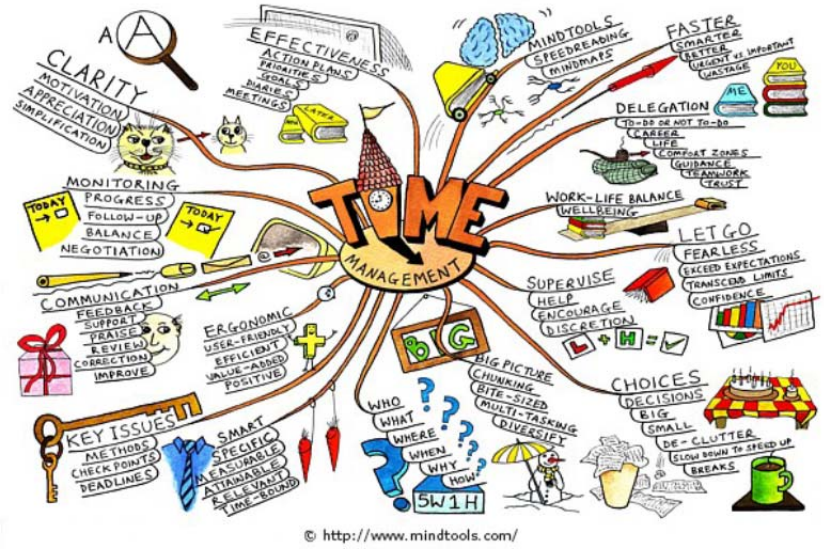
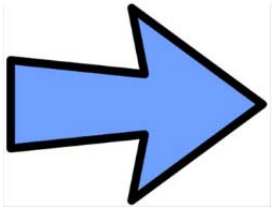
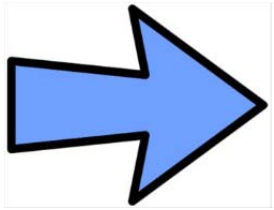
Developer Guide







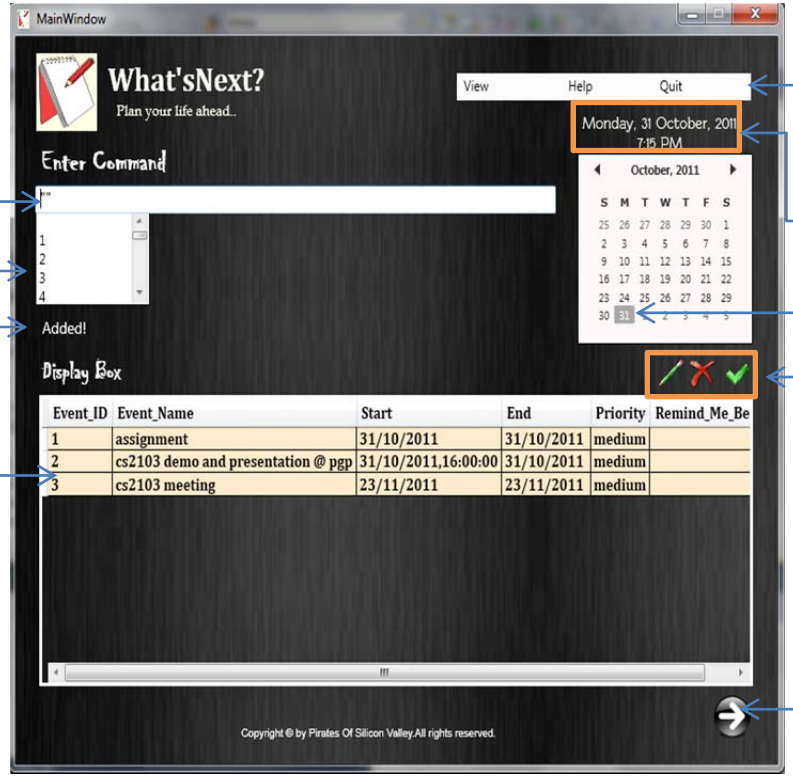
Contents

1. Introduction

What's Next? is an exceptional To-Do Manager that helps users manage their tasks efficiently with the help of its unique dual features : Simple-To-Use GUI and Intuitive Command Line Interface. While this application has been targeted primarily at power users to facilitate convenient keyboard usage, it also has an interactive User Interface that executes operations on mouse click. The following Developer's Guide explains in detail the software architecture, core features and functionalities, their implementation and the purpose for the different APIs used. This document also demonstrates the product's worth to power users with the help of use cases. Furthermore, instructions have been provided for extensive automated testing of the application. The future scope for improvements has also been planned out based on the analysis of existing functionality. The developer is advised to go through the User Manual to familiarize himself with the syntax of the commands.

2. Software Development

2.1. What's Next?-An Overview



Command Box: Commands are entered here and all event details should be within the double quotes that automatically appear.

IntelliSense : Shows a list of previously entered event details and syntax keywords to choose from

Status Message: Displays the outcome of the last performed operation .

Display View: Displays all the pending events according to impending deadlines. The events are dynamically updated every time an operation is performed.

Menu options for viewing history, performance, user manual and to close the application.

System Date and Time

Click on a date to see the tasks to be done for that day in Calendar view in the Display Box.

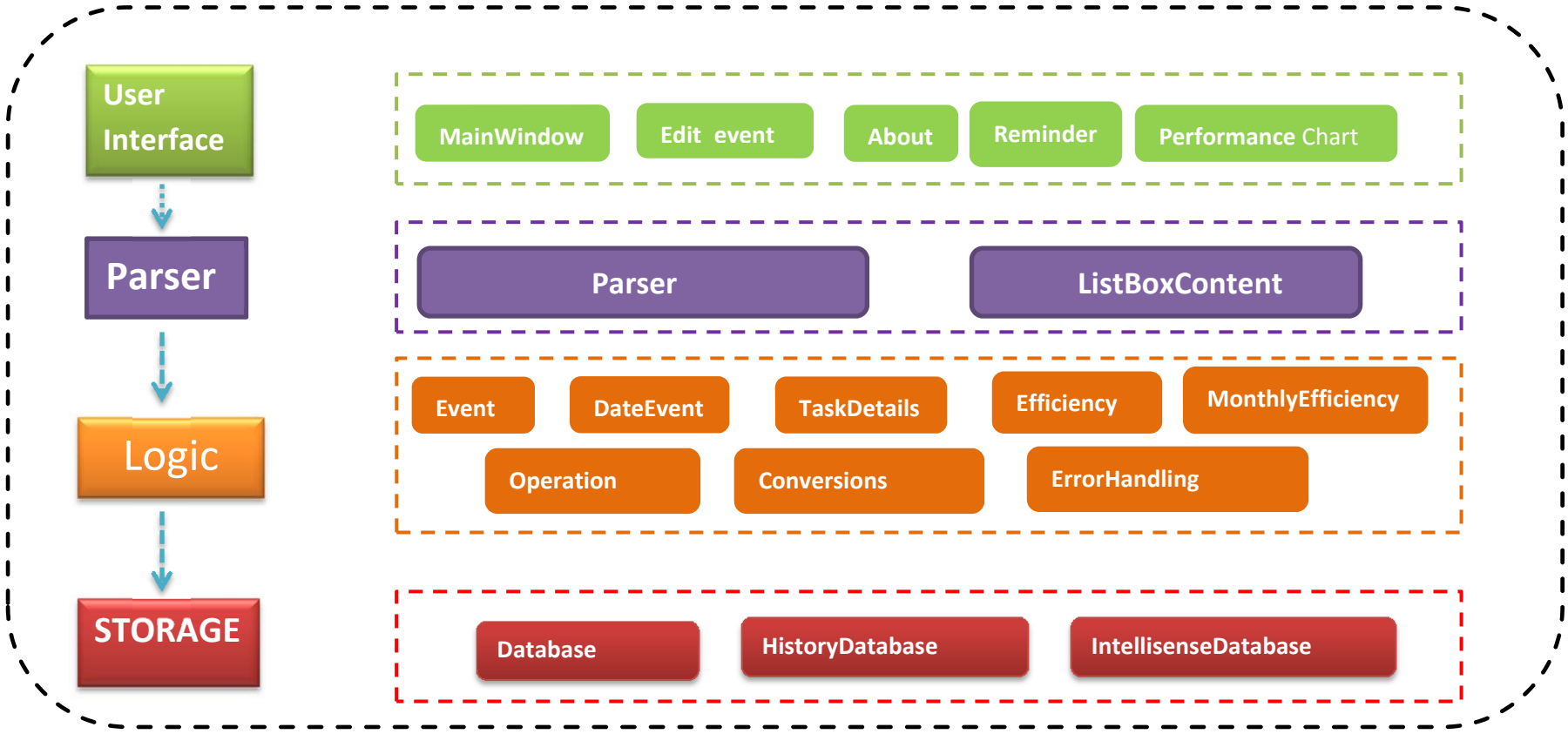
Instead of typing the command in the Command Box, select the task from the table andclick on the edit/delete/done buttons.

Flip View: Switch from Calendar view to Display view.

2.2. Software Architecture

An N-tier architecture has been adopted for this software as illustrated in Figure 1. It shows the one-way interactions between the four tiers: User Interface, Parser, Logic and Storage.

