………...3
3. Components and Example User Stories …………………………………………………4
   1. User Stories………………………………………………………………………4
      1. SELF……………………………………………………………………...4
      2. Help Me…………………………………………………………………..6
      3. Customization and HelpMeCustomization……………………………….7
      4. CustomizeCustomization and HelpMeCustomizeCustomization………..8
   2. Components………………………………………………………………………9
   3. Pages……………………………………………………………………………..10
      1. MainPage………………………………………………………………...10
      2. HelpMe…………………………………………………………………..12
      3. Customization…………………………….……………………………...14
      4. CustomizeCustomization………………. ……………………………….15
      5. HelpCustomization………………………………………………………16
      6. HelpCustomizeCustomization…………………………………………...16
4. Tutorial…………………………………………………………………………………..17

1. Summary of OiGO

OiGO is an augmentative and assistive communication (AAC) mobile application, designed to facilitate communication and directly express the emotions of the user. The application is meant to be used by any person, patient, etc. who suffers from some form of communication disorder. This can range from someone with autism, someone who is mute and/or deaf, someone with physical and/or mental trauma. The app needs to be accessible to all and any person who cannot properly communicate and cannot express their emotions. This app also needs to be accessible to any person regardless of age, sex, race, ethnicity, economic background, etc. The only requirement is that the users need to have basic reading comprehension skills and can operate a mobile device. In a short statement, the app is meant for “people who know what they want to say but are unable to say what they want”.

OiGO differentiates itself from other similar AAC mobile applications by

* Focusing on expressing the users’ emotions
* Bypassing the difficulties some users may have with typing on a mobile device and bypassing the use of indirect forms of communication, such as emojis, by presenting the user with clear, coherent options for communication
* Helping users in ways to correct their behavior
* Allowing customizability of components to fit to a user’s individual needs
* Being as accessible as possible

This 1.0.0 version of the OiGO project was made very simply due to the limited monetary, time, and personnel resources available. Hopefully, this documentation can help the next workers of the project better understand OiGO such that they will be able to vastly improve on the basic concept. While the version 1.0.0 app has been tested to meet user needs, the goal of future iterations should be to improve the quality, ease of use, and looks of the app while allowing room for additional features.

2. Xamarin Documentation

OiGO was created using the Microsoft Xamarin API (in order to make the application cross-platform between Android and iOS) and is written in C# and XAML. Prerequisites for working on the project include:

* The latest version of Visual Studio (the 1.0.0 version was made in Visual Studio 2017)
* The latest version of Microsoft Xamarin
* The Xamarin.Essentials library

Documentation for Xamarin can be found at: <https://docs.microsoft.com/en-us/xamarin/>. Heavy C# and XAML knowledge is not required (from personal experience). Intermediate programming knowledge, experience, and the Xamarin documentation should be enough to work on the project, although C# and XAML knowledge/experience is a plus. Experience working on other Xamarin applications will grant one the biggest advantage. To fully understand the OiGO code, one should review the Xamarin documentation for the following concepts:

* AbsoluteLayouts
* Common Xamarin objects, such as Images, Buttons, Frames, Labels, ObservableCollections, Pickers, Entry, etc.
* StackLayouts
* ListView
* ObservableCollection
* How to install Xamarin and Xamarin.Essentials
* Text-to-speech functionality in the Xamarin.Essentials library

3. User Stories , Components, and Pages

*User Stories*

The following user stories are meant to illustrate the primary method of use of the app from the user’s perspective. Understanding these stories will help in understanding the primary purpose and function of the components and pages.

