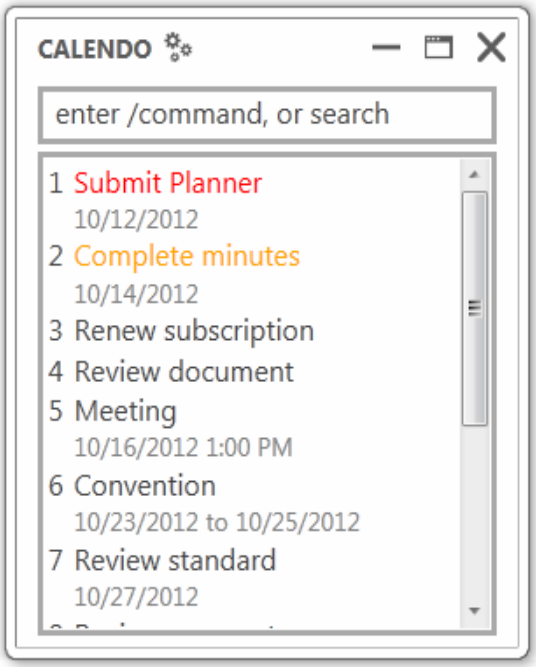
Calendo



			
Nicholas Kwan	**Rahij Ramsharan**	**Jerome Cheng**	**Pallav Shinghal**
Team Leader, Data Storage Developer, Specification Writer	Google Calendar Integration, Network Developer	User Interface Developer, System Tester	User Input Processing, Component Tester

User Guide

1. Introduction

1.1. About Calendo

Calendo is a task planner that allows you to manage tasks efficiently in an easy way. It integrates with Google Calendar so that tasks could be retrieved online from any computer. Its offline mode ensures that you can stay in check even without access to the Internet.

1.2. System requirements

Calendo is compatible with Windows XP, Windows Vista, Windows 7 and Windows 8.
Some features such as Google Calendar Integration may require Internet access and a Google account.

2. Features

2.1. User Interface

Calendo's user interface is designed to be easy to use. All commands can be performed by entering them into the command or search box. A command is of the form **/command** [parameters].

Tasks are displayed in the task list panel, and are sorted by importance. The importance of the task is determined by its time and date.

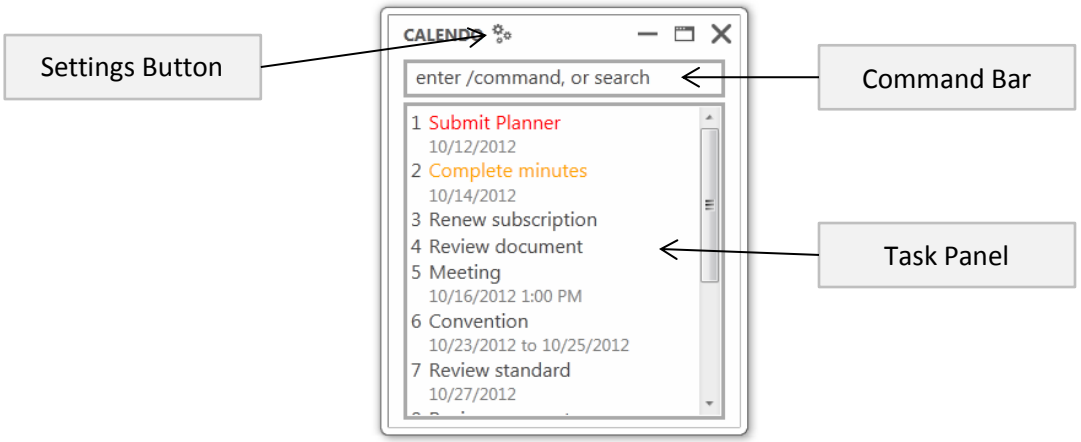


Figure 2.1: User Interface

2.2. Task Notification

There are 3 types of tasks: Floating, Deadline, and Timed. Calendo treats each task differently.

2.2.1. Floating Tasks

Floating tasks are tasks without any due date or time interval specified. Calendo will not provide any special notifications for floating tasks.

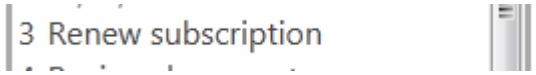


Figure 2.2: Floating Task

2.2.2. Deadline Tasks

Deadline tasks are tasks that have a specific due date or time.

Deadline tasks would be notified 24 hours before it is due. When this occurs, the task is highlighted in orange in the task list. When the task is overdue, the task would be highlighted in red, and Calendo would display a prompt to the user.