Figure 1

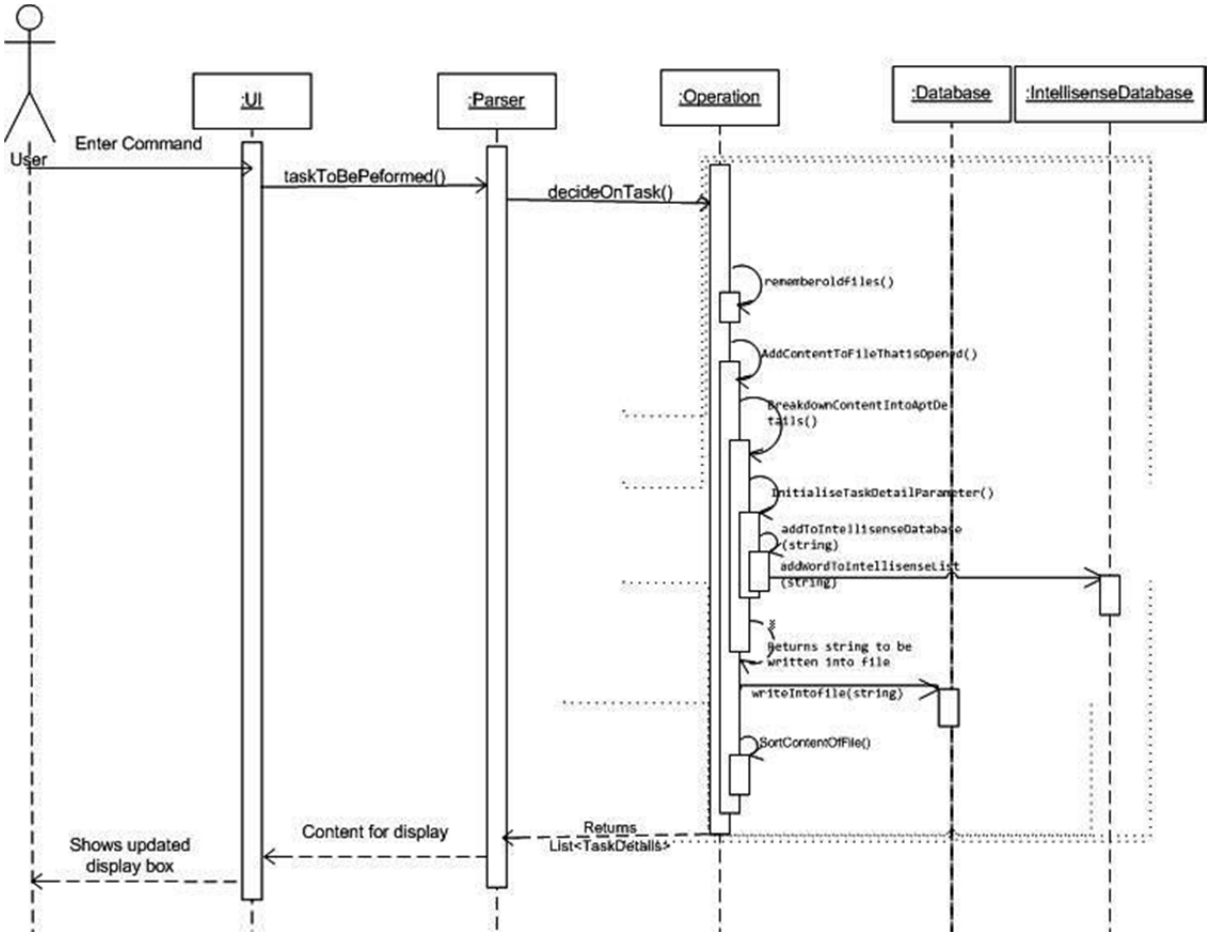


The User Interface interacts with the user and takes the command which specifies the operation to be performed. The UI then passes the command to the Parser unit to separate the keywords from the task details. This parsed information is then passed on to the Logic Tier where the operation to be carried out is deciphered from the command keyword. The Logic interacts with the Storage unit to retrieve existing tasks and manipulate the information accordingly. The tasks are written back into the files in Storage. The result of the operation is passed back to the UI through the Parser unit. The UI finally displays the updated information and a status message that informs the user of the outcome of the operation.

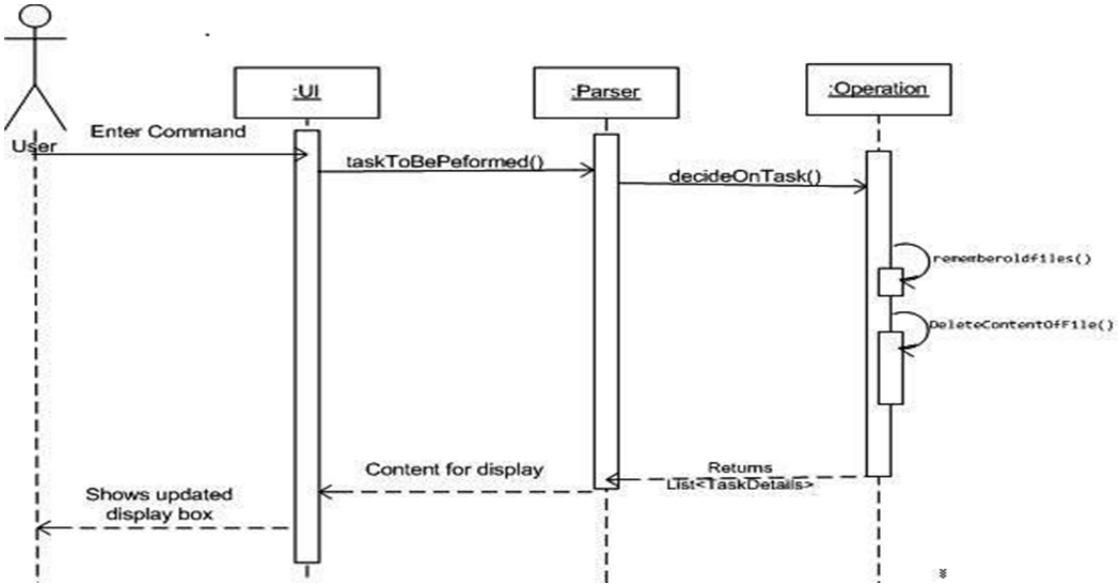
2.3. Implementation Details

2.3.1 Functionalities

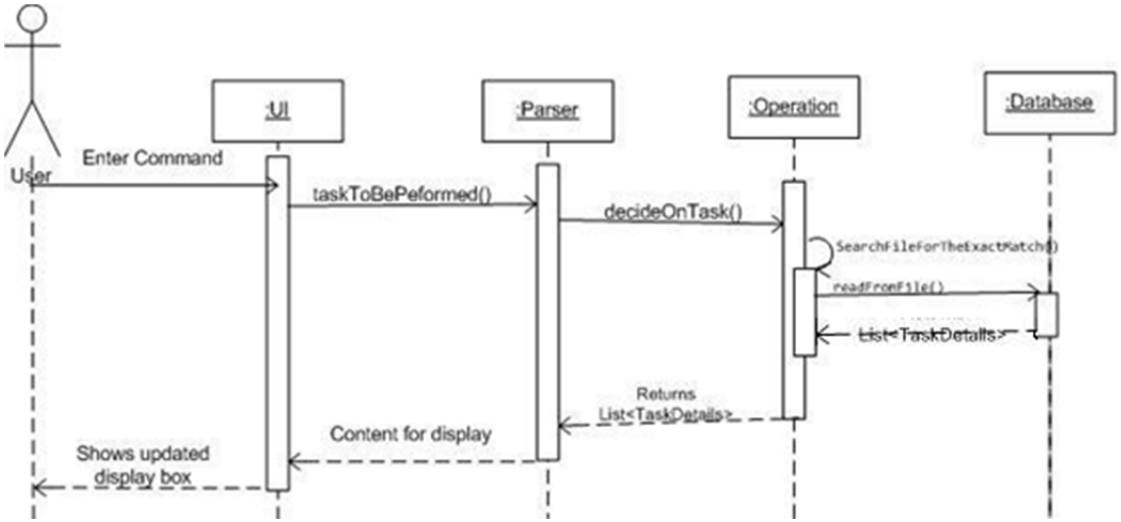
1. Add



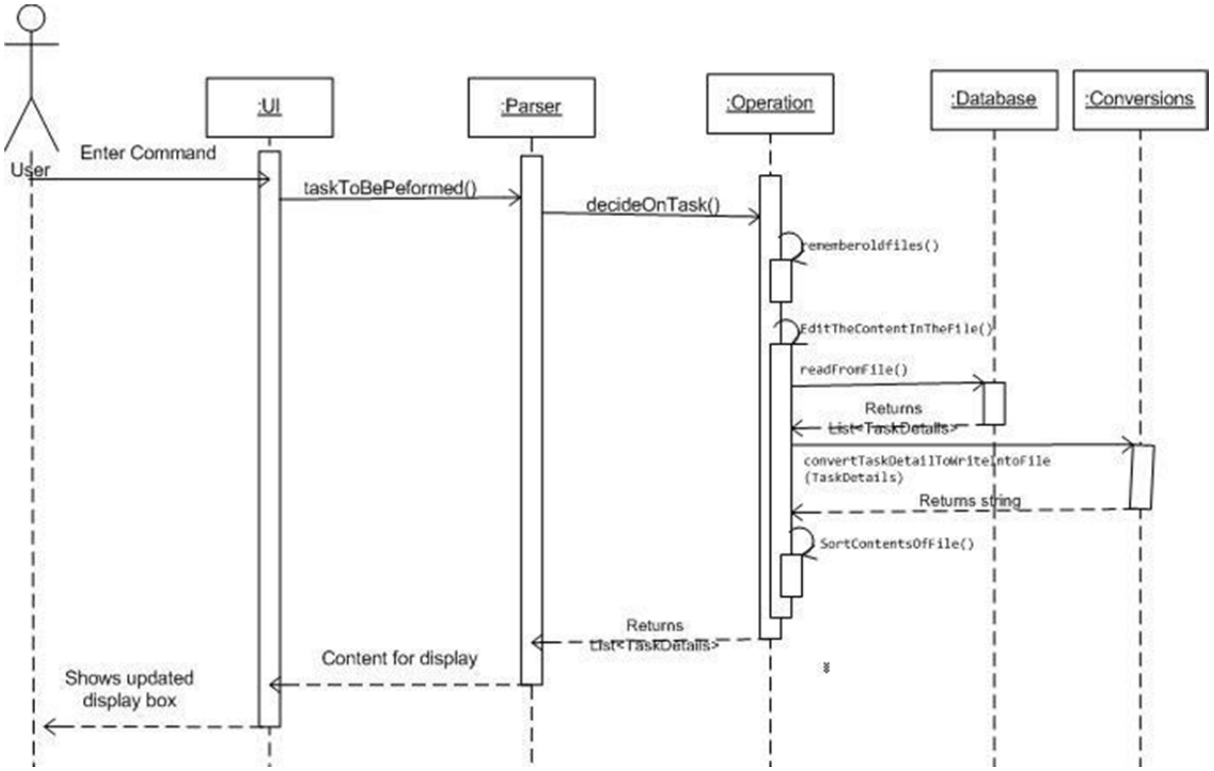
2. Delete


:UI
:Parser
:Operation
User
Enter Command
taskToBePeformed()
decideOnTask()
renemberoldfiles()
DeleteContentOfFile()
Content for display
Returns
List<TaskDetails>
Shows updated
display box

