Box-Picking Handoff Information

https://github.com/jmerolla/470-Box-Picking

Last Modified: 12/17/2020

2020 Team Members:

Graham Hallman-Taylor
Jessica Merolla
Spencer Dusi

NOTE: The master branch of the 470-Box-Picking repository holds the most up to date code, *not* in-progress.

Project Overview

It should first be noted that MobileScanApp is a Universal Windows Platform application. Any searches for how to accomplish certain tasks within the code should include UWP somewhere in the query as Windows Forms results show up far more often when the platform is not specified. Another very important piece of information about this project is that we are operating under the assumption that a system will be implemented within Avery Dennison that adds physical barcodes to items unique to each product. As it is now, the barcodes found on individual products have no correlation to the item they are printed on and instead signify some relation to the order they are within. The application could most likely be modified to read these barcodes as input instead, but it would not solve the problem at hand of over/underpacking. However, if the choice is still made to change the packing process from physical to digital, this implementation is most certainly possible.

Simplicity is a must when it comes to designing the aspects that will determine functionality and useability for this application; the process this program is attempting to replace is already quite efficient and is only up for replacement due to the margin of human error regarding the items picked and the amounts of said items. It is inevitable that some efficiency will be sacrificed in the transition from the current method to the utilization of this application.

The first major process that the application undergoes upon use is reading in the file that contains the order to be packed. The data contained within the order list must be parsed to allow for the interactive list to be generated. While it may seem that a .csv or comma separated value file would be the most logical format to read in the orders, there is

an error within the program that generates them causing them to data to be mangled and split in several places. We have no way of accessing the aforementioned program and we are not allowed access to it in any way, so I am assuming you will not either. Because of this, the program will only accept .txt files. While the program will not crash if it reads in an incorrectly formatted file, it will not function properly. Only .txt files generated by the program Avery Dennison uses should be fed into the application as the regular expressions we use are designed specifically for them and their quirks. To parse the order data, the application searches for a keyword that flags it to ignore all previous characters. The same process is done to know when to stop parsing data and ignore the remainder of the characters. Within the characters deemed important, a delimiter is used to separate the different columns of data from each other. All of the details regarding these variables can be found within the TextFileHandler class.

The name of the selected file is displayed after a correctly formatted file is selected and the user is then prompted to confirm their order selection with a button press. A list optimized for touch screen devices is then displayed with the following information deemed important in the TextFileHandler. The user selects an item by tapping on any part of the row an item is contained within. A page then asks the user to confirm the item they selected and offers them a back button if an item was mistakenly selected previously. A new scan page is generated on confirmation of the selected item and the full screen turns into the scanning interface. It has been suggested that it would be more practical to have a constant scan page running alongside the order list that would process any of the valid items. However, the configuration of the ScanPage would need major modifications for this to work. As it is now, a ScanPage is only generated once an item is selected from the list and

confirmed by the user. The data displayed on this new scanpage is dependent on the selected item and does not pull directly from the whole order list. To reiterate, this implementation would be possible, but ScanPage would have to be heavily edited for this to work. As for the current ScanPage, once a barcode is recognized, it is displayed to the user through an alert that determines and informs the user whether or not the barcode matches that of the product they selected. The user is then prompted to enter in a number symbolizing how many of the scanned product they intend to pack at this immediate moment. This scanning of a given number of some product can be conducted any number of times until the total for the order is met. Once the total is met, the user is locked out of selecting the completed item from the list as if the entered amounts were correct, this would certainly result in an overpacking of the item in question. Once every correct amount of every product has been packed, the order is completed and is logged to a file within a local folder on the host device generated on the first run of the application.

It is sensible to come to the conclusion after reading all of this information on the application's functionality that the problem of over/underpacking is not entirely safeguarded against using this approach. We are aware of this and between the three of us, this is the best we could come up with that does not sacrifice too much efficiency from the traditional method while still providing a considerable amount of security regarding the amount of items packed and correctness of said packed items. Some very minor changes can be made to the application to cut out confirmation screens and other things that slow down the packing process in exchange for security if it is so desired. The theoretical perfect balance of efficiency and accuracy will only be found once the application is put through

more thorough testing which is out of the scope of us to conduct within the remainder of this semester.

I feel that it is also significant to note that Avery Dennison has informed us that typically, only one employee will be packing orders at any given time. Therefore, no issues should arise from the fact that all order files and logs are saved locally on only one tablet computer as anyone packing orders will be using the same tablet each time.

MainPage

MainPage acts as the startup page of the app. The ultimate goal of this page is to select and parse an order sheet into a list of OrderItems, and then pass them into the OrderListView page.
The code in this method is the result of numerous tweakings and rewrites, and it might benefit from a complete overhaul.