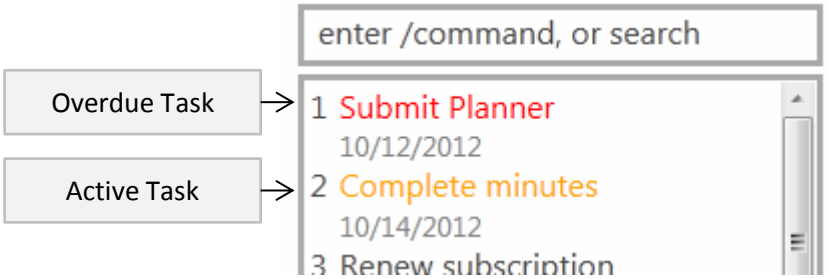


Figure 2.3: Task notification for Deadline Tasks

2.2.3. Timed Tasks

Timed tasks are tasks spanning over a period of time.

Timed tasks are similar to deadline tasks in that Calendo will provide a notification 24 hours before the timed task begins. When the time interval of the timed task has lapsed, the task will be marked as overdue.

Figure 2.4: Timed Task

2.3. Google Calendar Integration

Google Calendar integration allows you to modify tasks online via Google Calendar, and have them synchronized with Calendo so that you can access the tasks from within the application.

2.4. Auto-Suggest

The Auto-Suggest feature is designed to give immediate feedback as to what each command does. Typing a "/" immediately activates the feature, displaying a list of available commands along with their description.

3. Quick start

3.1. Setting up Calendo

1. Launch the application.
2. If you have a Google account, you can specify your account settings. You may skip this step and perform it later; however certain features such as Google Calendar Integration will be disabled.
3. Calendo is now ready for use.

3.2. Adding tasks

Tasks can be added by entering **/add** command into the search box, followed by the description of the task. Tasks can be assigned a date and time to the task using the **/date** and **/time** parameters respectively.
/add [Description] **/date** [Day/Month/Year] **/time** [Hour:Minutes]

The date and time parameters are optional.

> **Example**: To create a floating task
> **/add** Prepare chicken
>
> This adds "Prepare chicken" to the list of tasks. As no date and time is specified, it is treated as a floating task and would remain on the list of tasks until it is removed.

> **Example**: To create a deadline task at a certain date
> **/add** Meeting **/date** 18/10
>
> This adds "Meeting" to the list of tasks, and will notify the user of the task nearing the 18th of October. For more information on task notifications, refer to 2.2 Task Notification.
> The date is in the form of Day/Month/Year. Specifying the year is optional, and if it is omitted Calendo will assume that the event occurs on the closest matching date in the future.

Example: To create a deadline task at a certain time
/add Meeting **/time** 11:00 AM

Calendo will notify the user of the task at 11:00 AM on the same day. If the current time is past 11:00 AM, Calendo will assume it is for the following day.
If AM/PM is not specified, it is assumed that the time follows the 24-hour clock.

Example: To create a deadline task on a certain date and time
/add Meeting **/date** 18/9 **/time** 11:00 AM

Calendo will notify the user of the task on 18th September at 11:00 AM.

Example: To create a timed task on a certain time range
/add Exhibition **/date** 19/9-21/9

Calendo will notify the user of the task near the 19th to 21st of September.

3.3. Removing tasks

To remove the task, use the **/remove** command and specify the task number that appears to the left of the task. Calendo does not automatically remove tasks; users should remove them when they are completed.
/remove [Task number]

Example: To remove the first task
/remove 1

3.4. Modifying tasks

To modify a task, use the **/change** command and specify the task number. Immediately following the command, type the new description, the time and date and press enter.
/change [Task Number] [Description] **/date** [Day/Month/Year] **/time** [Hour:Minutes]

The description, date, and time parameters are optional. If a parameter is omitted, it will remain unchanged.

Example: To change the description of the first task on the list
/change 1 Flight to Hong Kong

Example: To change the time of the first task on the list
/change 1 **/time** 11:00 AM

If the task is originally a floating task, it would be converted to a deadline task.

Example: To convert a task from a deadline task to a floating task
/change 1 **/date /time**

The date and time parameter is provided but their parameter values are omitted.

Example: To change the date of the first task
/change 1 **/date** 19/11

Example: To change multiple parameters
/change 1 Flight to Hong Kong **/date** 19/11 **/time** 11:00 AM

3.5. Searching for tasks

To search for a particular task, simply type the search query into the Command Bar. The results should appear immediately. Tasks can be searched either by description or by due date.

Search queries must not begin with a "/". In the event you wish to search with the "/" included, add a space to the front of the query.

