

# **CGSocialApp Restful API Reference**

A document describing the restful API of the social interaction module CGSocialApp for CMS Made Simple.

**CGSocialApp Version:**  
1.1.9

**Author:**  
Robert Campbell

**Modified:**  
December 17, 2012

**Created:**  
December 26, 2011

## Table of Contents

|                           |    |
|---------------------------|----|
| Introduction:.....        | 4  |
| Features:.....            | 4  |
| Server Requirements:..... | 4  |
| Known Limitations:.....   | 5  |
| To Be Completed:.....     | 6  |
| Setup Guide:.....         | 6  |
| Base URL:.....            | 7  |
| Security:.....            | 7  |
| Authentication:.....      | 8  |
| Parameters:.....          | 8  |
| Return Data:.....         | 8  |
| Sample Request:.....      | 8  |
| Users API:.....           | 9  |
| OP: REGISTER.....         | 9  |
| OP: LOGIN.....            | 9  |
| OP: PING.....             | 10 |
| OP: LOGOUT.....           | 10 |
| OP: GETINFO.....          | 11 |
| OP: SETPROPS.....         | 11 |
| OP: LIST.....             | 12 |
| OP: BADGES.....           | 12 |
| Friends API:.....         | 13 |
| OP: REQUESTLIST.....      | 13 |
| OP: REQUEST.....          | 13 |
| OP: APPROVE.....          | 13 |
| OP: DENY.....             | 14 |
| OP: SUGGEST.....          | 14 |
| OP: DEL.....              | 14 |
| OP: LIST.....             | 15 |
| Locations API:.....       | 15 |
| OP: HIERARCHY.....        | 15 |
| OP: CATEGORIES.....       | 16 |
| OP: LIST.....             | 16 |
| OP: ADD.....              | 17 |
| OP: EDIT.....             | 18 |
| OP: PHOTOUPLD.....        | 19 |
| OP: PHOTODELETE.....      | 19 |
| OP: DELETE.....           | 20 |
| OP: GET.....              | 20 |
| Activities API:.....      | 20 |
| OP: LIST.....             | 20 |
| Checkins API:.....        | 21 |
| OP: BYLOCATION.....       | 21 |
| OP: FRIENDS.....          | 21 |
| OP: EDIT.....             | 22 |

|                         |    |
|-------------------------|----|
| OP: MYCHECKINS.....     | 22 |
| OP: DEL.....            | 23 |
| OP: GET.....            | 23 |
| OP: ADD.....            | 23 |
| OP: APPROVE.....        | 24 |
| Leaderboard API:.....   | 24 |
| OP: LIST.....           | 24 |
| OP: GET.....            | 25 |
| OP: LEADERS.....        | 25 |
| Badges API.....         | 25 |
| OP: LIST.....           | 25 |
| OP: GET.....            | 26 |
| OP: MEMBERS.....        | 26 |
| Social API:.....        | 26 |
| OP: LIST.....           | 26 |
| OP: GETKEY.....         | 27 |
| OP: SETKEY.....         | 27 |
| Checkin Approvals:..... | 27 |
| Sample Client:.....     | 28 |

## ***Introduction:***

The CGSocialApp module is a third party add on module written In PHP for CMSMS that provides social capabilities for mobile applications. This module is designed to be used only to serve APIs and house data, it provides little frontend functionality of its own. Instead this module provides a remote API where other applications can interact with CMSMS Data.

The module provides a secure method for authenticating client applications, and for authenticating users. It provides APIs for managing users, friends, locations, activities, and other aspects of a social website.

The primary concept behind the module is to provide a social mechanism whereby customers can build friends lists, and “checkin” to “locations” (such as parks, restaraunts, hotels, etc). Customers can attach comments and photos (and perhaps videos, or audio clips) to their checkins... and the friends can view the checkins of other friends, or browse the checkins at a location.

## ***Features:***

1. Flexible user capabilities. The system uses the well tested FrontEndUsers module to manage users. The administrator can define any number of 'properties' or attributes for users.
2. Flexible location capabilities. The system uses the well tested CompanyDirectory module to manage 'locationns'. This is a well tested system that has flexible querying capabilities, including radius based searches.
3. Friend relationships via requests, when a user approves a friend request the friendship relationship is created between both users. When one user deletes a friendship the relationship is removed on both sides.
4. Friend Suggestions. Using a 'Friends of Friends' mechanism the system can suggest possible friends to a user.
5. Rate Limiting: The checkin mechanism has rate limiting built in to prevent spamming or somebody artificially posting checkins from their couch.
6. Leaderboards and Badges: A flexible leaderboard mechanism with badges for achieving set scores.
7. This module is capable of displaying checkins given a number of criteria on the frontend in a completely customizable manner.

