Simplenote API2

Documentation **v2.1.3**: (April 18, 2011).

Recent documentation changes are listed on the last page.

Contents

Introduction & Basics 3
- Technical Foundation 3

Authentication 4
- Obtaining a token 4
- Using the token 4
- Security 4

Working with notes 5
- Note object (JSON): 5
- Creating a note 7
- Retrieving a note 7
- Updating a note 7
- Deleting a note 8

Getting the note index 9

Tags API 11
- Tag Index 11
  - GET Parameters 11
  - The "index" field 12
- Tag Names 12
- Create a tag 13
  - Request 13
  - Response 13
- Retrieve tag info 13
  - Request 13
  - Return 13
- Update tag info (change case and/or index): 13
  - Request 13
  - Response 13

Delete a tag from the index 13
Request 13
Response 13
Group (or shared) tags 14
Sharing 14
Caveats 14
API Examples 15
Checking for changes 19
Psuedo-code algorithm for syncing 19
Response codes 20
Recent Documentation Changes 21

Introduction & Basics

This is the second major version of the API for Simplenote, a platform for notes that supports sharing, tagging, and trashing. By using this API, you can create a client that reads and/or writes notes to a Simplenote user's account. Simplenote is also the name of our iOS app that uses the Simperium platform, and of our web app at http://simple-note.appspot.com.

Please take advantage of our Simplenote developer mailing list:
http://groups.google.com/group/simplenote-api/

This mailing list is used for:

- As a resource for you to look for solutions to problems you may encounter while developing this API
- Third-party developers to share tips, tricks, or libraries
- Any questions you may have regarding the API
- Gathering feedback regarding this API and the Simplenote ecosystem

Additionally, anyone interested in the next-generation Simperium platform is encouraged to sign up for an eventual announcement at http://simperium.com.

Technical Foundation

The Simplenote API v2 makes heavy use of JSON. Strings should be in the UTF-8 charset. You may also have to work with Base 64 encoding of data.

This API uses the HTTP verbs GET and POST, and communicates failure through use of HTTP status codes (see the section "Response Codes").

The base URL for all API calls is:
`https://simple-note.appspot.com/`
Thus if a method described in this document says to "GET /api2/tags," for example, that means you should make a GET request to the URL:
`https://simple-note.appspot.com/api2/tags`
(However, you may opt to use HTTP instead of HTTPS for any calls except login.)

Finally, we do request that your app identify itself via the HTTP `User-Agent` header:
`User-Agent: MyApp/0.5.0`
Note that depending on your app's framework, your app may already be setting this header appropriately.

Authentication

Simplenote uses a simple token based authentication system. All methods will require a valid authentication token to process.

Obtaining a token

Call the `/api/login` method supplying two fields, email and password. These should be the same as what you created your account with in the Simplenote application. The server will respond by attempting to set a cookie named `auth` with the token as the value. Also, the HTTP response body will contain the token.

HTTP POST to the following URL:

```
https://simple-note.appspot.com/api/login
```

(Note this is the only method in this document which begins `/api/` instead of `/api2/`.)

The body of the request should contain this, but base64-encoded:

```
email=[email address]&password=[password]
```

To be clear, you will concatenate `email=` with the user's email address (trimmed), `&password=`, and the user's password; then base64 the resulting string and send it as the HTTP request body. (Do not encode this request using the standard variable encoding used for POSTed web forms. We basically ignore the `Content-Type` header for this request, but `text/plain` makes the most sense here.)

Using the token

Subsequent calls to the API methods will check that either the 'auth' cookie is present with a valid token or that the token is supplied as a query parameter (&auth=[token]). Once issued, the token is valid for 24 hours. The email address should also be passed with each request either as a query parameter or as a cookie.

Security

The login method can only be accessed through HTTPS.

Working with notes

Each note has a unique string associated with it; using this `key`, individual notes can be modified, created, retrieved, or deleted. Once a note has been created it resides at the note URL and can be modified using that URL (`/api2/data/[note key]`).