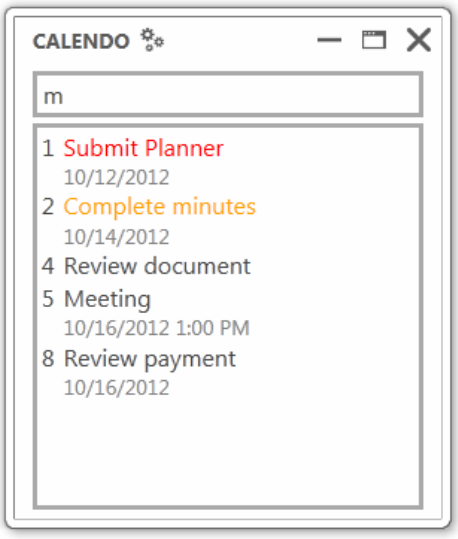


Figure 3.1: Searching for tasks

3.6. Undo an operation

To undo an operation performed by a command, use the **/undo** command. To undo an operation performed by the undo command, use the **/redo** command. Both commands do not have any parameters.

Example: To undo an operation
/undo

Example: To undo the undo operation
/redo

3.7. Synchronize with Google Calendar

Ensure that Calendo has the correct Google Account settings specified. You can manage account settings by clicking on the Settings Button.
To synchronize with the Google Calendar, use the **/sync** command to synchronize changes done in Calendo to Google Calendar and vice versa. The **/sync** command does not have any parameters.

Example: To synchronize with Google Calendar
/sync

Developer Manual

Table of Contents

1. **Introduction** **9**
    1.1. Component Overview 9

2. **User Interface** **10**
    2.1. MainWindow 10
    2.2. EntryToBrushConverter 11
    2.3. EntryToDateTimeStringConverter 11
    2.4. EntryToDateTimeVisibilityConverter 12

3. **Command Processing** **13**
    3.1. CommandProcessor 13

4. **Data Storage** **14**
    4.1. Storage<T> 14
    4.2. StateStorage<T> 15
    4.3. Data<T> 17
    4.4. State<T> 17

5. **Google Calendar** **19**
    5.1. GoogleCalendar 19

6. **Task Manager** **20**
    6.1. TaskManager 20
    6.2. SettingsManager 21

7. **References** 23

1. Introduction

Calendo is a task management application designed to help users manage their tasks efficiently. This developer guide aims to introduce you to the inner workings of Calendo, and assumes you have experience working with C#. **User Interface** developers should also have some knowledge of XAML and the Windows Presentation Foundation.

1.1. Component Overview

Calendo is broken up into several components.

- **User Interface** (Section 2): The **User Interface** component consists of the Graphical User Interface (GUI) and handles user interaction. The component is simply referred to as "Calendo" in code.
- **Command Processing** (Section 3): The **Command Processing** component handles commands that the user provides through the **User Interface**. This component decides what actions are to be performed based on the supplied command.
- **Data** (Section 4): The **Data** component handles data storage. The component stores data and allows for the retrieval of information to be used at runtime.
- **Google Calendar** (Section 5): The **Google Calendar** component handles synchronization with the Google Calendar web service, including Authentication with the service.
- **Task Manager** (Section 6): The **Task Manager** component is the "logic" module of Calendo, with the bulk of actions being performed by this component. This component acts as a wrapper between **Command Processing** and other components such as **Google Calendar** and **Data**.

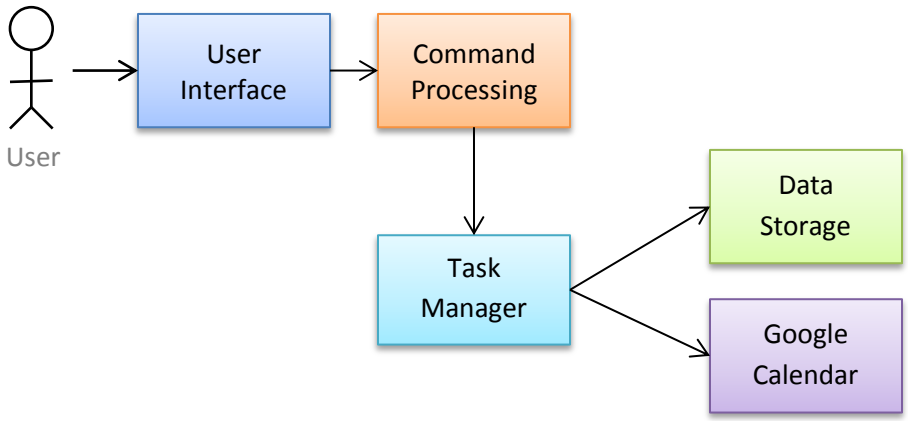


Figure 1.1: Component Overview

2. User Interface

The **User Interface** component is responsible for handling all user interactions, providing a means for users to input commands as well as displaying a list of all the tasks Calendo is handling. There is one main class in this component, `MainWindow`, as well as a corresponding XAML document containing the markup for the interface.

