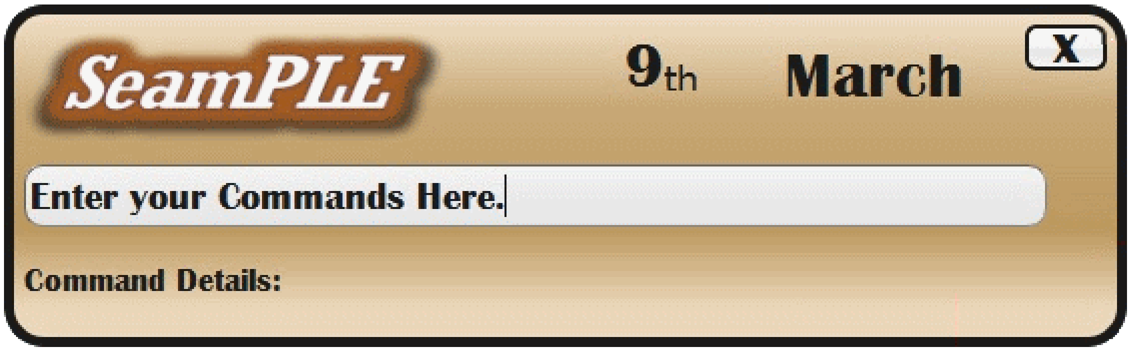
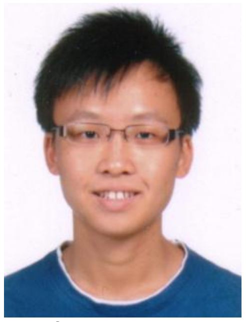
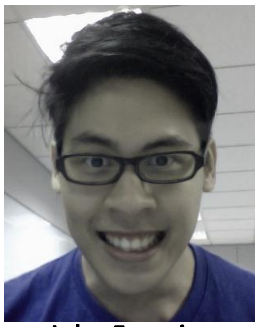
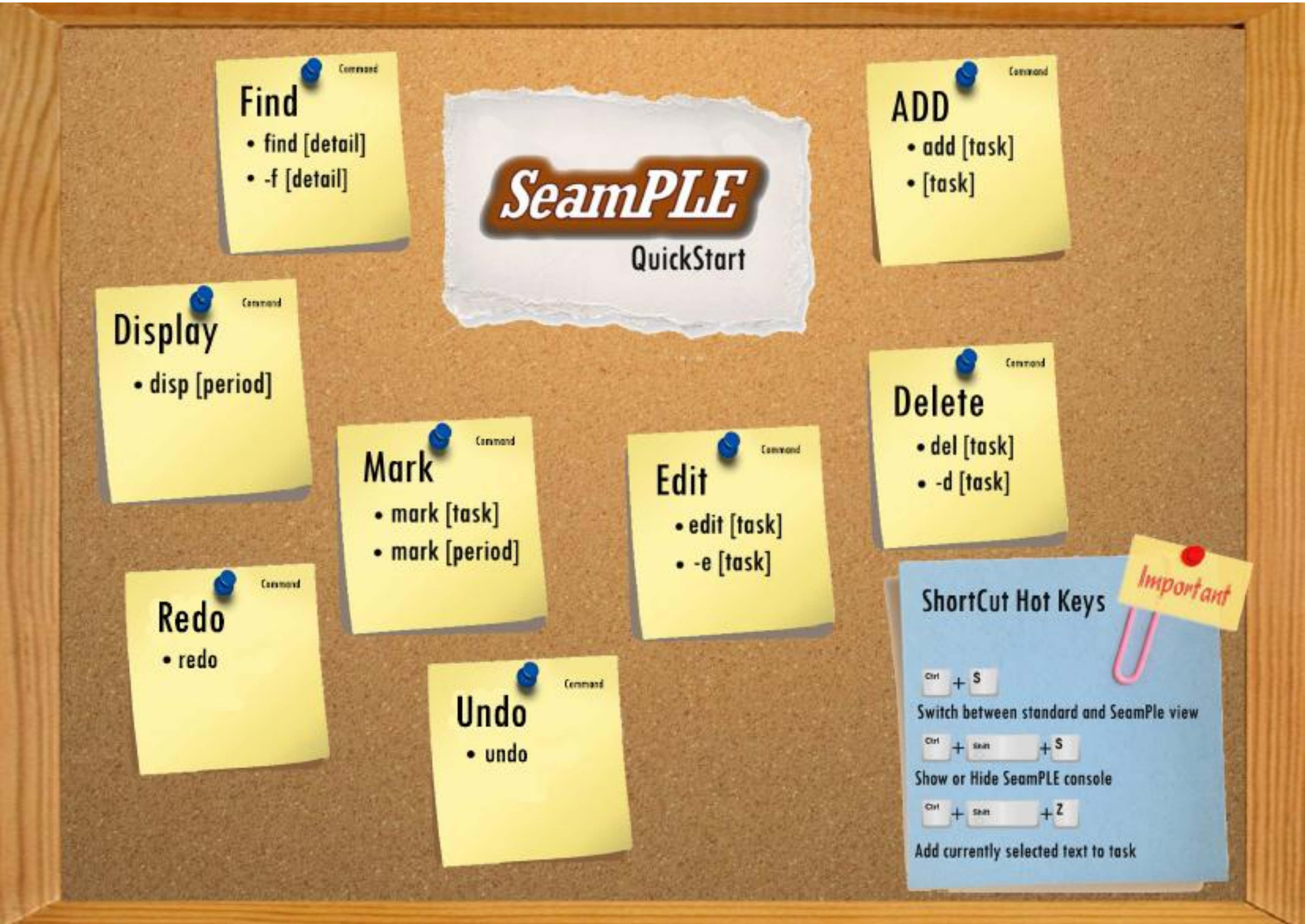
SeamPLE



A User Manual

V 0.1

Liu Weiyuan Programmer, Designer, Content Producer	**Riandy** Content Producer, Programmer, Time-Keeper	 **Cham Wen Bin** Leader, Designer, Programmer	**Pan Wenren** Programmer, Designer, Content Producer	 **John Francisco** Programmer, Designer, Content Producer



SeamPLE

QuickStart

Command

Find

- find [detail]
- -f [detail]

Command

ADD

- add [task]
- [task]

Command

Display

- disp [period]

Command

Mark

- mark [task]
- mark [period]

Command

Edit

- edit [task]
- -e [task]

Command

Delete

- del [task]
- -d [task]

Command

Redo

- redo

Command

Undo

- undo

Important

ShortCut Hot Keys

Ctrl + S

Switch between standard and SeamPle view

Ctrl + Shift + S

Show or Hide SeamPLE console

Ctrl + Shift + Z

Add currently selected text to task

What is SeamPLE Software?

