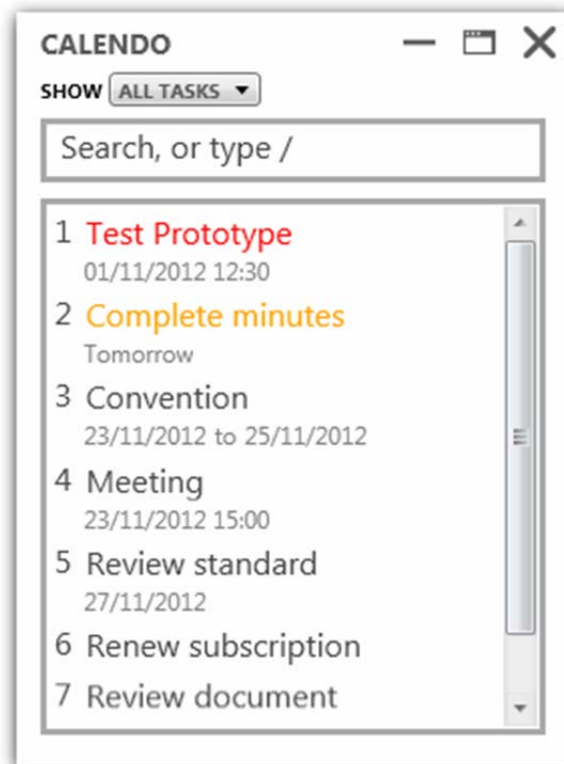






# Calendo



			
<b>Nicholas Kwan</b>	<b>Rahij Ramsharan</b>	<b>Jerome Cheng</b>	<b>Pallav Shinghal</b>
Team Leader, Data Storage Developer, Specification Writer	Google Calendar Integration, Network Developer	User Interface Developer, Interface Tester	User Input Processing, Command Testing

# User Guide

## 1. Introduction

### 1.1. About Calendo

Calendo is a task planner that allows you to easily and efficiently manage tasks. It integrates with Google Calendar so that tasks can 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. Google Calendar Integration requires 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 Search/Command Bar. A command is of the form **/command** [parameters].

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

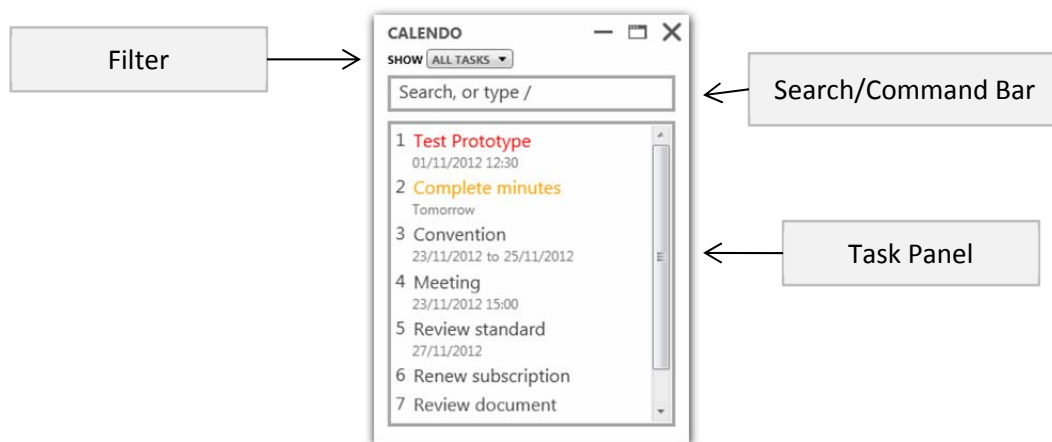


Figure 2.1: User Interface

### 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 are considered Active 24 hours before their due date. When this occurs, the task appears highlighted in orange in the task list. When the task is overdue, the task is highlighted in red.

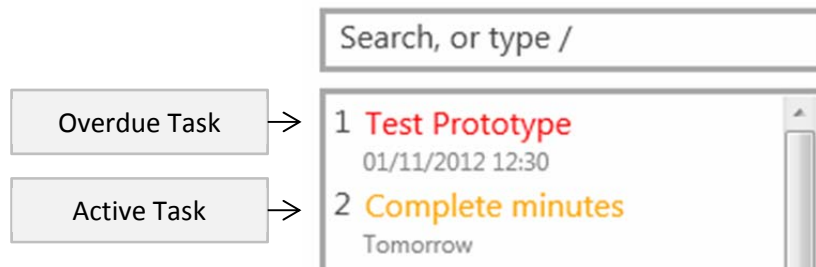


Figure 2.3: Task notification for Deadline Tasks

### 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 highlight them as Active 24 hours before the timed task begins. When the time interval of the timed task has lapsed, the task will be marked as overdue.

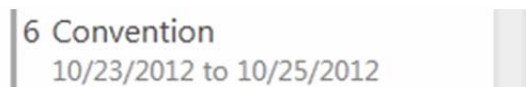


Figure 2.4: Timed Task

## 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.

**Note:** Google Calendar Integration is an experimental feature, and functionality might change in future updates. For more information, refer to section 4 Experimental Features.

## 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.

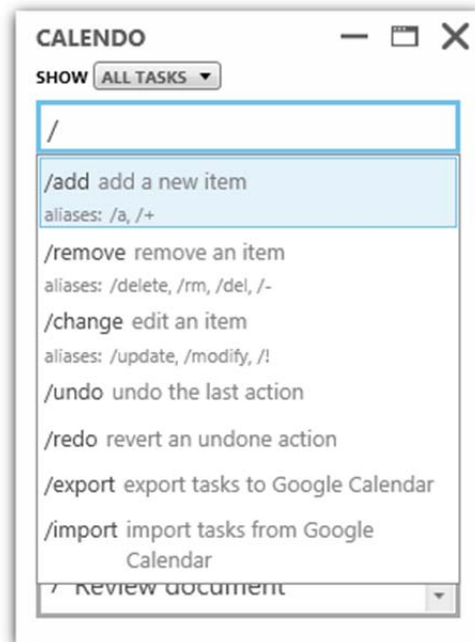


Figure 2.5: Auto Suggest

## 3. Quick start

### 3.1. Setting up Calendo

Calendo does not require installation; simply run the executable where it is. If Calendo is run for the first time, it will create several new files in the same folder.

### 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 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 will 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 18<sup>th</sup> of October. For more information on task notifications, refer to section 2.2 Task Notification.