Note object (JSON):

```
{
      key : (string, note identifier, created by server),
      deleted : (integer pseudo-boolean, in trash or not),
      modifydate : (last modified date, in seconds since epoch),
      createdate : (note created date, in seconds since epoch),
      syncnum : (integer, number set by server, track note changes),
      version : (integer, number set by server, track note content
changes),
      minversion : (integer, number set by server, minimum version
available for note),
      sharekey : (string, shared note identifier),
      publishkey : (string, published note identifier),
      systemtags : [ (Array of strings, some set by server) ],
      tags : [ (Array of strings) ],
      content : (string, data content)
}
```

Properties in green can be set by the client. Other properties are returned only by the server.

*New properties may be returned in the note object without notice. Please make sure your code can handle this! We'll make sure to announce any changes for existing properties though.

System Tags

`systemtags` is an array of strings. Each possible string conveys a bit of note meta-information, used primarily to tweak the user interface (UI).

Current possible `systemtags` values are:

- `pinned` — if present, the note is "pinned" should be displayed before unpinned notes in a list of notes. (Pinned notes are displayed at the top of our iOS and web clients' note lists, and we encourage you to do the same in your apps. You could also allow users to pin or unpin notes.)

- `unread` — If the 'unread' system tag is present, it should be removed when that note is read by the user. The `unread` system tag is applied when someone else has modified a shared note.
- `markdown` — indicates that the note is written using Markdown syntax (or a superset; we support the "Markdown Extra" variant, including raw HTML — sanitized, of course). It is up to you if and how to support Markdown preview. In our own web app, we support Markdown mode in this manner:
  - Users can toggle an option labeled "Markdown Formatted."
  - Markdown formatted notes can be toggled between "Edit" and "Preview" (Markdown transformed) modes.
  - Published notes that have been marked as "Markdown Formatted" are Markdown transformed (just as in "preview") when viewed as published pages.
- `list` — Premium users may, in our iOS app, designate a note as a "list" which completely changes the manner in which it is edited and displayed. Items may be easily rearranged and "crossed off". A leading hyphen and space(s) are visually suppressed as redundant (a line that reads "- bananas" will be shown as "bananas"). You are welcome to implement a list mode yourself as a third-party developer.

To add or remove a system tag just do an update of the note with the new `systemtags` array. That said, you should probably not be *adding* the `unread` system tag.

Other note properties: publishing, tags, available versions, trash status

`sharekey`, and `publishkey`, can currently only be generated by the iPad/iPhone apps, and have to do with sharing or publishing by URL; we plan to expose this functionality soon via the API.

`tags` is an array of strings, each representing a tag. Currently no tag should contain any spaces or commas, but this may change in the future. Setting this array changes the tags applied to the note. By tagging a note with an email address or user-created "group tag", the Simplenote backend knows to share the note with those users. Please note that tags are considered case-insensitive. Lastly, please note the order of the tags array has no defined meaning, and may change at any time; but in general, try to maintain the order of the tags array. Please see also the Tags API section of this document.

`minversion` is useful in that it indicates how far back the history of this note goes (the number of old note versions available per note currently depends on whether or not the Simplenote user is a premium user.)

`deleted` acts like a boolean value, but is a JSON number `0` or `1` (**not** JSON `true` or `false`). If `deleted` is 1, the note is in the Trash. (See the section "Deleting a note.")

Creating a note

To create a note, send a POST request to the following URL:

```
https://simple-note.appspot.com/api2/data?auth=[auth token]&email=[email]
```

The body of the POST request should be the new note object. At minimum, the `content` property **must** be supplied. This will create a note and return a note object with a new note identifier (in the `key` property).

Retrieving a note

To retrieve a note, send a GET request to the note URL:

```
https://simple-note.appspot.com/api2/data/[note key]?auth=[auth token]&email=[email]
```

The `key` parameter **must** be supplied, `auth` may be omitted if the '`auth`' cookie is present, likewise for `email`.

The response body will contain a note object.

Updating a note

To update a note, a POST request is made to the note URL:

```
https://simple-note.appspot.com/api2/data/[note key]?auth=[auth token]&email=[email]
```

The body should contain a partial note object. The minimum properties to send should be the `content`. To support merging, last `version` number received from the server for this note **must** be sent. If `version` is omitted, the new contents will automatically overwrite the current note contents, even if changes have been made on other clients.