SeamPLE is a todo list widget that is aimed at allowing users (you!) to have total control over day-to-day events. We help you achieve this through our simplistic yet intuitive user interface, as well as our intelligible widget command list.

Getting SeamPLE to run

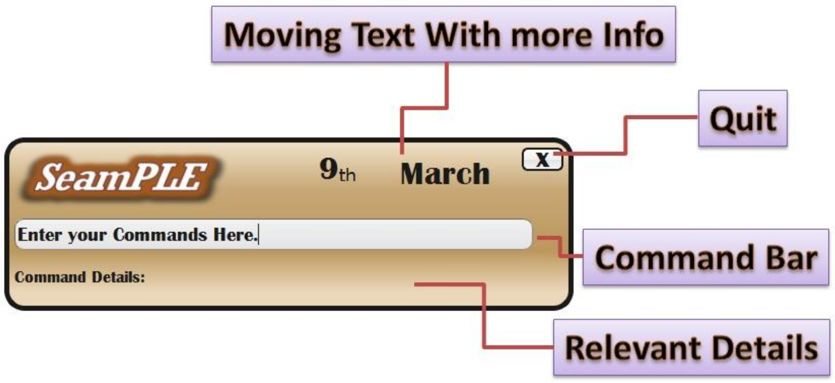
Operating System Requirement: WindowsXP/WindowsVista/Windows7**[RECOMMENDED]**

To run the program, double click SeamPLE.exe icon. (Simple, ain't it?)

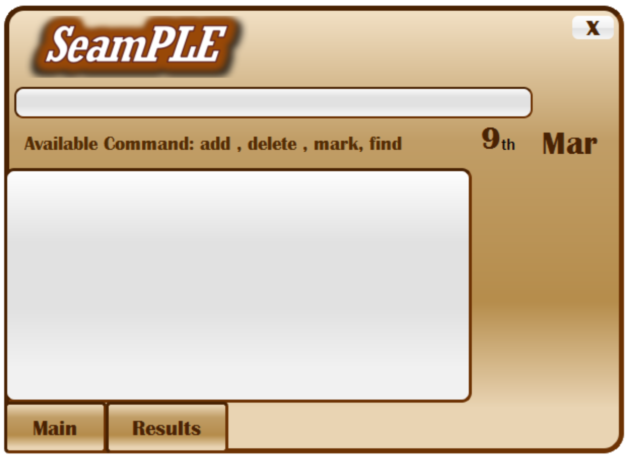
Experiencing SeamPLE

There are two views to our SeamPLE application: **SeamPLE view and Standard view**

SeamPLE view



Standard view



SeamPLE view is seen most regularly as a toolbar notification while **Standard view** is seen only when the program is maximised.

Using the Standard view offers the user more customization options, such as having the ability to check through the day's events and tasks or even searching for an event that is keyed into SeamPle previously. The SeamPLE view offers the user an alternate and more convenient way of using the program, by allowing the user to issue commands quickly without the need for maximizing the widget view.

To view the different commands that can be used with SeamPLE, please flip to the next page.

Command Walkthrough

ADD

Description: Use this command to add a specific task that you want.
Just type the detail that you want and SeamPLE will help you to manage them!

```
Command : add [task] [date] [category] [time] [priority]

        : [task] [date] [category] [time] [priority]
```

Ex:
```
add dinner With mary
Meet investors Monday
```
`add dinner With mary` → Add task into today's list of things to do.
`Meet investors Monday` → Add task into the upcoming Moday.

Edit

Description: Use this command to edit any details of a specific task.
Just type the details that you want to edit and SeamPLE will update accordingly!
Navigate across fields that you wish to edit with the **`tab`** button.

```
Command : edit [task] [date] [category] [time] [priority]

        : -e   [task] [date] [category] [time] [priority]
```

Ex:
`Edit meeting HIGH` → Edit task's Priority to be high .
`-e homework Monday` → Edit task's Date into the upcoming Moday.

Delete

Description: Use this command to delete any specific task.
Just type the task with the ID that you want to delete and SeamPLE will clear them!

```
Command : del [task ID]

        : -d  [task ID]
```

Ex:
del `101` → Delete Task "Tennis with Jamie" . (Task embedded with ID 101 is deleted)
`-d 34` → Delete Task "Walk dog". (Task embedded with ID 34 is deleted)

Find

Description: Use this command to find a specific task that you want or even list out several task with the same category. Just type the detail that you want and SeamPLE will help you list all of them!

```
Command : find [category]⏎ , find [event name]⏎,
          find [event ID]⏎
        : -f [category]⏎ , -f [event name]⏎, -f [event ID]⏎
```

Ex: `Find meeting` → List all tasks that contains the word "meeting"
`Find 101` → Show the task "Tennis with Jamie".

Display

Description: Use this command to display all tasks

```
Command : dsp ⏎
        : display⏎
```

Ex: `dsp` → Show all tasks that are saved in text file

Mark

Description: Use this command to mark/unmark your events/tasks. You can mark the tasks that you have done and put it in archive and also unmark an event if you accidentally mark it done. Besides that you can also mark all the past events or tasks that have passed it's deadline.

```
Command : mark [task] [period]⏎
```

Ex: `mark meeting with John` → Mark "meeting with John" event done
`mark yesterday` → Mark all yesterday events/tasks done.

Note : command in red is optional. But you have to provide at least one of it to guarantee that the command you want is executed.

Undo

Description: To revert the latest change made by the user. To allow the user to reverse the command of the alsto cancel or reverse the last command added