## ***Server Requirements:***

This module has certain requirements in excess of the normal CMSMS requirements.

1. CMSMS 1.10.2 or greater
2. PHP 5.3 or greater with the curl, mcrypt and xml extensions.
3. CompanyDirectory Module

The CompanyDirectory module provides the 'locations' capabilities. This module must be

properly configured.

4. FrontEndUsers Module

The FrontEndUsers module provides the user organization (including groups, users, and user properties). This module must be properly configured.

5. CGExtensions Module

6. The CGNotifier module and related sub-modules for notifications.

7. The CGJobMgr module.

8. CGFavs Module

The CGFavs module is used to store and manage friend relationships.

9. HTTPS

For authentication this module may require that secure connections between the client and the server be properly configured.

### ***Known Limitations:***

This system is not designed to be a full scale solution to the social application problem, such as facebook, gowalla or foursquare, etc. It is designed to work in conjunction with a single small to medium sized website, on a single host. As such there are the normal limitations regarding disk space, and the amount of files in a directory. As well, this system is subject to the normal PHP limitations (request size, memory limits, maximum upload size, maximum post size etc). In addition there are a few known limitations as described below:

1. Uploaded Files

Under most circumstances, uploaded Files (text files, videos, images, sound clips) for attachments are stored given a unique name in a single directory on the website specifically designated for the CGSocialApp module. There may be a filesystem limit on the number of files in a single directory, in addition to the normal disk space limits and PHP upload limits.

Optionally (using settings in the CGExtensions) module, thumbnail images may be generated for uploaded images. They are stored in the same physical directory. There is no further processing to uploaded images, and no limit to the size of files uploaded, or resolution of images.

2. Checkins

Checkins are stored in a single relational database table. And are tied to a record in the Locations (CompanyDirectory) database, a user (FrontEndUsers) database, and (optionally) a record in the activities table. If a company is deleted, or a user is deleted from their respective databases some checkins may be 'orphaned'. As a result, companies should not be deleted from the database.

3. Friends

The friend system uses a single record in an associative database table to store all of a users friendships. This data is serialized, and the maximum amount of data that can be stored in this record is 64k, therefore there is a realistic limit of a few thousand friends.

4. Cookies

The CGSocialApp module (and therefore the FrontEndUsers) module relies on HTTP requests. In order to identify users uniquely, and in a secure manner the system generates a session cookie. The client application must store, and re-send this session cookie with each request.

### ***To Be Completed:***

1. Events in CompanyDirectory (on Edit/Delete)

Though Locations (companies) shouldn't be deleted, when they are deleted we also need to delete or do something with the checkins. This will require a modification to the CompanyDirectory module to send out events, and then handling them in this module.

2. Handle FEU Events

When a user is deleted from the user database we need to delete the users checkins (or do something with them) also we need to remove that user from friends lists (or do something that will cross reference a friends list with the active users to only return active ones. This will be a modification to FEU and to the Social Module

3. API to Search Users

There is a 'recommended friends' function we need a method in the Users API to search friends by name, or location or something... this is optional though.

4. Facebook Integration for Friends.

We need to write a class for use in the API that if using facebook authentication we retrieve the friends list from Facebook rather than maintaining our own.

5. Add postcode search to the Location search API
6. Add rate limiting to the registration technology.
7. Add checks for HTTPS into the login/register code
8. Add file upload capabilities to the edit checkin code.

### ***Setup Guide:***

How to setup a test site for working with this module. Beside the normal CMSMS install there are some dependancies.

1. Install and configure CMSMS
  1. Setup Pretty URLs
2. Install and configure FEU + Dependants
  1. Create at least one property
  2. Create at least one group
  3. Mark group as default
  4. It'd be helpful to create a sample user or two in the FEU admin.
3. Install and configure CompanyDirectory

1. Define Hierarchy
2. Define Categories
3. Define custom fields
4. Install and configure CGJobMgr
5. Install CGFavs module, CGSimpleSmarty, and JQueryTools
6. Install the CGGoogleMaps module
7. Add locations to the CompanyDirectory module
  1. Be sure to use the address lookup functionality, or enter latitudes and longitudes for each location.
8. Install and configure the CGSocialApp module
  1. Define Activities
  2. Setup the required FEU group
  3. Create at least one API key
  4. Test out and play with the test client.
    - \* Be sure to modify the test client config.php to set the new API key
9. Develop your client application.

