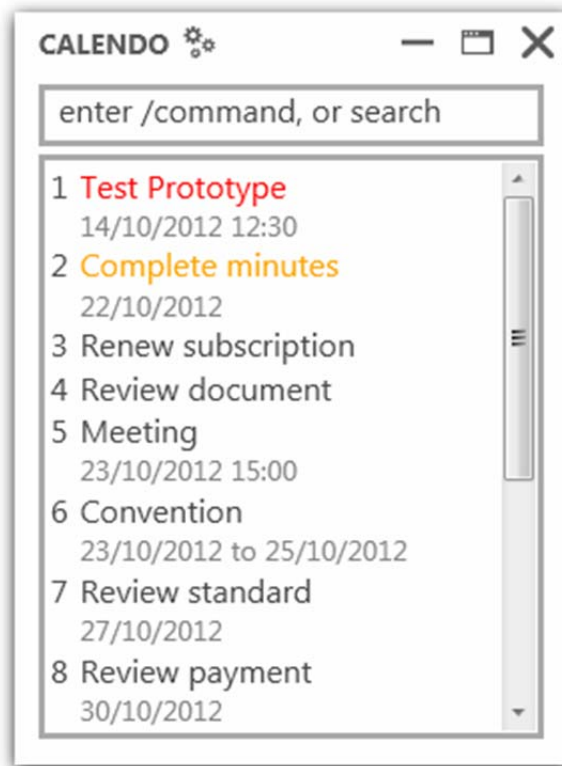






Calendo



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

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.

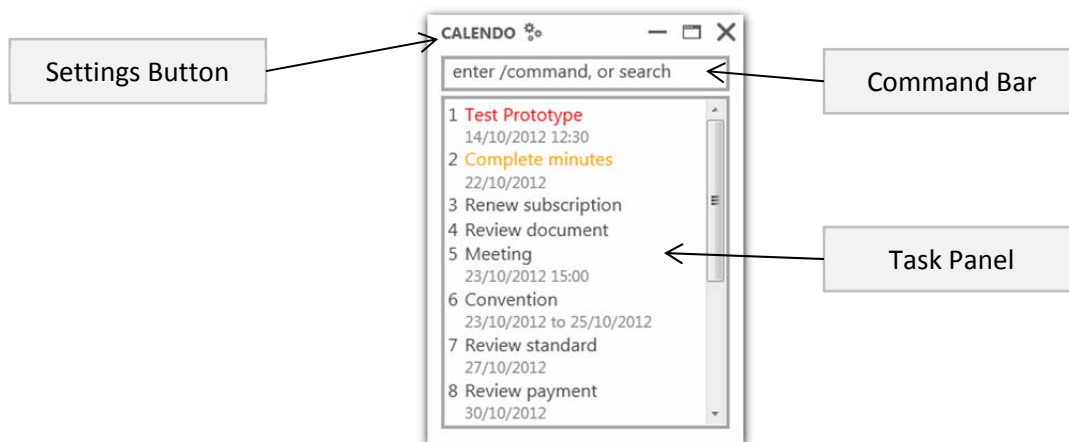


Figure 2.1: User Interface

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.

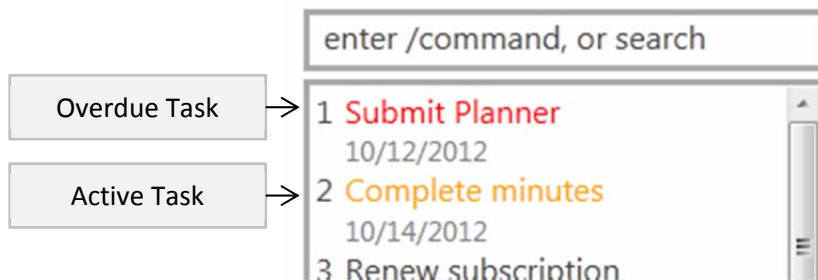


Figure 2.3: Task notification for Deadline Tasks

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.