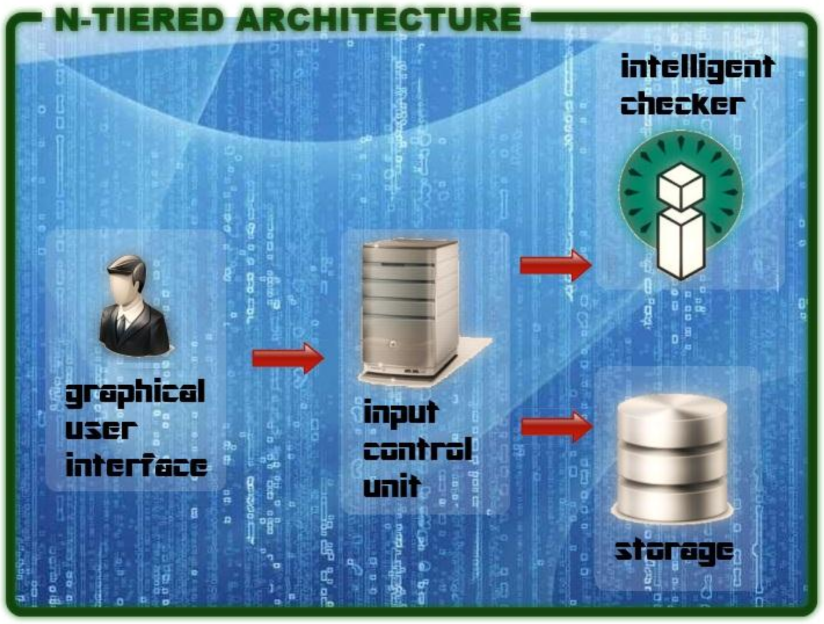
```
Command : undo Enter
```

Redo

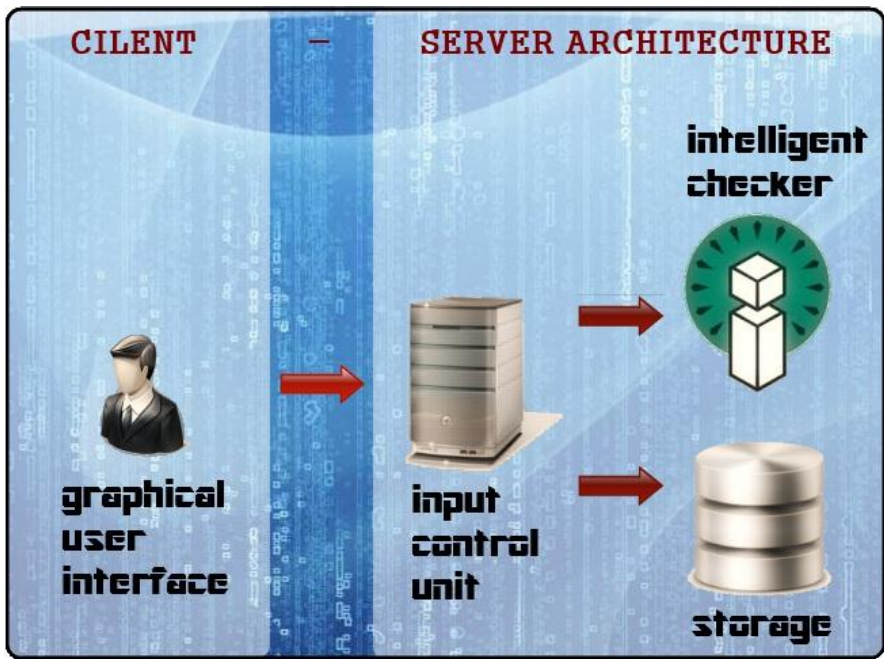
Description: To reverse an undo operation

```
Command : redo Enter
```

Architectural Style

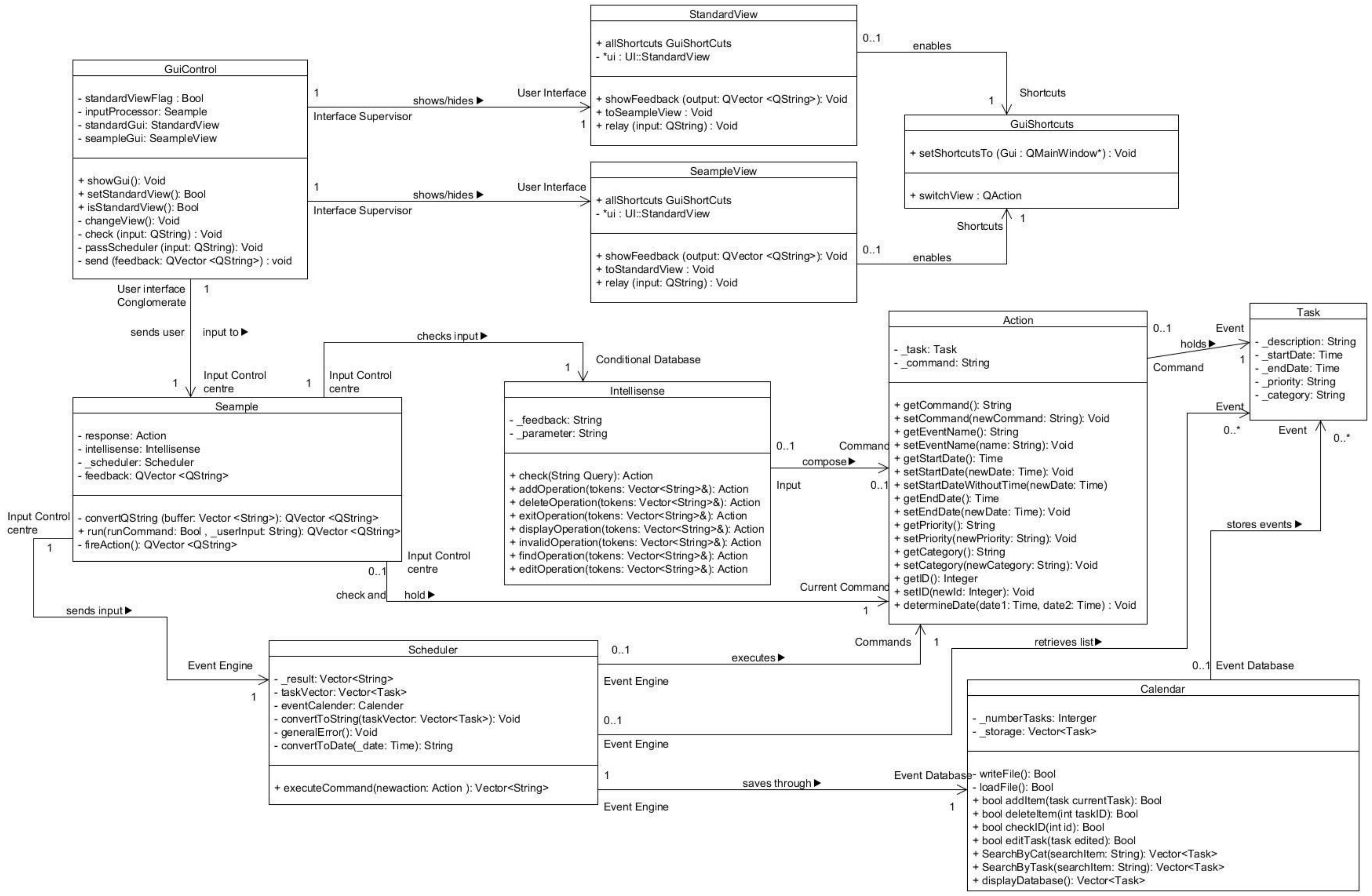


SeamPLE utilizes the N-Tiered Architectural style to power its operations. Starting with the highest level, users interact with various **Graphical User Interfaces (GUI)** that have been prepared specially for them. This input is sent to the **Input Control Unit (ICU)** which takes control of this input and decides whether it should be sent to the **Intelligent Checker (IC)** for validating the input or the **storage** for running the user inputs and storing or retrieving the appropriate details.



