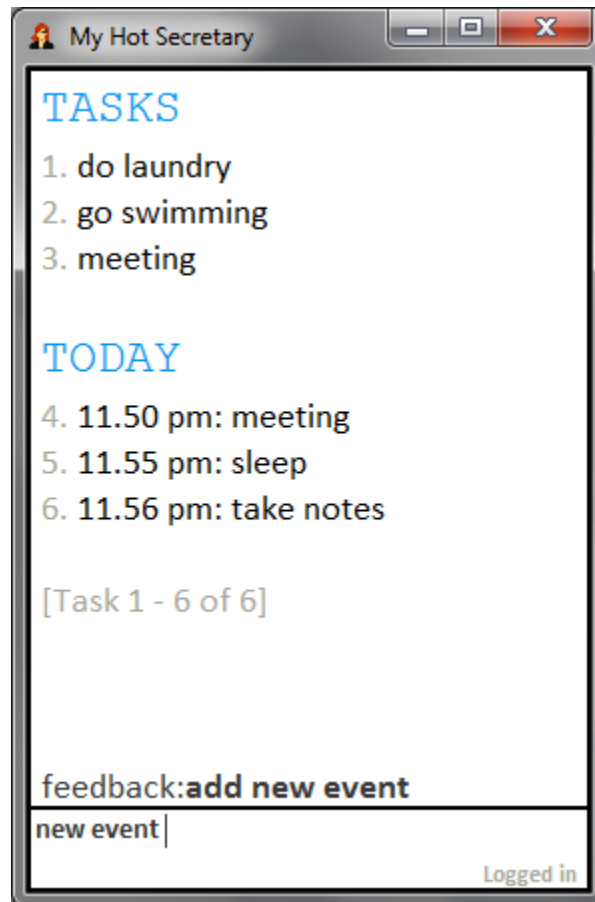


My Hot Secretary



Timothy Lim
Team leader



Shekhar B. R.
Test IC



John Wong
Integration IC



Kahou Cheong
Documentation IC

Contents

Installation Instructions	3
What is My Hot Secretary?	4
What is My Hot Secretary?	12
Who is this guide for?	12
Getting Started	12
Application Architecture.....	13
User Interface Package	14
Logic Package.....	16
Storage Package.....	19
Database.....	20
Database Creation.....	20
Database Overview	20
Synchronize	21
Synchronization Flow	22
Sync Task Operations Overview	22
Sync Logic.....	23
TaskValidator	23
Persistence Package	24
TaskLists.....	24
Persistence.Local Package	24
TaskRecordFile	24

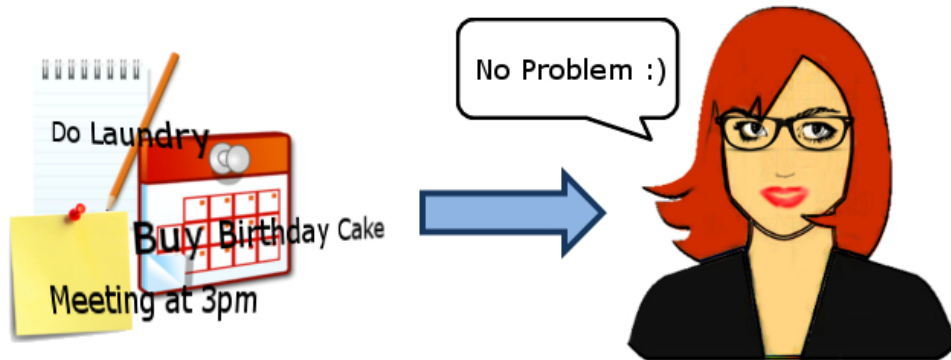
ConfigFile	24
Persistence.Remote Package.....	24
GoogleCalendar.....	24
Database Policies.....	25
Common Package	26
HtmlCreator.....	26
MhsLogger.....	26
MhsGson	26
DateTimeConverter.....	26
TaskTypeConverter	26
Logging.....	26
Testing	27
Integration	28
Build Automation.....	28
Appendices	28
Credits.....	28

Installation Instructions

My Hot Secretary requires no installation, just download and run!

What is My Hot Secretary?

My Hot Secretary is an application that makes task management a breeze, just pass all your tasks to My Hot Secretary and let her manage them for you.




With My Hot Secretary you can easily, create, retrieve, edit and delete all the tasks you need to keep track of. It's simple, here's how:


Adding a task

To add a task, just type the task name and the time.