### **Base URL:**

The base URL for all RESTFUL API requests will be <site base url>/app/v1/<api area>. Each api function has different specifications for the parameters, request method, or wither a secure connection is required.

### **Security:**

Numerous security mechanisms are included in the API to validate that the request is valid, by an authenticated user, and from an authenticated app:

1. Each request must include the apikey and that is validated (see below)
2. The userid of the known logged in user, and the userid sent in each requests (for user dependant requests) is validated.
3. Some methods include checksums of the request information to be verified by the server.
4. By default the same user account cannot be logged in from two places simultaneously
5. The app must continue to ping the server on a frequent basis or the user will be automatically logged out after a brief period of time.
6. All passwords are stored using a common salt and a one way encryption mechanism. Passwords cannot be recovered from the database.

## ***Authentication:***

The CGSocialApp module manages API keys for different applications. Apikeys may have expiry dates, and may limit the functionality that an app is permitted to access. The apikey must be sent with most requests.

## ***Parameters:***

Each request consists of multiple parameters, including (as described above) the apikey and an 'operation'. If insufficient or incorrect parameters are passed to the API an error will be returned.

## ***Return Data:***

All data returned from this API is encapsulated inside json encoded array, object, or array of objects. There will be a minimum of 2 elements in this array at all times: “\_status” which will either be OK for successful requests, or contain an error code or string. As well the “data” member of this associative array may contain other data representing the result of the request. A “\_message” member may also exist with a readable explanation.

## ***Sample Request:***

Below is some sample PHP code for initializing a login, and parsing the result. This code is from the sample client included with the system. It uses a class built into CMS Made Simple (and duplicated in the test client called cms\_http\_request, which in turn uses curl). This is a simple example of making a request to the social app and parsing its results.

In the code below you can see that we are creating a POST request to a users API method. Our 'op' (operation) is login. We are transmitting the apikey, the username, and the password, and for additional security reasons transferring an md5 key of the apikey, the username, and the password).

Once the request is sent using the ->execute() method, we decode the json encoded results, and process them.

```
function user_login($opts)
{
    if( count($opts) != 2 )
    {
        set_errormsg('register: insufficient arguments');
        return FALSE;
    }

    define('BASE_URL','http://mysite.com/app/v1'); // the base url
    $req = new cms_http_request();
    $req->setMethod('POST');
    $req->setTarget(BASE_URL.'/users'); // calling a users api function
    $req->addParam('apikey',_apikey()); // send our api key
    $req->addParam('op','LOGIN'); // send our requested operation
    $req->addParam('u',$opts[0]); // a username
    $req->addParam('p',$opts[1]); // password (plain text)
```



```

$req->addParam('v',md5(_apikey().$opts[0].$opts[1])); // verification
$res = $req->execute(); // do the request

$tmp = json_decode($res); // decode the results.
if( isset($tmp->_status) && $tmp->_status != 'OK' ) {
    $msg = $tmp->_status;
    if( isset($tmp->_message) ) {
        $msg .= ': '.$tmp->_message;
    }
    set_errormsg($msg);
    return FALSE;
}
if( isset($tmp->data) ) {
    set_appdata('uid',$tmp->data);
}
print_r($tmp);
echo "OK\n"; // woot, done.
return TRUE;
}

```

## Users API:

### OP: REGISTER

DESCRIPTION: Register a new user to the site. This creates a new user under FrontEndUsers . Depending upon the configuration of the website this user may also have the ability to login to the website using a normal browser.

REQUIREMENTS: FrontEndUsers module must be properly configured. The PERM\_USERS\_REGISTER permission must be associated with the supplied apikey and an feu\_usergroup must be specified in the CGSocialApp settings.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Status Only

**TODO: Use HTTPS**

PARAMETERS:

- apikey: The applications api key.
- op: REGISTER
- u: Desired Username
- p: Desired Password
- v: md5 hash of the concatenated apikey, username and password

### OP: LOGIN

DESCRIPTION: Register a new user to the site. This creates a new user under FrontEndUsers . Depending upon the configuration of the website this user may also have the ability to login to the website using a normal browser.