* Social Emotional Language Formulation (SELF)
  + SELF is the primary function of OiGO and it helps the user seamlessly formulate sentences based around emotion. Predetermined and customizable options will be presented to the user, and the user can choose which options acts as which part of a sentence. All the chosen options are strung together into a full sentence.
  + The structure of a sentence consists of a subject + emotion + conjecture + conclusion.
    - ‘Subject’ begins the sentence and determines the subject of the sentence.
      * As of the now, the only two subjects are the user and whoever the user is communicating with. The only options for a subject are ‘I feel’ and ‘Do you feel’.
    - ‘Emotion’ is the specific emotion the user is trying to convey.
      * The six emotions OiGO is designed to convey is ‘happy’, ‘sad’, ‘angry’, ‘worried’, ‘scared’, and ‘frustrated’.
    - ‘Conjecture’ is the part of the sentence that connects the ‘subject + emotion’ to the ‘conclusion’. Conjectures usually consist of ‘because’, ‘that’, ‘when’, etc. Conjectures are not selectable options, however. Each CustomConclusion object will have a conjecture associated to it (see below).
    - ‘Conclusion’ is the custom response that conveys why the subject is feeling the chosen emotion.
    - Behind the Scenes: All Subjects are stored in an instance of GlobalData. Each Subject contains a list of Emotions. All instances of CustomConclusion are stored in a list in an instance of Customization. See the Components and Pages sections for more details.
  + User Story breakdown (with some behind the scenes info transparent to the user)
    - User starts the app. They are taken to the SELF page first.
    - Users see a back button in the top left and a “label” bar (with the OiGO logo on it) at the top. A purple rectangle with some selectable buttons takes up most of the screen in the middle. They should first be showing options for the ‘subject’ of the sentence. On the bottom, they see 4 buttons. One ‘+’ sign button to go to Customization, one ‘SELF’ button that is unselectable, one ‘Help Me’ button to go to Help Me, and a ‘Speak’ button to turn their sentence from text to speach.
    - The user wants to form a sentence. They can select any ‘subject’ option button presented to them in the middle purple rectangle. Once selected, the top label will update and show the selected option in words. The options on the rectangle will change to show the options for an emotion. The user can now select an option to convey an emotion. Again, the top label is updated to show the subject + the emotion (also, the label border’s color changes to match the emotion select i.e. it turns red when ‘angry’ is selected), and the rectangle now shows options for conclusions.
      * Along with the conclusions, some ‘arrow’ buttons may appear on the rectangle. The rectangle can only show so many options at a time. If some options are left off of the screen, the user can navigate to those options using the arrows. When an arrow is selected, the old options are removed from the rectangle and the new options are shown.
      * Behind the Scenes: All conclusion options are stored in a singular list in an instance of Customization, but only a certain number a chosen to be shown at a time. 9 conclusions can be shown on one screen at max, but to make room for the arrow buttons, only 7 conclusions are shown at a time if there are 10 or more total conclusions to show. The first 7 are shown at first, but pressing the right arrow shows the next set of 7 or less conclusions. The right arrow can keep showing the next set until there are no more to show. The left arrow shows the previous set of conclusions until the first set is shown. The arrow buttons may not appear if it is not possible to select the next or previous set.
    - Picking up from the previous bullet point, once the user selects a conclusion, the label is updated to show the subject + emotion + selected conjecture and conclusion (conjectures are automatically associated with a conclusion and displayed with the conclusion). A pop-up window appears that also shows the full sentence. The user can remove the window by pressing ‘OK’. The conclusions are not wiped from the rectangle, and the user can choose a new conclusion to replace the previous conclusion.
    - Pressing the device’s back button or the back button in the app will bring the user back to the set of options for the previous part of the sentence. Ex) If the user sees the conclusion options and presses back, the label will remove the chosen emotion and the rectangle will show the emotion options. Pressing back when viewing emotion options will clear the label of the chosen subject and the rectangle will show the subject options. Pressing back when viewing subject options does nothing.
    - Pressing the ‘+’ button at any time will bring up the Customization page, where the user can customize their conclusions.
    - Pressing ‘Speak’ at any time will cause the device to ‘speak’ the phrase in the label. The phrase doesn’t have to be a full sentence. If there is nothing in the label, nothing will be spoken.
* Help Me
  + Help Me is a component of OiGO that works very similarly to SELF, but its primary purpose is to present customizable behavioral solutions to the user. Behavioral Solutions in psychology are actions/behaviors that adjust our mood. In times overwhelming emotion, users of the app may need to rely on behavioral solutions in order to correct their behavior. For example, if a user becomes too mad when using the app (or in any other scenario), they can use Help Me to remind themselves of the behavioral solutions that can be used to calm themselves down. Behavioral solutions are customizable to meet an individual user’s needs.
  + Behavioral Solutions shall be known as Solutions as in the app they are represented by the Solution.cs class (See Components for more). The structure of a full Solution includes emotion + connection/conjecture + solution. When a user is feeling [an emotion], they can [perform a Solution] to adjust their behavior.
    - The Emotion and Conjecture parts of a Solution as the same as they were as parts in a sentence in SELF.
    - Emotions are stored in a list in an instance of HelpMeCustomization. All Solutions are stored in a list in an instance of HelpCustomization. See Components and Pages for more information.
  + User Story Breakdown
    - To get to the Help Me page, a user must select the Help Me button at the bottom of the page. When that happens, the Help Me page is brought up, and the Help Me button becomes unselectable while the SELF button does become selectable. To return to SELF, one can select the back button (when the emotion buttons are present on the Help Me page) or select the SELF button at the bottom of the page. This’ll switch the page from Help Me to SELF and make the SELF button unselectable while making the Help ME button selectable.