The response will have the updated note object, including the new `version` and `syncnumber` properties assigned by the server. If no changes have been made by other clients since the last update, then `content` will not be included in the response note object. If there have been other changes then `content` will be returned and the client should update their local store accordingly.

`modifydate` and `createdate` can be set by the client. If not provided, the server will use the current time for these values. Sorting in the Simplenote clients are done based on these values and should be sent especially if the change did not happen at the time of updating the note on the server.

Deleting a note

Simplenote now includes a Trash feature. Deleting a note now means to send a note to the Trash, where it can also be restored.

The `deleted` property of the note object corresponds to whether or not a note is in the Trash. To delete a note, just update a note and set the `deleted` property to `1`. Restoring a note would mean to update and set the `deleted` property back to `0`.

To delete a note from the trash (permanent delete), make a DELETE request to the note URL:

```
https://simple-note.appspot.com/api2/data/[note key]?auth=[auth token]&email=[email]
```

This should not be done except at the user's explicit, confirmed request to permanently delete a note. In general, you should use the Simplenote Trash functionality to allow users to delete notes. Providing an "empty trash" or "permanently delete" functionality is an optional, but nice, extra step after supporting Trash functionality.

Getting the note index

To make it simple to synchronize your notes with other applications, the API provides an easy way to get an index of your notes. This method will return a JSON object with a count of the number of note objects and an array of the note objects themselves. The note objects are the same as retrieved by a normal GET call on a note, with the `content` omitted. Since there is a maximum number of notes that can be returned in one call, if there are more notes in the index, then the `mark` parameter will be set and it should be used in the next index call to continue retrieving notes. For example, if the user has 300 notes, this would require at least 3 calls to retrieve all notes.

To get the index, make a GET request to `/api2/index`. For example:

```
https://simple-note.appspot.com/api2/index?length=[number of notes]&mark=
[bookmark key]&since=[time value]&auth=[auth token]&email=[email]
```

The `auth` parameter is **optional** and is needed only if the `auth` cookie is not present. The `email` parameter is also **optional** and is needed only if the `email` cookie is not present. The `length` parameter determines the maximum number of note objects to return (maximum is 100). If there are more note objects, then the response object will include a `mark` property. This **should** be used as the value of the `mark` GET parameter in a subsequent call to retrieve the next set of note results. (As of Feb. 21, 2011, the `mark` parameter returned in index calls became to be much longer and no longer maps directly to a note key anymore. This change should not affect your app.)

The **optional** `since` parameter is a time value (like `modifydate` or `createdate`) and instead of returning all notes will return all notes that have been changed since the time specified.

Sample Index Response:

```
{
 "count": 1,
 "data":
     [
     {"modifydate": "1285591393.044700",
      "tags": [],
      "deleted": 0,
      "createdate": "1285591393.044700",
      "systemtags": [],
      "version": 1,
      "syncnum": 1,
      "key": "agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA",
      "minversion": 1}
```

```
        ]
}
```

Tags API

The tags API provides a way to retrieve and update a tag list (index of all tags) and provides an easy interface to sharing via the shared tags feature.

Tag Index

This call provides a way to retrieve a list of tags. This will not include tags in notes that have not been updated since the release of Simplenote 3.1. The tags field contains an array of tags objects. Much like the note index call, if there are more tags than are returned, then a "mark" field will be set. There are two ways to create tags, either explicitly using API, or, if a note is updated with a new tag, it will automatically update this index as well.

`GET /api2/tags`

GET Parameters

- (`auth` and `email` as per usual if the equivalent cookies are not present.)
- `length`: # of tags to return (because of a bug in the client, this will return up to 1000 tags currently, but behavior will likely change in the future)
- `mark`: As in `/api2/index`, if there are more tags than can be returned at once, the response will include a `mark` property which should be used as the value of the `mark` GET parameter in a subsequent call to retrieve the next set of tag results.