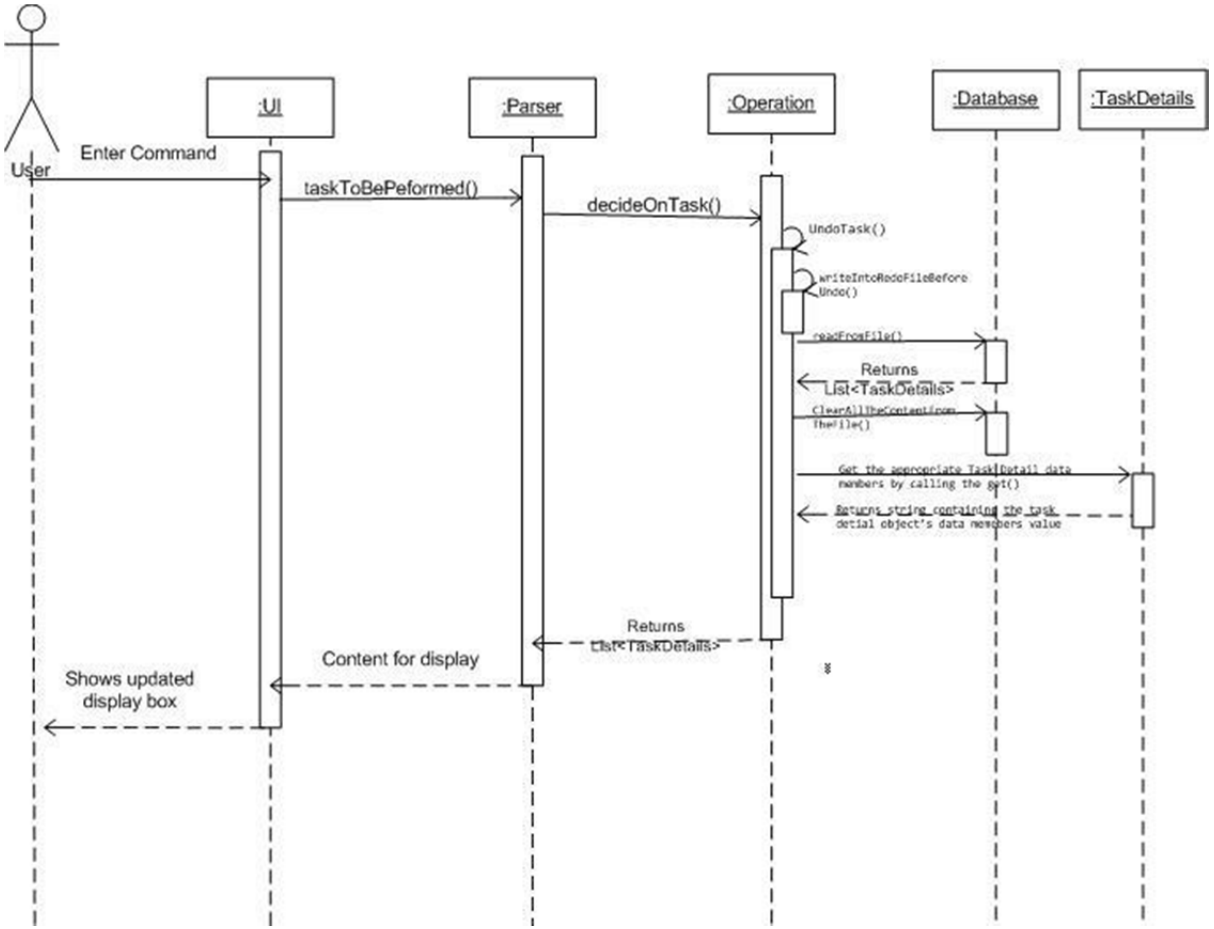
3. Search


User
:UI
:Parser
:Operation
:Database
Enter Command
taskToBePeformed()
decideOnTask()
SearchFileForTheExactMatch()
readFromFile()
List<TaskDetails>
Returns
List<TaskDetails>
Content for display
Shows updated
display box

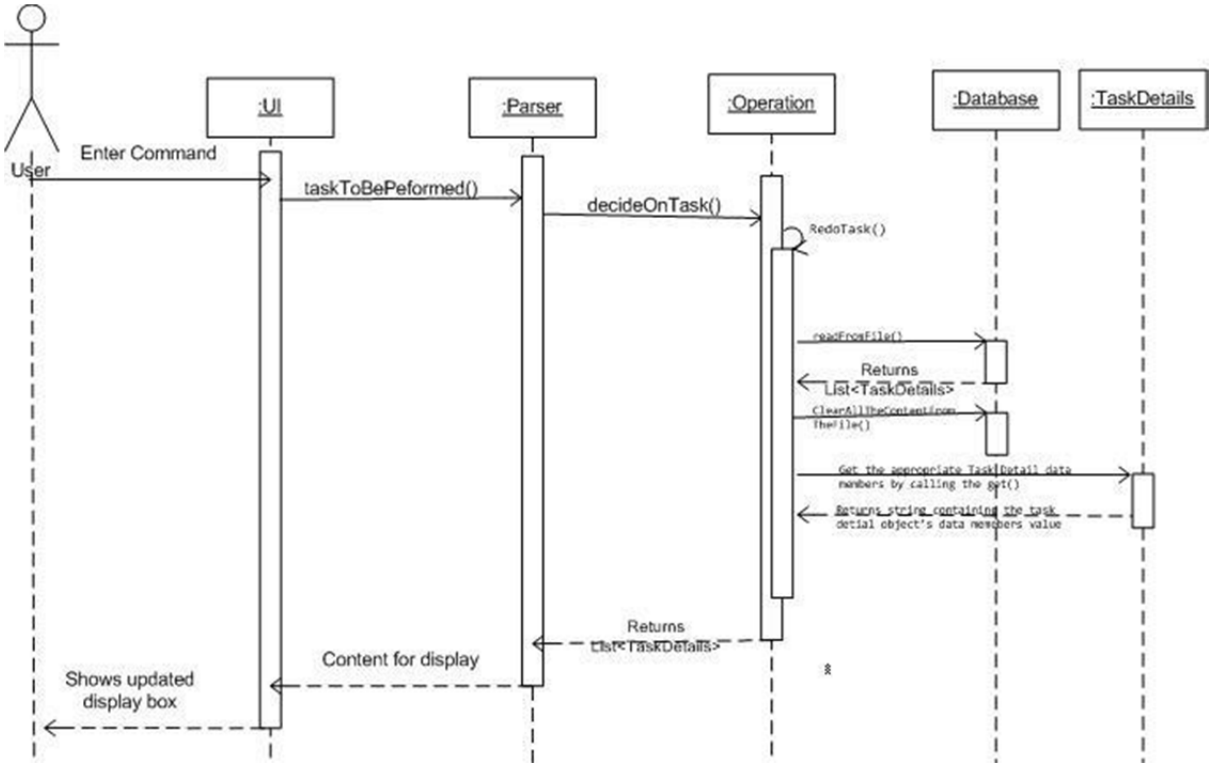
4. Edit


:UI
:Parser
:Operation
:Database
:Conversions
User
Enter Command
taskToBePeformed()
decideOnTask()
rememberoldfiles()
EditTheContentInTheFile()
readfromFile()
Returns
List<TaskDetails>
convertTaskDetailToWriteIntoFile
(TaskDetails)
Returns string
SortContentsOfFile()
Returns
List<TaskDetails>
Content for display
Shows updated
display box