This component also contains several converter classes, found in the `Calendo.Converters` namespace. These classes are `EntryToBrushConverter`, `EntryToDateTimeStringConverter`, and `EntryToDateTimeVisibilityConverter`. Each of these converter classes implements the `Convert(object value, Type targetType, object parameter, CultureInfo culture)` method in the `IValueConverter` interface.

Each converter is used with a data-binding that passes it an `Entry` object.

2.1. `MainWindow`

This class contains methods used as event handlers for user interaction, as well as an instance of the **Command Processing** component, used to handle user input.

Unlike other components in Calendo, `MainWindow` has no publicy accessible properties or methods, and is not designed to be instantiated or used by other modules. This portion of the document hence assumes the reader will be extending or modifying its functionality.

2.1.1. `MainWindow` Properties

Property Name	Description
`CommandProcessor: CommandProcessor`	Command Processing module, used to execute user commands.

2.1.2. `MainWindow` Methods

Method Name	Description
`MainWindow()`	Constructor. Creates and initializes the component.
`DefocusCommandBar()`	Method used to set keyboard focus away from the Command Bar.
`FilterListContents()`	Method used to search the Task List.
`UpdateItemsList()`	Updates the Task List with entries retrieved from `CommandProcessor`.
`BtnSettingsClick(object sender, RoutedEventArgs e)`	Event handler triggered when the Settings button is clicked.
`CloseWindow(object sender, RoutedEventArgs e)`	Event handlers triggered when the Close, Minimise, or Maximise buttons are clicked.
`MinimiseWindow(object sender, RoutedEventArgs e)`	
`MaximiseWindow(object sender, RoutedEventArgs e)`	
`GridMouseDown(object sender, MouseButtonEventArgs e)`	Event handler triggered when a mouse click occurs in the main Grid control. Used to set keyboard focus away from the Command Bar.
`TbxCommandBarLostFocus(object sender, RoutedEventArgs e)`	Event handlers triggered when the Command Bar has lost or gained focus, used to show or hide the “Enter Command” prompt appropriately.
`TbxCommandBarGotFocus(object sender, RoutedEventArgs e)`	

Method Name	Description
`TbxCommandBarKeyUp(object sender, KeyEventArgs e)`	Event handler triggered after a keystroke has been detected in the Command Bar. Used to send user input to `CommandProcessor` as well as filter the task list.
`UndoHandler(object sender, ExecutedRoutedEventArgs e)`	Event handlers triggered when the key combinations Ctrl+Z or Ctrl+Y are pressed, used to Undo or Redo the last action.
`RedoHandler(object sender, ExecutedRoutedEventArgs e)`	
`ItemsListDoubleClick (object sender, SelectionChangedEventArgs e)`	Event handler triggered when an item in the Task List has been double-clicked. Used to automatically fill a "change" command into the Command Bar.
`WindowStateChanged(object sender, EventArgs e)`	Event handler triggered when the window shifts between states (maximized, normal, or minimized). Used to prevent a quirk in WPF's handling of the window border.

2.2. `EntryToBrushConverter`

This class is used in the `MainWindow.xaml` control to colour tasks in the task list according to their current status (ongoing, overdue, or normal).

2.2.1. `EntryToBrushConverter` Properties

This class has no properties.

2.2.2. `EntryToBrushConverter` Methods

Method Name	Description
`Convert(object value, Type targetType, object parameter, CultureInfo culture)`	Takes in an `Entry`, processes it, and returns a `Brush` with a colour appropriate for the entry's status.
`IsTaskOverdue(Entry currentEntry)`	Takes in an `Entry`, returns `true` if it is overdue, `false` otherwise.
`IsTaskOngoing(Entry currentEntry)`	Takes in an `Entry`, returns `true` if it is ongoing, `false` otherwise.
`ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)`	Unimplemented Interface member. Will throw a `NotImplementedException` if called.

2.3. `EntryToDateTimeStringConverter`

This class is used in the `MainWindow.xaml` control to format and display the start and end dates of tasks in a human-readable format.

The data-binding used with this class must also pass it a `string` parameter of either "`StartDate`" or "`EndDate`" depending on which date is to be used.

2.3.1. `EntryToDateTimeStringConverter` Properties

This class has no properties.

2.3.2. `EntryToDateTimeStringConverter` Methods

Method Name	Description
`Convert(object value, Type targetType, object parameter, CultureInfo culture)`	Takes in an `Entry` and a `string` parameter, returns a `string` of the specified date (start/end) in a human-readable format.
`ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)`	Unimplemented Interface member. Will throw a `NotImplementedException` if called.