REQUIREMENTS: The PERM\_USERS\_LOGIN permission must be associated with the

supplied apikey and an feu\_usergroup must be specified in the CGSocialApp settings.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Returns the users unique integer identifier (uid)

**TODO: Use HTTPS**

PARAMETERS:

- apikey: The applications api key.
- op: REGISTER
- u: Desired Username
- p: Desired Password
- v: md5 hash of the concatenated apikey, username and password

### **OP: PING**

DESCRIPTION: Pings the server with the users credentials to verify the login session and keep the login session alive.

REQUIREMENTS: The PERM\_USERS\_LOGIN permission must be associated with the supplied apikey.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: Status only

PARAMETERS:

- apikey: The applications api key.
- op: PING
- uid: The users unique integer identifier
- v: md5 hash of the concatenated apikey, and uid

### **OP: LOGOUT**

DESCRIPTION: Terminate the login session for the specified user.

REQUIREMENTS: The PERM\_USERS\_LOGIN permission must be associated with the supplied apikey.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Status only

PARAMETERS:

- apikey: The applications api key.
- op: LOGOUT

- uid: The users unique integer identifier
- v: md5 hash of the concatenated apikey, and uid

### **OP: GETINFO**

DESCRIPTION: Retrieve the known properties about the current user. If an image property is associated with the user, and there is a value for that property then the full URL to the image will be returned.

REQUIREMENTS: The PERM\_USERS\_READ permission must be associated with the supplied apikey.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An associative array of all properties stored in the database for the specified user.

**TODO: Return user info (except for password), ensure that we supply full urls for images**

PARAMETERS:

- apikey: The applications api key.
- op: GETINFO
- uid: The users unique integer identifier
- v: md5 hash of the concatenated apikey, and uid

### **OP: SETPROPS**

DESCRIPTION: Set user properties. This method can be used to set any properties associated with the user (for example: gender, real name, location etc). Any data that is defined as a property in the FrontEndUsers module and associated with the proper group. This method can also be used to allow changing the user password. Note, multiple properties can be adjusted with a single request.

REQUIREMENTS: The PERM\_USERS\_WRITE permission must be associated with the supplied apikey.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Status Only

**TODO: Use HTTPS, Allow uploading image properties.**

PARAMETERS:

- apikey: The applications api key.
- op: SETPROPS
- uid: The users unique integer identifier
- v: md5 hash of the concatenated apikey, and uid
- ::password:: (optional) The desired new password

- `::repeatpassword::` (optional) The desired new password (must match the `::password::` field).
- `<propertyname>`: The property name to modify. i.e: `full_name: John Q. Public`

## **OP: LIST**

DESCRIPTION: Retrieve a list of users that match the specified criteria.

REQUIREMENTS: The `PERM_USERS_READ` permission must be associated with the specified apikey.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array of userid's and usernames for users that match the criteria.

PARAMETERS:

- `apikey`: The applications api key.
- `op`: LIST
- `uid`: The current users uid
- `n`: (integer, optional) the maximum number of results to return. Default is 50
- `o`: (integer, optional) the offset of results to return. Default is 0
- `userpattern`: (string, optional) A regular expression of usernames
- `property`: (string, optional) The name of a property to match
- `proppattern`: (string, optional) A regular expression pattern to match against the property.
- `idlist` (string, optional) A comma separated list of user ids
- `full` (integer, optional)

## **OP: BADGES**

DESCRIPTION: Retrieve a list of the current user (or any users) badges

REQUIREMENTS: The `PERM_USERS_READ` permission must be associated with the specified apikey.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array of badge id's that that the user possesses

PARAMETERS:

- `apikey`: The applications api key.
- `op`: BADGES
- `uid`: (optional) A user identifier. If not specified, the currently logged in users uid will be used.

## ***Friends API:***

### **OP: REQUESTLIST**

DESCRIPTION: Return a list of friend requests for the current user

REQUIREMENTS: The PERM\_FRIENDS\_READ permission must be associated with the specified apikey.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array of integer uids representing the people that have sent friend requests

PARAMETERS:

- apikey: The applications api key
- op: REQUESTLIST
- uid: The current users uid
- n: The number of friend requests to return. Default is 100000
- o: The offset of the friend requests to return. Default is 0

### **OP: REQUEST**

DESCRIPTION: Submit a friend request to another user.

REQUIREMENTS: The PERM\_FRIENDS\_WRITE permission must be associated with the specified apikey.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Status only

PARAMETERS:

- apikey: The applications api key
- op: REQUEST
- uid: The current users integer user id
- fuid: The potential friends integer user id.