    - When Help Me is brought up, it is nearly identical to how the SELF page looks. Refer to that for more details. The main differences are the ones listed above and that the center rectangle now holds buttons for emotions instead of subjects.
    - The beginning of bringing up a behavioral solution starts at selecting an emotion. The Help Me page begins with buttons for selecting an emotion. The user should select the emotion which they are feeling overwhelmed by and wish to correct. When they select an emotion button, the top label will display the emotion text, the label’s border will change to the color associated with the emotion, and the emotion buttons will be replaced by the buttons for the behavioral solutions.
      * Arrow buttons may accompany the solution buttons in the same way they accompany the conclusion buttons in SELF. Refer to the SELF User Story for more details.
    - The user can select the solution they’d like to try out. Clicking a solution button will update the label to show the emotion + conjecture (which is automatically associated with the solution) + solution. A popup window will also appear showing the full solution. The popup window can be closed by pressing ‘OK’.
      * Additional preset words can accompany the solution to have read in some coherent sense. In example, when displaying the full solution, the app should automatically structure the solution such that it reads ‘When I am [emotion], I [conjecture] [solution]’. Ex. ‘When I am angry, I can count to 10.’
    - When pressing the back button, the user will return to the SELF page if they haven’t selected an emotion yet. If the user has selected an emotion and can see the solution buttons, then pressing the back button will remove the selection of the emotion (the label’s text will be blank and its border will be colorless/gray), the solution buttons will be removed and the emotion buttons will be shown, as if the user had just started the Help Me component.
    - Pressing the ‘+’ button at any time will bring up the HelpMeCustomization page, where the user can customize their behavioral solutions.
    - Pressing ‘Speak’ at any time will cause the device to ‘speak’ the phrase in the label. The phrase doesn’t have to be a full sentence. If there is nothing in the label, nothing will be spoken.
* Customization and HelpCustomization pages
  + The customization page is to allow the user to customize their conclusions/solutions. The customization page to change conclusion must be accessed when the user is on the SELF page while the customization page to change behavioral solutions must be accessed when the user is on the Help Me page. While technically two separate pages, the Customization and Help Me Customization pages are so similar, they’ll both be explained here. We’ll also somewhat bring up the CustomizeCustomization pages.
  + CCs and Solutions are organized by emotions as each CC and Solution object has a String representing one of the emotions that a user can express through OiGO. This customization page allows the user to reach all of the CC and Solutions on the app so they can add new ones, change existing ones, or delete old ones. The CC pages are the pages that allow the user to actually customize and change CCs and Solutions.
  + User Story Breakdown:
    - To reach a Customization Page, one must press the ‘+’ button on the SELF or Help Me pages. ‘SELF’ will lead to the customization page to customize CustomConclusions. ‘Help Me’ will lead to the customization page (called HelpCustomization.xaml.cs) to customize Solutions. When a Customization page is brought up, a list of emotions is first presented. At the top of the page, there is a ‘+’ button to bring one to the CustomizeCustomization pages (will be explained later).
    - Selecting an emotion on the list will bring up the CCs or Solutions associated with that emotion. The emotions on the list will be replaced by the options associated with the selected emotion.
    - Holding a CC or Solution will bring up a menu (individual to the device) with a ‘Delete’ option. Selecting the ‘Delete’ option will remove the held option from the list of options.
    - Pressing back on the device when viewing options will bring back the list of emotions, such that the user can select another emotion to bring up options associated with another emotion. Pressing back while viewing a list of emotions will exit Customization and go back to the respective SELF or Help Me pages.
    - The user can select either a CC or Solution or the ‘+’ button on Customization. Both will bring up the CustomizeCusotmization page, but with different effects. Pressing ‘+’ will bring up the CustomizeCustomization page blank, while selecting an option will bring up the CustomizeCustomization page (HelpCustomizeCustomization page for Solutions) with the page filled with information associated with that option.
* CustomizeCustomization and HelpCustomizeCustomization
  + The names of these pages will be transparent to the user. These pages will seem as extensions of the Customization function. These pages will allow the user to make changes to selected options from the previous page or make new options. The CustomizeCustomization page will edit CCs and the HelpCustomizeCustomization page will edit Solutions.
  + If the user is changing/adding a CC, there will be a ‘Picker’ to allow a user to pick a subject. Otherwise, both pages will have a ‘Picker’ to select an emotion and two ‘Entry’ lines so the user can fill in a conjecture and a CC or Solution. These items will first be blank if the user is adding an option, but will otherwise have the
  + information of the selected option from the previous page.
  + User Story Breakdown
    - The user can manipulate the objects as they see fit. They can change the subject, emotion, conjecture, and conclusion/solution of the new/selected item using the Picker and Entry options. Pressing the ‘Save’ button at the bottom of the page will save the changes to the existing item or add it to the list of options if it is new. This’ll also bring the user back to the respective Customization pages
      * ‘Picker’ objects act as minimizable lists. One can bring down the list to make a new selection.
      * ‘Entry’ objects act as blank lines that can be typed on. Users can type on the line to enter whatever they want to act as the respective conjecture or CC/Solution.
    - Pressing back at any time will bring the user back to the Customization pages without saving, effectively canceling the addition/changes.