The date is in the form of Day/Month/Year. Specifying the year is optional, and if 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

**Example:** To create a timed task on a certain time range

**/add** Exhibition **/date** 19/9-21/9

**Note:** floating tasks may also be added by simply typing into the Search/Command Bar and pressing enter.

### 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`

**Note:** a quicker way to remove tasks is to click on the delete icon shown beside the task.

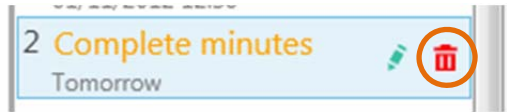


Figure 3.1: Quick Remove

### 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 will 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 parameters are provided but their 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

**Note:** a quicker way to modify tasks is to click on the edit icon. The time or date can be also be modified in this way using parameters similar to the change command.



Figure 3.2: Quick Edit

### 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 by description.

Search queries must not begin with a “/”.

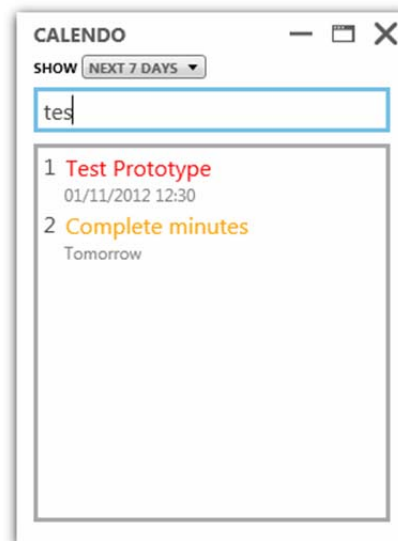


Figure 3.3: Searching for tasks

### 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.

**Note:** Undo and redo can also be performed using the Ctrl+Z and Ctrl+Y keyboard shortcuts respectively.

**Example:** To undo an operation  
`/undo`

**Example:** To revert changes done by undo  
`/redo`

## 4. Experimental Features

Experimental features are features that are still under development. However, we have chosen not to remove these features so that you are able to see what we have in store for the future. Be aware that these experimental features are not as well-tested as other features, and unexpected behavior may be encountered when using them.

### 4.1. Export to Google Calendar

To export tasks to Google Calendar, use the `/export` command. Tasks in Calendo will be transferred to the online service, and will replace your task list and schedule.

**Note:** The export command will remove all existing entries in Google Calendar. This operation is permanent and cannot be undone.

**Example:** Export to Google Calendar  
`/export`

When the command is used, you may be prompted for an authorization code. Simply sign in using your Google Account and paste the authorization code into the form shown.

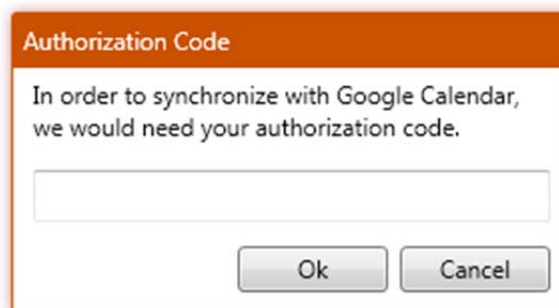
A dialog box titled "Authorization Code" with an orange header bar. The text inside reads: "In order to synchronize with Google Calendar, we would need your authorization code." Below the text is a white rectangular input field. At the bottom right of the dialog are two buttons: "Ok" and "Cancel".

Figure 4.1: Authorization form



## 4.2. Import from Google Calendar

To import your tasks from Google Calendar, use the `/import` command. Tasks in Calendo will be replaced by corresponding tasks on the online service.

**Note:** The import command will remove all existing entries in Calendo. The command can be reverted using the undo command.

**Example:** Import from Google Calendar  
`/import`

# Developer Manual

## Table of Contents

<b>1. Introduction</b>	<b>11</b>
1.1. Component Overview	11
<b>2. User Interface</b>	<b>12</b>
2.1. MainWindow	12
2.2. EntryToBrushConverter	13
2.3. EntryToDateTimeStringConverter	14
2.4. EntryToDateTimeVisibilityConverter	14
2.5. StringArrayToStringConverter	15
2.6. AutoSuggest	15
2.7. AutoSuggestEntry	16
<b>3. Logic</b>	<b>17</b>
3.1. CommandProcessor	18
3.2. Command	19
3.3. CommandExtractors	19
3.4. TaskManager	20
3.5. SettingsManager	21
3.6. TimeConverter	22
<b>4. Data Storage</b>	<b>24</b>
4.1. Storage<T>	24
4.2. StateStorage<T>	26
4.3. Data<T>	27
4.4. State<T>	27
<b>5. Google Calendar</b>	<b>29</b>
5.1. GoogleCalendar	29
5.2. JSON<T>	29
<b>6. Testing</b>	<b>31</b>
<b>7. Change Log</b>	<b>35</b>
<b>8. References</b>	<b>37</b>

## 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, as shown in figure 1.1.

- **User Interface** (Section 2, page 11): 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.
- **Logic** (Section 3, page 16): The **Logic** 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. This component acts as a wrapper between **User Interface** and other components such as **Google Calendar** and **Data**.
- **Data** (Section 4, page 21): 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, page 26): The **Google Calendar** component handles synchronization with the Google Calendar web service, including authentication.

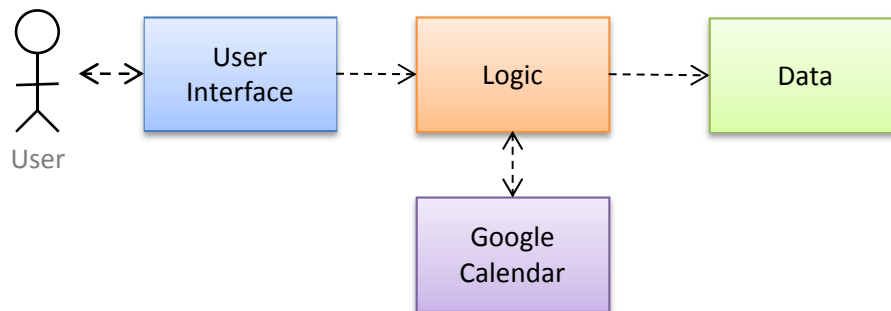


Figure 1.1: Architecture Overview

## 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. The main window and its behavior is defined in the `MainWindow` class, as well as the corresponding XAML document containing the markup for the interface.

The `UiViewModel` class is used as a façade for the `MainWindow` class to perform its functions: this class acts as a source task data and auto suggest information, and a sink for user input to be routed to the **Logic** component. Data from `UiViewModel` is displayed in `MainWindow` through the use of data bindings.

This component also contains several converter classes, found in the `Calendo.Converters` namespace. These classes are `EntryToBrushConverter`, `EntryToDateTimeStringConverter`, `EntryToDateTimeVisibilityConverter`, and `StringArrayToStringConverter`. Each of these converter classes implements the `IValueConverter` interface, and is used with a data-binding that passes it an `Entry` object with the exception of `StringArrayToStringConverter`, which takes in an array of strings.