### **OP: APPROVE**

DESCRIPTION: Approve a friend request from another user

REQUIREMENTS: The PERM\_FRIENDS\_WRITE permission must be associated with the specified apikey.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Status only

PARAMETERS:

- apikey: The applications api key
- op: REQUEST
- uid: The current users integer user id
- fuid: The potential friends integer user id.

#### **OP: DENY**

DESCRIPTION: Deny a friend request from another user.

REQUIREMENTS: The PERM\_FRIENDS\_WRITE permission must be associated with the specified apikey.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Status only

PARAMETERS:

- apikey: The applications api key
- op: REQUEST
- uid: The current users integer user id
- fuid: The potential friends integer user id.

#### **OP: SUGGEST**

DESCRIPTION: Suggest a number of friends for the current user

REQUIREMENTS: The PERM\_FRIENDS\_READ permission must be associated with the specified apikey.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array of integer uids for suggested friends if possible.

PARAMETERS:

- apikey: The applications api key
- op: SUGGEST
- uid: The current users uid

#### **OP: DEL**

DESCRIPTION: Remove a friend from the friends list.

ALIAS: REMOVE

REQUIREMENTS: The PERM\_FRIENDS\_WRITE permission must be associated with the specified apikey.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Status Only

PARAMETERS:

- apikey: The applications api key
- op: DEL (or REMOVE)
- uid: The current users uid
- fuid: (integer) The friends uid

### **OP: LIST**

DESCRIPTION: Retrieve a list of the current users friends.

REQUIREMENTS: The PERM\_FRIENDS\_READ permission must be associated with the specified apikey.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array of integer uids representing the users friends. If the deep parameter is specified, an array of objects is returned. Each objects will contain the uid and username of the friend.

PARAMETERS:

- apikey: The applications api key
- op: LIST
- uid: The current users uid
- n: (integer, optional) The maximum number of results to return. Default is 100000
- o: (integer, optional) The offset of results to return. Default is 0
- deep: (integer, optional) Whether to return username information about friends. Default is 0

## ***Locations API:***

### **OP: HIERARCHY**

DESCRIPTION: Retrieve a hierarchy tree for the locations.

REQUIREMENTS: Requires the PERM\_LOCATIONS\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: A nested array of hierarchy information.

PARAMETERS:

- apikey: The applications api key
- op: HIERARCHY

## **OP: CATEGORIES**

DESCRIPTION: Retrieve a list of category information for locations

REQUIREMENTS: Requires the PERM\_LOCATIONS\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array of category information (including extra data).

**TODO: Pagination parameters**

PARAMETERS:

- apikey: The applications api key
- op: HIERARCHY

## **OP: LIST**

DESCRIPTION: Return a list of locations matching the specified criteria. This method actually calls the same code that is used by the CompanyDirectory summary view. Most of the parameters accepted by that view will be accepted here.

REQUIREMENTS: Requires the PERM\_LOCATIONS\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array of locations matching the specified criteria. URLs Will be specified for images.

PARAMETERS:

- apikey: The applications api key.
- op: LIST
- sortby: (optional)
- sortorder: (optional)
- pagelimit: (optional integer) Default value is 50
- page: (optional integer) The page number. Default value is 1
- name: (optional string) Search by location name (wildcard characters accepted)
- category: (optional string) comma separated list of category names
- category2: (optional string) comma separated list of category names
- categoryid: (optional integer) a single category id
- hier: (optional string) Search by hierarchy name (wildcard characters accepted)
- expiredowners: (optional integer) Allows showing locations where the owner is a FrontEndUser and has expired. Default is 0
- address: (optional, string) Search by address (wildcard characters accepted)
- showall: (optional, integer) Allows returning all locations (even those marked as draft)



- **matchall:** (optional integer) Allows returns locations that match ANY of the conditions specified instead of ALL of the conditions. Default is 1.
- **status:** (optional string) Allows searching for locations with the specified status. Possible values are 'draft' and 'published'
- **lat:** (optional float) Allows specifying a latitude (useful only when long and radius are also specified).
- **long:** (optional float) Allows specifying a longitude (useful only when lat and radius are also specified).
- **idlist:** (optional, string) Search by specifying a comma separated list of location id's (*an array should also be accepted*).
- **radius:** (optional string) Allows specifying a radius in miles or kilometers. (useful only when lat and long are also specified). The units can be appended to the value. i.e: radius='15km' or radius='12.5mi'. Default units are miles.