2.4. `EntryToDateTimeVisibilityConverter`

This class is used in the `MainWindow.xaml` control to properly display tasks with or without start/end dates. If a task has no start or end date, it is displayed as a single-line entry consisting of only its description; if it does, the start (and end date, if applicable) are displayed underneath the description text. This converter is used to determine if the second line should be hidden or displayed, and if so, whether or not separator text is needed between start and end date.

The data-binding used with this class must also pass it a `string` parameter of either "StackPanel" or "RangeText", depending on if the converter is being used to show/hide the second line as a whole, or the separator text alone.

2.4.1. `EntryToDateTimeVisibilityConverter` Properties

This class has no properties.

2.4.2. `EntryToDateTimeStringConverter` Methods

Method Name	Description
`Convert(object value, Type targetType, object parameter, CultureInfo culture)`	Takes in an `Entry` and a `string` parameter, returns `Visibility.Visible` or `Visibility.Collapsed` depending on whether the Entry has a start/end time.
`ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)`	Unimplemented Interface member. Will throw a `NotImplementedException` if called.

3. Command Processing

The **Command Processing** component is used to convert text input from the user into executable commands, execute them, and return the "result". There is one class in this component: `CommandProcessor`.

3.1. `CommandProcessor`

An object of this class is constructed as a "module" to process and execute commands. The `ExecuteCommand` method is then called with a user-input string, interprets the input as an executable command, and finally executes it. `TaskList` is updated with the result of the command.

Example: Execution

```
CommandProcessor commandProcessor = new CommandProcessor();
string inputString = "/add Meeting for CS2103";
commandProcessor.ExecuteCommand(inputString);
```

Example: Result extraction

```
CommandProcessor commandProcessor = new CommandProcessor();
List<Entry> result = CommandProcessor.TaskList;
```

3.1.1. `CommandProcessor` Properties

Property Name	Description
`TaskList: List<Entry>`	Gets the list of tasks returned by the last executed command.

3.1.2. `CommandProcessor` Methods

Method Name	Description
`CommandProcessor()`	Constructor. Creates a `CommandProcessor` "module".
`ExecuteCommand(string userInput)`	Executes the command and stores the result in `TaskList`.

4. Data Storage

The **Data Storage** component is used to store information used by Calendo for later retrieval. There are two main classes in this component, `Storage` and `StateStorage`. These classes are part of the `Calendo.Data` namespace. External classes such as `SettingsManager` and `TaskManager` depend on the `Storage` and `StateStorage` classes respectively.

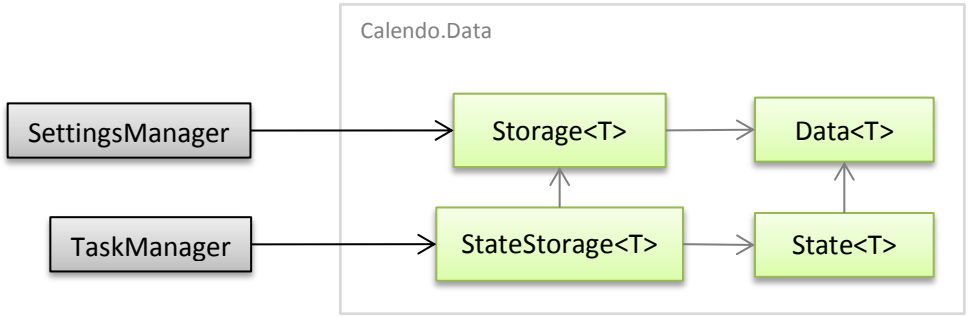


Figure 4.1: Data Storage component overview

4.1. `Storage<T>`

The `Storage` class is used as a general purpose data storage system. As it is a generic class (hence the "<T>"), runtime objects can be saved as they are, and returned back to their original state upon retrieval. The `Storage` class saves the object into an XML file, in a human-readable format.

The `Storage` class saves to "data.txt" by default.

Note: Only *publicly accessible* properties that can be modified are saved. The class must also be publicly accessible in order to be supported by the Storage class. Interfaces are not supported by the Storage class.

Example: Saving

```
Storage<Entry> myStorage = new Storage<Entry>();
myStorage.Entries = new Entry();
myStorage.Save();
```

Example: Loading

```
Storage<Entry> myStorage = new Storage<Entry>();
myStorage.Load();
Entry current = myStorage.Entries;
```

4.1.1. Storage<T> Properties

Property Name	Description
`Entries: T`	Gets or sets the object stored by Storage.

4.1.2. Storage<T> Methods

Method Name	Description
`Storage()`	Constructor. Use the default "data.txt" as file path.
`Storage(string filePath)`	Constructor. Sets the file path used by Storage class.
`Load(): bool`	Loads the current object from the file. Returns true on success, false if an error occurred.
`Save(): bool`	Saves the current object to the file. Returns true on success, false if an error occurred.