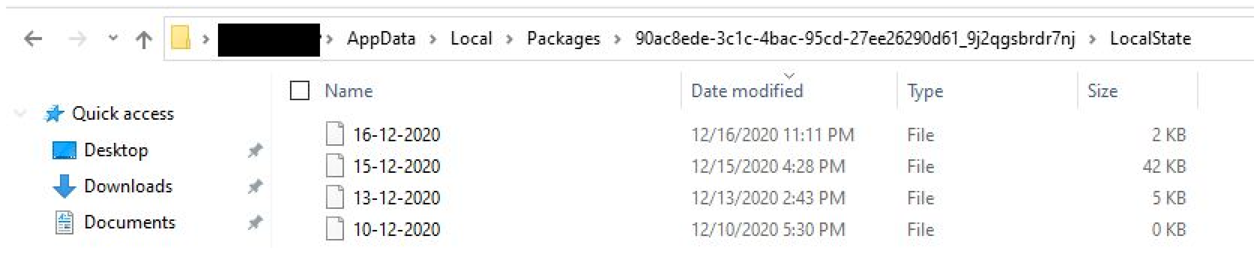
There are two Plugins to note:

- `PCLStorage`
    - **Used to get the path to the LocalStorage folder.**
    - **NOTE: This plugin is a bit of a headache, and only seemed to work selectively. If you can find a better way to save logs to the device, it is strongly recommended that you move away from this plugin entirely.**
- `Plugin.FilePicker`
    - **Allows cross-platform access to the file explorer. The focus of the application is no longer on cross-platform compatibility, but this plugin still works for Windows.**

The createLogFile() method looks for a file in LocalStorage with the current date, and creates one if it does not yet exist.

The log files are created in the AppData/local/Packages folder under a random string of letters and numbers, as seen in Figure 1. Navigate to this folder, and click on LocalState. Your logs for each date should appear here.

NOTE: The AppData folder is hidden from users by default, so you must open it from the search bar on Windows and type %AppData%. This might take you to a different folder within AppData, but you can just move back into the folder you do want.



[redacted] > AppData > Local > Packages > 90ac8ede-3c1c-4bac-95cd-27ee26290d61_9j2qgsbrdr7nj > LocalState

Name	Date modified	Type	Size
16-12-2020	12/16/2020 11:11 PM	File	2 KB
15-12-2020	12/15/2020 4:28 PM	File	42 KB
13-12-2020	12/13/2020 2:43 PM	File	5 KB
10-12-2020	12/10/2020 5:30 PM	File	0 KB

Figure 1

Navigating to the Log Files

The PickFileButton_Clicked(object sender, EventArgs e) method allows the user to select a file, and will loop until a valid text file has been chosen. It will then pass the file into the ReadInTXT method.

The ReadInTXT(FileData filedata) method reads in all the text from the selected file, then stores the header of the text file before removing the header and footer of the file (these portions do not have anything to do with the actual order item information). The remaining text is split by whitespace into a list, and passed into parseOrderItemsFromList where it will create a list of OrderItems.

NOTE: ReadInTXT returns a string of the order text without the header and footer, as it was once displayed on MainPage to check for accuracy. This returned value is still used for debugging, but could be removed if desired.

The ConfirmOrderButton_Clicked(object sender, EventArgs e) method locks in the order file to be picked. The order's header is added to the log file with its format preserved, and the application navigates to the OrderListView page, passing in the list of OrderItems.

OrderItem.cs

OrderItem.cs holds the various fields that each item on the order sheet will contain. A few things to note:

- `Boolean IsPacked`
  - **Note: Not on the OrderSheet, should be set to false when creating a new item.**
  - **Used to track when an item has been fully scanned.**
- `String Name`
  - **The name of this specific item.**
- `String LocationQOH`
  - **Holds the multiple locations and QOH (Quantity on Hand) in CTs for each location.**
- `String BarcodeID`
  - **Holds the barcode of the item.**
  - **NOTE: There is an OrderNumber, although the company does not currently have barcodes specific to the item as a whole. Rather, each individual iteration of an item has a unique barcode. The company has agreed to implement barcodes for each item as a whole, but it is unclear at this time if this change has occurred.**
- `decimal PalletQty`
  - **The expected amount of palettes of that item.**
  - **NOTE: Unused in the current codebase.**
- `decimal CartonQty`
  - **The expected amount of cartons of that item.**
  - **NOTE: Unused in the current codebase.**
- `decimal QtyOrdered`
  - **The quantity of the time ordered by the customer.**
- `decimal QtyOpen`
  - **The expected quantity of the item available in the warehouse.**
  - **NOTE: Unused in the current codebase**
- `String UM`
  - **Value unknown at this time, as it was consistently an empty field on all of our sample order sheets.**
- `String ExtPrice`
  - **Price of the item**
  - **Note: Unused in the current codebase**

- `String DueDate`
  - **The date the item is due.**
  - **Unused in the current codebase**

As you can see, our team only focused on a few fields of the OrderItem. If you find any of these fields confusing, or want more examples, the best thing you can do is ask your contact to clarify.