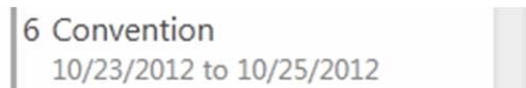


Figure 2.4: Timed Task

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.

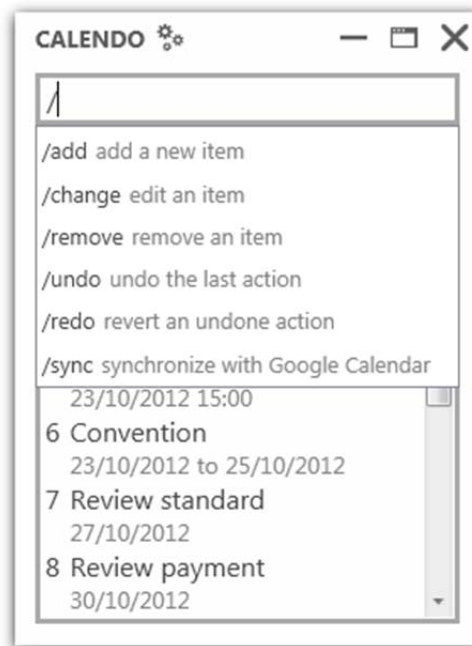


Figure 2.4: Auto suggest

3. Quick start

3.1. Setting up Calendo

Calendo does not require installation. Simply run the executable at where it is. If it Calendo is run for the first time, it would 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 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.

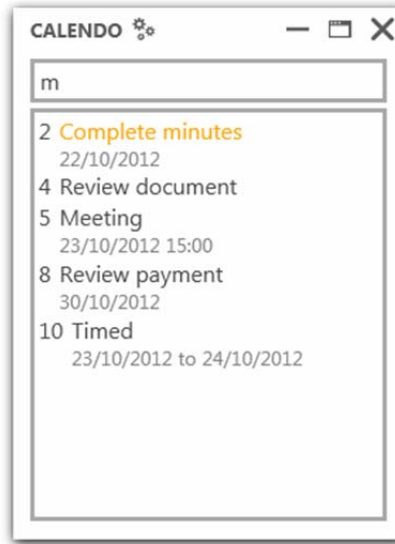


Figure 3.1: Searching for tasks

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`

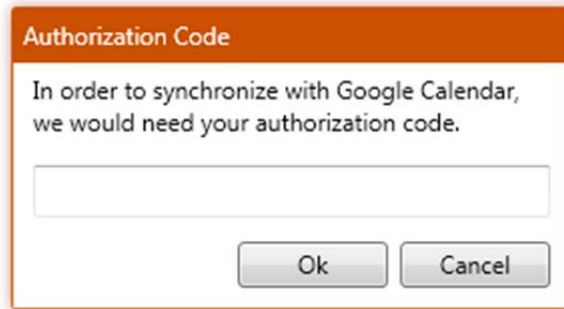
Example: To undo the undo operation
`/redo`

3.7. Synchronize with Google Calendar

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`

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 single-line text input field. At the bottom right are two buttons: "Ok" and "Cancel".

Figure 3.2: Authorization form

Developer Manual

Table of Contents

1. Introduction	10
1.1. Component Overview	10
2. User Interface	11
2.1. MainWindow	11
2.2. EntryToBrushConverter	12
2.3. EntryToDateTimeStringConverter	12
2.4. EntryToDateTimeVisibilityConverter	13
3. Logic	14
3.1. CommandProcessor	15
3.2. TaskManager	15
3.3. SettingsManager	17
4. Data Storage	18
4.1. Storage<T>	18
4.2. StateStorage<T>	19
4.3. Data<T>	21
4.4. State<T>	21
5. Google Calendar	23
5.1. GoogleCalendar	23
7. Testing	25
8. Change Log	26
9. References	27

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, 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 14): 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 18): 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 23): The **Google Calendar** component handles synchronization with the Google Calendar web service, including authentication.