*Components*

* Conclusion.cs
  + Conclusion.cs was the original class made to represent the conclusion component of the app but has since been depreciated.
* CustomConclusion.cs
  + CustomConclusion.cs is the updated conclusion class.
  + In Xamarin, the first main component is sentence construction. The user can select different options presented to them and the selected options form a coherent sentence. A sentence is split into different parts: the subject, the emotion, the conjecture, and the rationale/conclusion. There are different options for each part. The CustomConclusion class creates objects made of strings to hold each part of a sentence. Each object has a ‘subject’, ‘emotion’, ‘conjecture’, and ‘conclusion’ string, and each object essentially only have getter and setter methods for the strings. Each object essentially represents an end node when traversing the tree of possible options to choose from when constructing a sentence. In other words, each CustomConclusion object represents a possible sentence that can be constructed from the presented options. All CustomConclusion options stored in the application data make up every possible combination of ‘subject’, ‘emotion’, ‘conjecture’, and ‘conclusion’ options. See the SELF functionality user story for more information.
* Emotion.cs
  + The emotion class represents an emotion object. Each object has a ‘name’ string with a getter and setter. There were variables and methods to represent a conclusion linked to an emotion and to add and get those conclusions, but those variables and methods have been depreciated.
* GlobalData.cs
  + The application stores information onto files on the device. GlobalData reads the files and loads the information onto the device. The ‘library’ string variable is part of the path that must be appended to file names in order to access them. Init() is the function called when the app loads and is what actually reads the files and loads them into Customization, HelpCustomization, and Tutorial.
* Solution.cs
  + Very similar to the CustomConclusion.cs class, Solution.cs holds the strings for the emotions, conjecture/connection, and solutions for every behavioral solution in the Help Me component of the app. Help Me is implemented to remind users of behavioral solutions (activities they can perform) to adjust their mood and behavior. Every solution object represents an end node of traversing the tree of possible options to choose from when selecting a behavior solution. In other words, each Solution object represents a possible behavioral solution that can be made from the presented options. All Solution objects stored in the application data make up every possible combination of ‘emotion’ and ‘solution’. See the Help Me functionality user story for more information.
* Subject.cs
  + This class represents a subject object that holds a string ‘name’ that represents its value and a list of emotions. There are methods to return the ‘name’ and the list of emotions.

*Pages*

Xamarin allows the user to create individual and custom ‘page’ classes written in XAML and C#. These pages act as the user interface of the app. One page equates to one visible screen the user sees when using the app.