SeamPLE has been cleverly engineered to suit developers' needs and purposes. By adopting the approach of attaining minimum coupling between the **GUI** and the **ICU**, we allow the **GUI** and other components of the products to be easily exchangeable with other developer created classes. The "Client" Component, as represented by the **GUI**, can be easily exchanged with other developer created **GUI** or even the command prompt. The "Server" component, as represented by the rest of our system, can be replaced by developer classes to process any possible user inputs.

Next, we'll be looking at the overall class diagram of **SeamPLE**. The class diagram is an essential tool for any prospective developers to continue development or to create new innovations from out system.


StandardView
+ allShortcuts GuiShortCuts
- *ui : UI::StandardView
+ showFeedback (output: QVector <QString>): Void
+ toSeampleView : Void
+ relay (input: QString) : Void
0..1
enables
Shortcuts
1
GuiShortcuts
+ setShortcutsTo (Gui : QMainWindow*) : Void
+ switchView : QAction
GuiControl
- standardViewFlag : Bool
- inputProcessor: Seample
- standardGui: StandardView
- seampleGui: SeampleView
+ showGui(): Void
+ setStandardView(): Bool
+ isStandardView(): Bool
- changeView(): Void
- check (input: QString) : Void
- passScheduler (input: QString): Void
- send (feedback: QVector <QString>) : void
1
shows/hides ▶
User Interface
Interface Supervisor
1
1
shows/hides ▶
User Interface
Interface Supervisor
SeampleView
+ allShortcuts GuiShortCuts
- *ui : UI::StandardView
+ showFeedback (output: QVector <QString>): Void
+ toStandardView : Void
+ relay (input: QString) : Void
Shortcuts
1
0..1
enables
User interface
Conglomerate
1
sends user
input to ▶
checks input ▶
1
Conditional Database
1
Input Control
centre
1
Input Control
centre
Seample
- response: Action
- intellisense: Intellisense
- _scheduler: Scheduler
- feedback: QVector <QString>
- convertQString (buffer: Vector <String>): QVector <QString>
+ run(runCommand: Bool , _userInput: String): QVector <QString>
- fireAction(): QVector <QString>
Intellisense
- _feedback: String
- _parameter: String
+ check(String Query): Action
+ addOperation(tokens: Vector<String>&): Action
+ deleteOperation(tokens: Vector<String>&): Action
+ exitOperation(tokens: Vector<String>&): Action
+ displayOperation(tokens: Vector<String>&): Action
+ invalidOperation(tokens: Vector<String>&): Action
+ findOperation(tokens: Vector<String>&): Action
+ editOperation(tokens: Vector<String>&): Action
0..1
Command
compose ▶
Input
0..1
Action
- _task: Task
- _command: String
+ getCommand(): String
+ setCommand(newCommand: String): Void
+ getEventName(): String
+ setEventName(name: String): Void
+ getStartDate(): Time
+ setStartDate(newDate: Time): Void
+ setStartDateWithoutTime(newDate: Time)
+ getEndDate(): Time
+ setEndDate(newDate: Time): Void
+ getPriority(): String
+ setPriority(newPriority: String): Void
+ getCategory(): String
+ setCategory(newCategory: String): Void
+ getID(): Integer
+ setID(newId: Integer): Void
+ determineDate(date1: Time, date2: Time) : Void
0..1
Event
holds ▶
1
Command
Task
- _description: String
- _startDate: Time
- _endDate: Time
- _priority: String
- _category: String
Event
0..*
Event
0..*
stores events ▶
Input Control
centre
1
Input Control
centre
0..1
check and
hold ▶
Current Command
1
sends input ▶
Event Engine
1
Scheduler
- _result: Vector<String>
- taskVector: Vector<Task>
- eventCalender: Calender
- convertToString(taskVector: Vector<Task>): Void
- generalError(): Void
- convertToDate(_date: Time): String
+ executeCommand(newaction: Action): Vector<String>
0..1
executes ▶
Event Engine
Commands
1
retrieves list ▶
0..1
Event Engine
0..1
Event Database
1
saves through ▶
Event Database
Event Engine
1
Calendar
- _numberTasks: Interger
- _storage: Vector<Task>
- writeFile(): Bool
- loadFile(): Bool
+ bool addItem(task currentTask): Bool
+ bool deleteItem(int taskID): Bool
+ bool checkID(int id): Bool
+ bool editTask(task edited): Bool
+ SearchByCat(searchItem: String): Vector<Task>
+ SearchByTask(searchItem: String): Vector<Task>
+ displayDatabase(): Vector<Task>

Through the class diagram of **SeamPLE**, the different relations between the various components are explored. It is important to note that GuiControl must always contain exactly one instance of Seample and Seample must be owned by one instance of GuiControl. This supports the Client – Server architecture that was mentioned earlier on, where one unique client can have only one relation with another unique server.

API Descriptions

Next, we look at possible API descriptions for our program. The API descriptions support what we have implemented in the class diagram above. These descriptions can be used for further implementation (extending on current purposes) and testing (for writing stubs that emulates component behavior)

GUI CONTROL class

send

Description

Function to display feedback to users through the GUI. Call this function when a component wants to interact with the user. API will display all elements accordingly.

Syntax

void send(QVector <QString> feedback)

Input value(s): Qvector of Qstrings containing feedback strings

Return values (s) : None. Output displayed by GUI

passScheduler

Description

Event trigger function when "enter" key is pressed. Function will call scheduler to execute user's command.

Syntax

void passScheduler(QString input)

Input value(s): Qstrings containing user command input

Return values (s) : None. Output displayed by GUI

check

Description

Calls for intellisense.check to parse user command input.

Syntax

void check(QString input)

Input value(s): Qstrings containing user command input

Return values (s) : None.

INTELLISENSE class

check

Description

Parses user's command input in the textfield into Action files

Syntax

Action check(string query)

Input value(s): String containing user command input
Return values (s) : None.

getParameter

Description

Determine the input requirements from user command input and generates a feedback string to inform the user of the required fields.

Syntax

string getParameter()
Input value(s): None
Return values (s) : String of colour coded feedback indications

SCHEDULER class

executeCommand

Description

Performs the relevant actions based on the type of operation, input fields.

Syntax

vector<string> executeCommand(Action newaction)
Input value(s): Action class containing input field information
Return values (s) : Vector of string containing feedback messages depending on operation.