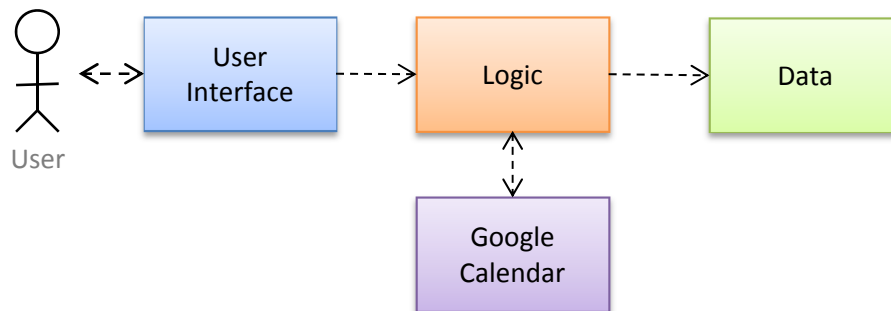


Figure 1.1: Architecture Overview

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

2.1.2. MainWindow Methods

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

Method Name	Description
<code>KeyEventArgs e)</code>	Command Bar. Used to send user input to <code>CommandProcessor</code> as well as filter the task list.
<code>UndoHandler(object sender, ExecutedRoutedEventArgs e)</code>	Event handlers triggered when the key combinations Ctrl+Z or Ctrl+Y are pressed, used to Undo or Redo the last action.
<code>RedoHandler(object sender, ExecutedRoutedEventArgs e)</code>	
<code>ItemsListDoubleClick (object sender, SelectionChangedEventArgs e)</code>	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.
<code>WindowStateChange(object sender, EventArgs e)</code>	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 color 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
<code>Convert(object value, Type targetType, object parameter, CultureInfo culture)</code>	Takes in an Entry, processes it, and returns a Brush with a colour appropriate for the entry’s status.
<code>IsTaskOverdue(Entry currentEntry)</code>	Takes in an Entry, returns <code>true</code> if it is overdue, <code>false</code> otherwise.
<code>IsTaskOngoing(Entry currentEntry)</code>	Takes in an Entry, returns <code>true</code> if it is ongoing, <code>false</code> otherwise.
<code>ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)</code>	Unimplemented Interface member. Will throw a <code>NotImplementedException</code> 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 NotSupportedException 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 NotSupportedException if called.

3. Logic

The **logic** component is used to perform commands issued by the user. The component is located in the `Calendo.Logic` namespace.

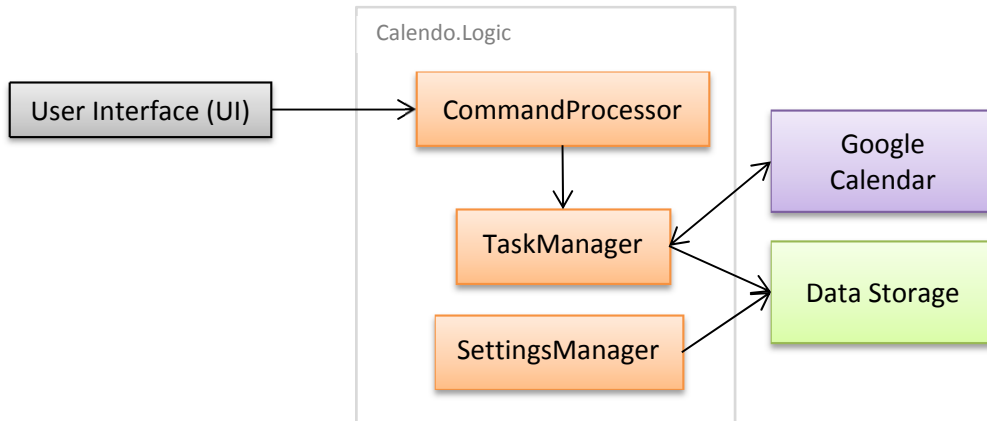


Figure 3.1: Task Manager Component Overview

When a user enters a command into the command box in the **User Interface**, the command gets passed into **CommandProcessor** as a string. **CommandProcessor** decides which action to take, and 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.