Sample tag index response (JSON):

```
{
  "count": 3,
  "time": "1297642347.727434",
  "tags": [ {"index": 2, "version": 2, "name": "Work"},
            {"index": 1, "version": 4, "name": "Later"},
            {"index": 0, "version": 4, "name": "Todo"}
          ]
}
```

Sample basic tag object:

```
{"index": 1, "version": 3, "name": "Todo"}
```

Sample group (or shared) tag object:

```
{
  "name"    : "Family",
  "index"   : 1,
  "version" : 2,
  "share"   : [ "mom@example.com", "dad@example.com", "sis@example.com" ]
}
```

The "index" field

The `index` field is used to represent the order the tag appears in, with 0 appearing first. There may be duplicates or -1 values in this field. For users of the Simplenote app, the app will automatically try to sort the tags according to this field and update the index values for any tags that require it (duplicates or -1 values).

Tag Names

Tags are case insensitive at the moment; "todo" is the equivalent of "Todo", though case is preserved when the tag is created. Tags also cannot currently contain spaces, but this may change. Tags should not contain commas, as our Simplenote clients treat commas as delimiters; when the user types a comma, this should indicate the termination of a tag (in the same manner as spaces currently work).

Valid Tag Names	Notes
Todo	
@Todo	
username@example.com	Automatically shares note with user
someone@@example.com	Valid tag (though not treated as email)
Markdown	Normal tag; also triggers Markdown support
markdown	Synonymous with previous tag
.	(period)
Some_Tag	
Contains-Hyphens	
Ελλάδα	

Currently Invalid Tag Names	Notes
My Notes	Contains a space
1,05	Contains a comma, problematic to edit
	(whitespace)
Leading_or_Trailing_Whitespace	Should be trimmed
x@example.com,y@example.com	Should be two separate tags; contains comma.

Create a tag

Request
```
POST /api2/tags

{"name" : "newtag"}
```
Where `newtag` is the name of the tag you are creating.

Response
```
{"name" : "newtag", "index" : 0, "version" : 1}
```

Retrieve tag info

(The tag index at `/api2/tags` returns entire tag objects, so there isn't much need for this call.)

Request
```
GET /api2/tags/newtag
```

Return
```
{"name" : "newtag", "index" : 0, "version" : 1}
```

Update tag info (change case and/or index):

Request
```
POST /api2/tags/newtag

{ "name":"NewTag", "index":10 }
```

Response
```
{"name" : "NewTag", "index" : 10, "version" : 2}
```

Delete a tag from the index

(will not currently remove tag from notes that still have this tag)

Request
```
DELETE /api2/tags/newtag
```

Response
A JSON tag object representing the deleted tag.

Group (or shared) tags

A group tag is a tag that contains a "share" field. The share field is a list of email addresses. If a tag is created and the name looks like an email address, it will be automatically created as a shared tag with "share" containing the email address.

Sharing

Sharing is backwards compatible with the tags field in notes. If your app supports tags then it already supports sharing.
When a note's tags are updated, the server will check the tags with the tag list. If any of these are shared tags, then the note is automatically shared. All share recipients (indicated in 'share' field) will receive an email informing them that a note has been shared with them including a link to shared version of the note. If the recipient is a Simplenote user (their email matches a Simplenote user account), then the note will be automatically created in their account.

Caveats

Batch tag operations are currently not supported by the API.

- Deleting a tag from the index will not also delete the tag from all notes that contain this tag.
- Likewise, updating a previously basic tag to become a shared tag will not trigger sharing for all notes that have that tag.

For these operations it's best to individually update each affected note from the client. Renaming a tag would be accomplished in the same way, for all notes that have the old name, delete the old tag and add the new one. Then delete the tag from the tag index, and create a new tag with the new name.