For example, type “do laundry”, hit enter, and My Hot Secretary will record it for you

Time information can also be specified, in fact, My Hot Secretary organizes your tasks into three categories, depending on the time information specified:

 Floating Tasks: tasks with no time information specified

 Deadline Tasks: tasks with only one time specified



Timed Tasks: Tasks with a start and end time



Viewing Tasks

Once you've added your tasks, My Hot Secretary helps you organize it by category, time and date.

Viewing tasks in the Home Page

When you start up My Hot Secretary she will automatically show you the tasks you have for today as well as any floating tasks you may have.

Viewing tasks on a specific date or date range

To view tasks on a specific date, simply type "display (date)", for example, "display 21/11" will show you the tasks you have on for 21st November this year. You can also view tasks over a specific time range by typing "display (date) (date)"

Viewing tasks by category

Lastly, you can view all tasks by typing "(category)", for example, "deadline" will show your deadline tasks.



Editing Tasks

Have a meeting that changed timing, or need to rename a task? Not to worry, editing existing tasks in My Hot Secretary is easy.

Renaming a task

Type “rename (original task name) to (new name)”, for example “rename meeting to meeting with secretary” will edit the task with title “meeting” to “meeting with secretary”.

Editing a task's timing

Type “edit (original task name) to (new time)”, for example, “edit meeting to 3pm” will change the timing of the task with title “meeting” to 3pm.

Changing a task

Type “edit (original task name) to (new task name and date)”, for example, “edit meeting to movie 4pm” will change the remove the task meeting and instead add a task movie at 4pm.



Deleting Tasks

Created a task that you no longer need to do? You can easily remove it from My Hot Secretary. Just type “delete (task name)”, for example, “delete laundry” will delete the task with title “laundry”.



Everyone makes mistakes; My Hot Secretary understands that, which is why it is easy to undo any mistakes you may have made, simply type “undo” to undo your last action.



Need to access your tasks on the go? My Hot Secretary helps you sync all your tasks with your Google Calendar account, just type “sync”, type in your account details, and you’re good to go!



There’s no greater satisfaction then completing tasks, increase the satisfaction by telling My Hot Secretary about it, just type “done (task name)”, for example “done laundry” will mark the task with title “laundry” as done, this will also move the task into a separate Google Calendar: “Completed Tasks (MHS)”.

Command Reference

Single Date

General Format	Example	Remarks
dd/mm	15/06	Recognized as 15 th June this year, if that has already passed, it will be taken to mean 15 th June next year
dd mmm	15 Jun	Recognized as 15 th June this year, if that has already passed, it will be taken to mean 15 th June next year
dd/mm/yyyy	15/06/2012	Recognized as 15 th June 2012
dd mmm yyyy	15 Jun 2012	Recognized as 15 th June 2012

Date Range

General Format	Remarks
(1 st date) to (2 nd date)	Recognized as 12am of 1 st date to 23.59pm of 2 nd date

Single Time

General Format	Example	Remarks
hh.mm	15.30	Recognized as 3.30pm
hh.mm am/pm	3.30 pm	Recognized as 3.30pm

Time Range

General Format	Remarks
----------------	---------

(1 st time) to (2 nd time)	Time range recognized as 1 st time to 2 nd time
--	---

Adding a Task

Task Type	General Format	Example	Remarks
floating	(task name)	do laundry	Floating task with title “do laundry” will be added
deadline	(task name) (time)	do laundry 7pm	Adds a deadline task “do laundry” with time information recognized as “7pm”
timed	(task name) (1 st time) (2 nd time)	do laundry 7pm to 8pm	Adds a timed task “do laundry” with time information recognized as “7pm to 8pm”

Viewing Tasks

Command	Remarks
home	Displays floating tasks, deadline and timed tasks for today
today	Displays deadline and timed tasks for today
tomorrow	Displays deadline and times tasks for tomorrow
(category)	“Floating”: Displays all floating tasks “Deadline”: Displays all deadline tasks “Timed”: Displays all timed tasks
(date)	Displays all tasks from 12am to 11.59pm of the specified date
(1 st date) to (2 nd date)	Displays all tasks from 12am of the 1 st date to 11.59pm of the 2 nd date

Editing Tasks

General Format	Example	Remarks
rename (task name) to (new task name)	rename laundry to wash clothes	Renames task with title “laundry” to “wash clothes”
schedule (task name) to (time)	schedule meeting to 3pm	Schedules task with title “meeting” to be at time “3pm”
float (task name)	float meeting	Removes all time information for task with title “meeting” and converts it to a floating task