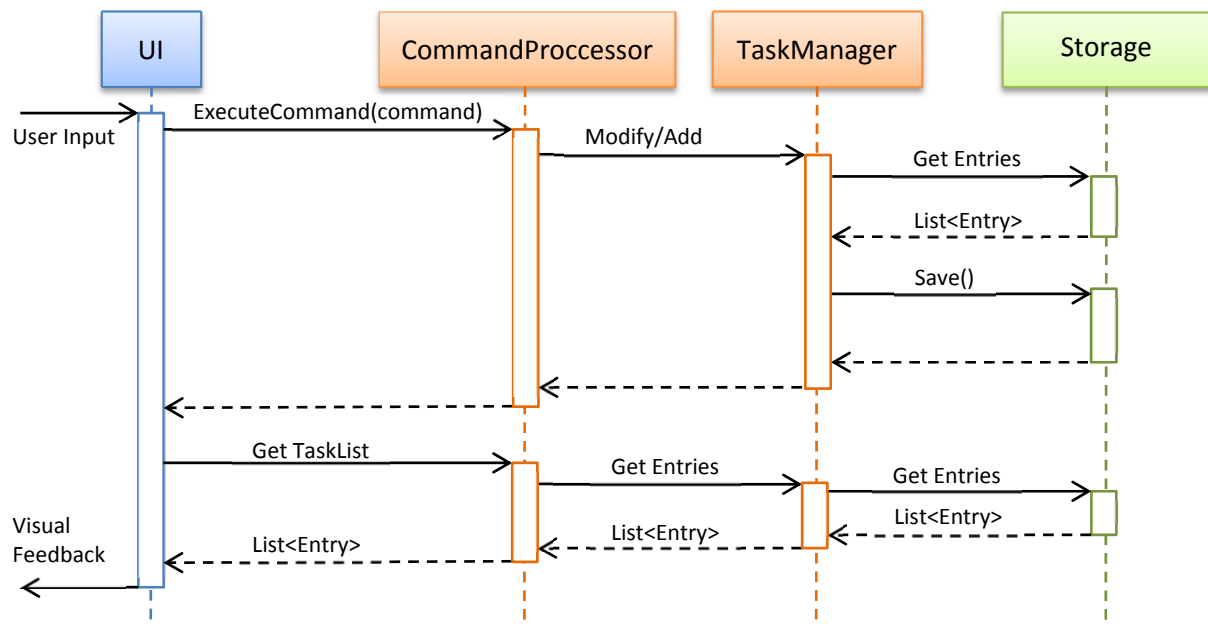


Figure 3.2: General processing of user input

3.1. CommandProcessor

The CommandProcessor class is used for processing user input, and determines what actions are to be performed based on the provided input. 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: 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.

3.2. TaskManager

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

3.2.1. TaskManager Properties

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

3.2.2. TaskManager Methods

Method Name	Description
<code>TaskManager()</code>	Private constructor. Load entries from archive file. To obtain an instance of TaskManager, use the Instance property.
<code>Add(string description, string startDate, string startTime, string endDate, string endTime): void</code>	Adds a task.
<code>AddSubscriber(Delegate handler): void</code>	Adds a handler to the list of subscribers.
<code>Change(int id, string description, string startDate, string startTime, string endDate, string endTime): void</code>	Change parameters of a task. Parameters that are empty would be ignored.
<code>Get(int id): Entry</code>	Get the task by ID.
<code>Redo(): void</code>	Revert changes performed by undo
<code>Remove(int id): void</code>	Remove a task by ID.
<code>Import(): void</code>	Import tasks from Google Calendar.
<code>Export(): void</code>	Export tasks from Google Calendar.
<code>Undo(): void</code>	Undo the last operation (excluding itself).

3.2.3. 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 many 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.