#### **OP: ADD**

**DESCRIPTION:** Create a new location. A setting in the admin panel of the module controls whether any location can be added. Additionally, the appropriate permission must be associated with the api key.

**REQUIREMENTS:** Requires the PERM\_LOCATIONS\_WRITE permission.

**REQUEST METHOD:** POST

**PROTOCOL:** HTTP

**RESULTS:** OK message, or error message.

**PARAMETERS:**

- **apikey:** The applications api key.
- **op:** ADD
- **owner:** The current users uid
- **company\_name:** The name of the new location
- **latitude:** The location latitude

It is recommended that this come automatically from the submitters device, and the user not be allowed to directly edit this value.

- **longitude:** The location longitude

It is recommended that this come automatically from the submitters device, and the user not be allowed to directly edit this value.

- **address:** (optional) The address of the location
- **telephone:** (optional) Telephone number
- **fax:** (optional) Fax number
- **contact\_email:** (optional) contact email address

- website: (optional) The location website
- details: (optional) HTML for the details of this company
- hier\_id: (optional) hierarchy id.

Categories:

It is possible to set a location into multiple categories. By specifying numerous category arguments:

- C:categoryname1=1
- C:categoryname2=1

Custom Fields:

NOTE: Image fields are not supported.

It is possible to specify numerous custom field values in the format of:

- F:fieldname=value

## **OP: EDIT**

DESCRIPTION: Create a new location. A setting in the admin panel of the module controls whether any location can be added. Additionally, the appropriate permission must be associated with the api key.

REQUIREMENTS: Requires the PERM\_LOCATIONS\_WRITE permission.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: An array of locations matching the specified criteria. URLs Will be specified for images.

PARAMETERS:

- apikey: The applications api key.
- op: ADD
- owner: The current users uid  
The owner must match the owner attached to the location.
- company\_name: (optional) The name of the new location
- latitude: (optional) The location latitude  
It is recommended that this come automatically from the submitters device, and the user not be allowed to directly edit this value.
- longitude: (optional) The location longitude  
It is recommended that this come automatically from the submitters device, and the user not be allowed to directly edit this value.
- address: (optional) The address of the location
- telephone: (optional) Telephone number

- fax: (optional) Fax number
- contact\_email: (optional) contact email address
- website: (optional) The location website
- details: (optional) HTML for the details of this company
- hier\_id: (optional) hierarchy id.

Categories:

It is possible to set a location into multiple categories. By specifying numerous category arguments:

- C:categoryname1=1
- C:categoryname2=1

Custom Fields:

NOTE: Image fields are not supported.

It is possible to specify numerous custom field values in the format of:

- F:fieldname=value

## **OP: PHOTOUPLoad**

DESCRIPTION: Upload image files and attach them with a location.

REQUIREMENTS: A field of type 'Album' must be created in the companydirectory module, and marked as public. Only the owner of a location can upload photos.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: An array of hashes representing information about the uploaded files.

SEE ALSO: <http://blueimp.github.com/jQuery-File-Upload/>

PARAMETERS:

- apikey: The applications api key
- location: (integer) The location identifier
- owner: (integer) The owner of the location
- series of uploaded files

This function accepts one or more files using either the standard multi-part file upload method, with a field name of “file” or files uploaded asynchronously via the XMLHttpRequest standard using the HTTP\_X\_FILE\_NAME, HTTP\_X\_FILE\_SIZE, and HTTP\_X\_FILE\_TYPE headers.

## **OP: PHOTODELETE**

DESCRIPTION: Delete uploaded image files that are attached to a location.

REQUIREMENTS: A field of type 'Album' must be created in the companydirectory module,

and marked as public. Additionally, there must be files in that field. Only the owner of a location can delete attached files.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: On success, there is no output

PARAMETERS:

- apikey: The applications api key
- location: (integer) The location identifier
- owner: (integer) The owner of the location
- file: (array of strings). The filenames (basenames are fine) to remove.

#### **OP: DELETE**

DESCRIPTION: Delete an existing location that is owned by the current user

**NOT IMPLEMENTED**

#### **OP: GET**

DESCRIPTION: Retrieve information about a single company

REQUIREMENTS: Requires the PERM\_LOCATIONS\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array representing detailed information for a single location.