* StartPage.xaml and StartPage.xaml.cs
  + This was originally the start screen for the app but has since been depreciated. The app now automatically starts at the screen used for the SELF function.
* MainPage.xaml and MainPage.cs
  + This is the new start screen of the app, and serves as the main page for the SELF functionality of the app.
  + XAML
    - The xaml file is one big content page encapsulating one big AbsoluteLayout. Within that is one more AbsoluteLayout, who’s meant to just be a purple background. Within that are four components.
    - The first is an image that is meant to take up the entire screen. All other components exist on top of that image.
    - The next component is an AbsoluteLayout with several components. It has a button, meant to act as a back button. It also has a Frame that encompasses a Label called “text” and an image that acts as a logo. The constructed sentences are meant to appear on the Label, and the Frame changes colors based on the currently selected emotion option.
    - The next component is a Frame with an AbsoluteLayout within it, called “topFlex”. It has no children populating it in the code, but it is meant to hold Buttons that represent the selectable options for sentence construction. Buttons are removed and added based on which part of a sentence the user is constructing.
    - The last component is an AbsoluteLayout with 4 buttons as its children. The first button is a “+” sign that leads to the Customization page. The second buttons should say “SELF”, be grayed out and can’t be selected. This shows that the user is already in the SELF functionality page. The next button should say Help Me and lead to the Help Me functionality. The last button reads the text value on the “text” Label and speaks it back to the user using text-to-speech functionality.
  + C#
    - More specific documentation for the MainPage can be found within the comments of the code. MainPage.xaml.cs has a subjectHolder, emotionHolder, and a customConclusionHolder to hold those objects when a user selects an option for a sentence part. I.E., the subjectHolder will hold the selected subject chosen by the user.
    - MainPage also has a Time variable and various functions such as Handle\_Past/Present/Future, and Change\_Time, but these variables and functions have been depreciated.
    - The ‘page’ variable is used to determine which ‘page’ of options the user is looking at. In context, not all options for a conclusion/solution can be displayed at once. When there are more than 10, only 7 at a time can be displayed. ‘Page’ is used to determine which 7 options are displayed on topFlex at a time. When ‘Page’ is 0, the first 7 options are displayed. When ‘Page’ is 1, the next 7 options are displayed. And so on and so forth.
    - Strsize, proportion, and ff are variables used to determine the font size of certain text on buttons and labels.
    - Counter is used in functions that create buttons and helps calculate the position of buttons.
    - MainPage() is an initializer, that creates buttons for the selectable subject options and places them in topFlex.
    - Handle\_Back() determines what to do on a back button press. In MainPage, pressing back will bring up the previous options for the previous sentence part the user just chose from. Handle\_Back() determines what options are being displayed and uses the Handle\_GoTo\_[Blank]() functions to “go to” the previous set of options.
    - Handle\_Subject() handles when the user presses a subject button. It clears topFlex, gets the selected subject, and creates buttons for every emotion associated with the chosen subject and adds those buttons to topFlex. The ‘text’ label is also updated with the chosen subject.
    - Handle\_Emotion() handles when the user presses an emotion button. It clears topFlex, changes the color of the ‘border’ AbsoluteLayout, gets the selected emotion, and creates buttons for every CustomConlcusion (CC) associated with the emotion (CCs are stored in List ‘custom’, which is filled using the GetConclusion(emotion) function in Customization). The buttons are placed on topFlex, and the “text” label is updated.
      * Depending on the number of CCs, the function will display all of them or display some of them and also create “arrow” buttons. These arrow buttons use the “Handle\_Page\_Increase/Decrease” functions, but essentially, depending on the value of page, only certain CCs are shown, and the arrows adjust that page value.
    - Handle\_CustomConclusion() handles when the user pushes a CC button. It gets the selected CC, constructs a full sentence using the holder variables, and displays that sentence on the “text” label and an alert display box.
    - Handle\_Goto\_Subject() handles when a user goes back to the subject options. It is very similar to MainPage(), so see that for reference.
    - Handle\_Goto\_Emotion() handles when the user goes back to the emotion options. It is very similar to Handle\_Subejct(), so see that for reference.
    - Handle\_Goto\_Conclusion() handles when the user goes back to the conclusion options. It is primarily used when the user adjusts the page variable. It is very similar to Handle\_Emotion(), so see that for reference. Also see Handle\_Page\_Increase/Decrease().
    - Handle\_Page\_Increase/Decrease() respectively increases or decreases the page value and calls Handle\_Goto\_Conclusion(). The “arrow” buttons displayed with groups of CCs call this function. Depending on page number, Handle\_Goto\_Conclusion() will display a group of 7 or less CCs along with a left or right arrow. The left arrow decreases the page number and the right increases it. The arrows don’t appear if the page number can’t further be decrease/increased. See the ‘page’ variable.
    - Help\_Me\_Clicked() and Customize\_Clicked() change the page to the Help Me and Customization pages.
    - On\_Main\_Page() is used in MainActivity.cs to determine if Handle\_Back() can be called.
    - Speak() implements a text-to-speech function
    - AdjustFont() adjusts the font on buttons and labels.
* HelpMe.xaml and HelpMe.xaml.cs
  + This is the page implementing the Help Me functionality of the app.
  + XAML
    - The page layout is nearly identical to the layout in MainPage.xaml. See MainPage.xaml for reference.
  + C#
    - More specific documentation for HelpMe can be found in the comments of the code. Helpme.xaml.cs has an array ‘emotions’ of the 6 primary emotions used in the app. String ‘emotion’ and Solution ‘solution’ respectively hold the chosen emotion and behavioral solution.
    - Variables ‘page’, ‘strsize’, ‘proportion’, ‘ff’, and ‘counter’ maintain the same purpose as they did in MainPage.xaml.cs [see for reference].
    - HelpMe() is an initializer, that creates buttons for the selectable emotion options and places them in topFlex.
    - Handle\_Back() determines what to do on a back button press. In MainPage, pressing back will bring up the previous options for the previous sentence part the user just chose from. Handle\_Back() determines what options are being displayed and uses the Handle\_GoTo\_[Blank]() functions to “go to” the previous set of options.
    - Handle\_Emotion() handles when the user presses an emotion button. It clears topFlex, changes the color of the ‘border’ AbsoluteLayout, gets the selected emotion, and creates buttons for every behavioral solution associated with the emotion (solutions are stored in List ‘custom’, which is filled using the GetSolutions(emotion) function in HelpCustomization). The buttons are placed on topFlex, and the “text” label is updated.
      * Depending on the number of solutions, the function will display all of them or display some of them and also create “arrow” buttons. These arrow buttons use the “Handle\_Page\_Increase/Decrease” functions, but essentially, depending on the value of page, only certain solutions are shown, and the arrows adjust that page value.
    - Handle\_Solution() handles when the user pushes a solution button. It gets the selected soltuion, and displays that behavioral solution on the “text” label and an alert display box.
    - Handle\_Goto\_Emoion() handles when a user goes back to the subject options. It is very similar to HelpMe(), so see that for reference.
    - Handle\_Goto\_Solution() handles when the user goes back to the conclusion options. It is primarily used when the user adjusts the page variable. It is very similar to Handle\_Emotion(), so see that for reference. Also see Handle\_Page\_Increase/Decrease().