Auto Suggest logic is found in the `AutoSuggest` class, with an accompanying `AutoSuggestEntry` class used to model entries that appear in the suggestion list.

Finally, the `UiTaskHelper` class contains several methods used to assist in comparing `Entry` objects in order to sort them.

### 2.1. MainWindow

This class contains methods used as event handlers for user interaction, as well as an instance of the `UiViewModel` class.

Unlike other components in Calendo, `MainWindow` has no publicly 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
ViewModel: <code>UiViewModel</code>	<code>UiViewModel</code> instance. Used to display task and auto suggest data and handle commands.

#### 2.1.2. MainWindow Methods

Method Name	Description
<code>MainWindow()</code>	Constructor. Creates and initializes the component.
<code>CategoryFilter(object o)</code>	Predicate used in <code>FilterListContents</code> . Used to filter the task list for items in a specific category (overdue, active, or by date.)
<code>DeleteSelectedTask()</code>	Method used to delete a task when a user clicks the Delete button on it in the Task List.
<code>FocusOnTaskList()</code>	Method used to set keyboard focus away from the Command Bar.

Method Name	Description
FilterListContents()	Method used to filter the Task List.
SearchFilter(object o)	Predicate used in FilterListContents. Used to filter the task list for items matching the user's search term.
SetCommandFromSuggestion()	Method used to fill in the appropriate command when the user selects an item from the Auto Suggest list.
AutoSuggestListKeyDown(object sender, KeyEventArgs e)	Event handler triggered after a keystroke has been detected in the Auto Suggest list. Used to move keyboard focus between the list and the Command Bar, and select a suggestion from the list.
AutoSuggestListMouseUp(object sender, MouseButtonEventArgs e)	Event handler triggered after a mouse click occurs in the Auto Suggest list. Used to select a suggestion from the list.
BtnSettingsClick(object sender, RoutedEventArgs e)	Event handler triggered when the Settings button is clicked.
CommandBarLostFocus(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.
CommandBarGotFocus(object sender, RoutedEventArgs e)	
CommandBarKeyUp(object sender, KeyEventArgs e)	Event handler triggered after a keystroke has been detected in the Command Bar. Used to send user input to UiViewModel as well as filter the task list.
CloseWindow(object sender, RoutedEventArgs e)	Event handlers triggered when the Close, Minimize, or Maximize buttons are clicked.
MinimiseWindow(object sender, RoutedEventArgs e)	
MaximiseWindow(object sender, RoutedEventArgs e)	
DeleteHandler(object sender, ExecutedRoutedEventArgs e)	Event handler triggered when the Delete key is pressed on the keyboard. Used to delete the selected item from the list.
DragWindow(object sender, MouseButtonEventArgs e)	Event handler triggered when a mouse click occurs on the title bar area. Used to toggle the maximize state of the window and allow it to be moved around the screen.
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.
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)	
WindowStateChange(object sender, EventArgs e)	Event handler triggered when the window shifts between states (maximized, normal, or minimized). Used to adjust the window borders.

## 2.2. EntryToBrushConverter

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

### 2.2.1. 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.

Method Name	Description
IsTaskOverdue( <a href="#">Entry</a> currentEntry)	Takes in an Entry, returns true if it is overdue, false otherwise.
IsTaskOngoing( <a href="#">Entry</a> currentEntry)	Takes in an Entry, returns true if it is ongoing, false otherwise.
ConvertBack( <a href="#">object</a> value, <a href="#">Type</a> targetType, <a href="#">object</a> parameter, <a href="#">CultureInfo</a> 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 Methods

Method Name	Description
Convert( <a href="#">object</a> value, <a href="#">Type</a> targetType, <a href="#">object</a> parameter, <a href="#">CultureInfo</a> culture)	Takes in an Entry and a string parameter, returns a string of the specified date (start/end) in a human-readable format.
ConvertBack( <a href="#">object</a> value, <a href="#">Type</a> targetType, <a href="#">object</a> parameter, <a href="#">CultureInfo</a> 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. EntryToDateTimeStringConverter Methods

Method Name	Description
Convert( <a href="#">object</a> value, <a href="#">Type</a> targetType, <a href="#">object</a> parameter, <a href="#">CultureInfo</a> 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( <a href="#">object</a> value, <a href="#">Type</a> targetType, <a href="#">object</a> parameter, <a href="#">CultureInfo</a> culture)	Unimplemented Interface member. Will throw a NotImplementedException if called.

## 2.5. StringArrayToStringConverter

This class is used in the MainWindow.xaml control to format the list of aliases that appear under each suggestion in the Auto Suggest list.

### 2.5.1. StringArrayToStringConverter Methods

Method Name	Description
Convert(object value, Type targetType, object parameter, CultureInfo culture)	Takes in a string array, returns a string consisting of each of the elements in the array separated with commas (or “none” if the array is empty.)
ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)	Unimplemented Interface member. Will throw a NotImplementedException if called.

## 2.6. AutoSuggest

This class is used to display a list of suggested commands based on the user’s input. It maintains a basic dictionary of the various commands accepted by Calendo, as well as an accompanying description for each.

### 2.6.1. AutoSuggest Properties

Property Name	Description
COMMAND_INDICATOR: char	Constant, set to the command prefix used by Calendo (“/” at the time of this writing.)
SuggestionList: List<AutoSuggestEntry>	The list of suggestions to be displayed in the User Interface.

### 2.6.2. AutoSuggest Methods

Method Name	Description
AutoSuggest(Dictionary<string, string[]> aliasDictionary)	Constructor. Initializes this module, and prepares the Master List of entries. Takes in a dictionary containing command types and the corresponding aliases.
AddEntryFromString(Dictionary<string, string[]> aliasDictionary, string currentCommand)	Method used to populate the list of suggestions. Takes in the dictionary of aliases and the current command to be added to the list.
SetSuggestions(string input)	Method used to trigger the generation of suggestions. Takes in the input for which suggestions should be provided.
GenerateSuggestionsFromInput(string input)	Method used to determine if Auto Suggest should be providing command suggestions, or displaying help instructions for a specific command. Called from SetSuggestions.
MatchInputToInstruction(string inputCommand)	Method used if Auto Suggest is to display help instructions for a specific command. Takes in the command to display instructions for.
MatchInputToCommandSuggestion(string inputCommand)	Method used if Auto Suggest is to display command suggestions. Takes in the input for which suggestions should be provided.