Furthermore, note that multiple fields are decimals. Originally, we treated these numbers as integers, which is reflected in some workarounds in certain parts of the codebase. We changed these fields to decimals, as the official order sheets do all have decimal values for each of these fields.

TextFileHandler.cs

TextFileHandler.cs is responsible for parsing the actual order sheets. Originally this method worked by parsing csv files, but it was later decided that passing the order in as a text file was the only way our team could consistently parse the data in the correct format.

Though there are several methods in this class, the most important one to note is parseOrderItemsFromList(List<string> ItemsList). Ideally, the user would pass in a list of strings parsed from the original order sheet. This method takes this list, and looks for the different fields for each OrderItem, populating a new list of OrderItems.

This method relies heavily on the use of regex to match the different OrderItem fields:

- `Regex ln = new Regex(@"[0-9]{1,2}", RegexOptions.Compiled | RegexOptions.Singleline);`
    - **"ln" matches numbers from 0-99. This assumes that orders will not have more than 99 items to pick at a time.**
- `Regex ItemNumber = new Regex(@"[0-9]{10,}", RegexOptions.Compiled | RegexOptions.Singleline);`
    - **"ItemNumber" is treated as the barcode of the item.**
    - **It matches a number of at least 10 digits. This assumes a barcode will not be shorter than a length of 10.**
- `Regex ItemName = new Regex(@"([a-zA-Z]{2,})|([0-9]{4})", RegexOptions.Compiled | RegexOptions.Singleline);`
    - **"ItemName" refers to the descriptive name of the item. Our team did not have an official understanding of what ItemNames could look like, so we had to allow for multiple words and numbers. This means "Tape 0030" and "The Tape" are equally valid.**
    - **This method assumes each word will either be a word of at least two letters, or a number with exactly four digits.**
- `Regex Location = new Regex(@"[a-zA-Z]{1}[0-9]+", RegexOptions.Compiled | RegexOptions.Singleline);`
    - **"Location" refers to the physical place where that particular item is stored. An example location would be "x02".**
    - **We did not have an exact explanation of how many locations there are, so "Location" matches a single letter with one or more numbers after it.**
    - **NOTE: There can be multiple locations for a single item.**

The actual method loops through the list, first looking for a match for "ln". When there is a match, a counter, j, loops through and finds the multiline fields for each item.

Because the order sheets have a consistent format, we know that 10 strings over from ln will be the first location value. An inner loop will occur to find each location, not stopping until it hits the ItemName field. The order of these fields is:

(QOH in CTs, Location, Item Name) where QOH in CTs and Location can repeat multiple times.

Each time a location is found, the locationAndQOH String will add the location and its QOH in CTs to itself. Note that the line break (\r\n) at the end of the location is removed, and then added back in after the QOH in CTs value.

Once all locations are found, j is then used to grab all matches of ItemName and store them in the NameOfItem String.

From here, the method attempts to try and add a new OrderItem to the list OrderItems. Because the format of the order sheet is preserved, once ln, the full location with QOH in CTs information, and the full name of the item have been found, the spots of all other item fields can all be predicted. It might be worth making these values constants in the future.

NOTE: Any number fields that could potentially be used in mathematical operations are parsed as decimals, although this was a later change from being parsed as integers.

ScanPage.cs

ScanPage.cs is responsible for capturing the barcode on the box of an item at Avery Dennison. The scanner is to scan however many times the quantity specified. We made this process easier for the workers by implementing a button to enter a custom amount up to one less than the quantity remaining to scan.

Though this class is large in size, there are only two methods created to check our scans. These methods are barCodeMatcher() and RemainingScans()

barCodeMatcher()

- `This method returns a boolean; true if the barcode that was just scanned matches the barcode from the order sheet.`
- `If this method returns false nothing will be added to qtyScanned.`

RemainingScans()

- `This method uses the QtyOrdered to determine how many scans remain.`
- `decimal qtyOrdered = scannableItem.QtyOrdered; is how we see how many of this item we need to scan`
- `We add to qtyScanned to subtract from QtyOrdered. If the user selects to enter a custom amount we add the amount entered to qtyScanned.`

Once remaining scans become 0, our application will redirect you back to the OrderListView page. This is done by popping the page we are on which is the ScanPage, then by popping the confirm item page immediately after. To show we have completed our scans for a certain item we show a check mark next to the name of the item in the list. To do this, in the ScanPage we must use this command.

```
OIList[OIList.IndexOf(scannableItem)].IsPacked = true;
```

If all scannableItems IsPacked is set equal to true, then once the last item is scanned, you will be redirected all the way to the MainPage to pick a new file with another order.