Marking Tasks

General Format	Example	Remarks
mark (task name)	mark laundry	Marks the task with title “laundry” as completed
unmark (task name)	unmark laundry	Marks the task with title “laundry” as uncompleted

Deleting Tasks

General Format	Example	Remarks
delete (task name)	delete laundry	Deletes the task with title “laundry”

Sync

Command	Remarks
sync	Syncs all tasks from 3 months before current date to 1 year after current date. My Hot Secretary will prompt for login information if you are not currently logged in
login	My Hot Secretary will prompt for Google account email and password
logout	Logs out of your Google account. Tasks will not be synced until next login.

Index

General Format	Example	Remarks
(command) (index)	delete 1	At any point in time, the index of a task is substitutable for the task name. So “delete 1” will delete the task at index 1. This feature is available for all commands.

My Hot Secretary Developer Guide

What is My Hot Secretary?

My Hot Secretary is a desktop application that helps users to keep track of their tasks.

Functionality summary:

1. Support for natural language commands
2. Command hinting
3. Creating tasks
4. Retrieving and viewing tasks
5. Editing existing tasks
6. Deleting tasks
7. Synchronization with user's Google Calendar
8. Index based commands

Who is this guide for?

This guide is for anyone who wants to work on My Hot Secretary to understand the code behind the application.

Getting Started

My Hot Secretary is hosted on Google Code, the source code can be found at:

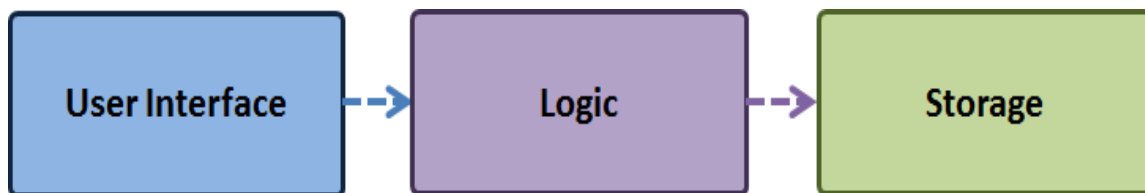
<http://code.google.com/p/cs2103aug12-t14-2j/>

[T14-2j][V0.4]

To work on My Hot Secretary, just retrieve the code from the above repo. We recommend using Mercurial as this is the Revision Control System we are currently using: <http://mercurial.selenic.com/>

Application Architecture

My Hot Secretary (MHS) utilizes an N-Tier architecture:



User Interface

The User Interface-tier handles user-program interaction; it provides the interface for the user to input commands and for the program to display output to the user. It is also responsible for updating the logic-tier with user input and updating what is displayed to the user at the appropriate times.

Logic

The Logic-tier is responsible for processing user commands. It decides what to do with the user input sent from the User Interface-tier. It is responsible for parsing the command and leading it to execution. It also decides what is to be displayed to the user.

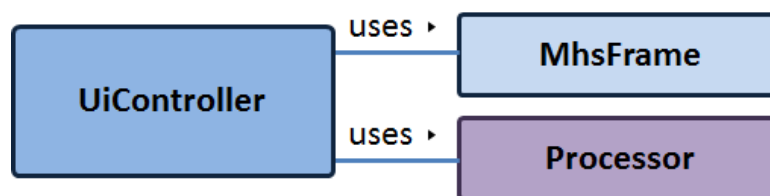
Storage

The Storage-tier provides the methods for the Logic-tier to store and retrieve data. It is responsible for storing data on the user's hard disk so that it is not lost when the user

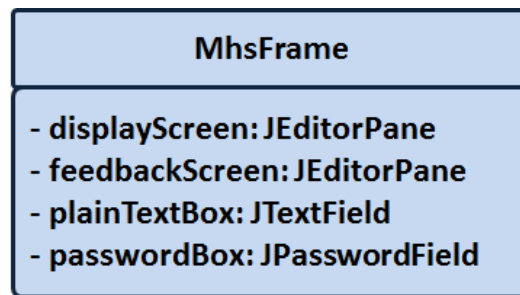
closes the program. It is also responsible for ensuring that tasks stored locally are synchronized with the user's Google Calendar if the user is logged in.

