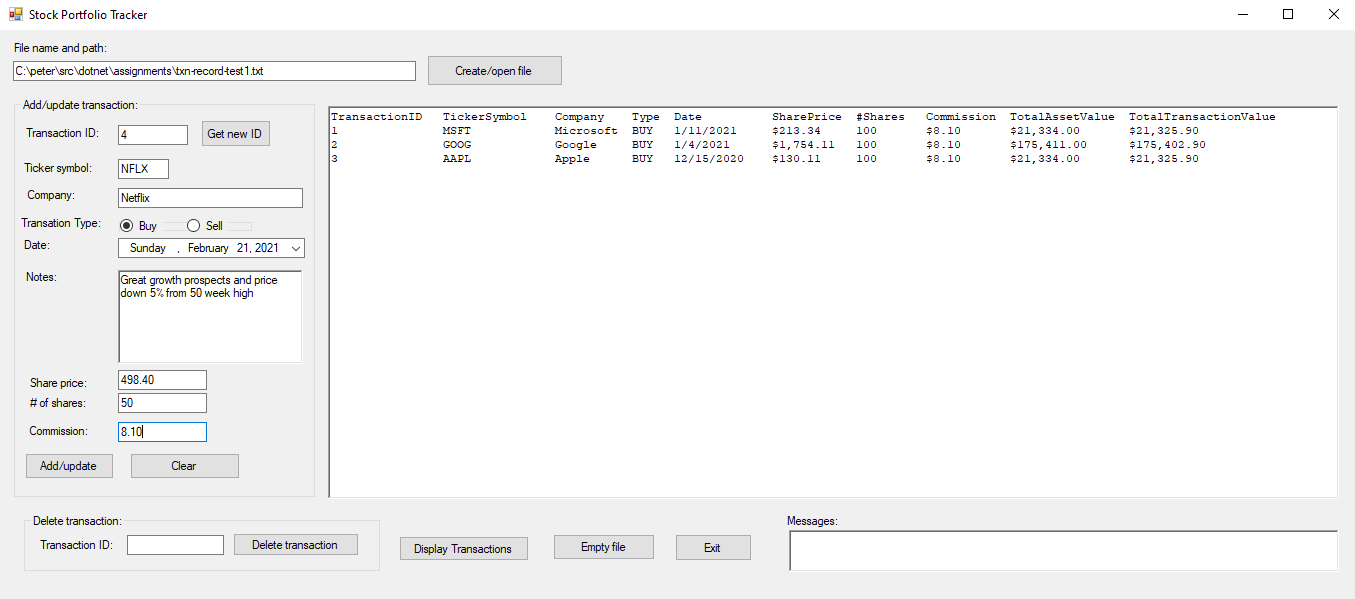
This is an exercise in file I/O management through an encapsulated class, not an exercise in List or Array management. If you declare a collection such as an array or List outside of a method or as a property, I will delete the object before I start marking. You have to think of a multi-user environment: if you retain your data in memory and just dump it to file periodically, then you will obliterate data added or updated by others … and they will do the same to you.

## Assignment 3 – File I/O

An investor wants an app to keep track of the stock transactions that he or she makes in their personal investment portfolio. The transactions (e.g. the investor bought 100 shares of MSFT at $216.34 per share) will be stored in a text file where each row represents a single transaction. The fields within each row will be separated by a delimiter of your choice.

1. The starting point for the app is being able to open an existing transaction record file, or create the file if it doesn’t exist. If one didn’t exist, display a message to the screen notifying that a new file was created. If the file does exist, the program will display it's records on the rich text box when it is opened. If you are interested in exploring the “*Data Grid View*” control you could all use it to display all the transactions.
2. The investor can add new, update existing, and delete transactions and all these changes need to be reflected immediately in the text file. More specifically:
   1. The list of existing transactions will be displayed in a rich text box.
      1. Be sure to include a header row and pad each column heading and field into a fixed length.
      2. Additionally, notice that there is *not* a 1 to 1 mapping between every field that will be stored in the file and the fields to be shown on the rich text box. For example, the notes the investor makes when recording the transaction are not shown in the rich text box but need to be stored in the file. The columns that your rich text box should display are specified in the image below.
   2. Format the Date as shown on the image below (e.g. 2021/02/21).
   3. Display the Share Price, Commission Fee, Total Asset Value, and Total Transaction Value as currency.
   4. When data is being inserted, updated, or deleted, a *Filename* needs to have already been specified.
3. Catch all file I/O exceptions (open/read/write) and display them as error messages on the form.
4. All new/update transaction input fields are required. If the transaction does not exist in the file, insert the record. Otherwise, update the record. Any errors during the insert or update, are displayed all at once, one line per error, and give the focus to the first field in error. Also, the following constraints must be met:
   1. The transaction ID must be a unique integer, and greater than zero.
   2. The transaction date cannot be in the future.
   3. The transaction notes may contain commas and can be multi-line.
   4. The share price and commission need to be greater than zero, and # shares an integer greater than zero.
   5. The total asset & transaction values will not be entered by the user. Instead, the program will calculate them.
5. In order to delete a transaction, a valid transaction ID needs to be entered.
6. The “Display Transactions” button should display in the rich text box the transactions found in the current record file whenever the user suspects the data currently displayed in the rich text box is stale in comparison to what’s stored in the file.
7. The “Exit” button exits the program and the “Empty record file” button removes all transaction entries from the record file.



## Hand In

1. Bundle and upload your solution folder to the assignment drop-box on eConestoga as a ***single*** zip file.