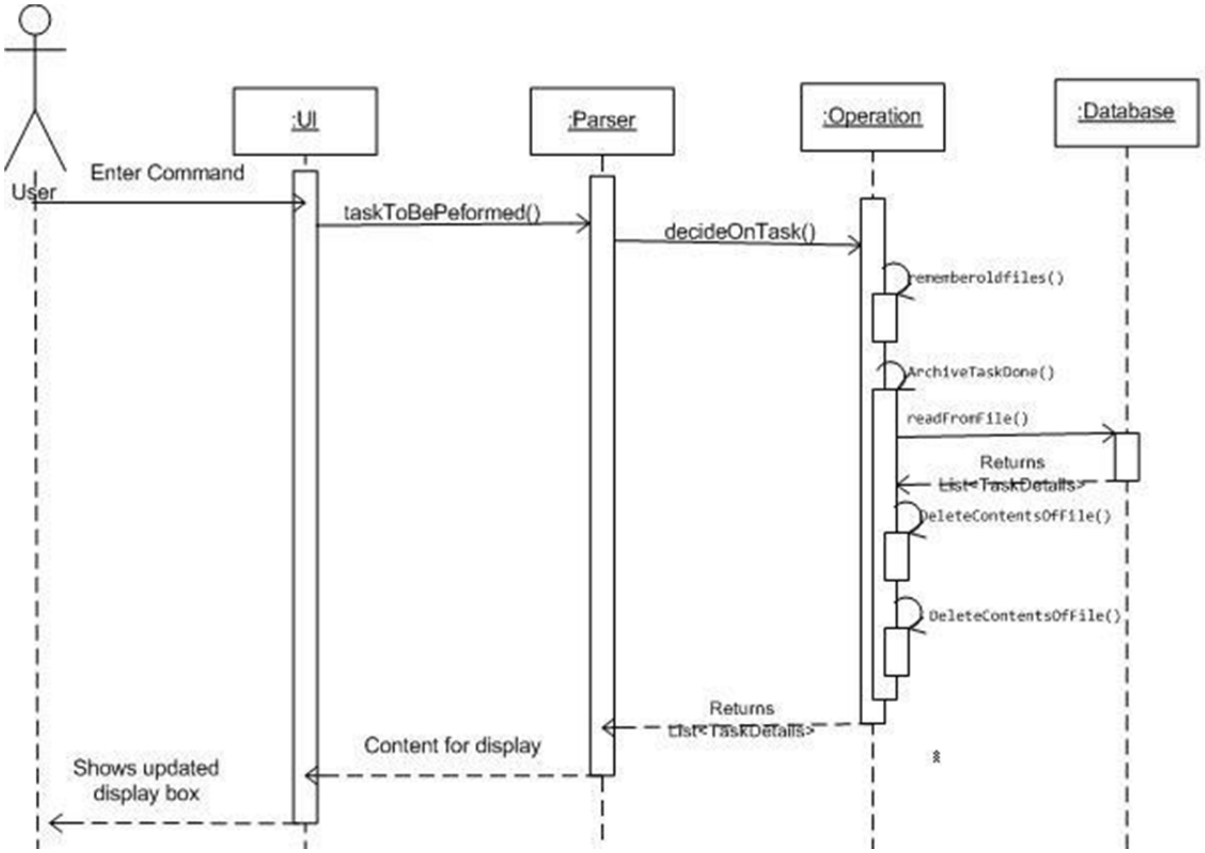
5. Undo


User
:UI
:Parser
:Operation
:Database
:TaskDetails
Enter Command
taskToBePeformed()
decideOnTask()
UndoTask()
writeIntoRedoFileBefore Undo()
readFromFile()
Returns List<TaskDetails>
clearAllTheContentsFrom TheFile()
Get the appropriate TaskDetail data members by calling the get()
Returns string containing the task detial object's data members value
Returns List<TaskDetails>
Content for display
Shows updated display box

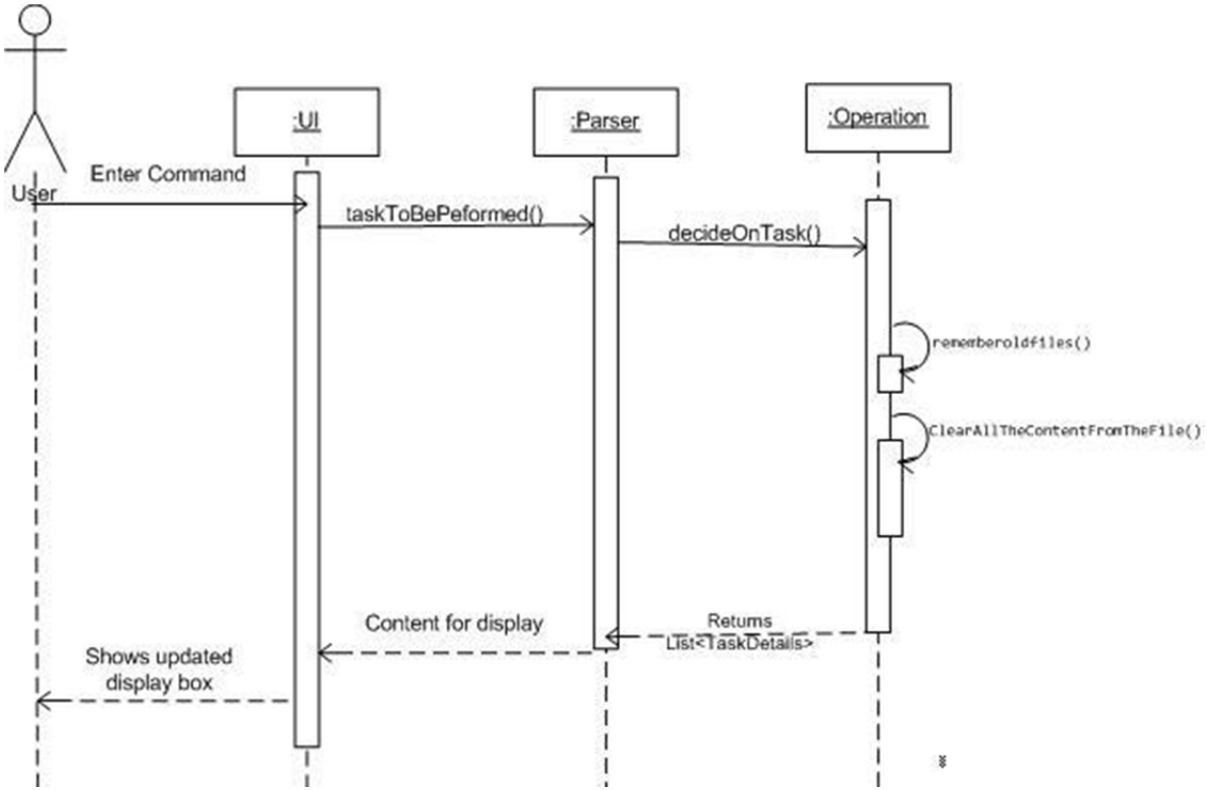
6. Redo


:UI
:Parser
:Operation
:Database
:TaskDetails
User
Enter Command
taskToBePeformed()
decideOnTask()
RedoTask()
readFromFile()
Returns
List<TaskDetails>
ClearAllTheContentsFrom
TheFile()
Get the appropriate TaskDetail data
members by calling the get()
Returns string containing the task
detial object's data members value
Returns
List<TaskDetails>
Content for display
Shows updated
display box

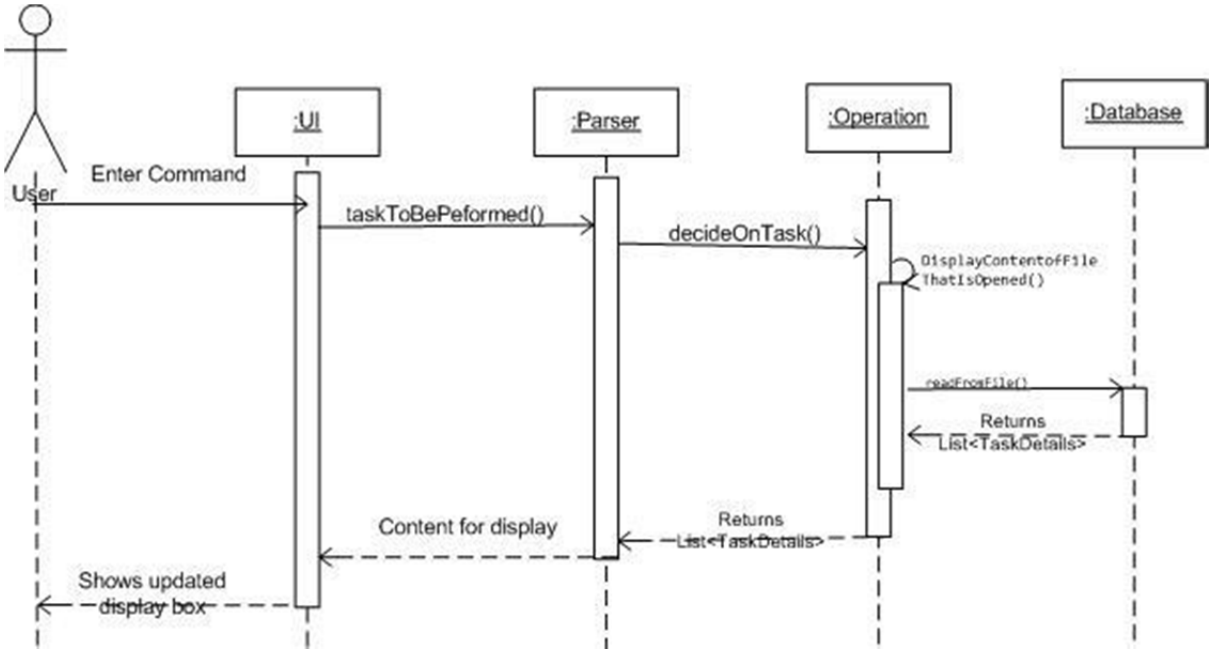
7. Done


:UI
:Parser
:Operation
:Database
User
Enter Command
taskToBePeformed()
decideOnTask()
rememberoldfiles()
ArchiveTaskDone()
readfromFile()
Returns
List<TaskDetails>
DeleteContentsOfFile()
DeleteContentsOfFile()
Returns
List<TaskDetails>
Content for display
Shows updated
display box