User Interface Package

UiController Class Diagram



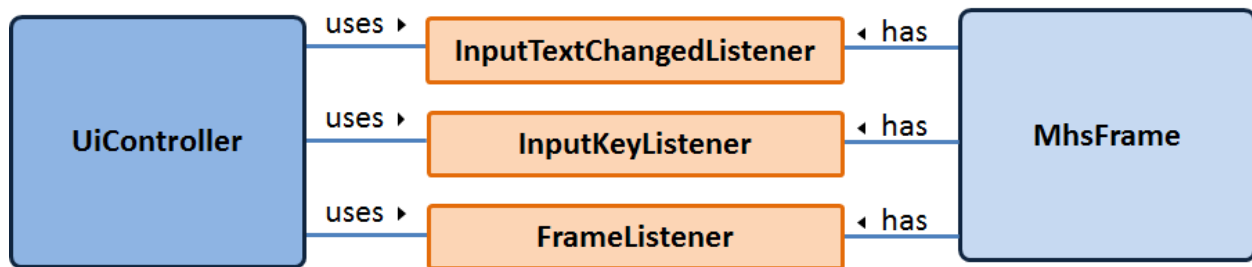
The responsibility of the UiController is to control interaction between the Processor and the MhsFrame. To understand this interaction, let us first look at the MhsFrame class.



MhsFrame provides the means for the user to interact with the application. It comprises an area to display output, an area to display one line feedback or confirmation messages, these two areas support the display of HTML formatted strings. There is also an area for the user to input commands; two types of input formats are available: password input, where user input is replaced with dots, and plain text input.

Observer Pattern

UiController makes use of the observer pattern in order to update the Processor when the user changes the state of MhsFrame:



InputTextChangedListener alerts **UiController** when the user changes the text in the input box of **MhsFrame**. **UiController** then updates **Processor** with the current user command.

InputKeyListener alerts the **UiController** when the user hits the enter key in **MhsFrame**, **UiController** then tells the **Processor** to execute the current command.

FrameListener alerts the **UiController** when the user resizes **MhsFrame**, **UiController** then takes note of this change and updates the maximum number of lines being displayed to the user.



Lastly, **ProcessorStateListener** alerts the **UiController** to update the display areas of **MhsFrame** when the **Processor** has processed a command and there is a change in the state of **Processor**.

Logic Package

Processor Class Diagram

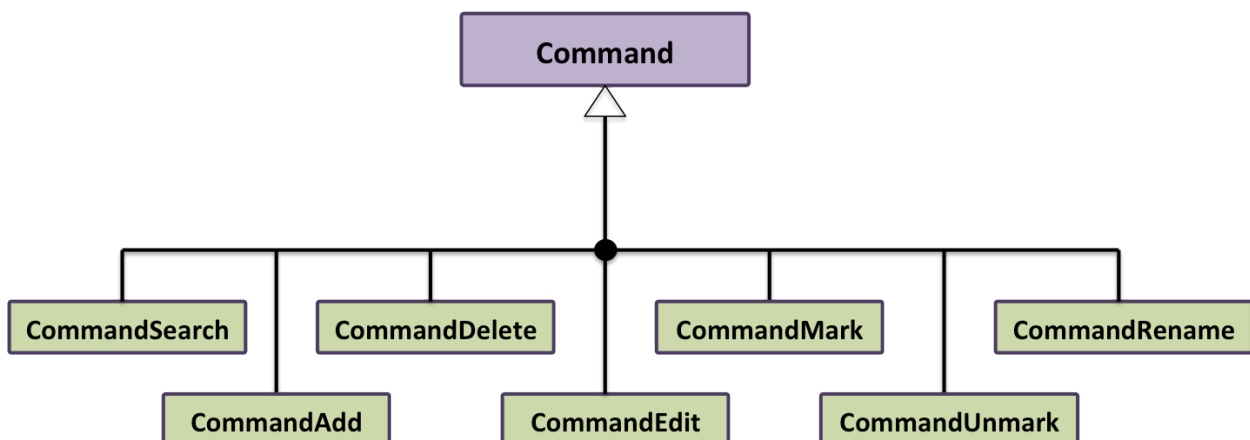
Processor class along with its associated Logic classes acts as the controller in MHS. It manages the flow of the user's command and leads it to execution. In this process it interfaces with the UI, Command Parser and Database classes.



Processor receives a command string from the UI. If this is a Google Login Command then the processor handles it internally and interacts with the Database to complete login and sync. Other commands are processed in associated processor classes and finally the command is completed using the database. The confirmation message flows back through all the associated classes and is updated on the user's screen.