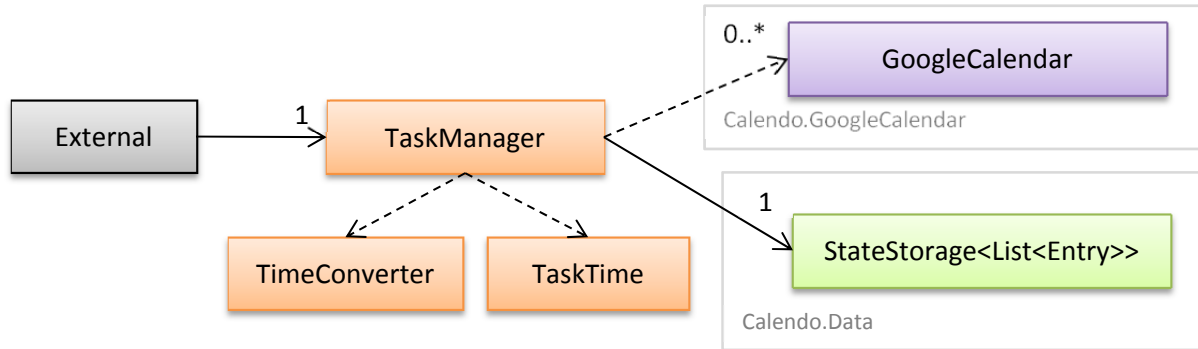


Figure 3.3: TaskManager Dependencies

3.3. SettingsManager

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

3.3.1. SettingsManager Properties

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

3.3.2. 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 overwritten.

3.3.3. 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 and is only used when loading and saving settings.

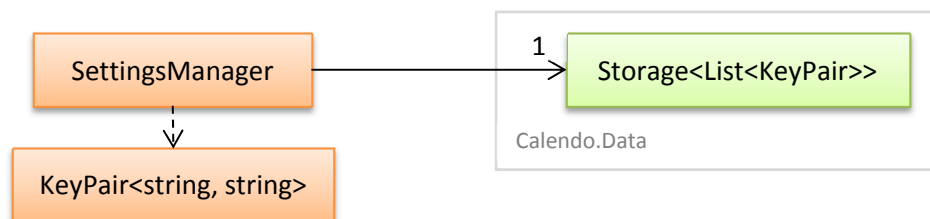


Figure 3.4: SettingsManager Dependencies

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.

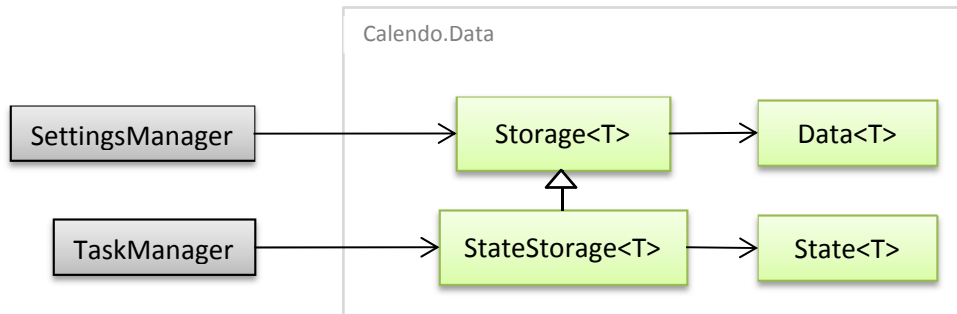


Figure 4.1: Data Storage component overview

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.

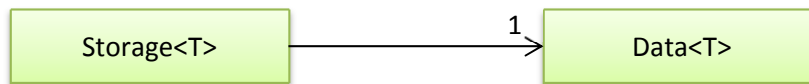


Figure 4.2: Storage<T> Dependencies

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.

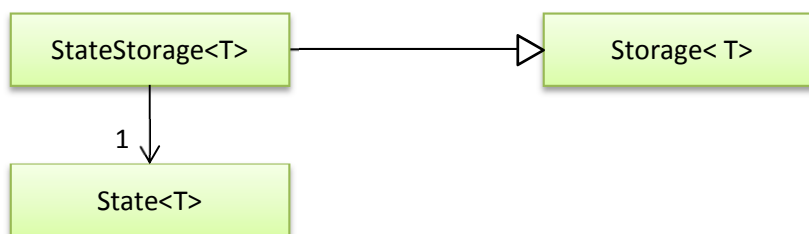


Figure 4.3: StateStorage<T> Dependencies

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.

