Taskly

USER MANUAL REPORT

Command-Line TMS

Authored by:
- Dominicus Dylan HARYOTO
- Bohan YANG
- Qinye ZHANG
- Shusen ZHENG

1. Introduction

Welcome to Taskly, a command-line task management system (TMS) that streamlines task management for users. Whether it's managing simple tasks or complex ones, Taskly offers a streamlined solution to help you stay on top of your responsibilities.

In Taskly, a task is either primitive or composite. Primitive tasks are the fundamental units of work, while composite tasks are collections of other tasks, which could be either primitive or composite.

Primitive tasks are designated by a unique name, followed by a description, a duration, and a set of prerequisites. Composite tasks, on the other hand, are designated by a unique name, a description, and a list of subtasks.

Apart from primitive and composite tasks, Taskly also has criteria, which are a group of tasks selected based on the given condition. It is designated by a unique name, a property, a condition operator, and a value.

Through this user report manual, we will guide you on how to define, query, and modify tasks using Taskly.

2. Getting Started

Before starting with Taskly, it's important to have the IntelliJ IDE installed on your computer, as Taskly runs in this environment. If you haven't already installed IntelliJ, please follow the steps below:

1. Visit the official JetBrains website at https://www.jetbrains.com/idea/download/.
2. Choose the version suitable for your operating system (Windows, macOS, or Linux).
3. Download either the Ultimate (paid, with a free trial) or the Community (free) version, depending on your needs.
4. Once the download is complete, install IntelliJ IDEA by following the on-screen instructions.

After you have successfully installed IntelliJ, you'll be ready to start using Taskly.

Extracting the Taskly Project

1. Locate the downloaded Taskly project zip file on your computer.
2. Right-click on the zip file and select "Extract All..." (Windows) or double-click the zip file (macOS/Linux).
3. Choose a location on your computer where you want to extract the Taskly project folder. Make sure it's a location that you can easily access, such as your Desktop or Documents folder.

Opening the Taskly Project in IntelliJ

1. Open IntelliJ IDEA.
2. On the IntelliJ welcome screen, click on "Open".
3. Navigate to the location where you extracted the Taskly project folder.

4. Select the Taskly project folder and click "OK". IntelliJ IDEA will now load the Taskly project.

Navigating to the "Source Code" Directory

1. In IntelliJ IDEA, with the Taskly project open, you'll see the project explorer on the left side. This displays a tree structure of the project files and directories.
2. Click the arrow next to the Taskly project name to expand the project directories.
3. Navigate through the directories by clicking the arrows beside each directory name. Follow the path to the "Source Code" directory.

Running the Applicationjava File

1. Inside the "Source Code" directory, locate the Application.java file. This is the main entry point of your Taskly application.
2. Click on the Application.java file to open it in the editor.
3. Look towards the top right of the IntelliJ IDEA interface, you'll see a green "Run" button (represented by a green play icon). Click on this button.
4. IntelliJ IDEA will then compile and run the application.

IntelliJ IDEA will open a Run window at the bottom of the IDE, displaying the output of the program. If the application starts successfully, you will see a message indicating that the application is running.

3. Commands

This manual provides detailed instructions on how to use each command, including its purpose, syntax, parameters, and examples. If you wish to test the TMS, you can input the examples provided for each command into the designated black box.

CreatePrimitiveTask

This method creates a new primitive task and adds it to the database. It validates the name, description, duration, and prerequisite(s) of the task before creating it. The name must contain only English letters and digits, cannot start with a digit, and must be 8 characters or less. The description may only contain English letters, digits, and the hyphen character (-). The duration must be a positive real number. The prerequisite(s) must already exist in either the primitiveTask or compositeTask HashMaps.