    - Handle\_Page\_Increase/Decrease() respectively increases or decreases the page value and calls Handle\_Goto\_Solution(). The “arrow” buttons displayed with groups of solutions call this function. Depending on page number, Handle\_Goto\_Solution() will display a group of 7 or less solutiuons along with a left or right arrow. The left arrow decreases the page number and the right increases it. The arrows don’t appear if the page number can’t further be decrease/increased. See the ‘page’ variable.
    - Sentence\_Button\_Clicked() changes the page to the MainPage page.
    - Customize\_Clicked() changes the page to the HelpCustomization page.
    - OnHelpMePage() is used in MainActivity.cs to determine if Handle\_Back() can be called.
    - Speak() implements a text-to-speech function
    - AdjustFont() adjusts the font on buttons and labels.
* Customization.xaml, Customization.xaml.cs
  + Customization.xaml is the page responsible for allowing the user to customize the specific conclusion options available to them. They can add new conclusions, change and delete existing ones.
  + XAML
    - The page layout is a content page encapsulating a StackLayout. The StackLayout has two components.
    - The first component is a button that just has a “+” sign as its text value and calls “Button\_Clicked” when it is selected
    - The second component is a ListView called “listView” and gets its items from the ObservableCollection “items” in Customization.xaml.cs. Each ViewCell (every item) in the ListView consists of a StackLayout that encapsulates a Label. These labels display the text of every item. Labels in the ListView can be held down to display the MenuItem with the text “delete” and which calls “Delete\_Clicked()” in Customization.xaml.cs.
  + C#
    - More specific documentation for Customization.xaml.cs can be found in the comments of the code. Customization.xaml.cs has a Boolean “loaded” to check if the page has been loaded. String “customFilename” holds the name of the file to be read when loading in items (this file name should be set to ././customConclusions.csv). A list of CustomConclusions called “conclusions” holds all CustomConclusions.
    - Boolean “emotionList” determines which items should be displayed in the ListView. Several functions read “emotionList” in order to determine which instructions to follow. When “emotionList” is true, the ListView will display a list of emotions to choose from. When “emotionList” is false, the ListView will display a list of conclusions associated with a specific emotion).
    - String “emotionSelected” holds the currently selected emotion.
    - Customization() initializes the page. It sets ‘emotionList = true’ and checks ‘loaded’. If the page isn’t loaded, then ‘Reload()’ is called. If it is, ‘ReloadItems()’ is called.
    - GetConclusion() is a function primarily used by other functions. It returns a list of CustomConclusion associated with the parsed strings ‘emotion’ and ‘subject’.
    - GetItem() returns a specific CustomConclusion based on a parsed string ‘conclusion’.
    - ReplaceCustomConclusion() replaces a current CustomConclusion from the list ‘conclusions’. It removes a parsed in ‘orginalConclusion’ and adds a parsed ‘newConclusion’. This function is primarily used in the CustomziationCustomize page.
    - IsLoaded() returns ‘loaded’ to check if the page is loaded.
    - Reload() reloads the CustomConclusions held in ‘conclusions’. It reads the file ‘customFilename’, creates new CustomConclusions based on the data in the file and adds them to conclusions. Then it sets ‘emotionList = true’ and calls ReloadItems() (see function below). Lastly, it sets ‘loaded = true’
    - ReloadItems() reloads the items on the ListView. It first empties the ListView of any current items, and reads ‘emotionList’. If ‘emotionList == true’, then a list of emotions are added to items (the emotions are just strings of the emotions used in the app). Else, it searches through conclusions to find CustomConclusions whose CustomConclusion.emotion == emotionSelected and adds them to ‘items’.
    - Delete\_Clicked() removes a selected item from ‘conclusions’ and calls Save().
    - Save() writes a string using all the CustomConclusions in conclusions and writes that string to the file ‘customFilename’. Then it calls ReloadItems().
    - ItemSelected() handles when a user selects an item on the ListView. If ‘emotionList==true’, then the item tapped was an emotion. In that case, the function sets ‘emotionList = false’, stores the selected emotion in emotionSelected, and calls ReloadItems(). Should ‘emotionList == false’, then the page changes to the CustomizationCustomize page, parsing in the selected item.
    - Button\_Clicked() handles when the user presses the “+” button to add a new CustomConclusion. It changes the page to the CustomizationCustomize page but parses null.
    - Handle\_Back() handles when the user presses the device’s back button. If ‘emotionList == false’, then the function sets ‘emotionList = true’ and calls ReloadItems(). Else, then the page is changed to the MainPage page.
    - OnCustomize() is used to check if Handle\_Back() can be called.
* CustomizationCustomize.xaml and CustomizationCustomize.xaml.cs
  + This page handles the adding or changing of a CustomConclusion. When nothing or null is parsed through CustomizationCustomize(), then items in the page are left blank. However, when a CustomConclusion is parsed through, the items are filled with the information stored in that CustomConclusion. Changes can be made to the items, and then saved.
  + XAML
    - The content page encapsulates a StackLayout which encapsulates 3 components.
    - The first component is a Label known as ‘preview’.
    - The second component is a StackLayout with several children. The first two are two Pickers, one with items for subjects and the other with items for emotions, and the pickers are respectively names ‘subject’ and ‘emotion’. The last two are Entry objects with the names ‘conjunction’ (to enter in a conjunction for the CustomConclusion) and ‘conclusion’ (to enter in a conclusion for the CustomConclusion).
    - The last component is a button with the text ‘Save’ and which calls “Save\_Pressed” when selected.
  + C#
    - CustomizationCustomize.xaml.cs had only one CustomConclusion private variable known as originalConclusion. This is meant to hold any CustomConclusion parsed to it.
    - CustomizationCustomize() constructs the page. It parses in a CustomConclusion and stores it in originalConclusion. It the parsed CustomConclusion isn’t null, then the ‘subject’, ‘emotion’, ‘conjecture’, and ‘conclusion’ Pickers and Entry objects in the XAML are filled in with the appropriate information that is associated with the parsed CustomConclusion.
    - Task Save() creates a new CustomConclusion and calls Customization.ReplaceCustomConclusion(originalConclusion, newConclusion) and Customization.Save()
    - Save\_Pressed() handles when the save button is pushed. It checks if the Picker and Entry objects are not null. Then, it pushes the page to the LoadingIndicator.xaml page and calls and awaits Save(). Once Save() is completed, the page is changed back to the Customization page.
* HelpCustomization.xaml and HelpCustomization.xaml.cs
  + These two pages are nearly identical to the Customzation xaml and cs pages. Refer to those pages for reference. Differences will be listed here.
  + HelpCustomization doesn’t deal with CustomConclusions and instead deals with Solutions. The list that holds all the Solutions is called “Solutions”. The customFilename should be set to “././customSolutions.csv”. Many of the variables and functions that had the word “conclusion” or were CustomConclusions have been replaced replaced by “solution” and Solutions.
* HelpCustomizationCustomize.xaml and HelpCustomizationCustomize.xaml.cs
  + These two pages are nearly identical to the CustomizationCustomize xaml and cs pages. Refer to those pages for reference. Differences will be listed here.
  + Since Solutions have no need for Subjects, the Picker object used to represent the subject have been removed from the XAML.
  + In the .cs files, other than changing some CustomConclusions to Solutions and some variables with “conclusion” to “solution”. The HelpCustomizationCustomize.xaml.cs and CustomizationCustomize.xaml.cs pages are identical.

