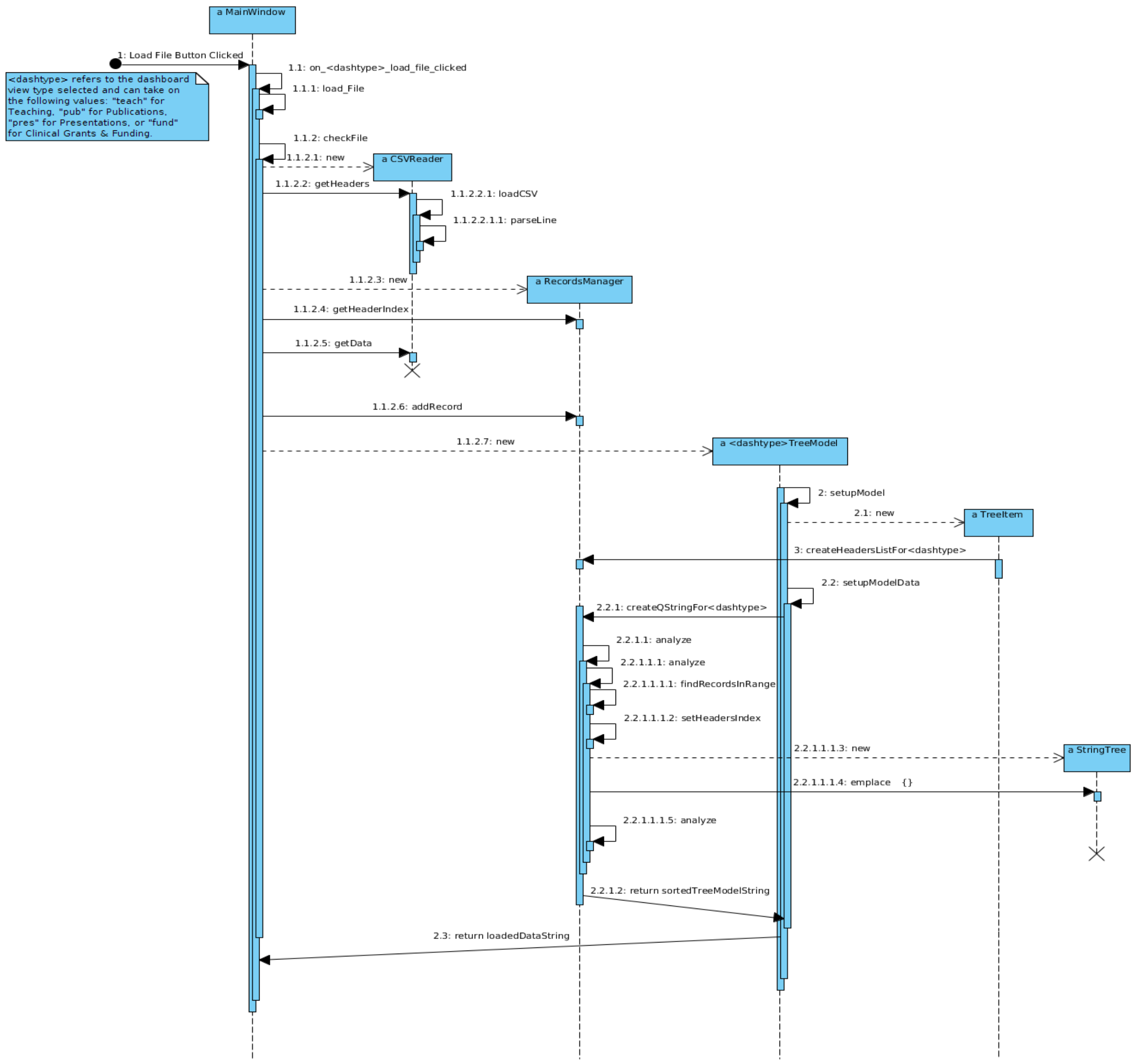
CS3307A – Acuity STAR Project – Team Peach Stage 2 – Sequence Diagram – Load Data From File