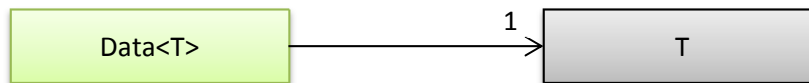


Figure 4.4: State<T> Dependencies

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 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<Data<T>></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.

Property Name	Description
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(): <code>void</code>	Adds a state.
Redo(): <code>bool</code>	Revert to a state in the redo stack. Returns true if the current state has changed, false otherwise.
Undo(): <code>bool</code>	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.

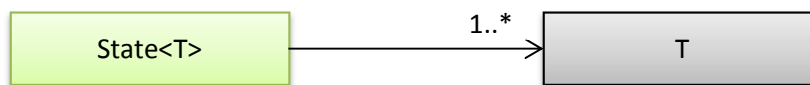


Figure 4.5: State<T> Dependencies

5. Google Calendar

The **Google Calendar** component is responsible for synchronizing the user's tasks with his/her Google Account.

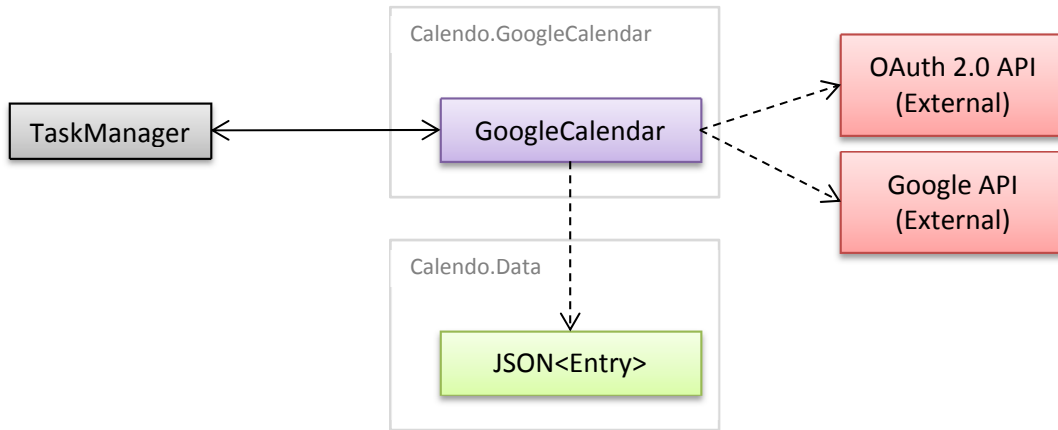


Figure 5.1: Google Calendar Component Overview

5.1. GoogleCalendar

5.1.1. GoogleCalendar Methods

Method Name	Description
Sync(List<Entry> entries): bool	Gets list of tasks and adds it to Google Calendar.
Authorize(): void	Authorizes user's Google Account.
GetAuthentication(NativeApplicationClient provider): string	Returns access token for authentication.
PostTasks(Entry entry, string access_token): bool	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.

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.

7. 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.

Unit testing is currently supported by the Data Storage and Logic components.

7.1. 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.
DataJSON	Tests the JSON serialization and deserialization methods.

7.2. Logic

The following tests are supported by the SettingsManager class.

Test Name	Description
SMAAdd	Tests if settings can be added.
SMLoad	Tests if settings can be loaded from file.
SMMModify	Tests if settings can be modified and persist after saving.

The following tests are supported by the TaskManager class.

Test Name	Description
TMAdd	Tests if entries can be added.
TMAddInvalid	Tests if malformed entries can be handled properly.
TMChange	Tests if entries can be modified.
TMRemove	Tests if entries can be removed.
TMCreate	Tests if TaskManager can be initialized.
TMUndoRedo	Tests the undo and redo functionality.

8. Change log

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 UIViewController, based upon the MVC (Model View Control) pattern.

9. 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/>