Tutorial

One of the last additions to the version 1.0.0 of the app was the inclusion of a tutorial. There are some important changes to the code that were not listed above. They’ll be list in this section instead.

Enable/Disable and InputTransparent

* Throughout the tutorial, there is code which sets values of IsEnabled and InputTransparent to true or false. IsEnabled controls whether an element is selectable or not and InputTransparent controls whether input should pass through an element or not. These are only applicable to XAML elements.

New XAML elements in MainPage.xaml and HelpMe.xaml

* In both of these files, a ContentView named ‘tutorial’ is placed outside of the AbsoluteLayout where the other elements reside. Within ‘tutorial’ is a StackLayout known as ‘Tutorial\_Window’, and within that is a StackLayout known as ‘Tutorial\_Box’.
* When the tutorial is active, ‘tutorial’ may be a transparent gray and may be the only element enabled and without input transparency. But sometimes, it may turn completely transparent and have input transparency, so the user can interact with the app during the tutorial.
* In addition, the logo on MainPage now has a button in front of it that starts the tutorial

keepTut found in MainPage.xaml.cs

* keepTut is the primary Boolean variable used to keep track of whether the tutorial should be active or not during run time. It is set within setTutorialVisible() (explained later). When ‘true’, the tutorial should be active. When ‘false’, it shouldn’t. keepTut is what is read by many functions within and outside of MainPage to see if the tutorial should be played out.