Command Pattern

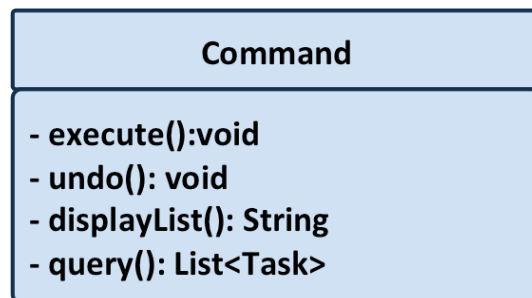
To ensure scalability and ease of development Command Pattern has been implemented in the processor and its associated Logic classes.



Flow of Logic in Command Pattern

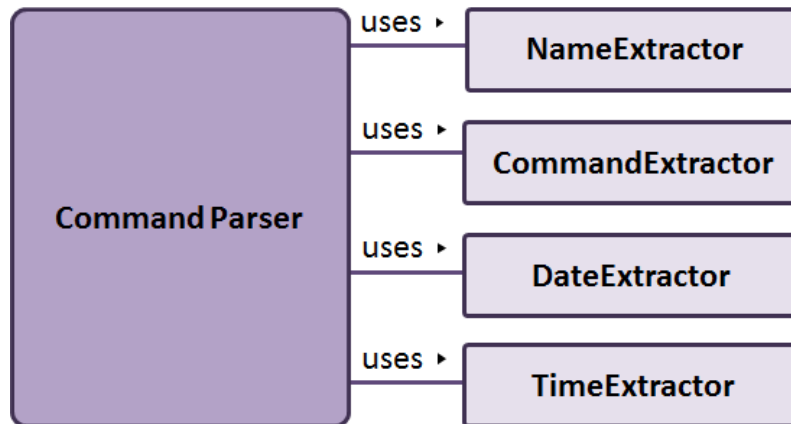
When a command is received by the processor which is not a login/sync command, the Command Pattern logic comes into play.

- Input string is passed to Command Parser for parsing which returns an object of CommandInfo.
- This object is passed to a method in **CommandCreator**. CommandCreator creates a new Command based on the input command type.
- Each subclass of **Command** has two methods execute() and undo() that handle the execution and undo operations respectively.



- Adding new Command Types
 - Add a subclass to command and implement all the abstract methods inherited from Command.
 - Add a case for that command type in Command Creator
- Reason for excluding a command queue from the command pattern
 - Since this software is being used only by one client at a time, there is no need to queue commands.

CommandParser Class Diagram



The responsibility of the CommandParser is to extract the task name, command keyword, date and time elements of a user's command. It accomplishes this through four independent extractors.

NameExtractor extracts the name of a task from the user's command by picking out all strings that are not dates, times, commands or special keywords (such as "to" or "on"). If a command contains quotation marks, for example "new task", the string within the quotation marks will be extracted as the task name.

CommandExtractor extracts command keywords, by comparing each string of a user's command with a command keyword list.

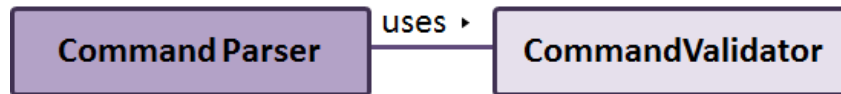
DateExtractor extracts strings that match various date formats.

TimeExtractor extracts strings that match various time formats.

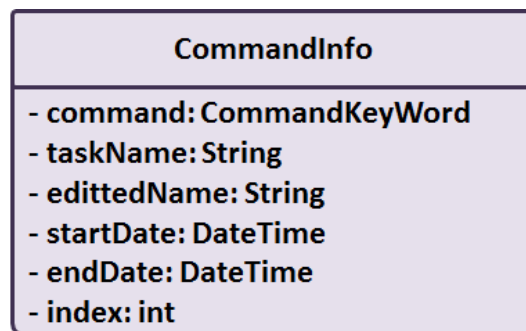
Besides the task name, command keyword, date and time elements, a fifth element index is also extracted; this is done within CommandParser.

[T14-2j][V0.4]

After all these elements are extracted, CommandValidator is used to combine all the elements together to form a logical command. For example, the date and time parameters extracted need to be combined for them to make sense.



Finally, CommandValidator creates a CommandInfo object which encapsulates the extracted and arranged data from the user's command:



Storage Package

The Storage package contains classes for the storage tier.