Method Name	Description
CheckAliasesForCommand( <a href="#">string</a> inputCommand, <a href="#">AutoSuggestEntry</a> entry)	Method used to determine if an <code>AutoSuggestEntry</code> contains an alias entered by the user. Takes in the entered alias and an entry, returns <code>true</code> if the alias is found in the entry and <code>false</code> otherwise.

## 2.7. AutoSuggestEntry

This class is used to model the entries that appear in the Auto Suggest list. There are two types of entries: Master and Detail.

Master entries appear when the user starts to enter a command in the Command Bar, and consist of the accepted command, an accompanying description, and the various aliases (alternate commands that have the same action) that can be entered instead.

Detail entries appear when the user has entered a command, and consist of the command and a parameter guide (showing the various parameters that command is used with.)

### 2.7.1. AutoSuggestEntry Properties

Property Name	Description
Command: <a href="#">string</a>	The command keyword.
Description: <a href="#">string</a>	The description of the command (for Master entries), or the parameter guide (for Detail entries).
Type: <a href="#">EntryType</a>	The type of this entry: Master or Detail.
Aliases: <a href="#">string</a> []	Any aliases that can be used as a replacement for this command.
IsMaster: <a href="#">bool</a>	Whether or not this entry is a Master entry: <code>true</code> if so, <code>false</code> otherwise.
HasAliases: <a href="#">bool</a>	Whether or not this entry has any aliases: <code>true</code> if so, <code>false</code> otherwise.



### 3. Logic

The **logic** component performs commands issued by the user. The component is located in the `Calendo.Logic` namespace. The component classes are illustrated in figure 3.1.

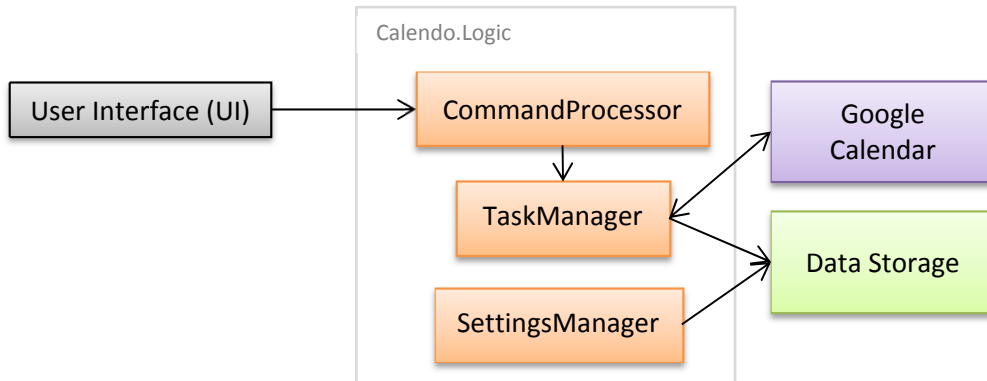


Figure 3.1: Task Manager Component Overview

When a user enters a command into the command box in the **User Interface**, the command gets passed into **CommandProcessor** as a string. **CommandProcessor** creates a **Command** object for the input string that represents details about the action to be taken. **Command** uses a **CommandExtractors** object to extract these details. **CommandProcessor** then calls the corresponding method in **TaskManager**. **TaskManager** manipulates the information in **Storage**. After the procedure, the **User Interface** component would request the list of tasks and display it to the user. The sequence of events when a command is invoked is illustrated by figure 3.2. **CommandExtractors** is omitted for brevity.

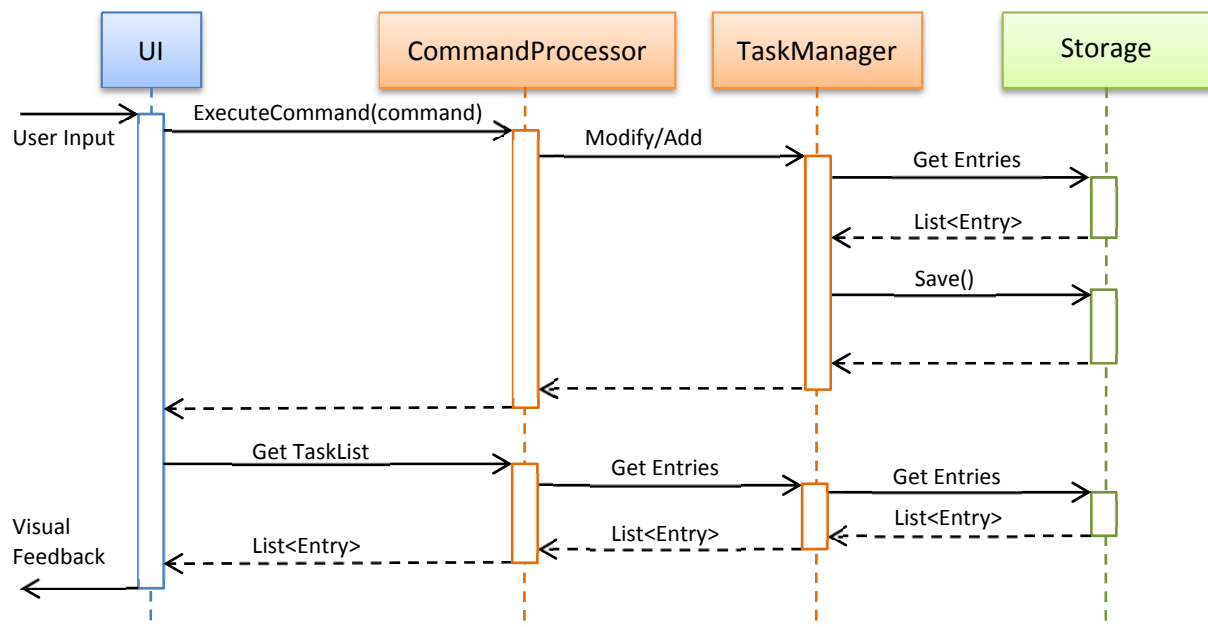


Figure 3.2: General processing of user input

### 3.1. CommandProcessor

The `CommandProcessor` class is used for processing user input, and determines which actions performed based on the information extracted by `CommandExtractors` from the provided input, through a `Command` object. The command entered by the user is passed into the `ExecuteCommand` method by the **User Interface** component. The `TaskList` property is updated after the method is called.

#### Example: Command Execution

```
CommandProcessor commandProcessor = new CommandProcessor();
string inputString = "/add Task Description";
commandProcessor.ExecuteCommand(inputString);
```

#### Example: Accessing the current task list

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

#### 3.1.1. CommandProcessor Properties

Property Name	Description
TaskList: <code>List&lt;Entry&gt;</code>	Gets the list of tasks returned by the last executed command.