4.1.3. Storage<T> Dependencies

The Storage class is dependent on the Data class. The Data class serves as a wrapper to for the data to be stored, and is mainly used for controlling the resulting XML format. The Storage class uses the `XMLSerializer` class (part of the .NET framework) to serialize and deserialize objects to and from XML.

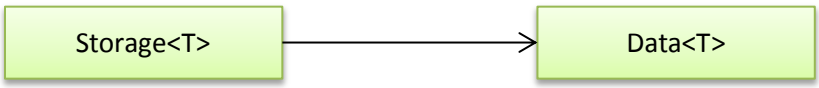


Figure 4.2: Storage<T> Dependencies

4.2. StateStorage<T>

The `StateStorage` class has almost the same functionality as the `Storage` class. Unlike the `Storage` class, `StateStorage` supports the undo and redo feature, allowing you to revert changes made to your object.

The `StateStorage` class can be used in exactly the same way as the `Storage` class.

Note: The file produced by `StateStorage` is generally not compatible with Storage. However, the `Storage` class can be used to read files produced by `StateStorage` by using the `State` class to wrap the original class which was saved.

```
Storage<State<Entry>> compatibleStorage = new Storage<State<Entry>>();
```

Note: The class to be used with `StateStorage` must be *serializable*. In order to tag a class as serializable, add the `[Serializable]` tag to the beginning of the class.

```
[Serializable]
public class Entry {
…
}
```

4.2.1. `StateStorage<T>` Properties

Property Name	Description
`Entries: T`	Gets or sets the object stored by StateStorage.

4.2.2. `StateStorage<T>` Methods

Method Name	Description
`StateStorage()`	Constructor. Use the default "data.txt" as file path.
`StateStorage(string filePath)`	Constructor. Sets the filePath used by StateStorage class.
`HasRedo(): bool`	Returns true if there are available redo states, false otherwise.
`HasUndo(): bool`	Returns true if there are available undo states, false otherwise.
`Load(): bool`	Loads the current object from the file. Returns true on success, false if an error occurred.
`Redo(): bool`	Reverses changes done by the Undo() method. Returns true if the current state has changed, false if no changes were made.
`Save(): bool`	Saves the current object to the file. Saving adds a state to the storage. Returns true on success, false if an error occurred.
`Undo(): bool`	Reverts the current state to the previous state. Returns true if the current state has changed, false if no changes were made.

4.2.3. StateStorage<T> Dependencies

The `StateStorage` class is dependent on both the `Storage` and `State` class. The State class is used to maintain the current state of the object, and stores copies of past revisions of the object. The `Storage` class is used to save the `State` class into a file.

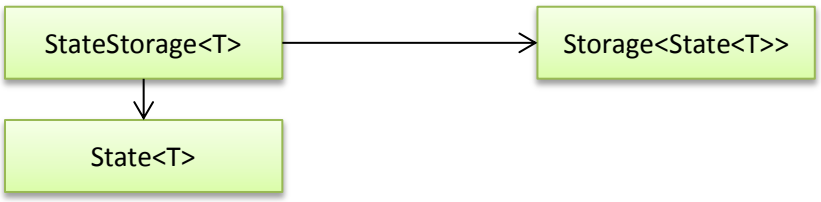


Figure 4.3: StateStorage<T> Dependencies

4.3. Data<T>

The Data class is a wrapper that is used by the `Storage` class. This class implements the `ICloneable` interface, and supports the `Clone()` method. Cloning of an object is done via binary serialization. The Data class is independent.

> **Note**: The supplied type T must have the default constructor.

> **Note**: The clone() method requires the supplied type T to be serializable. Refer to 4.2. StateStorage<T> for an example of how to tag a class as serializable.

4.3.1. Data<T> Properties

Property Name	Description
`Value: T`	Gets or sets the object represented by Data. Used for serialization.

4.3.2. Data<T> Methods

Method Name	Description
`Data()`	Constructor. Uses the default constructor for T.
`Clone(): object`	Returns a deep copy of the object.

4.4. State<T>

The `State` class maintains the current and past states of an object. The `AddState()` method is used to add a state to the list of states. This class supports the `Undo()` and `Redo()` methods, which are used for going back and forward in the list of states respectively.

> **Note**: The supplied type T must have the default constructor and must be serializable. Refer to 4.2. StateStorage<T> for an example of how to tag a class as serializable.

4.4.1. State<T> Properties

Property Name	Description
`Value: T`	Gets or sets the object represented by the current state.
`States: List<Data<T>>`	Gets or sets the list of states. Used for serialization.