API Examples

1. Get index

```
GET /api2/index
```

Response:

```
{"count": 0, "data": []}
```

2. Create a note

```
{"content" : "New note!"}
POST /api2/data
```

Response:

```
{"modifydate": "1285591393.044700", "tags": [], "deleted": 0,
"createdate": "1285591393.044700", "systemtags": [], "version": 1,
"syncnum": 1, "key": "agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA",
"minversion": 1}
```

3. Get index again should show newly created note

```
GET /api2/index
```

Response:

```
{"count": 1, "data": [{"modifydate": "1285591393.044700", "tags": [],
"deleted": 0, "createdate": "1285591393.044700", "systemtags": [],
"version": 1, "syncnum": 1, "key":
"agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA", "minversion": 1}]}
```

4. Retrieve note using note url

```
GET /api2/data/agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA
```

Response:

```
{"modifydate": "1285591393.044700", "tags": [], "deleted": 0,
"createdate": "1285591393.044700", "systemtags": [], "content": "New
note!", "version": 1, "syncnum": 1, "key":
"agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA", "minversion": 1}
```

5. Update note with new *content*, server returns new *version* and new *syncnum*

```
{"content" : "New note! with change", "version" : 1}
POST /api2/data/agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA
```

Response:

```
{"modifydate": "1285591647.688616", "tags": [], "deleted": 0,
"createdate": "1285591393.044700", "systemtags": [], "version": 2,
"syncnum": 2, "key": "agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA",
"minversion": 1}
```

6. Send note to trash by setting *deleted* to 1, only *syncnum* is changed because *content* was not modified

```
{"deleted" : 1}
POST /api2/data/agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA
```

Response:

```
{"modifydate": "1285591687.123246", "tags": [], "deleted": 1,
"createdate": "1285591393.044700", "systemtags": [], "version": 2,
"syncnum": 3, "key": "agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA",
"minversion": 1}
```

7. Permanently remove note from trash, only works if note is deleted

```
DELETE /api2/data/agtzaW1wbGUtbm90ZXIKCxIETm90ZRhsDA
```

Response:
Nothing returned, status code 200

8. Create 4 new notes (not shown), then retrieve index contains all 4 notes

```
GET /api2/index
```

Response:

```
{"count": 4, "data": [{"modifydate": "1285617794.872000", "tags": [],
"deleted": 0, "createdate": "1285617793.685000", "systemtags": [],
"version": 1, "syncnum": 1, "key":
"agtzaW1wbGUtbm90ZXIKCxIETm90ZRhvDA", "minversion": 1}, {"modifydate":
"1285617791.608000", "tags": [], "deleted": 0, "createdate":
"1285617790.018000", "systemtags": [], "version": 1, "syncnum": 1,
"key": "agtzaW1wbGUtbm90ZXIKCxIETm90ZRhwDA", "minversion": 1},
{"modifydate": "1285617788.655000", "tags": [], "deleted": 0,
"createdate": "1285617787.044000", "systemtags": [], "version": 2,
"syncnum": 2, "key": "agtzaW1wbGUtbm90ZXIKCxIETm90ZRhtDA",
"minversion": 1}, {"modifydate": "1285617783.647000", "tags": [],
"deleted": 0, "createdate": "1285617780.555000", "systemtags": [],
"version": 1, "syncnum": 1, "key":
"agtzaW1wbGUtbm90ZXIKCxIETm90ZRhuDA", "minversion": 1}]}
```

9. Retrieve index again using *length* = 2, response contains only 2 notes and also *mark* parameter to continue retrieving notes

```
GET /api2/index?length=2
```

Response:

```
{"count": 2, "data": [{"modifydate": "1285617794.872000", "tags": [],
"deleted": 0, "createdate": "1285617793.685000", "systemtags": [],
"version": 1, "syncnum": 1, "key":
"agtzaW1wbGUtbm90ZXIKCxIETm90ZRhvDA", "minversion": 1}, {"modifydate":
"1285617791.608000", "tags": [], "deleted": 0, "createdate":
"1285617790.018000", "systemtags": [], "version": 1, "syncnum": 1,
"key": "agtzaW1wbGUtbm90ZXIKCxIETm90ZRhwDA", "minversion": 1}],
"mark": "agtzaW1wbGUtbm90ZXIKCxIETm90ZRhtDA"}
```

10. Continue retrieving notes using bookmark *mark*. Response contains remaining 2 notes and *mark* parameter is omitted since there are no more notes.

```
GET /api2/index?length=2&mark=agtzaW1wbGUtbm90ZXIKCxIETm90ZRhtDA
```

