# Welcome to gkeepapi's documentation!

**Contents**



**gkeepapi** is an unofficial client for programmatically interacting with Google Keep:

```
keep = gkeepapi.Keep()
keep.login('...', '...')

note = gkeepapi.createNote('Todo', 'Eat breakfast')
note.pinned = True
note.color = gkeepapi.node.ColorValue.Red

keep.sync()
```

```
print note.title
print note.text
```

The client is mostly complete and ready for use, but there are some hairy spots. In particular, the interface for manipulating labels and blobs is subject to change.

# Client Usage

All interaction with Google Keep is done through a **`Keep`** object, which is responsible for authenticating, syncing changes and tracking modifications.

## Logging in

gkeepapi leverages the mobile Google Keep API. To do so, it makes use of **`gpsoauth`**, which requires passing in the username and password. This was necessary as the API we're using is restricted to Google applications (put differently, there is no way to enable it on the Developer Console):

```
keep = gkeepapi.Keep()
keep.login('...', '...')
```

To reduce the number of logins you make to the server, you can store the master token after logging in. Protect this like a password, as it grants full access to your account:

```
import keyring
# <snip>
token = keep.getMasterToken()
keyring.set_password('google-keep-token', username, token)
```

You can load this token at a later point:

```
import keyring
# <snip>
token = keyring.get_password('google-keep-token', username)
keep.resume(email, master_token)
```

Note: Enabling TwoFactor and logging via an app password is recommended.

## Syncing

gkeepapi automatically pulls down all notes after login. It takes care of refreshing API tokens, so there's no need to call **`Keep.login()`** again. After making any local modifications to notes, make sure to call **`Keep.sync()`** to update them on the server!:

```
keep.sync()
```

## Caching notes

The initial sync can take a while, especially if you have a lot of notes. To mitigate this, you can serialize note data to a file. The next time your program runs, it can resume from this state. This is handled via **`Keep.dump()`** and **`Keep.restore()`**:

```
# Store cache
state = keep.dump()
fh = open('state', 'w')
```

```
json.dump(state, fh)

# Load cache
fh = open('state', 'r')
state = json.load(fh)
keep.restore(state)
```

You can also pass the state directly to the **`Keep.login()`** and **`Keep.resume()`** methods:

```
keep.login(username, password, state=state)
keep.resume(username, master_token, state=state)
```

# Notes and Lists

Notes and Lists are the primary types of notes visible to a Google Keep user. gkeepapi exposes these two notes via the **`node.Note`** and **`node.List`** classes. For Lists, there's also the **`node.ListItem`** class.

## Creating Notes

New notes are created with the **`Keep.createNote()`** and **`Keep.createList()`** methods. The **`Keep`** object keeps track of these objects and, upon **`Keep.sync()`**, will sync them if modifications have been made:

```
gnote = keep.createNote('Title', 'Text')

glist = keep.createList('Title', [
    ('Item 1', False), # Not checked
    ('Item 2', True)  # Checked
])

# Sync up changes
keep.sync()
```

## Getting Notes

Notes can be retrieved via **`Keep.get()`** by their ID (visible in the URL when selecting a Note in the we-bapp):

```
gnote = keep.get('...')
```

To fetch all notes, use **`Keep.all()`**:

```
gnotes = keep.all()
```

## Searching for Notes

Notes can be searched for via **`Keep.find()`**:

```
# Find by string
gnotes = keep.find(query='Title')

# Find by filter function
gnotes = keep.find(func=lambda x: x.deleted and x.title == 'Title')

# Find by labels
gnotes = keep.find(labels=[keep.findLabel('todo')])
```

```
# Find by colors
gnotes = keep.find(colors=[gkeepapi.node.ColorValue.White])

# Find by pinned/archived/trashed state
gnotes = keep.find(pinned=True, archived=False, trashed=False)
```

# Manipulating Notes

Note objects have many attributes that can be directly get and set. Here is a non-comprehensive list of the more interesting ones.

Notes and Lists:

- **`node.TopLevelNode.id`** (Read only)
- **`node.TopLevelNode.parent`** (Read only)
- **`node.TopLevelNode.title`**
- **`node.TopLevelNode.text`**
- **`node.TopLevelNode.color`**
- **`node.TopLevelNode.archived`**
- **`node.TopLevelNode.pinned`**
- **`node.TopLevelNode.labels`**
- **`node.TopLevelNode.annotations`**
- **`node.TopLevelNode.timestamps`** (Read only)
- **`node.TopLevelNode.collaborators`**
- **`node.TopLevelNode.blobs`** (Read only)

ListItems:

- **`node.TopLevelNode.id`** (Read only)
- **`node.TopLevelNode.parent`** (Read only)
- **`node.TopLevelNode.parent_item`** (Read only)
- **`node.TopLevelNode.indented`** (Read only)
- **`node.TopLevelNode.text`**
- **`node.TopLevelNode.checked`**

## Getting Note content

Example usage:

```
print gnote.title
print gnote.text
```

## Getting List content

Retrieving the content of a list is slightly more nuanced as they contain multiple entries. To get a serialized version of the contents, simply access **`node.List.text`** as usual. To get the individual **`node.ListItem`** objects, access **`node.List.items`**:

```
# Serialized content
print glist.text

# ListItem objects
glistitems = glist.items

# Checked ListItems
cglistitems = glist.checked
```

```
# Unchecked ListItems
uglistitems = glist.unchecked
```

## Setting Note content

Example usage:

```
gnote.title = 'Title 2'
gnote.text = 'Text 2'
gnote.color = gkeepapi.node.ColorValue.White
gnote.archived = True
gnote.pinned = False
```