4.4.2. State<T> Methods

Method Name	Description
`State()`	Constructor. Creates an empty state.
`AddState(): void`	Adds a state.
`Redo(): bool`	Revert to a state in the redo stack. Returns true if the current state has changed, false otherwise.

Method Name	Description
`Undo(): bool`	Revert to the state before the current state. Returns true if the current state has changed, false otherwise.

4.4.3. `State<T>` Dependencies

The `State` class is dependent on the `Data` class. The `Data` class is used to provide deep copying of the object, and acts as a wrapper to store the objects representing a state.

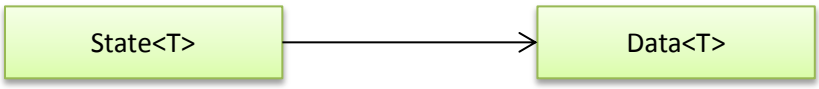


Figure 4.4: State<T> Dependencies

5. Google Calendar

The **Google Calendar** component is responsible for synchronizing the user's tasks with his/her Google Account. Instances of this component are created at runtime only when required.

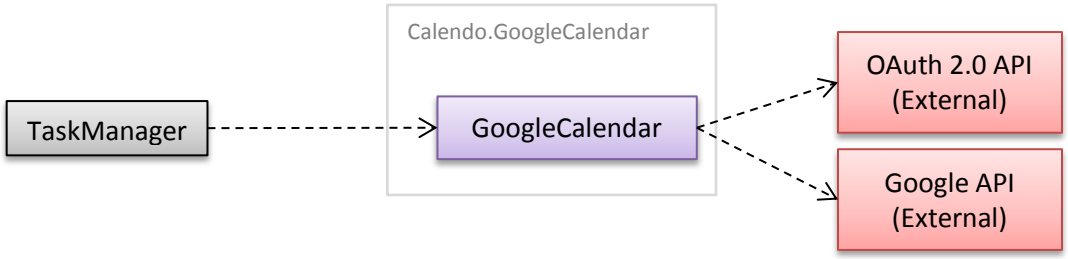


Figure 5.1: Google Calendar Component Overview

5.1. GoogleCalendar

5.1.1. GoogleCalendar Methods

Method Name	Description
`Sync(List<Entry> entries): boolean`	Gets list of tasks and adds it to Google Calendar
`Authorize(): string`	Authorises user's Google Account
`GetAuthentication(NativeApplicationClient provider): string`	Returns access token for authentication
`PostTasks(Entry entry,string access_token): Boolean`	Posts a task
`Import(string access_token): string`	Imports tasks from Google Calendar

5.1.2. GoogleCalendar Dependencies

The `GoogleCalendar` class is dependent on the `TaskManager` class, which is used to obtain entries to be added to Google Calendar (through the `Sync()` method). The `TaskManager` class also calls the `Import()` method of `GoogleCalendar`.

The `GoogleCalendar` component uses several libraries from the Google Calendar API [1]. Authentication is performed using the OAuth 2.0 authorization protocol [2]. Google's API libraries are part of the `Google.Apis` namespace, while the OAuth 2.0 library is part of the `DotNetOpenAuth` namespace; both namespaces belong to external libraries.

Additionally, the **Google Calendar** component is indirectly dependent on the JSON parsers from the `Json.Net` library [3], which is used by the Google API libraries to serialize runtime objects into a format suitable for web requests or responses.

6. Task Manager

The **Task Manager** component acts as a layer between the **Command Processing** and **Data** components. This component is part of the main `Calendo` namespace.

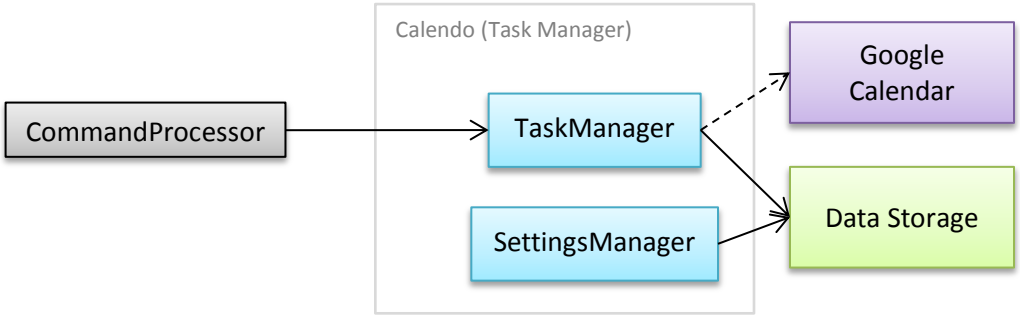


Figure 6.1: Task Manager Component Overview

6.1. TaskManager

The `TaskManager` class performs the actions requested by the **Command Processing** component. Some of these actions are passed directly to other components (i.e. a pass-through), such as the `Sync()` method to the **Google Calendar** component.