Command → `CreatePrimitiveTask name description duration prerequisites`
Example → `CreatePrimitiveTask task1 boil-water 0.3 ,`

```
TaskManagementSystem@COMP2021 ~ % CreatePrimitiveTask task1 boil-water 0.3 ,
task1 has been successfully created!
TaskManagementSystem@COMP2021 ~ % CreatePrimitiveTask task2 boil-egg 0.7 task1
task2 has been successfully created!
TaskManagementSystem@COMP2021 ~ %
```

CreateCompositeTask

This method creates a new composite task and adds it to the database. It validates the name, description, and subtasks of the task before creating it. The name must contain only English letters and digits, cannot start with a digit, and must be 8 characters or less. The description may only contain English letters, digits, and the hyphen character (-). The k subtasks $(k \geq 2)$ must already exist in either the primitiveTask or compositeTask HashMaps.

Command → `CreateCompositeTask name0 description name1,name2,...,namek`

Example → `CreateCompositeTask comtask1 make-coffee primitive1,composite0`

```
TaskManagementSystem@COMP2021 ~ % CreateCompositeTask comtask1 cook-egg task1,task2
comtask1 has been successfully created!
TaskManagementSystem@COMP2021 ~ %
```

DeleteTask

This method tries to delete a task, either primitive or composite, by its name. If the task is a prerequisite for another task or if it is a subtask of a composite task, it will not be deleted and an error message will be printed. If the task to be deleted is a composite task, this method will also try to delete all its subtasks recursively. If any subtask cannot be deleted (because it's a prerequisite for another task), the deletion of the composite task will be aborted and an error message will be printed. If the task does not exist, an error message will be printed.

Command → `DeleteTask name`

Example → `DeleteTask task1`

```
TaskManagementSystem@COMP2021 ~ % DeleteTask task2
task2 has been successfully deleted!
TaskManagementSystem@COMP2021 ~ %
```

ChangeTask

This method changes a specified property of a task (either primitive or composite) to a new value. The properties that can be changed are: "name", "description", "duration" (for primitive tasks), “prerequisite” (for primitive tasks), and "subtasks" (for composite tasks). If the task or property does not exist, or if the new name already exists as a task, an error message will be printed. If the new duration is not a positive real number, an error message will be printed. Name changes are also propagated to all tasks where the task being modified is a prerequisite or a subtask.

Command → `ChangeTask name property newValue`

Example → `ChangeTask task1 duration 0.5`

```
TaskManagementSystem@COMP2021 ~ % ChangeTask task1 duration 0.5
task1's duration has been successfully changed!
TaskManagementSystem@COMP2021 ~ %
```

PrintTask

This method prints the name, description, duration (for primitive tasks), prerequisites (for primitive tasks), or subtasks (for composite tasks) of a specified task. If the task does not exist, an error message will be printed.

Command → `PrintTask name`

Example → `PrintTask task1`

```
TaskManagementSystem@COMP2021 ~ % PrintTask task1
task1 -> description: boil-water, duration: 0.5, prerequisite(s): null
TaskManagementSystem@COMP2021 ~ %
```

PrintAllTasks

This method prints the information about all tasks, both simple and composite, currently in the task management system. For each task, it prints its name, description, duration (for primitive tasks), prerequisites (for primitive tasks), or subtasks (for composite tasks). The tasks are grouped by their type (simple or composite) and each group is preceded by a header.

Command → `PrintAllTasks`

```
TaskManagementSystem@COMP2021 ~ % PrintAllTasks
Here is the list of Simple Task(s)
task1 -> description: boil-water, duration: 0.5, prerequisite(s): null

Here is the list of Composite Task(s)
comtask1 -> description: cook-egg, subtasks: task1 task2
TaskManagementSystem@COMP2021 ~ %
```

ReportDuration

This method reports the duration of a task given its name. If the task is a primitive task, it prints out the duration. If it is a composite task, it prints out the least time required to complete the task. If the task does not exist, it outputs a message indicating that the task was not found.

Command → `ReportDuration name`

Example → `ReportDuration task1`

```
TaskManagementSystem@COMP2021 ~ % ReportDuration task1
0.5
TaskManagementSystem@COMP2021 ~ %
```

ReportEarliestFinishTime

This method reports the earliest finish time of a task given its name. It prints out the minimum time required to complete the task.