Calender

addItem

Description

Stores a user task into the database.

Syntax

bool addItem(task currentTask)
Input value(s): task class containing input field information
Return values (s) : Boolean true if operation is successful false otherwise.

deleteItem

Description

Remove a user task from the database .

Syntax

bool deleteItem(int taskID)
Input value(s): task class containing input field information
Return values (s) : Boolean true if operation is successful false otherwise.

editTask
Description
Edit a user task in the database .
Syntax
```
bool editTask(task edited)
```
Input value(s): task class containing input field information
Return values (s) : Boolean true if operation is successful false otherwise.

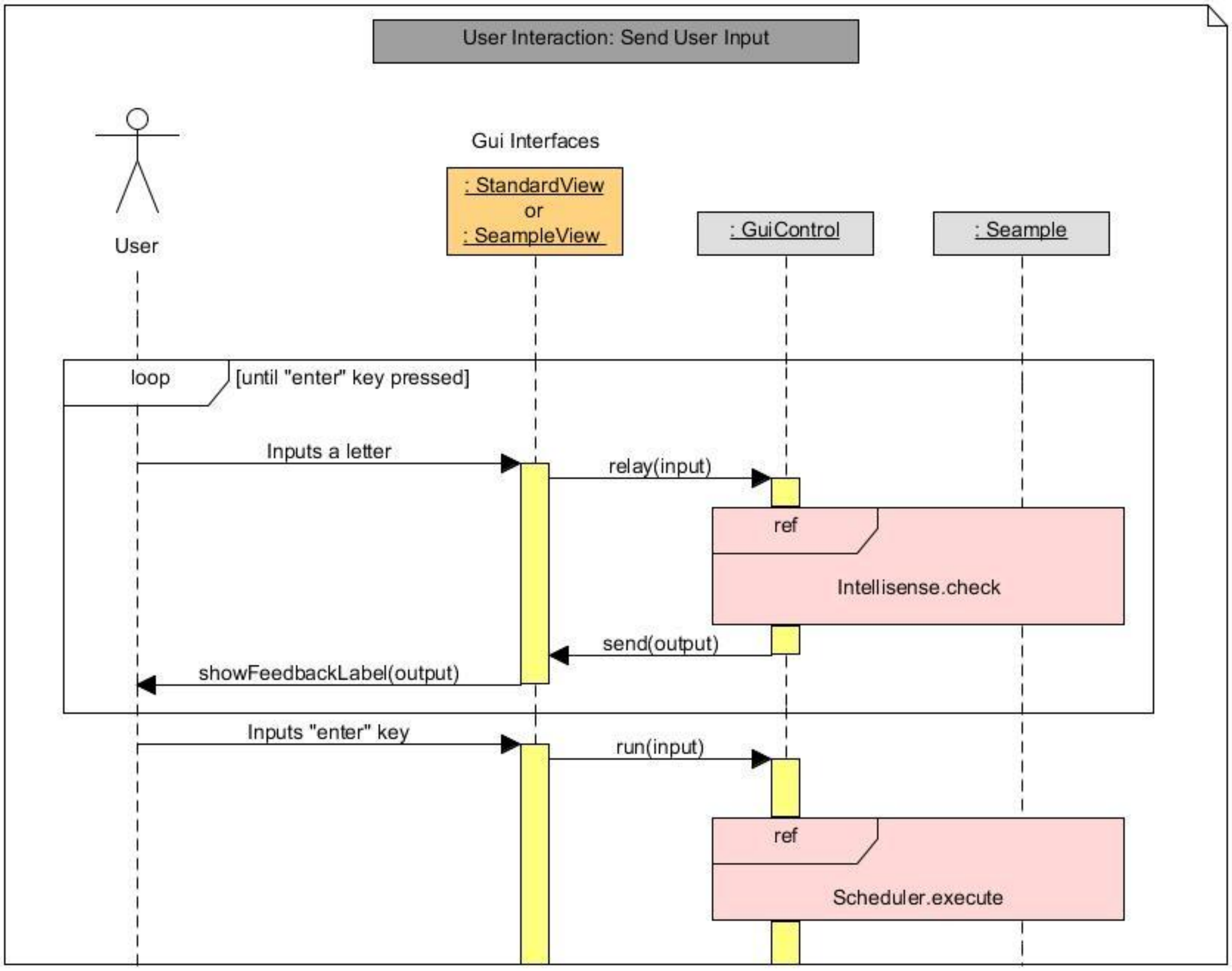
displayDatabase
Description
Display all user tasks in the database .
Syntax
```
vector<task> displayDatabase()
```
Input value(s): None
Return values (s) : Vector containing all task information