#### 3.1.2. CommandProcessor Methods

Method Name	Description
<code>CommandProcessor()</code>	Constructor. Creates a <code>CommandProcessor</code> “module”.
<code>ExecuteCommand(string userInput): void</code>	Executes the command and stores the result in <code>TaskList</code> .

#### 3.1.3. CommandProcessor Methods

The `CommandProcessor` class depends on the `Command` class for representing the commands, and `CommandExtractors` for extracting information used in a command from user input. The dependencies are illustrated in figure 3.3.

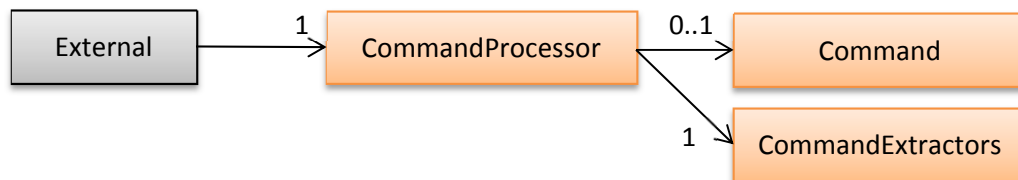


Figure 3.3: `CommandProcessor` Component Overview

### 3.2. Command

Objects of the Command class are used for storing all the details of a command that are required for its execution. The details are extracted using a CommandExtractors object passed to it by CommandProcessor.

**Example:** Command object construction

```
string inputString = "/add Task Description";
CommandExtractors extractors = new CommandExtractors();
Command command = new Command(inputString, ref extractors);
```

**Example:** Accessing the details of the command

```
Command command = new Command(inputString, ref extractors);
string commandType = command.Type;
```

#### 3.2.1. CommandProcessor Properties

Property Name	Description
StartDate: <code>string</code>	Get the details of the command, extracted by CommandExtractors from userInput.
StartTime: <code>string</code>	
EndDate: <code>string</code>	
EndTime: <code>string</code>	
Type: <code>string</code>	
Text: <code>string</code>	

#### 3.2.2. CommandProcessor Methods

Method Name	Description
Command( <code>string</code> userInput, <code>ref</code> <code>CommandExtractors</code> extractors)	Constructor. Creates a Command object.

### 3.3. CommandExtractors

The CommandExtractors class is used for extracting the executable details of a command from the user input.

**Example:** Command extraction

```
CommandExtractors extractors = new CommandExtractors();
string userInput = "/add Task Description";
extractors.Extract(userInput, ref type, ref startDate, ref startTime, ref endDate, ref endTime, ref text);
```

### 3.3.1. CommandExtractors Methods

Method Name	Description
CommandExtractors()	Constructor. Creates a CommandExtractors instance.
Extract(string userInput, ref string type, ref string startDate, ref string startTime, ref string endDate, ref string endTime, ref string text): void	Extracts the components of the command from userInput into the strings passed by reference.

### 3.4. TaskManager

The TaskManager class performs the actions requested by the CommandProcessor class. Some of these actions are passed directly to other components, such as the Save() method to the **Data Storage** component.

**Note:** The TaskManager class is a *singleton*, which means there can only be one instance of TaskManager. This instance can be accessed from the Instance static property.

TaskManager has a subscriber functionality that allows other components to be notified when an operation has made modifications to the information stored.

**Example:** Adding a subscription

```
TaskManager.Instance.AddSubscriber(new TaskManager.UpdateHandler(this.UpdateMethod));
```

#### 3.4.1. TaskManager Properties

Property Name	Description
Entries: List<Entry>	Gets the list of entries.
Instance: TaskManager	Static property. Gets a TaskManager instance.

#### 3.4.2. TaskManager Methods

Method Name	Description
TaskManager()	Private constructor. Load entries from archive file. To obtain an instance of TaskManager, use the Instance property.
Add(string description, string startDate, string startTime, string endDate, string endTime): void	Adds a task.
AddSubscriber(Delegate handler): void	Adds a handler to the list of subscribers.
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.

Method Name	Description
Redo(): void	Revert changes performed by undo
Remove(int id): void	Remove a task by ID.
Import(): void	Import tasks from Google Calendar.
Export(): void	Export tasks from Google Calendar.
Undo(): void	Undo the last operation (excluding itself).

### 3.4.3. 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 TimeConverter class is used to convert strings into a suitable time format.

The TaskManager can have multiple instances of the GoogleCalendar class, as each invocation of the import and export methods create an instance of the class on a separate thread. GoogleCalendar runs on a separate thread since operations in the GoogleCalendar class may require significant resource usage depending on the number of tasks involved.

There can only be one instance of TaskManager. This is to allow for concurrent modification of information shared between different components.

The dependencies for TaskManager are shown in figure 3.4.

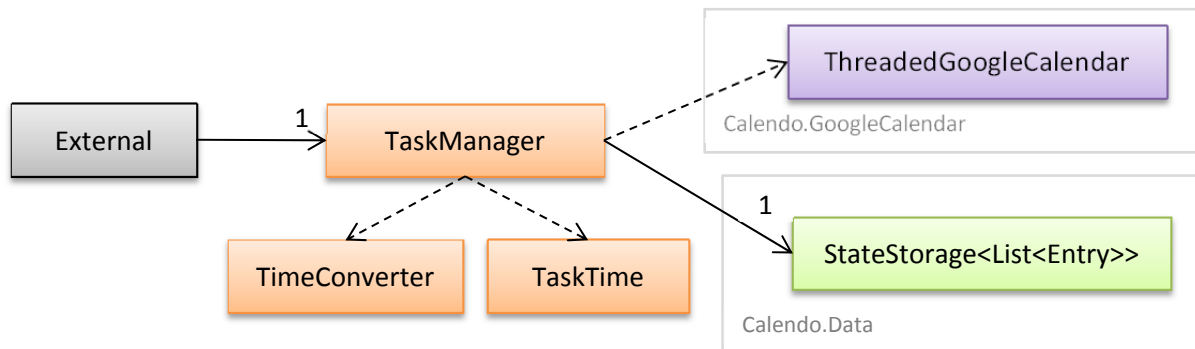


Figure 3.4: TaskManager Dependencies

## 3.5. SettingsManager

The SettingsManager class provides a means to access and modify settings. All settings are in string format. Objects can be converted into string format using serialization formats such as XML or JSON.

### 3.5.1. SettingsManager Methods

Method Name	Description
SettingsManager()	Constructor. Load settings from settings file.

Method Name	Description
GetSetting( <b>string</b> settingName): <b>string</b>	Retrieves the value of the setting for the specified setting.
SetSetting( <b>string</b> settingName, <b>string</b> settingValue): <b>void</b>	Sets the value of the setting for the specified setting. If the setting exists, its value would be overwritten.