Command → `ReportEarliestFinishTime name`

Example → `ReportEarliestFinishTime task1`

```
TaskManagementSystem@COMP2021 ~ % ReportEarliestFinishTime comtask1
0.5
TaskManagementSystem@COMP2021 ~ %
```

DefineBasicCriterion

This method defines a new basic criterion by specifying a name, property, operation (op), and value. The criterion can be applied to tasks to filter or sort them. If a criterion or task with the provided name already exists, an error message is printed. The name must be at most 8 characters long, must not start with a digit, and can only contain English letters and digits. The property must be one of: "name", "description", "prerequisites", "subtasks", or "duration". The operation must be "contains" for the first four properties and a comparison operator for "duration". The value must be a positive

real number for "duration". The new criterion is saved in the criterion list of the task management system.

Command → `DefineBasicCriterion name1 property op value`

Example → `DefineBasicCriterion criterion1 duration > 0.1`

```
TaskManagementSystem@COMP2021 ~ % DefineBasicCriterion cri1 description contains boil
cri1 has been successfully defined.
TaskManagementSystem@COMP2021 ~ %
```

DefineNegatedCriterion

This method defines a new negated criterion based on an existing criterion. The new criterion has the opposite effect of the original criterion. If a criterion with the provided new name already exists, or if the original criterion does not exist, an error message is printed. The name of the new criterion must be at most 8 characters long, must not start with a digit, and can only contain English letters and digits. The operations are negated as follows: ">" becomes "<=", "<" becomes ">=", ">=" becomes "<", "<=" becomes ">", "==" becomes "!=", "||" becomes "&&", "&&" becomes "||", "contains" becomes "not-contains", "not-contains" becomes "contains", "IsPrimitive" becomes "IsComposite", "IsComposite" becomes "IsPrimitive". The new negated criterion is saved in the criterion list of the task management system.

Command → `DefineNegatedCriterion name1 name2`

Example → `DefineNegatedCriterion criterion2 criterion1`

```
TaskManagementSystem@COMP2021 ~ % DefineNegatedCriterion cri2 cri1
cri2 has been successfully defined.
TaskManagementSystem@COMP2021 ~ %
```

DefineBinaryCriterion

This method defines a new binary criterion based on two existing criteria. The new criterion is the result of applying the given operation to the two original criteria. If a criterion with the provided new name already exists, or if either of the original criteria

does not exist, an error message is printed. The name of the new criterion must be at most 8 characters long, must not start with a digit, and can only contain English letters and digits. If the new name is "IsPrimitive" or "IsCompos", the binary criterion is created by combining one of the original criteria with a built-in criterion. The properties, operations, and values of the binary criterion are derived from those of the original criteria and the operation.

Command → `DefineBinaryCriterion name1 name2 logicOp name3`

Example → `DefineBinaryCriterion criterion3 criterion1 || criterion2`

```
TaskManagementSystem@COMP2021 ~ % DefineBinaryCriterion cri3 cri1 || cri2
cri3 has been successfully defined.
TaskManagementSystem@COMP2021 ~ %
```

PrintAllCriteria

This method retrieves all criteria from the criterion list and prints them. For each criterion, the name, property, value, and operation are printed. If the name of a criterion is "IsPrimitive" or "IsCompos", only the name is printed. Otherwise, the name is printed followed by the property, value, and operation of the criterion.

Command → `PrintAllCriteria`

```
TaskManagementSystem@COMP2021 ~ % PrintAllCriteria
Here is the list of Criteria(s)
IsPrimitive
cri2 -> property: description, value: boil, op: not-contains
cri3 -> property: description || description, value: boil || boil, op: contains || not-contains
cri1 -> property: description, value: boil, op: contains
TaskManagementSystem@COMP2021 ~ %
```

Search

This method searches for tasks in the task management system that meet the criterion specified by the given name. If the criterion does not exist, an error message is printed. If the criterion is a basic criterion, tasks that meet this criterion are searched for and

printed. If the criterion is a binary criterion, tasks that meet both of the sub-criteria are searched for and printed. If no tasks meet the criterion, a message is printed to indicate this.