[T14-2j][V0.4]

Storage's main role is to perform CRUD operations on tasks and any other persistent data. It interfaces persistent data storage operations on local disk as well as remote storage (Google Calendar Service) and synchronizing.

Database

Database Creation

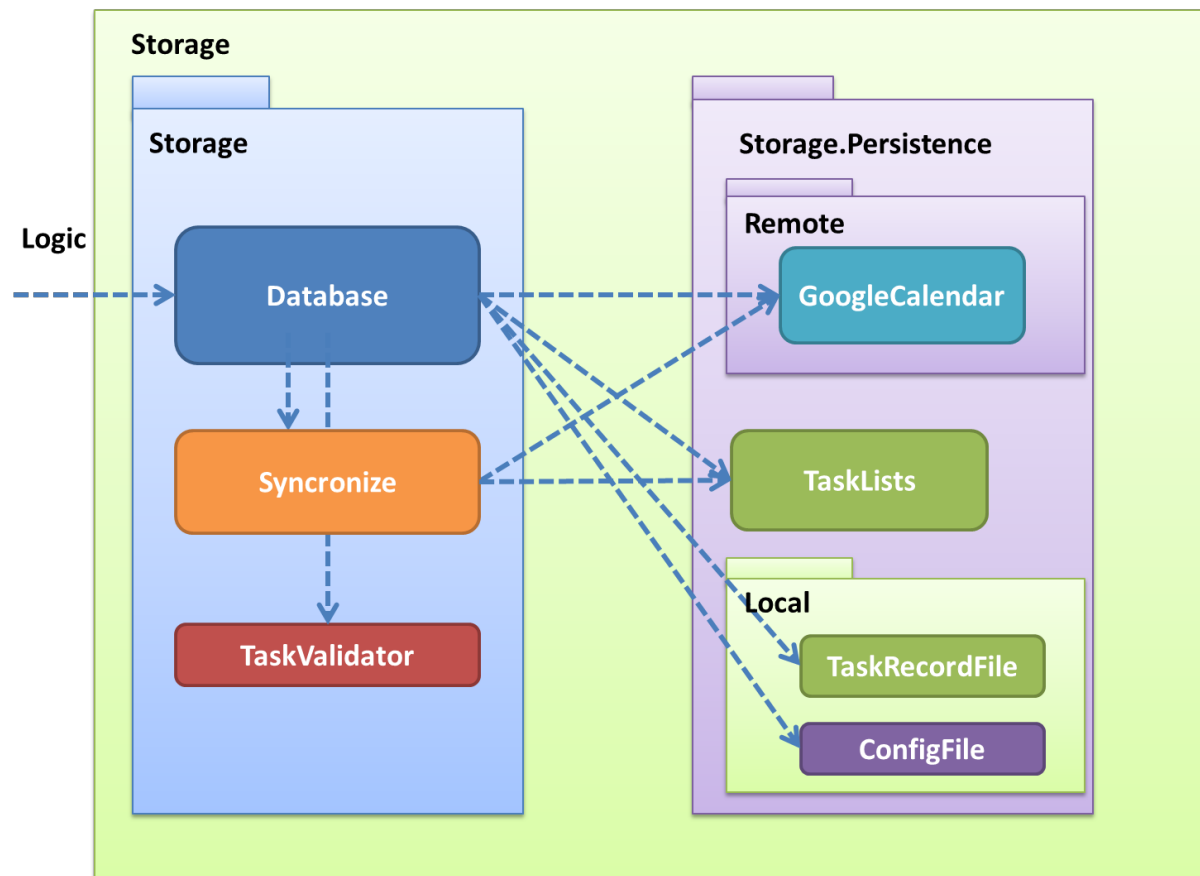
DatabaseFactory encapsulates **Database** Creation and its life-cycle handling to ensure a one-hot Singleton Database with user specific pre-set configurations.



Database Creation

Database Overview

Database is a singleton whose instance is used application-wide, modelling the **Storage Tier** following the n-tier architecture.



Storage Package Structure

Façade

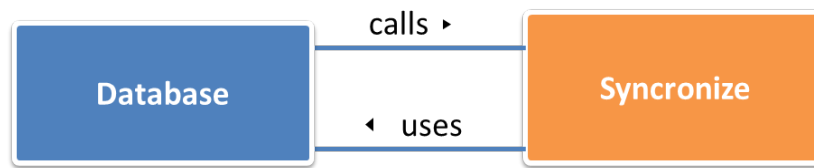
The **Database** Class abstracts packages **Persistence** and its sub-packages, **Local** and **Remote**. It acts as a façade for storage classes, interfacing operations on local and remote storage with a set of publicly available API.