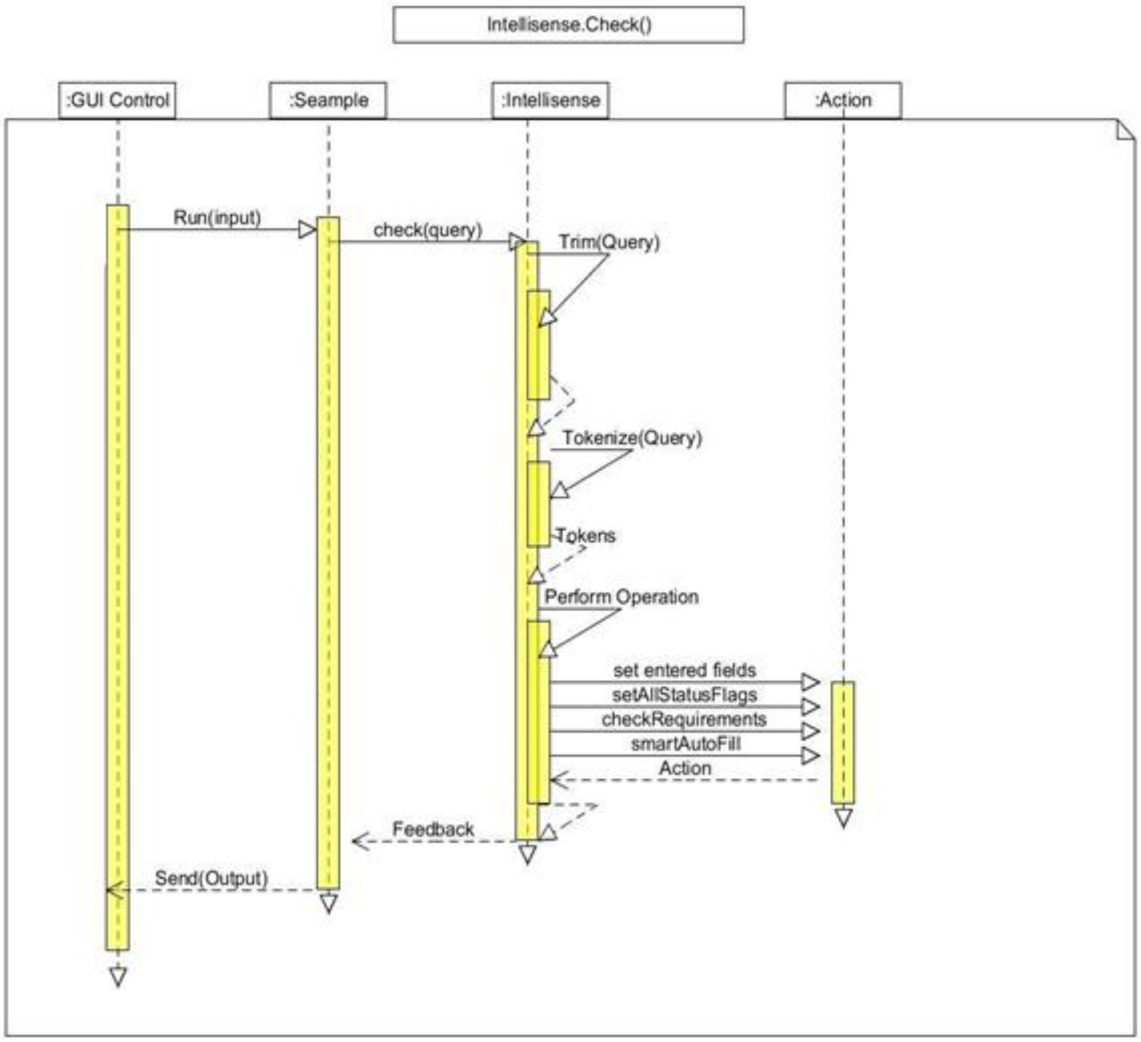
CALENDAR class

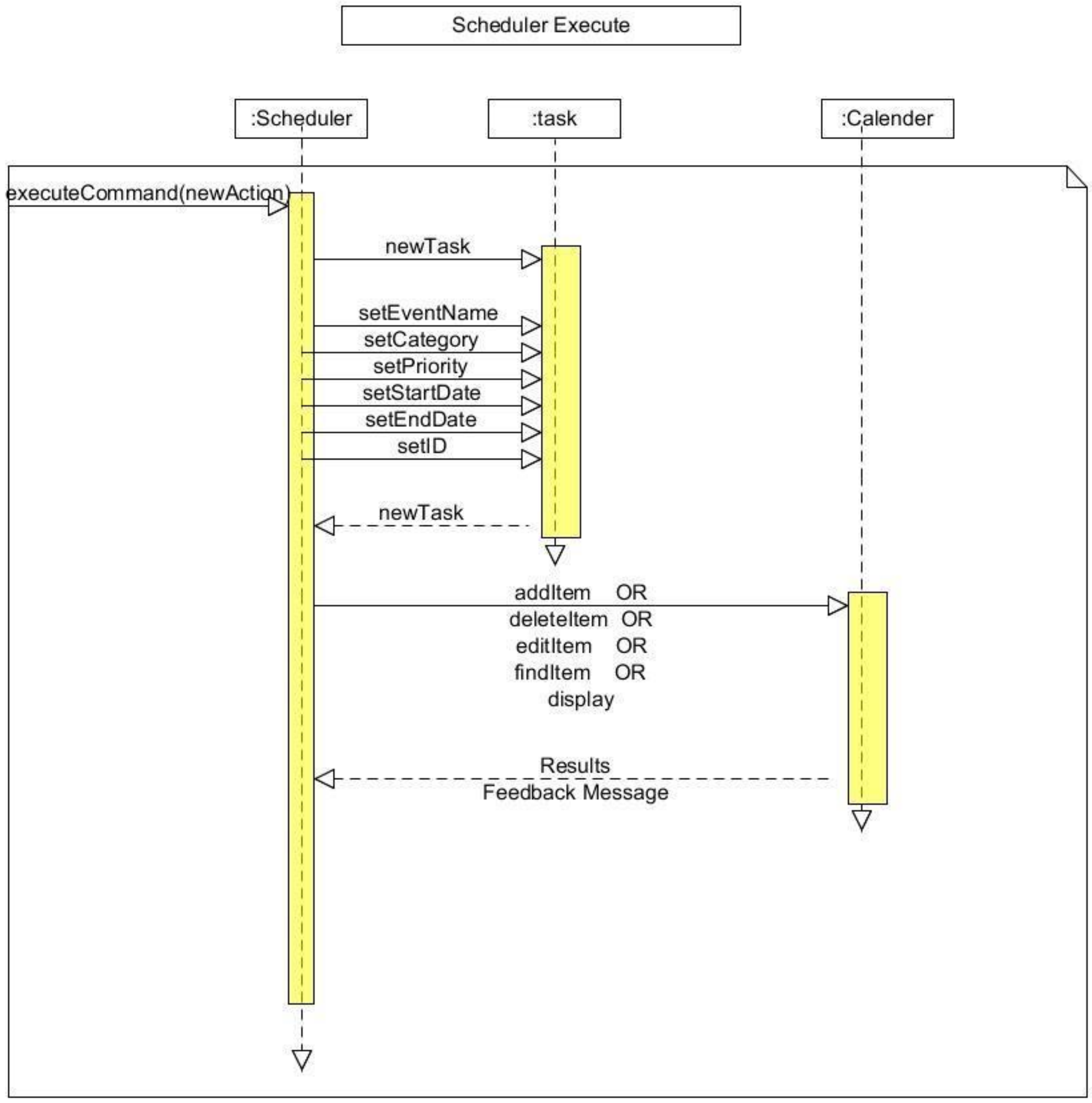
SearchByTask
Description
Search all user tasks in the database based on task ID.
Syntax
```
vector<task> SearchByTask(string searchItem)
```
Input value(s): string containing search field
Return values (s) : Vector containing all task information

SearchByCat
Description
Search all user tasks in the database based on task category.
Syntax
```
vector<task> SearchByCat(string searchItem)
```
Input value(s): string containing search field
Return values (s) : Vector containing all task information