### 3.5.2. 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 and is only used when loading and saving settings. The dependencies of SettingsManager are shown in figure 3.5.

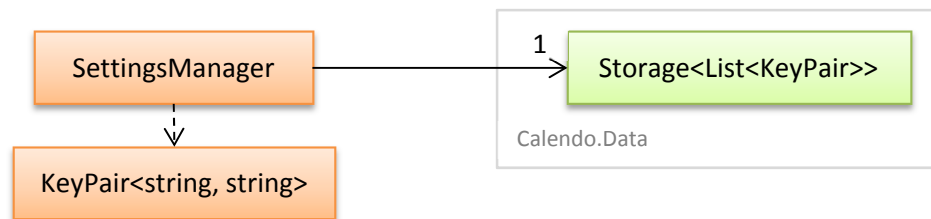


Figure 3.5: SettingsManager Dependencies

### 3.6. TimeConverter

The TimeConverter class is used for converting time formats in the form of strings into the DateTime format that can be used by other components.

#### 3.6.1. TimeConverter Methods

Method Name	Description
TimeConverter()	Constructor.
Convert( <b>string</b> date, <b>string</b> time): <b>TaskTime</b>	Converts date and time to the TaskTime format.
MergeTime( <b>TaskTime</b> source, <b>TaskTime</b> destination, <b>ModifyFlag</b> flag): <b>TaskTime</b>	Merges two times together.

#### 3.6.2. TimeConverter Dependencies

The TimeConverter uses the TaskTime format to represent DateTime format as well as additional information involved in the conversion process. The class also uses helper methods from the TimeHelper class to perform extraction of individual fields inside the time string. The dependencies of TimeConverter are shown in figure 3.6.

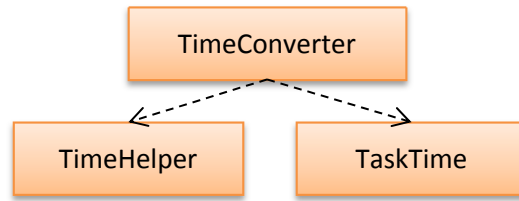


Figure 3.6: TimeConverter Dependencies

## 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. The component is illustrated in figure 4.1.

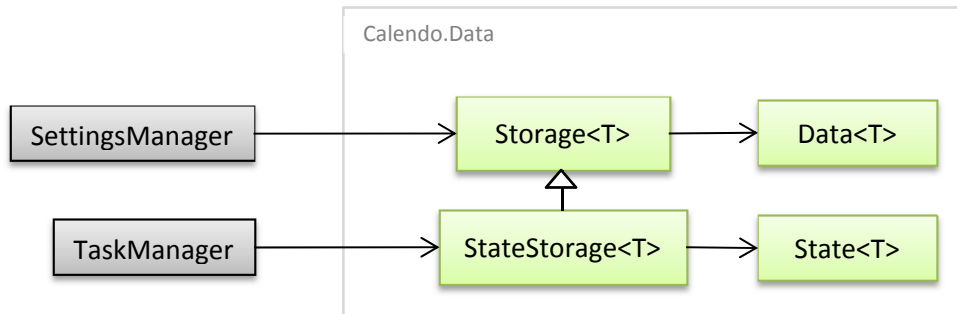


Figure 4.1: Data Storage component overview

### 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, which is 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 and non-serializable classes 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: <code>T</code>	Gets or sets the object stored by Storage.

#### 4.1.2. Storage<T> Methods

Method Name	Description
<code>Storage()</code>	Constructor. Use the default “data.txt” as file path.
<code>Storage(string filePath)</code>	Constructor. Sets the file path used by Storage class.
<code>Load(): bool</code>	Loads the current object from the file. Returns true on success, false if an error occurred.
<code>Save(): bool</code>	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. The Storage class dependencies is shown in figure 4.2.

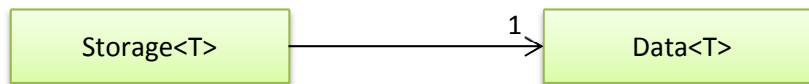


Figure 4.2: Storage<T> Dependencies

## 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 the same way as the Storage class.

**Note:** The file produced by StateStorage is usually 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 generic type T 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: <code>T</code>	Gets or sets the object stored by StateStorage.
HasRedo: <code>bool</code>	Returns true if there are available redo states, false otherwise.
HasUndo: <code>bool</code>	Returns true if there are available undo states, false otherwise.

#### 4.2.2. StateStorage<T> Methods

Method Name	Description
StateStorage()	Constructor. Use the default "data.txt" as file path.
StateStorage( <code>string</code> filePath)	Constructor. Sets the filePath used by StateStorage class.
Load(): <code>bool</code>	Loads the current object from the file. Returns true on success, false if an error occurred.
Redo(): <code>bool</code>	Reverses changes done by the Undo() method. Returns true if the current state has changed, false if no changes were made.
Save(): <code>bool</code>	Saves the current object to the file. Saving adds a state to the storage. Returns true on success, false if an error occurred.
Undo(): <code>bool</code>	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 inherits from the Storage and depends on the 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. The dependencies are illustrated in figure 4.3.

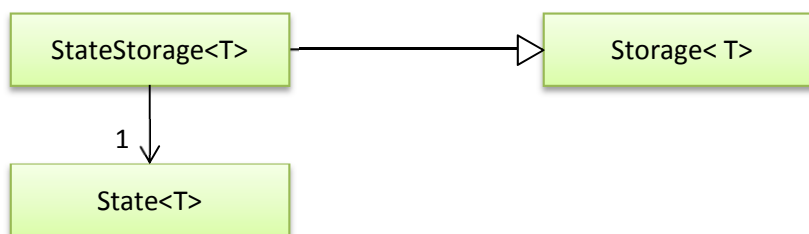


Figure 4.3: StateStorage<T> Dependencies

### 4.3. Data<T>

The Data class is a wrapper that is used by the Storage class. The wrapper class controls the formatting produced by the serializer.

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

#### 4.3.1. Data<T> Properties

Property Name	Description
Value: <code>T</code>	Gets or sets the object represented by Data. Used for serialization.

#### 4.3.2. Data<T> Methods

Method Name	Description
<code>Data()</code>	Constructor. Uses the default constructor for T.

#### 4.3.3. Data<T> Dependencies

The Data class has no dependencies with other classes in Calendo, and is only used to contain a value of the generic type. The dependencies are shown in figure 4.4.