Changes to existing functions

* There are too many changes made to existing functions to be listed, but many of them are similarly: read keepTut 🡪 if true, disable and enable some elements and do some function in Tutorial.cs (explained later). You’ll see many of these changes within MainPage.xaml.cs and HelpMe.xaml.cs, mainly within the “Handle\_[blank]” functions.

GlobalData.cs and Oigo.Droid.tutStart.txt

* In GlobalData.Init(), part of the functions tries to find tutStart.txt. If it can’t find it, it reads Oigo.Droid.tutStart.txt, which is just a text file that reads ‘true’ on it, and creates a new tutStart.txt in a different location so that it can write to it. tutStart.txt will either hold true or false, and whichever value it is determines whether the app should start with the tutorial active or not.

Tutorial.cs

* Tutorial.cs is the class responsible for initializing elements used in the tutorial.
* tutorialStart was the variable used to keep track if the tutorial is active or not, but that responsibility has shift to MainPage.self.keepTut.
* customFilename is the exact path to tutStart.txt
* tut[number]Text are strings that are used in the tutorial.
* textList is an array that holds some strings used in the tutorial
* Tutorial(int x) – is the constructor function. It takes in ‘x’ to determine if readFile() should be called. This is important because readFile() may called tutBegin(), restarting the tutorial while in the middle of it. While going to the HelpMe page, a Tutorial object is made, passing 1 to the constructor to avoid readFile() while in a tutorial.
* readFile() – reads customFilename to determine if the app should begin the tutorial or not
* writeFile() – writes either true or false onto customFilename
* createButton() – depreciated function that made button elements
* tutBegin() – begins the tutorial. Clears elements in the Tutorial\_Box on MainPage and adds new ones. These new button elements let the user start, stop, or permanently stop the tutorials.
* tutSetPageOne() – clears the Tutorial\_Box and places new elements in Tutorial\_Box. Essentially sets up one particular page of the tutorial within MainPage
* tutSetPageTwo() – creates a label and StackLayout (that acts as a border) with text that continues the tutorial. It calls MainPage.self.addtoTutorial() to remove the Tutorial\_Box and replace it with the created elements. See MainPage.xaml.cs below for more on addtoTutorial().
* tutSetPageThree() and tutSetPageFour() – the page three function imitates the page one function and the page four function imitates the page two function, but instead these functions adjust the Help Me page, not the Main Page.
* tutSetPageFive() – similar to tutSetPageOne(), but is recursive and goes through textList to create labels on the Tutorial\_Box. It goes through all the dialogue in textList before ending the tutorial
* endTutorial() – enables all non-tutorial elements on the MainPage and HelpMe page and disables the tutorial elements, as well as sets keepTut to false.
* noTutorialPrompt() – same as endTutorial(), but writes ‘false’ onto tutStart.txt so that when the app opens again, the tutorial doesn’t start.
* resetTut() – writes ‘true’ to tutStart.txt so that the app begins the tutorial when opened. Used on the button over the logo on the MainPage.

New MainPage.xaml.cs and HelpMe.xaml.cs variables and functions

While existing functions have changed too, these are entirely new variables and functions.

* Tut[number]Text – strings used in the tutorial
* addWindowElements() and removeTutorialChildren() – adds and removes children to Tutorial\_Box.
* setTutorialVisible(bool b) – sets whether the tutorial is active or not. If ‘b == true’, then keepTut is true, and the tutorial elements are visible and are not input transparent. The elements also have a gray color to them and Tutorial\_Box is appended to Tutorial\_Window (Tutorial\_Box may be removed during parts of the tutorial. This function resets that). If ‘b == false’, then keepTut is false, the tutorial elements are invisible and can’t be interacted with, and the tutorial is effectively deactivated.
* addToTutorial() – disables certain non-tutorial elements (keeps topFlex enabled as the user will interact with it), makes the tutorial elements input transparent, removes Tutorial\_Box from Tutorial\_Window and adds in a new label. It also resizes and repositions the tutorial elements. This effectively places labels on the screen with instructions for the user to follow (the instructions are specified by the string that is passed to it).
* Disabel() – disables certain elements
* DisEnable() – disables and enables certain elements. May be depreciated
* Enabel() – enables all non-tutorial elements
* Getters and setters for keepTut
* Handle\_Tutorial – the function called when the logo on the MainPage is tapped. It calls resetTut() and resets the tutorial. If users permanently disabled tutorials, they can re-enabled it with this.

MainActvity.cs changes

The override function that handles device back button pushes has been updates. It reads keepTut and, if the tutorial is active, the user can’t press the back button on their device.