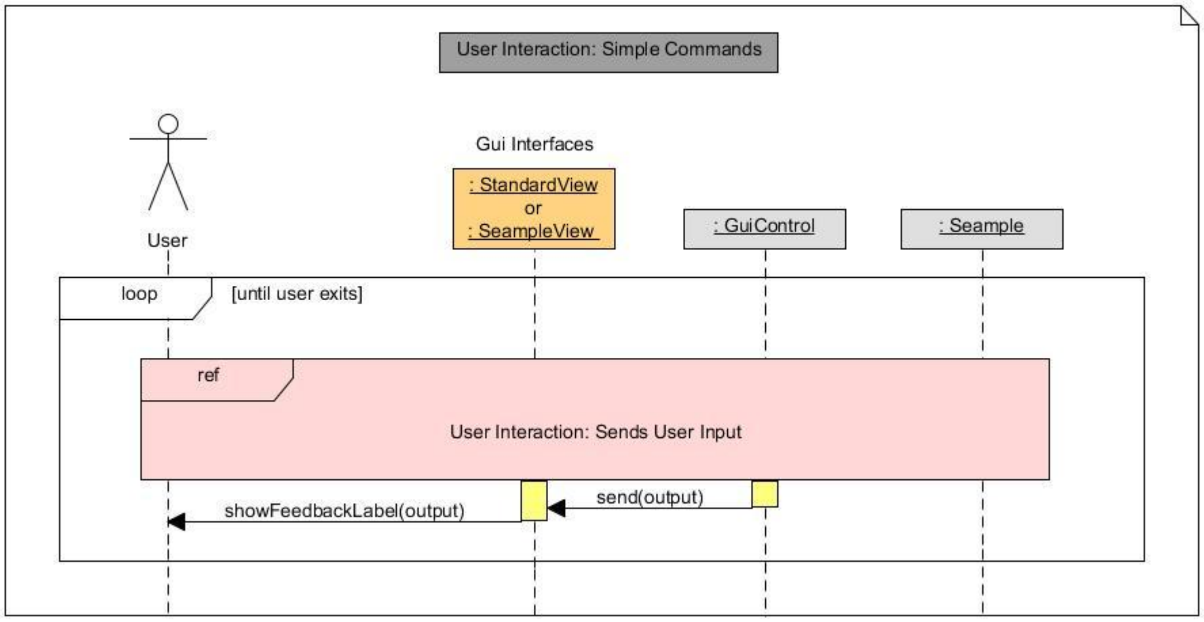
Sequence Diagrams



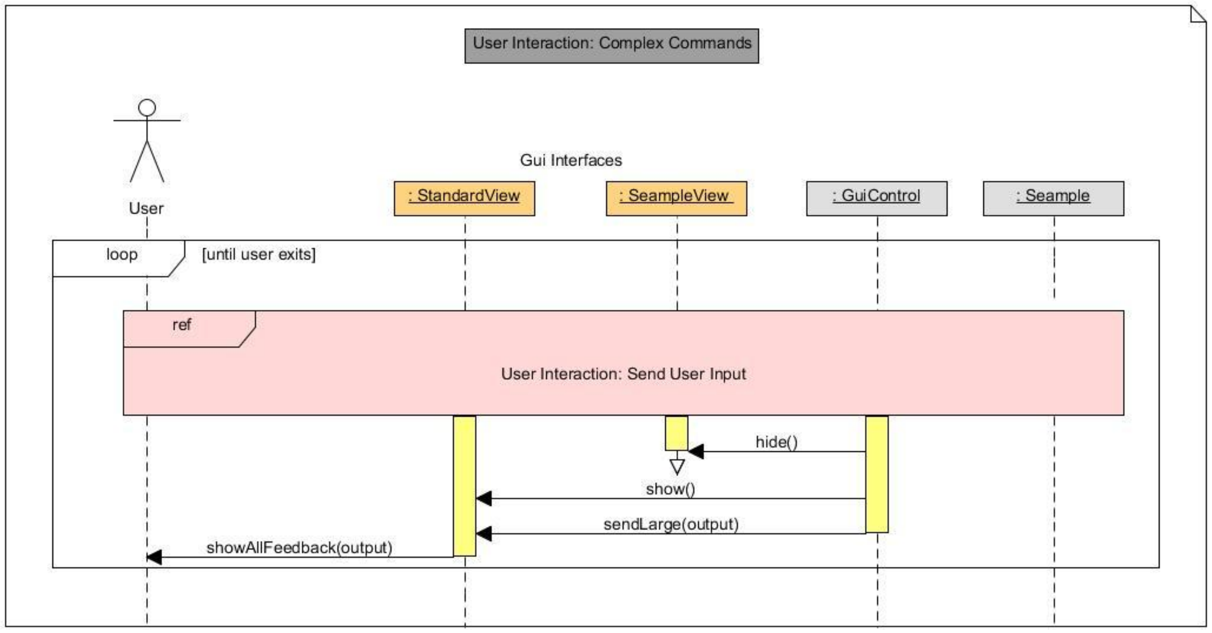

Intellisense.Check()
:GUI Control
:Seample
:Intellisense
:Action
Run(input)
check(query)
Trim(Query)
Tokenize(Query)
Tokens
Perform Operation
set entered fields
setAllStatusFlags
checkRequirements
smartAutoFill
Action
Feedback
Send(Output)