8. Clear

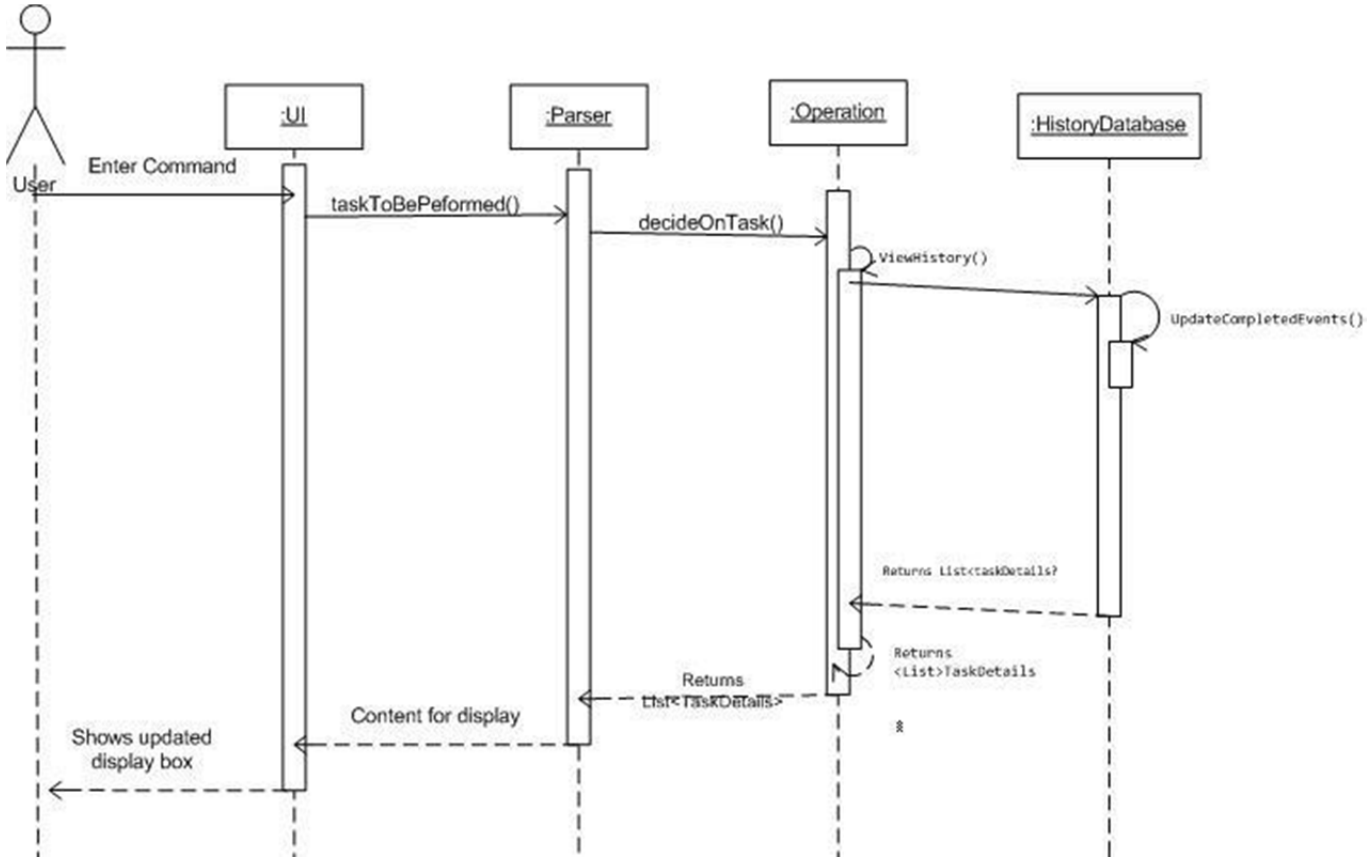

:UI
:Parser
:Operation
User
Enter Command
taskToBePeformed()
decideOnTask()
rememberoldfiles()
ClearAllTheContentFromTheFile()
Content for display
Returns
List<TaskDetails>
Shows updated
display box

9. Display

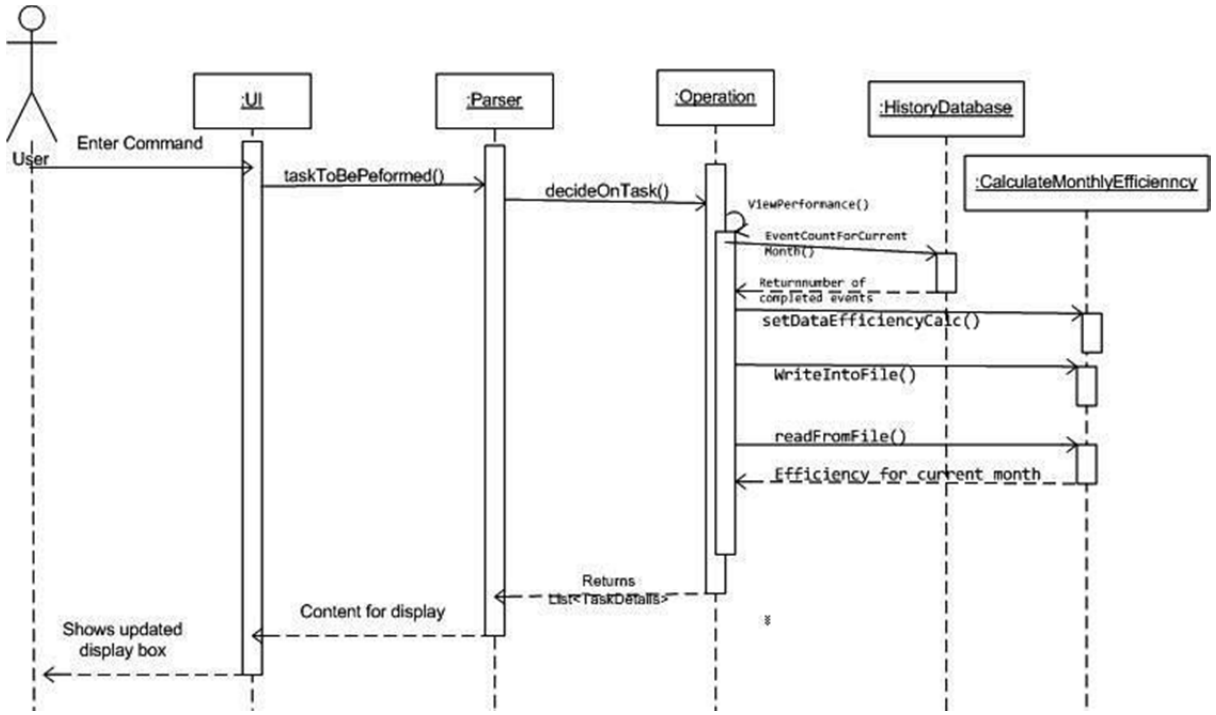

:UI
:Parser
:Operation
:Database
User
Enter Command
taskToBePeformed()
decideOnTask()
DisplayContentofFile
ThatIsOpened()
readFromFile()
Returns
List<TaskDetails>
Content for display
Returns
List<TaskDetails>
Shows updated
display box

Functionality for Advanced Features

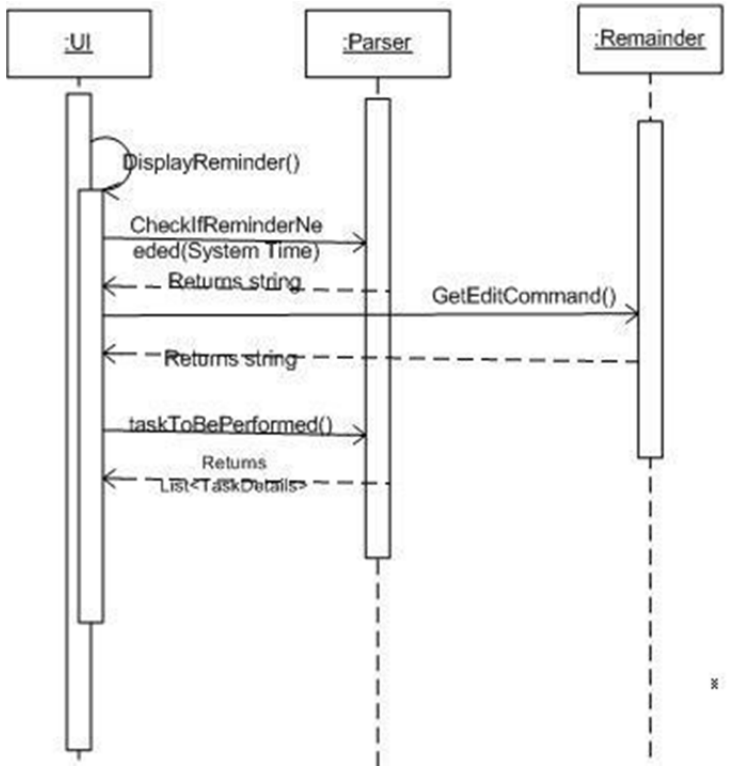
- History



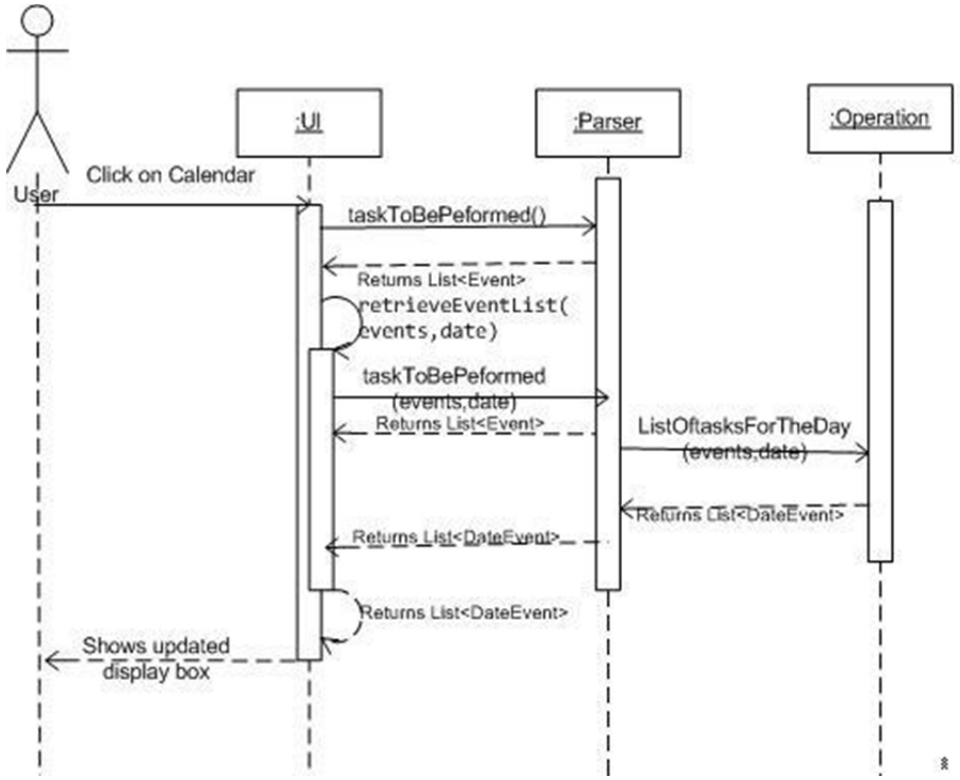
- **Performance**



- **Reminder Service**



- **Calendar**


:UI
:Parser
:Operation
User
Click on Calendar
taskToBePeformed()
Returns List<Event>
retrieveEventList(
events,date)
taskToBePeformed
(events,date)
Returns List<Event>
ListOftasksForTheDay
(events,date)
Returns List<DateEvent>
Returns List<DateEvent>
Returns List<DateEvent>
Shows updated
display box