a MainWindow
1: Load File Button Clicked
<dashtype> refers to the dashboard view type selected and can take on the following values: "teach" for Teaching, "pub" for Publications, "pres" for Presentations, or "fund" for Clinical Grants & Funding.
1.1: on_<dashtype>_load_file_clicked
1.1.1: load_File
1.1.2: checkFile
1.1.2.1: new
a CSVReader
1.1.2.2: getHeaders
1.1.2.2.1: loadCSV
1.1.2.2.1.1: parseLine
1.1.2.3: new
a RecordsManager
1.1.2.4: getHeaderIndex
1.1.2.5: getData
1.1.2.6: addRecord
1.1.2.7: new
a <dashtype>TreeModel
2: setupModel
2.1: new
a TreeItem
3: createHeadersListFor<dashtype>
2.2: setupModelData
2.2.1: createQStringFor<dashtype>
2.2.1.1: analyze
2.2.1.1.1: analyze
2.2.1.1.1.1: findRecordsInRange
2.2.1.1.1.2: setHeadersIndex
2.2.1.1.1.3: new
a StringTree
2.2.1.1.1.4: emplace {}
2.2.1.1.1.5: analyze
2.2.1.2: return sortedTreeModelString
2.3: return loadedDataString

Above is the sequence diagram for the scenario "Load Data from File" in the use case diagram for the Acuity STAR project, including the main scenario and extensions. Although the notation of the sequence diagram itself is quite self-explanatory, we present below the parts corresponding to each step in the scenario.

1. User clicks Load button

This step is represented by the found message, an arrow with a dotted end in the top left of the diagram entitled "Load Data Button Click". It activates the MainWindow self-call "on_<dashtype>_load_file_clicked", catalyzing the entire sequence.

2-3. System displays file structure – user selects file

These two steps are contained in a single activation. MainWindow self-calls the “loadFile” method, which opens a dialog box for the user to select the desired CSV file. It then returns the file path and, if successful, self-calls the “checkFile” to make sure the CSV file type is compatible with the dashboard view type. It is natural to combine these steps in such a way because the user will typically click the “Load Data” button and select an appropriate CSV file. If this is not the case (i.e. the user cannot find the desired file or accidentally closes the dialog box) then it does not make sense to try and load data.

4. System verifies file is of proper type

This step is accounted for in the MainWindow “checkFile” method self-call, which first checks that the filepath is valid and the filetype matches the appropriate dashboard view. If either of these conditions fail, MainWindow displays the following message: “Not a valid <dashtype> file”. If however the conditions are met, MainWindow proceeds with loading the data. This approach was chosen in order to allow the program to terminate gracefully and give the user the option to select a new (proper) CSV file. This design decision stemmed from the conclusion that a simple file path error should not crash the entire application.

5. System loads data from file

This last step is the most resource-consuming of all and is thus described by the bulk of the sequence diagram. The first activation creates a new CSVReader object which parses the file, while the next call returns the headers. Then MainWindow creates a new RecordsManager object and gets the sorted header indices, which in general are different for each of the four file types. Next the data is retrieved from CSVReader and stored into RecordsManager. However the data is not sorted; to do this MainWindow creates a new <dashtype>TreeModel, which when activated self-calls its “setupModel” and creates a new TreeItem (that is, the root node) and its appropriate header list. The data is now ready to be sorted and so “setupModel” calls the RecordsManager “analyze” function to accomplish this task. The “analyze” function has many self-calls and is internally overridden; it also calls “findRecordsInRange” to filter by date and creates an instance of the StringTree helper class. It builds the data string to return as well as any necessary accumulators for the dashboard view. The result is a TreeModel filled with the sorted data, ready to be used for visualization. The classes are designed to maximize cohesion and minimize coupling to support the general goals of readability and maintainability. CSVReader and StringTree's destructor methods are called as they are no longer of use.