6.1.1. TaskManager Properties

Property Name	Description
`Entries: List<Entry>`	Gets the list of entries.

6.1.2. TaskManager Methods

Method Name	Description
`TaskManager()`	Constructor. Load entries from archive file.
`Add(string description): void`	Adds a floating task.
`Add(string description, string date, string time): void`	Adds a deadline task.
`Add(string description, string startDate, string startTime, string endDate, string endTime): void`	Adds a timed task.
`Change(int id, string description, string startDate, string startTime, string endDate, string endTime): void`	Change parameters of a task. Parameters that are empty would be ignored.
`Get(int id): Entry`	Get the task by ID.
`Redo(): void`	Undo changes performed by undo
`Remove(int id): void`	Remove a task by ID.
`Sync(): void`	Synchronize tasks with Google Calendar (pass-through to Google Calendar component)
`Undo(): void`	Undo the last operation (excluding itself)

6.1.3. `TaskManager` Dependencies

The `TaskManager` class is dependent on `StateStorage` (from `Calendo.Data`). The `StateStorage` class is used for storing entries into a file for later retrieval. The `TaskManager` class also depends on the `TaskTime` class, but only as a temporary data representation for date and time. The `TaskTime` class is not intended to be used by other classes.

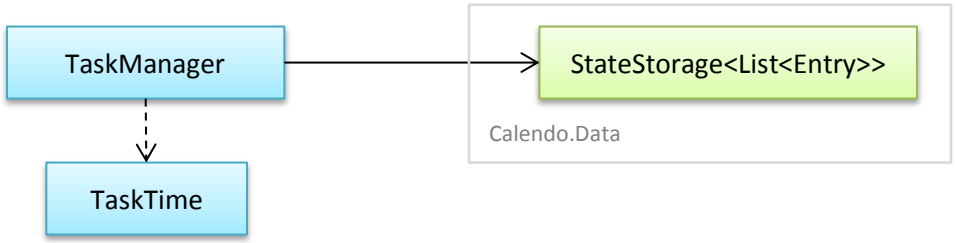


Figure 6.2: TaskManager Dependencies

6.2. `SettingsManager`

The `SettingsManager` class provides a means to access and modify settings. This class is intended for the **Google Calendar** component, although it can be used to store any other setting. All settings are in string format. Objects can be converted into string format using serialization formats such as XML or JSON.

6.2.1. `SettingsManager` Properties

There are no properties for `SettingsManager`. Settings are retrieved and modified via methods.

6.2.2. `SettingsManager` Methods

Method Name	Description
`SettingsManager()`	Constructor. Load settings from settings file.
`GetSetting(string settingName): string`	Retrieves the value of the setting for the specified setting.
`SetSetting(string settingName, string settingValue): void`	Sets the value of the setting for the specified setting. If the setting exists, its value would be overridden.

6.2.3. `SettingsManager` Dependencies

The `SettingsManager` class is dependent on the `Storage` (from `Calendo.Data`) and `KeyPair` classes. The `Storage` class is used for storing settings into a file for later retrieval. The `KeyPair` class is a serializable class used for storing Key-Value pairs.

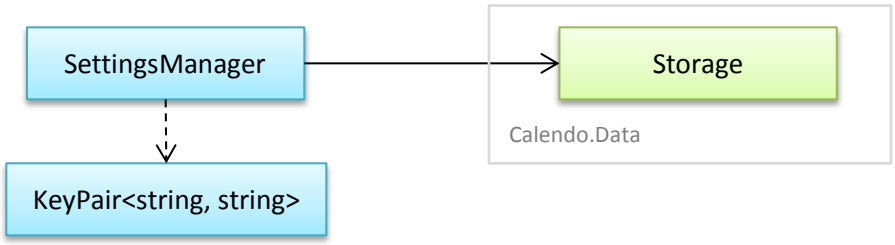


Figure 6.3: SettingsManager Dependencies

7. References

[1] Google. (2012 Sep). *Google Apps Calendar API: Downloads.* Retrieved 12 October, 2012 from the World Wide Web: https://developers.google.com/google-apps/calendar/downloads

[2] Internet Engineering Task Force. (2012 March). *The OAuth 2.0 Authorization Protocol.* Retrieved 13 October, 2012 from the World Wide Web: http://tools.ietf.org/html/draft-ietf-oauth-v2-22/

[3] Codeplex. (2012 Oct). Json.NET. Retrieved 13 October, 2012 from the World Wide Web: http://json.codeplex.com/