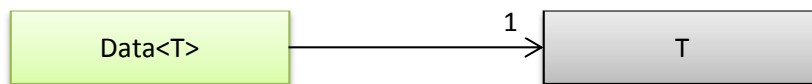


Figure 4.4: Data<T> Dependencies

### 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 generic type T must have the default constructor and must be serializable. Refer to section 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: <code>T</code>	Gets or sets the object represented by the current state.
States: <code>List&lt;Data&lt;T&gt;&gt;</code>	Gets or sets the list of states. Used for serialization.
HasRedo: <code>bool</code>	Returns true if there are available redo states, false otherwise.
HasUndo: <code>bool</code>	Returns true if there are available undo states, false otherwise.

#### 4.4.2. State<T> Methods

Method Name	Description
State()	Constructor. Creates an empty state if there is none.
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.
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 has no dependencies with other classes in Calendo. However, there must be at least one copy of the generic type T in the list of states. If all copies are removed, the State class would create one using the default value of the generic type. The dependencies are shown in figure 4.5.

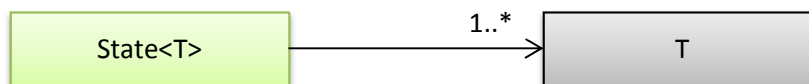


Figure 4.5: State<T> Dependencies

## 5. Google Calendar

The **Google Calendar** component is responsible for synchronizing the user's tasks with their Google Account. The component layout is shown in figure 5.1.

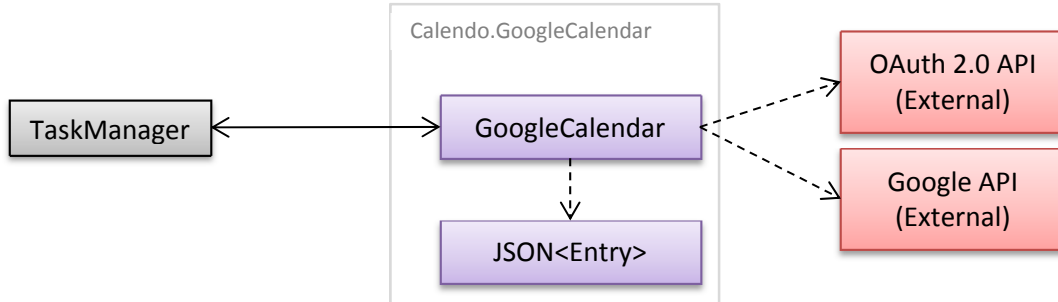


Figure 5.1: Google Calendar Component Overview

### 5.1. GoogleCalendar

#### 5.1.1. GoogleCalendar Methods

Method Name	Description
Authorize(): void	Request authorization of Google Account.
Export(): bool	Export tasks to Google Calendar.
Import(): bool	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 Export() 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.

The **Google Calendar** component depends on the JSON class in the Data component to serialize and deserialize objects. The JSON (JavaScript Object Notation) format is used by the Google API to represent runtime objects in a format suitable for web requests or responses.

### 5.2. JSON<T>

The JSON class is a JSON parser which converts objects to and from the JavaScript Object Notation format. This class does not have other external dependencies.

### 5.2.1. JSON<T> Methods

Method Name	Description
JSON()	Constructor. Initializes serialization tools.
Serialize(T obj): string	Converts object to JSON format.
Deserialize(string json): T	Converts JSON to an object.
DateToJSON(DateTime date): string	Converts DateTime to JSON format.
JSONToDate(string json): DateTime	Converts JSON to DateTime format.

## 6. Testing

Automated testing is performed via a separate project solution, which is located in CalendoUnitTests. Visual Studio 2010 Professional or Visual Studio 2012 Express is required for unit testing.

**Note:** The solution file for unit testing is separate from that of the main Calendo project solution.

### 6.1. Data Storage

The following tests are supported by the Data Storage component. The Data Storage component consists of the Storage and StateStorage classes.

Test Name	Description
DataEntry	Tests if entries can be modified.
DataIncompatible	Tests situations where the data file is corrupted.
DataLoad	Tests if entries can be loaded from file.
DataSave	Tests if entries persist after saving.
DataState	Tests State class functionality.
DataStateStorage	Tests StateStorage class functionality.
DataUnwritable	Tests situations where the data file is unreadable/locked.

### 6.2. Logic

The following tests are supported by the SettingsManager class.

Test Name	Description
SettingsAdd	Tests if settings can be added.
SettingsLoad	Tests if settings can be loaded from file.
SettingsModify	Tests if settings can be modified and persist after saving.

The following tests are supported by the TaskManager class.

Test Name	Description
TaskAdd	Tests if entries can be added.
TaskAddInvalid	Tests if malformed entries can be handled properly.
TaskChange	Tests if entries can be modified.
TaskChangeInvalid	Tests if malformed change requests can be handled properly.
TaskCreate	Tests if TaskManager can be initialized.
TaskGoogleCalendar	Tests multithreading of Google Calendar.
TaskRemove	Tests if entries can be removed.
TaskSubscriber	Tests the subscriber functionality.
TaskUndoRedo	Tests the undo and redo functionality.

The following tests are supported by the CommandProcessor class.

Test Name	Description
CommandAdd	Tests the add command.

Test Name	Description
CommandChange	Tests the change command.
CommandDelete	Tests the remove command.
CommandInvalid	Tests involving invalid commands.

### 6.3. Google Calendar

The following tests are supported by the Google Calendar component.

Test Name	Description
GCJSON	Tests the JSON serialization and deserialization methods.
GCJSONTime	Tests JSON time conversion.

### 6.4. User Interface

The following tests are supported by the User Interface component.

Test Name	Description
UiAutoSuggestMatchAlias	Tests Auto Suggest's matching of input to an alias.
UiAutoSuggestEntryHasNoAliases	Tests the HasAlias property of AutoSuggestEntry, for an instance with no aliases.
UiAutoSuggestEntryHasAliases	Tests the HasAlias property of AutoSuggestEntry, for an instance with aliases.
UiAutoSuggestSetSuggestionFullCommand	Tests Auto Suggest's matching of input to a command.
UiAutoSuggestSetSuggestionSearchString	Tests Auto Suggest's handling of input intended as a search query.
UiAutoSuggestSetSuggestionCommandAndWords	Tests Auto Suggest's matching of input to a command guide (i.e. when a command has been entered and instructions should be displayed)
UiOverdueTestDeadlineTaskOverdueJustNow	Tests the method used to determine if a task is overdue, using a deadline task which is 1 second overdue.
UiOverdueTestDeadlineTaskNotOverdueVerySoon	Tests the method used to determine if a task is overdue, using a deadline task which will be overdue in 1 second.
UiOverdueTestTimedTaskOverdueJustNow	Tests the method used to determine if a task is overdue, using a timed task which is 1 second overdue.
UiOverdueTestTimedTaskNotOverdueVerySoon	Tests the method used to determine if a task is overdue, using a timed task which will be overdue in 1 second.
UiOverdueTestFloatingTask	Tests the method used to determine if a task is overdue, using a floating task.
UiOngoingTestDeadlineTaskOngoingNow	Tests the method used to determine if a task is ongoing (active), using a deadline task which will be overdue in 1 second.
UiOngoingTestDeadlineTaskOngoingUnder24Hours	Tests the method used to determine if a task is ongoing (active), using a deadline task which has just become active.
UiOngoingTestDeadlineTaskNotOngoing24Hours	Tests the method used to determine if a task is ongoing (active), using a deadline task which is about to become active.
UiOngoingTestDeadlineTaskNotOngoingPast	Tests the method used to determine if a task is ongoing (active), using a deadline task which is overdue.