Clients

The **Logic Tier** is using the Façade Pattern to access resources from the abstracted packages.

Synchronize

Synchronize handles synchronization operations between local and remote storage.



Database and Synchronize Association

Synchronize is called by Database to handle synchronization operations between local storage and remote storage.

Synchronization operations are done on a separate thread so that it appears as a background operation to the end-user. Remote storage operations, such as communicating with **GoogleCalendar**, will not block the application.

Once the sync task is done, Synchronize uses Database methods to update the local storage.

Syncronization Flow

The following Sync Task Operations are done through a single-threaded executor on a FIFO basis.

Sync Task Operations Overview

The following operations are scheduled only when user is authenticated with an active Google Calendar Service.

1. *SyncAllTasks* – Performs full push-pull sync between local storage and remote storage. This operation is scheduled during the start of MHS and whenever a 'Sync' command is issued.

2. *SyncPushTask* – Performs a push sync for a single task. This operation is scheduled whenever the user enters an operation that is supported with Google Calendar.
3. *SyncTimedPullTasks* – Performs a timed pull sync from remote storage. This operation is timed and scheduled to execute at a fixed rate.

Sync Logic

1. Synchronize pulls events from **GoogleCalendar** into a list (*googleCalendarEventsList*) and this is compared with a local task list (*gCalTaskList*).
2. Pull Sync is performed by iterating over *googleCalendarEventsList*.
 - a. Create new local task if no existing task with Google Calendar Unique Id (*iCalUid*) exists.
 - b. Update existing local task if Google calendar event is newer.
 - c. Delete existing local task if Google calendar event is deleted.
3. Push Sync is performed by iterating over *gCalTaskList*.
 - a. Create new *GoogleCalendarEntry* if no existing *iCalUid* exists.
 - b. Update existing *GoogleCalendarEntry* if local task is newer.
 - c. Delete existing *GoogleCalendarEntry* if local task is deleted.

FailSyncSilentlyPolicy: Only *UnknownHostException*s due to internet connectivity are thrown during sync operations, other *ServiceExceptions* are caught and silently handled.

TaskValidator

TaskValidator performs validation and check on Tasks' format (Timed, Deadline or Floating) and type (Sync, Un-Synced)

Persistence Package

TaskLists

TaskLists abstracts CRUD operations on tasks in all task lists (taskList and gCalTaskList).

Persistence.Local Package

TaskRecordFile

TaskRecordFile handles CRUD operations of tasks on the task record file (taskRecordFile.json) as well as file operations, such as creation, saving and loading of tasks. (From object to json and vice-versa).

Calls to save to File are synchronized to prevent File I/O exceptions.

ConfigFile

ConfigFile handles CRUD operations of configuration parameters on the configuration file (configFile.json) as well as file operations such as creation, saving and loading.

Persistence.Remote Package

GoogleCalendar

This class provides methods for the application to easily create, retrieve, update and delete events in a user's Google Calendar. It utilizes the Google Calendar Data API and java code provided from:

<http://code.google.com/p/gdata-java-client/downloads/list>

To use this class, the user's email and password is passed into the class, this email and password is used to retrieve an access token from Google's Calendar API. The retrieved access token then associates and authorizes all future method calls with the specified user's Google Calendar.

Database Policies

Database public methods that have return type of tasks / list of tasks returns clones of task(s) so that any modification on those task(s) do not change the original task.

Adding new task

Creates new task and assigns a new unique task id, created time and updated time.

InvalidTaskFormatException thrown when task specified does not meet the required format.

Updating Task

Updates existing task's editable fields and updates created time and updated time.

Non-editable fields : CalTaskId, TaskCreated, TaskLastSync are preserved so as not to interfere with synchronization.

TaskNotFoundException thrown when task specified by taskId does not exist.

InvalidTaskFormatException thrown when task specified does not meet the required format.

Deleting Task

Deletes existing task and updates created time and updated time.

Tasks are marked as deleted and not removed for synchronization purposes.

TaskNotFoundException thrown when task specified by taskId does not exist.

InvalidTaskFormatException thrown when task specified does not meet the required format.