## Setting List content

New items can be added via **`node.List.add()`**:

```
# Create a checked item
glist.add('Item 2', True)

# Create an item at the top of the list
glist.add('Item 1', True, gkeepapi.node.NewListItemPlacementValue.Top)

# Create an item at the bottom of the list
glist.add('Item 3', True, gkeepapi.node.NewListItemPlacementValue.Bottom)
```

Existing items can be retrieved and modified directly:

```
glistitem = glist.items[0]
glistitem.text = 'Item 4'
glistitem.checked = True
```

Or deleted:

```
glistitem.delete()
```

## Setting List item position

To reposition an item (larger is closer to the top):

```
# Set a specific sort id
glistitem1.sort = 42

# Swap the position of two items
val = glistitem2.sort
glistitem2.sort = glistitem3.sort
glistitem3.sort = val
```

## Indent/dedent List items

To indent a list item:

```
gparentlistitem.indent(gchildlistitem)
```



To dedent:

```
gparentlistitem.dedent(gchildlistitem)
```

## Deleting Notes

The **`node.TopLevelNode.delete()`** method marks the note for deletion (or undo):

```
gnote.delete()
gnote.undelete()
```

To send the node to the trash instead (or undo):

```
gnote.trash()
gnote.untrash()
```

## Getting media

To fetch media (images, audio, etc) files, you can use the **`Keep.getMediaLink()`** method to get a link:

```
blob = gnote.blobs[0]
keep.getMediaLink(blob)
```

# Labels

Labels are short identifiers that can be assigned to notes. Labels are exposed via the **`node.Label`** class. Management is a bit unwieldy right now and is done via the **`Keep`** object. Like notes, labels are automatically tracked and changes are synced to the server.

## Getting Labels

Labels can be retrieved via **`Keep.getLabel()`** by their ID:

```
label = keep.getLabel('...')
```

To fetch all labels, use **`Keep.labels()`**:

```
labels = keep.labels()
```

## Searching for Labels

Most of the time, you'll want to find a label by name. For that, use **`Keep.findLabel()`**:

```
label = keep.findLabel('todo')
```

Regular expressions are also supported here:

```
label = keep.findLabel(re.compile('^todo$'))
```

## Creating Labels

New labels can be created with **`Keep.createLabel()`**:



```
label = keep.createLabel('todo')
```

## Editing Labels

A label's name can be updated directly:

```
label.name = 'later'
```

## Deleting Labels

A label can be deleted with `Keep.deleteLabel()`. This method ensures the label is removed from all notes:

```
keep.deleteLabel(label)
```

## Manipulating Labels on Notes

When working with labels and notes, the key point to remember is that we're always working with `node.Label` objects or IDs. Interaction is done through the `node.NodeLabels` class.

To add a label to a note:

```
gnote.labels.add(label)
```

To check if a label is on a note:

```
gnote.labels.get(label.id) != None
```

To remove a label from a note:

```
gnote.labels.remove(label)
```

# Constants

- `node.ColorValue` enumerates valid colors.
- `node.CategoryValue` enumerates valid note categories.
- `node.CheckedListItemsPolicyValue` enumerates valid policies for checked list items.
- `node.GraveyardStateValue` enumerates valid visibility settings for checked list items.
- `node.NewListItemPlacementValue` enumerates valid locations for new list items.
- `node.NodeType` enumerates valid node types.
- `node.BlobType` enumerates valid blob types.
- `node.RoleValue` enumerates valid collaborator permissions.
- `node.ShareRequestValue` enumerates vaild collaborator modification requests.
- `node.SuggestValue` enumerates valid suggestion types.

# Annotations

READ ONLY TODO

# Settings

TODO

# Collaborators

Collaborators are users you've shared notes with. Access can be granted or revoked per note. Interaction is done through the **`node.NodeCollaborators`** class.

To add a collaborator to a note:

```
gnote.collaborator.add(email)
```

To check if a collaborator has access to a note:

```
email in gnote.collaborator.all()
```

To remove a collaborator from a note:

```
gnote.collaborator.remove(email)
```

# Timestamps

All notes and lists have a **`node.NodeTimestamps`** object with timestamp data:

```
node.timestamps.created
node.timestamps.deleted
node.timestamps.trashed
node.timestamps.updated
node.timestamps.edited
```

These timestamps are all read-only.

# FAQ

1. I get a "NeedsBrowser" *exception.APIException* when I try to log in.

Your account probably has Two Factor enabled. To get around this, you'll need to generate an App Password for your Google account.

2. I get a "CaptchaRequired" *exception.LoginException* when I try to log in.

If you're using Python 2.x, try switching to Python 3.x. See this issue for more information.

# Known Issues

The **`Keep`** class isn't aware of new **`node.ListItem`** objects till they're synced up to the server. In other words, **`Keep.get()`** calls for their IDs will fail.

# Debug

To enable development debug logs:

```
gkeepapi.node.DEBUG = True
```

# Notes

- Many sub-elements are read only.
- **`node.Node`** specific **`node.NewListItemPlacementValue`** settings are not used.

## Reporting errors

Google occasionally ramps up changes to the Keep data format. When this happens, you'll likely get a **`exception.ParseException`**. Please report this on Github with the raw data, which you can grab like so:

```
try:
    # Code that raises the exception
except gkeepapi.exception.ParseException as e:
    print(e.raw)
```

If you're not getting an **`exception.ParseException`**, just a log line, make sure you've enabled debug mode.

# Indices and tables

- Index
- Module Index
- Search Page