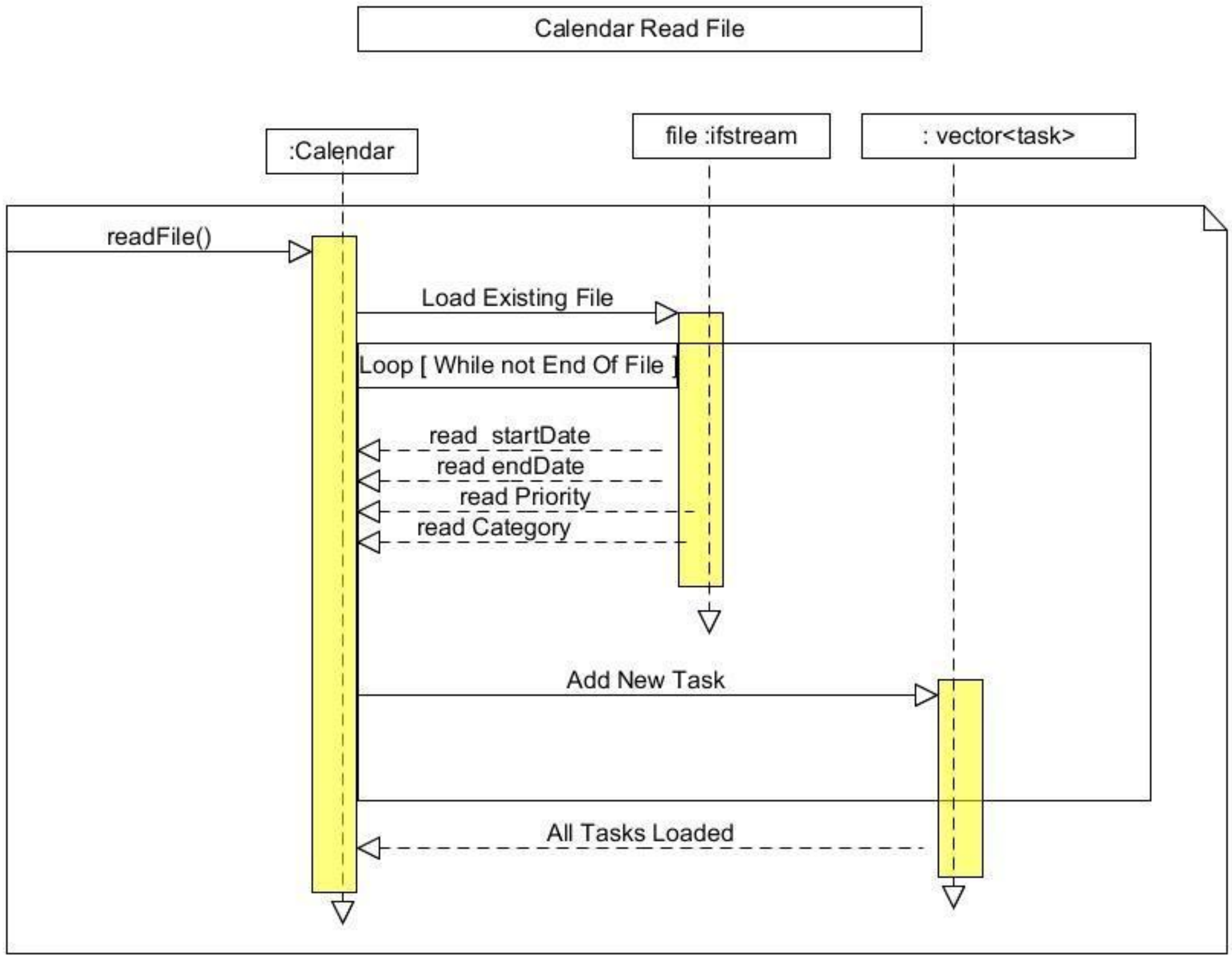

Scheduler Execute
:Scheduler
:task
:Calender
executeCommand(newAction)
newTask
setEventName
setCategory
setPriority
setStartDate
setEndDate
setID
newTask
addItem OR
deleteItem OR
editItem OR
findItem OR
display
Results
Feedback Message

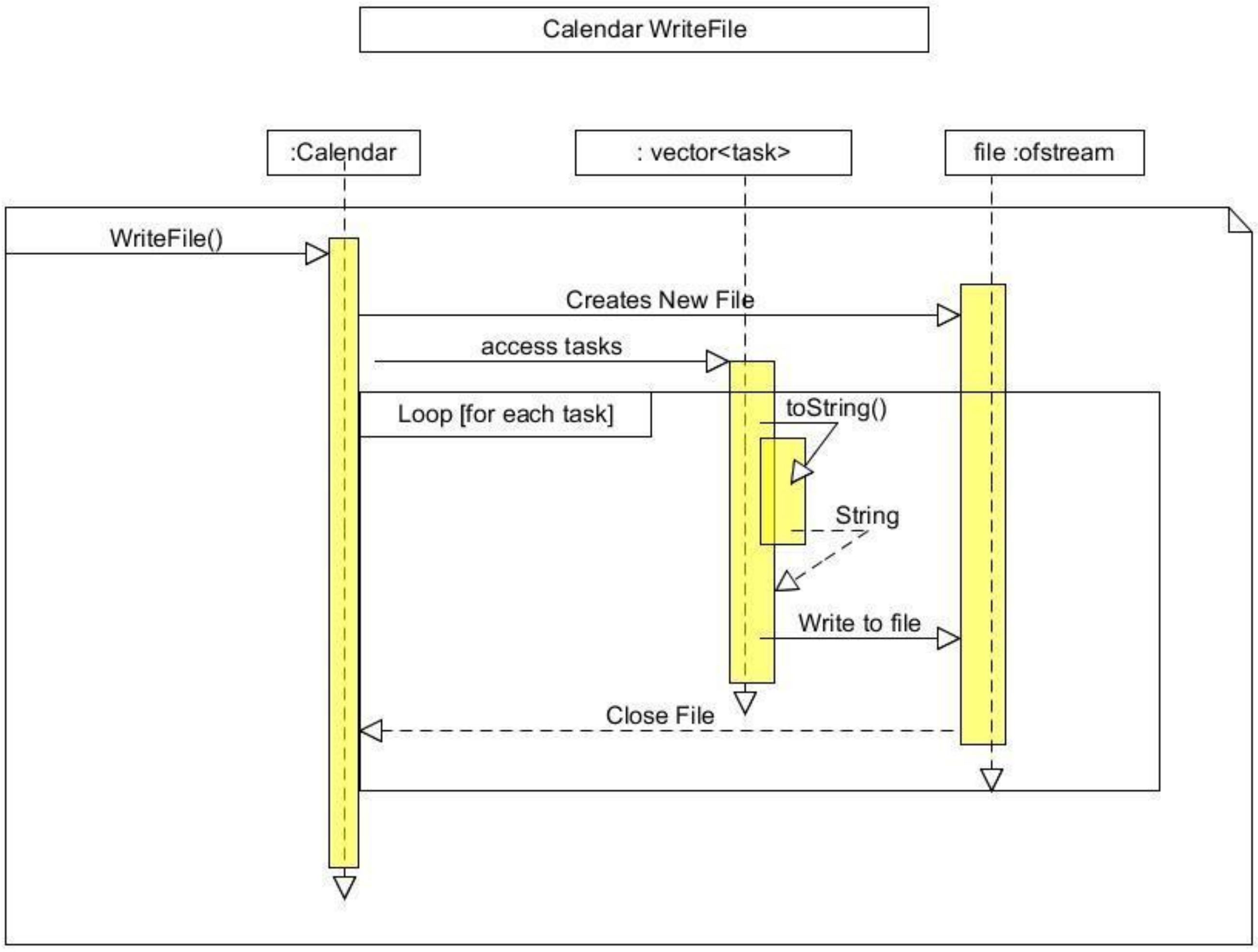
Note: Simple Commands refer to commands such as “add”, “delete” and “edit” where output consist of only one string



Note: Complex Commands refer to commands such as “find” and “display” where output can consist of more than one string due to multiple results.




Calendar Read File
:Calendar
file :ifstream
: vector<task>
readFile()
Load Existing File
Loop [While not End Of File]
read startDate
read endDate
read Priority
read Category
Add New Task
All Tasks Loaded


Calendar WriteFile
:Calendar
: vector<task>
file :ofstream
WriteFile()
Creates New File
access tasks
Loop [for each task]
toString()
String
Write to file
Close File