Command → `Search name`

Example → `Search criterion3`

```
TaskManagementSystem@COMP2021 ~ % Search cri3
Here is the list of task(s) of cri3
task1
TaskManagementSystem@COMP2021 ~ %
```

Store

This method stores the current tasks and criteria in a file. This method stores the current state of the primitive tasks, composite tasks, and criteria in a file with the specified path. The tasks and criteria are formatted in a specific way for easy retrieval:

- Primitive tasks are denoted with `$` before and after their details.
- Composite tasks are denoted with `%` before and after their details.
- Criteria are denoted with `^` before their details.

Each detail of the tasks and criteria is written on a new line, and multiple values are comma-separated. If the method succeeds, a success message is printed to the console. If an exception occurs during the write operation, the stack trace is printed to the console.

Command → `Store path`

Example → `Store d:\tasks.data`

```
TaskManagementSystem@COMP2021 ~ % Store d:\tasks.data
Tasks and criteria stored successfully.
TaskManagementSystem@COMP2021 ~ %
```

Load

This method reads a file with the specified path and loads the primitive tasks, composite tasks, and criteria from it. The file should be formatted in a specific way for correct parsing:

- Primitive tasks are denoted with `$` before and after their details.
- Composite tasks are denoted with `%` before and after their details.
- Criteria are denoted with `^` before their details.

Each detail of the tasks and criteria is read from a new line, and multiple values are comma-separated. If the method succeeds, a success message is printed to the console. If a `FileNotFoundException` occurs, an error message and the stack trace are printed to the console. If another `IOException` occurs, a different error message and the stack trace are printed to the console.

Command → `Load path`

Example → `Load d:\tasks.data`

```
TaskManagementSystem@COMP2021 ~ % Load d:\tasks.data
Tasks and criteria read successfully.
TaskManagementSystem@COMP2021 ~ %
```

Quit

This command is used to terminate the current execution of the Task Management System (TMS).

Command → `Quit`

```
TaskManagementSystem@COMP2021 ~ % Quit
Thank you for using the TMS. See you again!

Process finished with exit code 0
```

Undo

This method allows the user to undo their most recent action. If there are no actions to undo (e.g., at the start of the program or after a clear operation), the method will not perform any operations.

```
Command → undo
```

Redo

This method allows the user to redo an action that they've just undone. If there's no action to redo (e.g., no undo operation has been performed before or after a clear operation), the method will not perform any operations

```
Command → redo
```

4. Troubleshooting

Error in command syntax

Make sure that the command syntax is correct. Each command should have the correct number of arguments as defined in the switch case. If the number of arguments isn't correct, an error message will be displayed.

The task name doesn't exist

When performing operations on tasks such as DeleteTask, ChangeTask, PrintTask, ReportDuration, and ReportEarliestFinishTime, make sure that the task name exists in the system. Otherwise, the program won't find the task and will likely throw an error or exception.

Incorrect file path for Store and Load

When storing or loading tasks, make sure the file path is correct. The program may throw a FileNotFoundException if the path is invalid or the file doesn't exist.

Invalid operation for ChangeTask

The ChangeTask operation may have specific properties that it can change. If an invalid property is given, the operation might fail.

Issues with undo/redo operations

The undo and redo operations depend on how they are implemented in the TMS class. If there's a logic error in these implementations, these operations might not work as expected.

5. Additional

Documentation & References

Java Documentation → Comprehensive resource for all things Java. You can find detailed explanations and usage examples for most classes and methods. Visit Java Documentation

Stack Overflow → A community-driven platform where developers can ask questions and share their programming knowledge. Use the search function to find previous questions and solutions about Java. Visit Stack Overflow

FAQs & Support

For specific issues related to the Taskly Task Management System, you can reach out to our technical support team. Call us at +852 64753741 or email us at dylan.haryoto@connect.polyu.hk.