Response:

```
{"count": 2, "data": [{"modifydate": "1285617788.655000", "tags": [],
"deleted": 0, "createdate": "1285617787.044000", "systemtags": [],
"version": 2, "syncnum": 2, "key":
"agtzaW1wbGUtbm90ZXIKCxIETm90ZRhtDA", "minversion": 1}, {"modifydate":
"1285617783.647000", "tags": [], "deleted": 0, "createdate":
"1285617780.555000", "systemtags": [], "version": 1, "syncnum": 1,
"key": "agtzaW1wbGUtbm90ZXIKCxIETm90ZRhuDA", "minversion": 1}]}
```

11. Using same bookmark *mark* from example 10, retrieve index with maximum *length* = 1. Response contains only 1 note and another *mark* to retrieve last remaining note.

```
GET /api2/index?length=1&mark=agtzaW1wbGUtbm90ZXIKCxIETm90ZRhtDA
```

Response:

```
{"count": 1, "data": [{"modifydate": "1285617788.655000", "tags": [],
"deleted": 0, "createdate": "1285617787.044000", "systemtags": [],
"version": 2, "syncnum": 2, "key":
"agtzaW1wbGUtbm90ZXIKCxIETm90ZRhtDA", "minversion": 1}], "mark":
"agtzaW1wbGUtbm90ZXIKCxIETm90ZRhuDA"}
```

12. Using bookmark *mark* from example 11, retrieve last remaining note in index, response does not contain *mark* since there are no more notes.

```
GET /api2/index?length=2&mark=agtzaW1wbGUtbm90ZXIKCxIETm90ZRhuDA
```

Response:

```
{"count": 1, "data": [{"modifydate": "1285617783.647000", "tags": [],
"deleted": 0, "createdate": "1285617780.555000", "systemtags": [],
"version": 1, "syncnum": 1, "key":
"agtzaW1wbGUtbm90ZXIKCxIETm90ZRhuDA", "minversion": 1}]}
```

Checking for changes

To check for changes you can use `syncnum` and *version*. `syncnum` will increment whenever there is any change to a note, content change, tag change, etc. `version` will increment whenever the content property is changed. You **should** store both these numbers in your client to track changes and determine when a note needs to be updated or saved.

Psuedo-code algorithm for syncing

```
For any note changed locally (including new notes):
      Save note to server, update note with response
      // (new syncnum, version, possibly newly-merged content)

Get note index
For each remote note,
      if remote syncnum > local syncnum,
            Retrieve note, update note with response
      if new note (key is not in local store),
            Retrieve note, update note with response

For each local note not in index,
      Permanent delete, remove note from local store
```

Response codes

The HTTP response status code should always be checked to be sure that an operation has completed successfully. Successful operations return with a status code of 200. Any other code is an error.

Some common error codes:

400 - Bad Request *(Is a parameter missing?)*
401 - Unauthorized *(Credentials expired or not provided?)*
403 - Forbidden
404 - Not Found
412 - Precondition Failed *(we use this to indicate failure due to throttling/rate limiting)*
500 - Server Error

Best practices for error codes in common API methods:

`https://simple-note.appspot.com/api/login`

Any error code, retry or ask for password

`https://simple-note.appspot.com/api/index`

401 - User invalid, either authorization key expired or user incorrect, retry login
Any other error, retry

`https://simple-note.appspot.com/api/note`

401 - User invalid, either authorization key expired or user incorrect, retry login
404 - Note does not exist, do not retry
Any other error, retry

`https://simple-note.appspot.com/api/delete`

401 - User invalid, either authorization key expired or user incorrect, retry login
404 - Note does not exist, do not retry
Any other error, retry

Recent Documentation Changes

v.2.1.3:
- Expanded `systemtags` documentation to discuss List and Markdown modes

v.2.1.2:
- Corrected response returned after making a call to delete a tag from the tag index

v2.1.1:
- A clarification that tags should not include commas
- Examples of valid and invalid tags

v2.1.0:
- A new method for tags: `/api/tags`
- Small change to the “mark” key returned by the note index
- Corrected JSON type of the `deleted` property of note objects
- Other clarifications