PARAMETERS:

- apikey: The applications api key
- location: (integer) The location identifier.

### ***Activities API:***

#### **OP: LIST**

DESCRIPTION: Retrieve a hierarchical list of activities

REQUIREMENTS: Requires the PERM\_ACTIVITIES\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: A nested array representing activities along with their ids

PARAMETERS:

- apikey: The applications api key

## **Checkins API:**

### **OP: BYLOCATION**

DESCRIPTION: Retrieve a list of checkins for a specific location. This option has the ability to query checkins that are not approved IF approval of locations is enabled in the modules config section.

REQUIREMENTS: Requires the PERM\_CHECKINS\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: A list of checkins matching the criteria specified.

PARAMETERS:

- apikey: The application api key
- lid: (integer) The location identifier.
- timelimit: (optional integer). Specify a time limit for the search, a negative value is assumed to be in seconds relative to the current time. A positive value is assumed to be an absolute unix timestamp.
- limit: (optional integer) Specify a limit for the number of results to return. Default is 50
- offset: (optional integer) Specify the offset to start counting at when returning results. Default is 0
- deep: (optional integer) Any non zero integer indicates that deep information about this checkin should be returned (including username, activity name (If supplied) and location name.
- unapproved: (optional integer). Either 0 or 1. If sent, and then the following additional parameters are also required. This parameter additionally requires the PERM\_CHECKINS\_APPROVE permission and the uid parameter.
- uid: (optional integer). The current logged in users uid.

### **OP: FRIENDS**

DESCRIPTION: Retrieve the latest checkins of the current users friends.

REQUIREMENTS: Requires the PERM\_CHECKINS\_READ and PERM\_FRIENDS\_READ permissions.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: A list of checkins matching the criteria specified.

PARAMETERS:

- apikey: The application api key
- uid: The current users identifier
- n: (optional integer) Specify a limit for the number of results to return. Default is 50
- o: (optional integer) Specify the offset to start counting at when returning results. Default is

0

- deep: (optional integer) Any non zero integer indicates that deep information about this checkin should be returned (including username, activity name (If supplied) and location name.

### **OP: EDIT**

DESCRIPTION: Make modifications to an existing checkin

REQUIREMENTS: Requires the PERM\_CHECKINS\_EDIT permission

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Status only

**TODO: Add ability to upload files here.**

PARAMETERS:

- apikey: The application api key
- owner: (integer) The current users uid
- cid: (integer) The checkin id to edit.
- activity: (optional integer) An activity id.
- location: (optional integer) A location identifier.
- msg: (optional string, 255 chars) An optional message.
- del: (optional integer) Indicates that the file at index n should be deleted

### **OP: MYCHECKINS**

DESCRIPTION: Retrieve a list of checkins for the current logged in user

REQUIREMENTS: Requires the PERM\_CHECKINS\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array of checkin information matching the criteria specified.

PARAMETERS:

- apikey: The application api key
- owner: (integer) The current users uid
- timelimit: (integer) Specify a time limit for checkins. Negative value indicates relative time from now. Postive value is assumed to be a unix timestamp.
- limit: (integer) Specify a limit for the amount of items to be returned. Default is 50
- offset: (integer) Specify an offset for the number of items to return.
- deep: (optional integer) Any non zero integer indicates that deep information about this checkin should be returned (including username, activity name (If supplied) and location

name.

#### **OP: DEL**

DESCRIPTION: Delete a single checkin

REQUIREMENTS: Requires the PERM\_CHECKINS\_DEL permission.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: Status only

**TODO: Ensure that files are deleted.**

PARAMETERS:

- apikey: The application api key
- owner: (integer) The current users user id
- cid: (integer) The integer id of the checkin to be deleted

#### **OP: GET**

DESCRIPTION: Retrieve information about a single checkin

REQUIREMENTS: Requires the PERM\_CHECKINS\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: An array representing the data about a single checkin.

PARAMETERS:

- apikey: The application api key
- cid: (integer) The id for the desired checkin.
- deep: (optional integer) Any non zero integer indicates that deep information about this checkin should be returned (including username, activity name (If supplied) and location name.

#### **OP: ADD**

DESCRIPTION: Create a checkin for a location. This action will also accept uploaded image files.

REQUIREMENTS: Requires the PERM\_CHECKINS\_ADD permission.