Test Name	Description
UiCompareTaskFloatingOverdue	Tests the method used to compare two tasks, comparing a floating task and an overdue task.
UiCompareTaskBothFloating	Tests the method used to compare two tasks, comparing two floating tasks.
UiCompareTaskDifferentDates	Tests the method used to compare two tasks, comparing two timed tasks with different dates.
UiCompareTaskSameDates	Tests the method used to compare two tasks, comparing two timed tasks with the same date.
UiTaskFloating	Tests the method used to determine if a task is a floating task, using a floating task.
UiTaskFloatingDeadline	Tests the method used to determine if a task is a floating task, using a deadline task.
UiTaskFloatingTimed	Tests the method used to determine if a task is a floating task, using a timed task.
UiTimedTaskOngoing	Tests the method used to determine if a task is ongoing (active), using timed tasks which are ongoing.
UiTimedTaskNotOngoing	Tests the method used to determine if a task is ongoing (active), using timed tasks which are not ongoing.
UiStringArrayToStringTestValidArray	Tests StringArrayToStringConverter with a valid array of strings.
UiStringArrayToStringTestNullArray	Tests StringArrayToStringConverter with a null object.
UiEntryToDateTimeStringTestStartDateToday	Tests EntryToDateTimeStringConverter with a task due on the current date.
UiEntryToDateTimeStringTestStartDateTomorrow	Tests EntryToDateTimeStringConverter with a task due on the following day.
UiEntryToDateTimeStringTestTime	Tests EntryToDateTimeStringConverter with a task due at a specific time.
UiEntryToDateTimeStringTestDateTime	Tests EntryToDateTimeStringConverter with a task due at a specific date and time.
UiEntryToDateTimeStringTestDateTimeEndDate	Tests EntryToDateTimeStringConverter's handling of end date conversion, with a task due at a specific date and time.
UiEntryToDateTimeStringTestDateTimeToday	Tests EntryToDateTimeStringConverter with a task due on the current date at a specific time.
UiEntryToDateTimeStringTestDateTimeTomorrow	Tests EntryToDateTimeStringConverter with a task due on the following day at a specific time.
UiEntryToDateTimeStringTestDate	Tests EntryToDateTimeStringConverter with a task due at a specific date.
UiEntryToDateTimeStringTestNone	Tests EntryToDateTimeStringConverter with a task that has no date or time associated with it.
UiEntryToBrushOngoing	Tests EntryToBrushConverter with an ongoing task.
UiEntryToBrushOverdue	Tests EntryToBrushConverter with an overdue task.
UiEntryToBrushNormal	Tests EntryToBrushConverter with a task that is neither ongoing nor overdue.
UiEntryToVisibilityStackPanelBothFormatsValid	Tests EntryToDateTimeVisibilityConverter when it is used to determine if the date line should be displayed, with a task that has both a start and end date.
UiEntryToVisibilityStackPanelStartNone	Tests EntryToDateTimeVisibilityConverter with a task that has no start date, but has an end date.

Test Name	Description
UiEntryToVisibilityStackPanelEndNone	Tests EntryToDateTimeVisibilityConverter with a task that has a start date, but no end date.
UiEntryToVisibilityStackPanelBothNone	Tests EntryToDateTimeVisibilityConverter when it is used to determine if the date line should be displayed, with a task that has no start date and no end date.
UiEntryToVisibilityRangeTextVisible	Tests EntryToDateTimeVisibilityConverter when it is used to determine if the “to” text should be displayed, with a task that has an end date.
UiEntryToVisibilityRangeTextCollapsed	Tests EntryToDateTimeVisibilityConverter when it is used to determine if the “to” text should be displayed, with a task that has no end date.
UiViewModelAutoSuggestRowTest	Used to test UiViewModel’s AutoSuggestRow property.
UiViewModelSorterTest	Used to test UiViewModel’s task sorting.

## 7. Change log

### 7.1. Current

Version 0.5 is a production release build.

- Increased code coverage for automated tests.
- Fixes implemented for several components.

### 7.2. Version 0.4

Version 0.4 is a maintenance build. Major changes to its feature set are not expected.

- Refactoring has been performed on several components used by the application.
- In-line task editing has been added.
- Task sorting behavior has been updated to sort by type, followed by time, followed by description.
- **Google Calendar** import and export functionality has been refined. Both import and export operations will override existing tasks.

### 7.3. Version 0.3

Version 0.3 had several changes to its feature set since version 0.2.

- **TaskManager** and **DebugTool** have a notification functionality based on the subscriber pattern.
- **TaskManager** is now a singleton class to allow for concurrent data modification.
- **StateStorage** is a subclass of **Storage**.
- **Google Calendar** export and import functionality has been implemented.
- **Google Calendar** is multi-threaded to allow asynchronous operations for faster performance.
- **User Interface** has been changed to increase usability and user appeal.
- Main feature has been changed to **User Interface**.
- Changes to code to improve readability.
- Part of UI functionality has been moved to **UiViewModel**, loosely based upon the MVVM (Model-View-ViewModel) pattern.

#### 7.4. Version 0.2

Version 0.2 had several architecture changes since version 0.1, some of which are influenced by external feedback.

- Both TaskManager and SettingsManager classes have been moved to the **Logic** namespace.
- CommandProcessor class has been moved to the **Logic** namespace.
- The early stages of the AutoSuggest feature have been implemented.
- Several unit tests have been added, particularly those concerning the **Logic** namespace.
- Portions of the **Google Calendar** component, particularly those concerning authentication, has been integrated to the main application.
- Redundant dependencies in **Data** namespace have been reduced.
- JSON parser has been added to the **Data** namespace.
- TimeConverter class has been added to **Logic** namespace, and subsumes time conversion function of TaskManager class.
- **User Interface** has been changed to increase usability and user appeal.

#### 7.5. Version 0.1

Version 0.1 is the initial build.

- Early progress has been made on the **Data** component.
- A prototype has been made for the **User Interface** component.
- Basic command processing has been implemented.

## 8. 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/>