- **Re-Schedule**
  While performing the add operation, in the BreakDownContentIntoAptDetails() function, a check is made to see if any function is already occupying the specified time slot by calling the checkIfAutpScheduling() function. If the latter returns a true value, the task is added but a message is shown to the user informing him of the possible clash and advising him to re-schedule. The functions called for performing Re-Schedule are:

- **Intellisense**

  When the user is typing the command in the Command Box, a drop down list containing a list of previously entered event details and keywords is shown below the Command Box. The user can scroll down the list and choose the required word with the right arrow key. This feature is implemented through an object of type ListBoxContent which takes the last sequence of characters entered in the Command Box, searches for it in the "IntellisenseDatabase" file. If found, it returns a <List<String> that is bound to the ListBox. If not found, then the new word is added to the "IntellisenseDatabase" file. The main functions that are called for executing this operation are:
  - ✓ addItemToListBoxContent()
  - ✓ searchForsubString()
  - ✓ addWordToIntellisenseDatabase()

Refer to the sequence Diagram for Add functionality in the beginning of this section for additional information.

2.4. Storage Details:

2.4.1. Internal Storage

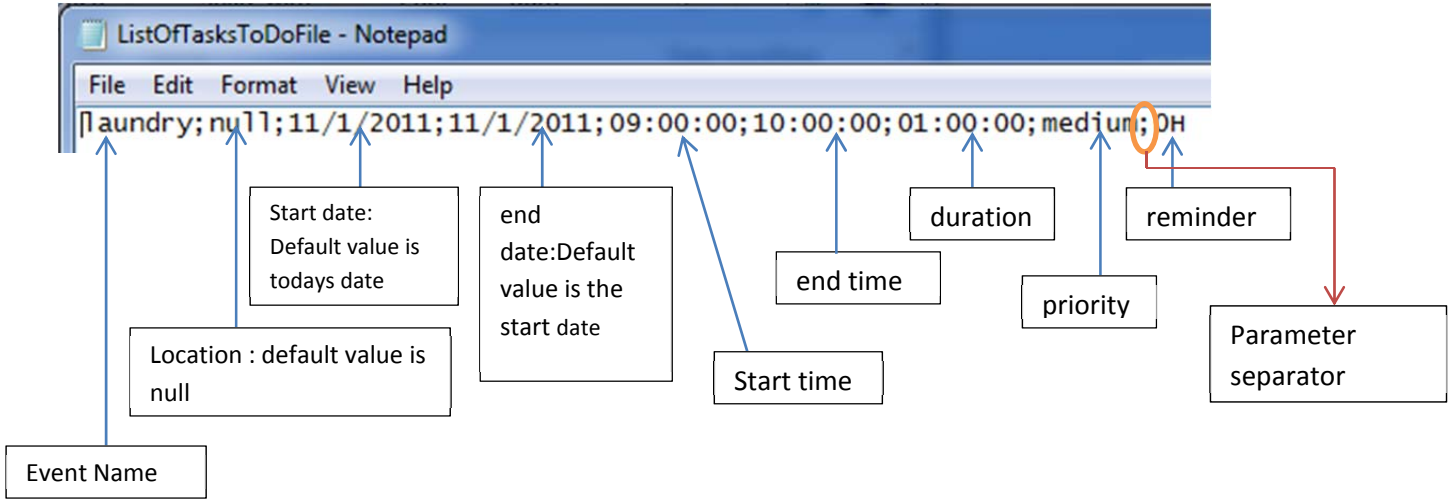
- TaskDetails : Each event is stored as a TaskDetails object which contains the following datamembers:
  - ✓ `Id`
  - ✓ `Eventname`
  - ✓ `Location`
  - ✓ `StartDate`
  - ✓ `EndDate`
  - ✓ `Startime`
  - ✓ `Endtime`
  - ✓ `Duration`
  - ✓ `Priority`

- ✓ Reminder
- ✓ Repeat
- ✓ EndRepeat

- Event : Each event is converted to an Event object for the purpose of displaying it in the datagrid. This class contains the following attributes:
  - ✓ Event_ID
  - ✓ Event_Name
  - ✓ Start
  - ✓ End
  - ✓ Priority
  - ✓ Reminder
- DateEvent :TaskDetails objects are converted to DateEvent objects for displaying the events on a particular day in Calendar View in the Display Box. The following are the members of the Date Event class:
  - ✓ Time
  - ✓ EventName

2.4.2. External Storage

The diagram below shows the format in which each event is stored in "ListOfTasksToDoFile.txt", the master file that stores the entire database of event information.



2.5. APIs and Class Diagrams:

User Interface Unit:

1. **MainWindow:**
   This class acts as the main interface between users and the program. The user specifies the operation to be performed thorough one of the following interactive tools in the GUI window:
   A. Command Box
   B. Calendar
   C. Menu
   D. Edit/Delete/Done buttons
   E. Flip View button

   - If the user has entered a command in the Command Box, it calls the Parser class for parsing the command.
   - If the user has selected one of the Help menu options, it invokes the About.xaml class to generate the help window. If he has selected View History or View Performance, then History.xaml class or PerformanceChart.xaml is invoked respectively.
   - In case he has clicked on one of the dates in the calendar, then the Logic unit is called through the Parser class to retrieve the events for that day from Storage Unit and display them in the datagrid. He can go back to the Display view through the FlipView button.
   - The user can also select an event from the datagrid and choose to edit, delete or mark the event as done. For this the Parser class is again invoked to execute the required operation.

2. **About**:
   This class generates the help window to display the User Manual and Syntax for Command.
3. **Performance Chart:**
   This class generates a window which displays a chart to track the user's efficiency across all the months of the year based on the number of completed and pending tasks. It consists of a user control that has been customised to show a WPF chart which in turn has been bound to an object of Efficiency class. The information required for generating this chart is calculated using the Monthly Efficiency class.
   A. Efficiency :
      This class has two data members: efficiency and month, which are the X and Y-axis values for the chart.
   B. Monthly Efficiency :

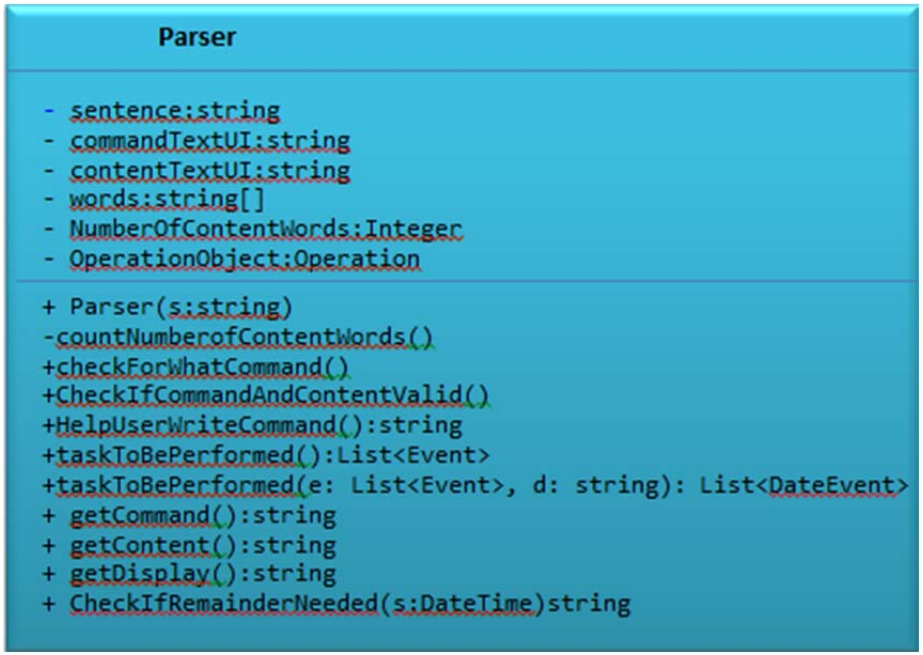
This class is used to compute the efficiency of the user for each month. It calculates the efficiency for the current month by reading information from "ListOfTasksToDoFile" and "Completed.txt". The calculated efficiency is then stored in the "Efficiency.txt" file which contains the statistics for monthly efficiency.

4. **Reminder:**
   This class generates the Reminder window that pops out every time a deadline is nearing or if a deadline has been overshot. It allows the user to postpone the deadline for the task if he wishes to.
5. **Edit_event:**
   This class generates the Edit Dialog Box that opens up when the user tries to edit a task using the Edit button. It shows the different fields with the existing information and the user can change any of these fields.

Parser Unit:

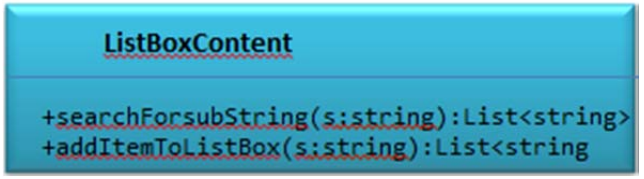
1. **Parser**:

The Parser class takes in the command entered by the user in the Command Box and separates the keywords from the event information through the taskToBePerformed() function. This parsed information is then passed on to the Logic unit for performing the operation. The Operation class of the Logic Unit invokes the decideOnTask() function to which the command word is passed.