Querying Task(s)

[T14-2j][V0.4]

IllegalArgumentException thrown when parameter is illegal.

TaskNotFoundException thrown when task specified by taskId does not exist.

Common Package

HtmlCreator

HtmlCreator contains the constants and methods to create basic HTML for MHS

MhsLogger

MhsLogger adopts a singleton pattern to creating single instance of logger (java.util.logger) for application-wide usage.

MhsGson

MhsGson is a singleton class that creates and uses a single instance of *Gson* configured to work with Tasks (Timed, Deadline, and Floating) and JodaTime DateTime objects.

DateTimeConverter

Gson Converter Class for Joda DateTime to json and vice-versa.

TaskTypeConverter

Gson Converter Class MHS Tasks to json and vice-versa.

Logging

My Hot Secretary implements logging for debugging purposes.

Log Levels Usage

Log Level	Purpose
INFO	<ul style="list-style-type: none">Application messages at key points

FINER	<ul style="list-style-type: none">• Tracing messages for debugging purposes• Tracking messages for exceptions and throws
--------------	---

Testing

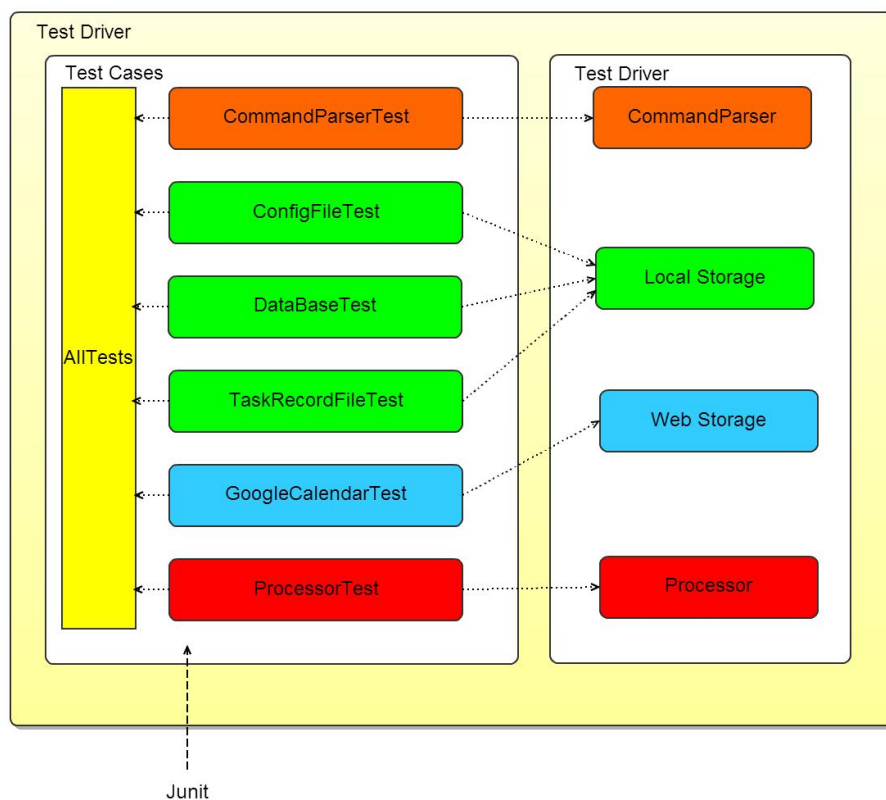


Figure 1 : Testing Package Diagram

Tests are contained with the Test Package - mhs.test.

1. Manual Testing
2. System Testing
3. Unit Testing

a. Test Suite - All Tests

- Individual junit tests

Note: Database Test may have failed test cases if anti-virus is enabled since it is dependent on I/O operations, disable anti-virus before running tests.

Integration

New features should be implemented in a separate branch. The new branch should be merged with the trunk only after it is certain that the system with the new feature is stable. Assessment of stability is done through the testing process documented below.

Build Automation

Build automation is achieved using Apache Ant.

The following are built automatically:

- Jar distributable
- Javadocs
- Combined dependencies into a single jar file

Appendices

- MHS Javadoc - <http://cs2103aug12-t14-2j.googlecode.com/hg/doc/index.html>

Credits

1. Google Calendar API(GData) and java client library:

<http://code.google.com/p/gdata-java-client/downloads/list>

[T14-2j][V0.4]

2. Date Time library : <http://joda-time.sourceforge.net/>
3. GSON: <http://code.google.com/p/google-gson/>