2. **ListBoxContent**:

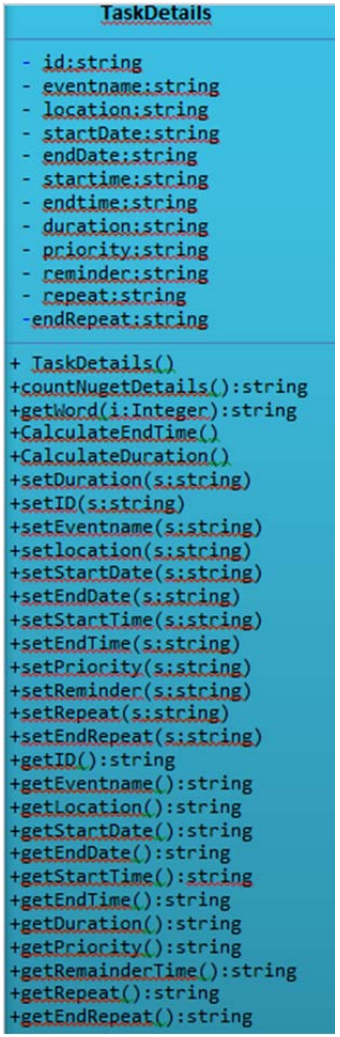
This class is responsible for taking in the sequence of characters entered by the user and searching for that substing in the "IntellisenseDatabase" using the searchForsubString() function. If the word does not already exist in the file, it is added through the addItemToListBox() function.



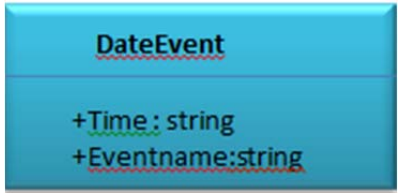
Logic

- **Helper Classes:**

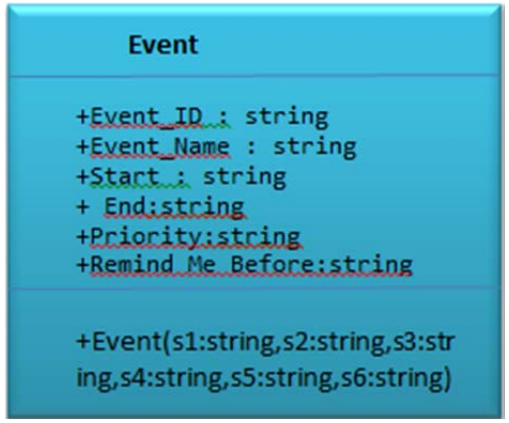
1. Task Details



2. **DateEvent**

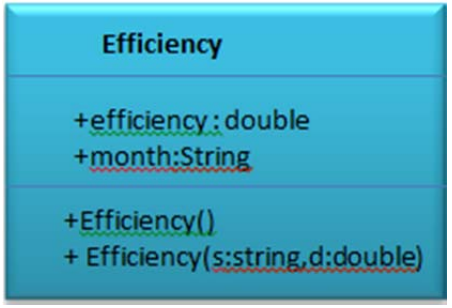


3. **Event**

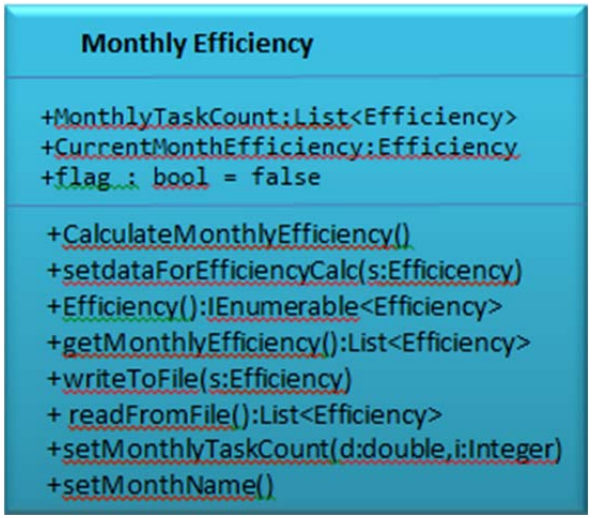


These three classes are used to store the task details in different formats as required by the operation. This has been explained in Internal Storage Details in Section 2.4.

4. Efficiency



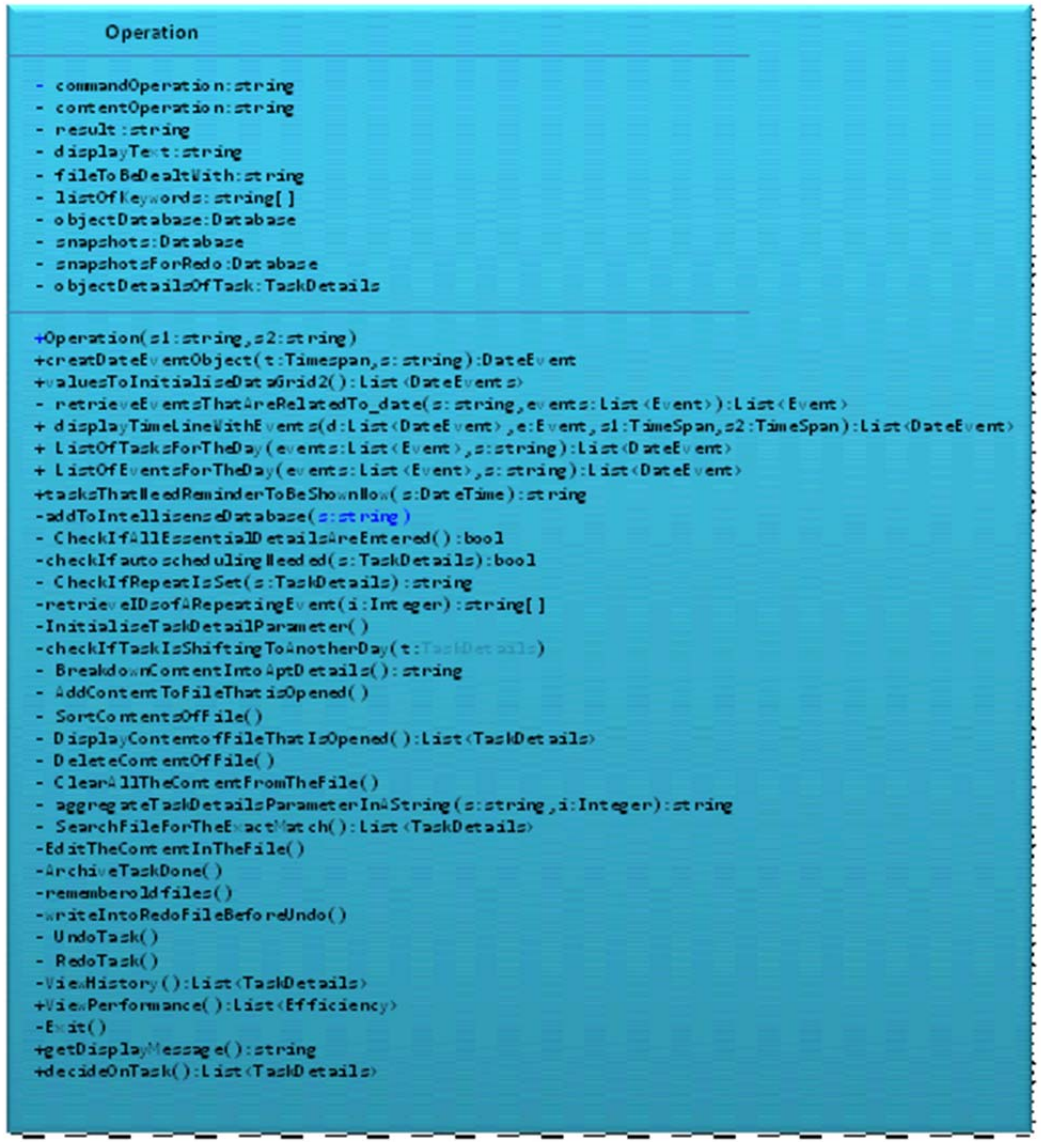
5. Monthly Efficiency



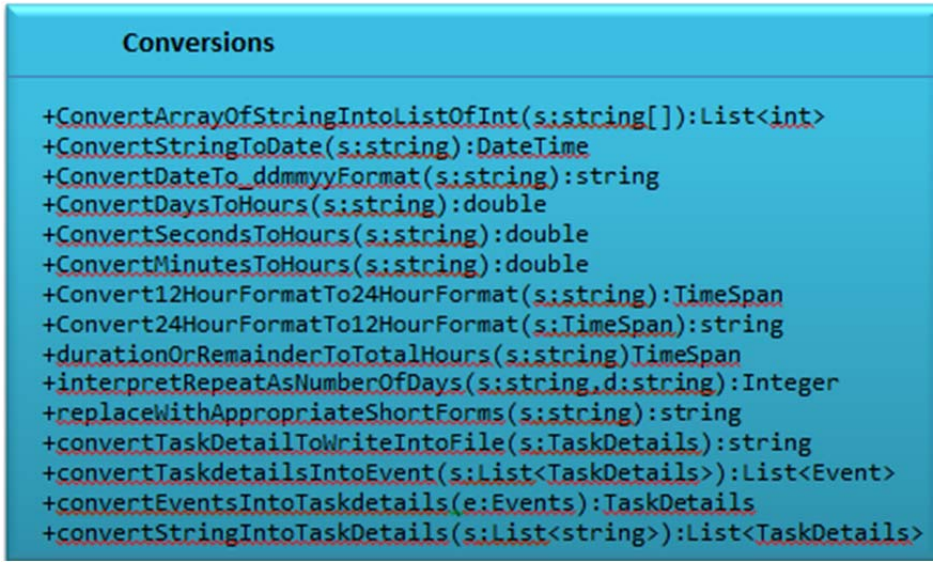
These two classes are used to display the performance chart as has been elaborated before in Section 2.3.

- **Logic Implementation Classes:**
  1. **Operation**:

     This class forms the crux of the logic for the entire program. It takes the command keyword and event information from the Parser class and calls the appropriate function based on the command using the decideOnTask() function. The list of functions which are invoked for each operation has been illustrated through the sequence diagrams (Should the sequence diagrams come here?……..)below. This class is also the one which interacts with the files in storage Unit to manipulate the task details stored there.



2. **Conversions**:

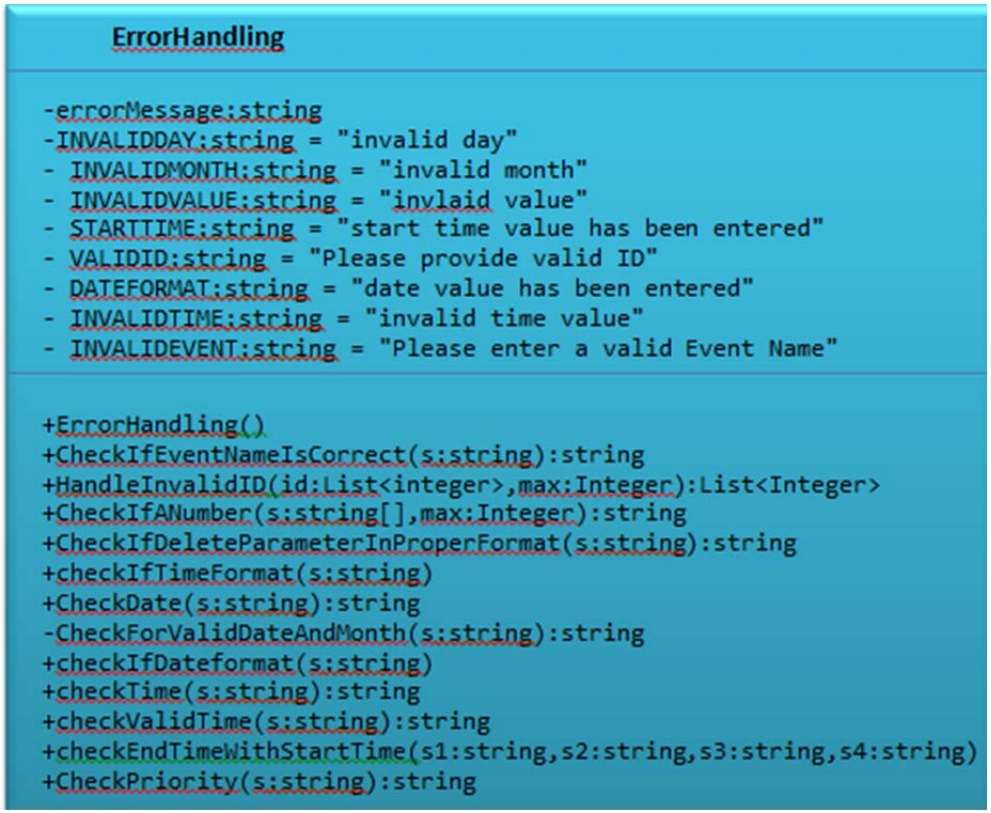


This class takes care of the different conversions required to change the parameters form one format to the other. Some of the main conversions that take place are:

A. String to Date/TaskDetails Object: ConvertStringToDate() , convertStringIntoTaskDetails()
B. 12 Hour Time Format to 24 Hour Time Format: Convert12HourFormatTo24HourFormat()
C. Days/ Minutes/ Seconds to Hours: ConvertDaysToHours(), ConvertSecondsToHours(), ConvertMinutesToHours()
D. TaskDetails object to Event Object and vice-versa: convertTaskdetailsToEvent(), convertEventsIntoTaskdetails()

3. Error Handling:

As the name suggests, this class takes care of the different errors that could creep up during runtime through defensive coding, exception handling, logging and assertion. In case of invalid syntax, the program tries to extract valid information and updates the required field on its own thereby relaxing the constraints enforced on the user. If the entire input is invalid, then an error message is returned to the UI to notify him of the invalid field. An important point to be noted is that the program offers flexibility to the user to enter 'start' and 'end' dates in 'from' and 'to' fields as well and vice-versa.

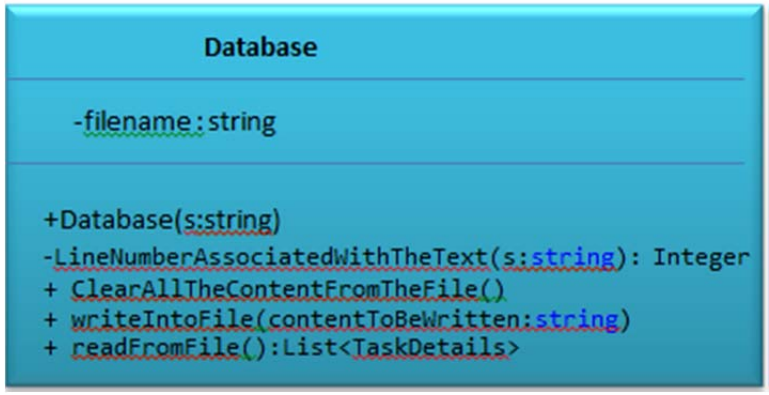


Storage

1. **Database**:

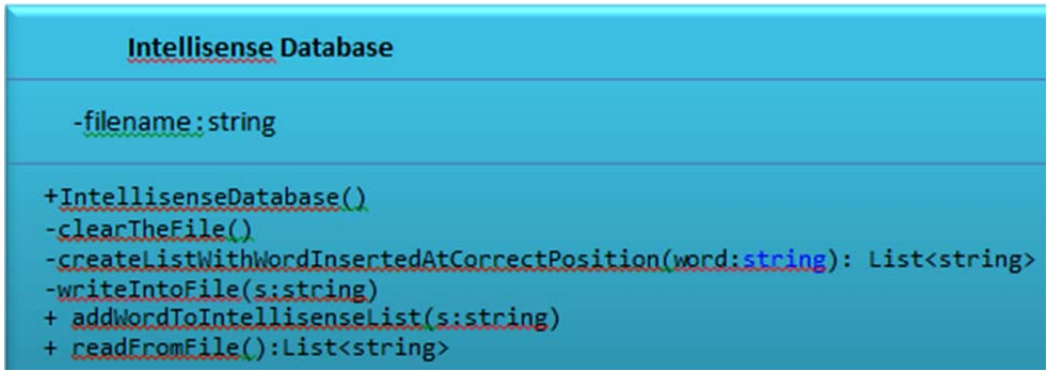
   The main purpose of this class is to take care of writing to and reading from the different files that store the task details. It contains the following main functions for implementing file manipulation:

   A. writeIntoFile()
   B. readFromFile()
   C. ClearAllTheContentFromTheFile()



2. **IntellisenseDatabase**:

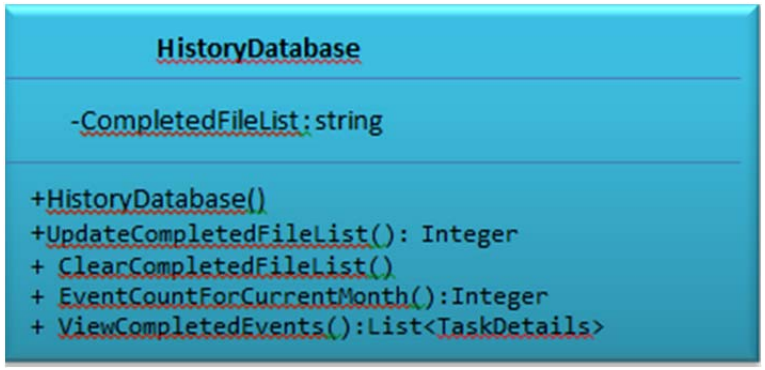
This class implements functions to manipulate the "Intellisense.txt" file. In addition to the common write, read and clear operations, its functions enable addition of new words to the file in the appropriate format and in the correct position through the createListWithWordInsertedAtCorrectPosition() function.



3. **HistoryDatabase**:

This class archives completed tasks in the "Completed.txt" file so that the user can view them later on. It provides the following functions to retrieve, clear and update the application history.

A. ViewCOmpletedEvents()
B. UpdateCompletedFileList()
C. ClearCompletedFileList()



3. What's Different About What's Next?

1. **Implicit Event Organizer:**
   What's Next? implicitly helps the user manage his tasks efficiently through its Re-schedule feature that prompts the user in case of clashes between events. Also the Calendar View helps the user view his schedule on any day, facilitating him to plan his future tasks better.
2. **Simple and Intuitive UI:**
   The command line interface is highly intuitive with easy-to-remember keywords and flexibility in syntax.
3. **Reminder Service System:**
   What's Next? goes beyond just scheduling tasks for the user and notifies him of impending deadlines even when the application is not running in the front end. The user can just minimise the application to an icon on his Task Manager while continuing with his other work.
4. **Efficiency Indicator:**
   Our To-Do Manager motivates the user to improve his efficiency through its Performance Indicator feature that shows him his efficiency graph for the year.

4. Testing Instructions:

4.1. Development Environment:

What's Next has been coded in C# programming language. It has been developed on Microsoft Visual Studio Professional 2010 IDE. Compilation and testing were done on Windows 7 Operating System.

4.2. Runtime Environment:

It is recommended that the user install WPF Toolkit in Windows before running the application.

5. Scope for Future Improvements:

What's Next? can further be improved if the following features are also incorporated into the system:

1. **Natural Language Processing:**
   The application should allow the user to specify event information using natural sentences and should be able to extract required information from the details entered.
   Example: Meet auditor on Sunday, 10am to file income tax returns.
2. **Integration with GCal:**
   Integrating the application with Google Calendar will add value to the user experience as events can be imported from here to GCal. What's Next? is a desktop utility and this integration will enable the user to view his schedule no matter where he is.

6. Credits:

http://wpf.codeplex.com : Download of WPF Toolkit