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: The integer checkin id of a newly created checkin

PARAMETERS:

- apikey: The application api key
- owner: (integer) The users user id

- location: (integer) The location identifier
- activity: (optional integer) An activity identifier.
- parent\_id: (optional integer) An optional parent checkin id
- lat: (optional, float) A latitude
- lng: (optional float) A longitude
- msg: (optional string) A message (up to 255 characters)
- tag: (optional string) A comma separated list of integer user identifiers identifying users to tag in this checkin.
- extra\_XXXX (optional). Any number of extra parameters can be associated with the checkin (64KB max, when serialized).

#### **OP: APPROVE**

DESCRIPTION: Approve unapproved checkins. Note: Depending upon the configuration of the CGSocialApp module, an additional password (pw) parameter may be required to ensure that the user has the ability to approve checkins for the specified location.

REQUIREMENTS: Required the PERM\_CHECKINS\_APPROVE permission

REQUEST METHOD: POST

PROTOCOL: HTTP

RESULTS: On success this action returns no results

PARAMETERS:

- apikey: The application api key
- owner: (integer) The users user id
- lid: (integer) The location identifier
- cid: (integer) The checkin identifier
- pw: (optional) The md5sum of the password associated with the company.

### ***Leaderboard API:***

#### **OP: LIST**

DESCRIPTION: List all leaderboards

REQUIREMENTS: Required the PERM\_LEADERBOARDS\_READ permission

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: On success this action returns an array of leaderboards

**TODO: Add pagination and filtering**

PARAMETERS:

- apikey: The application api key



**OP: GET**

DESCRIPTION: Retrieve full information for a single leaderboard.

REQUIREMENTS: Required the PERM\_LEADERBOARDS\_READ permission

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: On success an array representing the information for a single leaderboard.

PARAMETERS:

- apikey: The application api key
- board: (integer) The leaderboard identifier.

**OP: LEADERS**

DESCRIPTION: Retrieve a list of the leaders (by score) for a particular leaderboard.

REQUIREMENTS: Required the PERM\_LEADERBOARDS\_READ and the PERM\_USERS\_READ permissions.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: On success this action an array of integer user ids for the score leaders of the leaderboard, in order of decreasing score.

PARAMETERS:

- apikey: The application api key
- board: (integer) The board identifier.
- n: (integer) (optional) The number of items to return (by default 25).
- o: (integer) (offset) The start position to return results from (by default 0)

***Badges API*****OP: LIST**

DESCRIPTION: List all badges

REQUIREMENTS: Required the PERM\_BADGES\_READ permission

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: On success this action returns an array of badges

PARAMETERS:

- apikey: The application api key

**OP: GET**

DESCRIPTION: Retrieve full information for a single badge

REQUIREMENTS: Required the PERM\_BADGES\_READ permission

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: On success an array representing the information for a single badge

PARAMETERS:

- apikey: The application api key
- badge: (integer) The badge identifier.

**OP: MEMBERS**

DESCRIPTION: Retrieve a list of the members for a particular badge

REQUIREMENTS: Required the PERM\_BADGES\_READ and the PERM\_USERS\_READ permissions.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: On success this action an array of integer user ids for the users that have this badge.

PARAMETERS:

- apikey: The application api key
- badge: (integer) The board identifier.
- n: (integer) (optional) The number of items to return (by default 25).
- o: (integer) (offset) The start position to return results from (by default 0)

***Social API:***

The social API is a brief set of commands for interacting with other social networks.

**OP: LIST**

DESCRIPTION: Retrieve a list of modules that handle interaction with social networks, and the names of those social networks.

REQUIREMENTS: Requires the PERM\_SOCIAL\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: A hash containing module names as keys and social network names as values. Or an empty result.

PARAMETERS:

- apikey: The application API key

### OP: GETKEY

DESCRIPTION: Retrieves the user token and secret for the specified social network module for the currently logged in user.

REQUIREMENTS: Requires the PERM\_SOCIAL\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: A nested array representing activities along with their ids

PARAMETERS:

- apikey: (string) The applications API key
- uid: (integer) The currently logged in frontend-users identifier (will be verified)
- network: (string) The name of the module handling the social network.
- v: (string) md5 checksum of the concatenated apikey,uid and network

### OP: SETKEY

DESCRIPTION: Set the user token and secret for the specified social network module for the currently logged in user. User tokens and secrets are stored encrypted in the database

REQUIREMENTS: Requires the PERM\_SOCIAL\_READ permission.

REQUEST METHOD: GET

PROTOCOL: HTTP

RESULTS: None

**TODO: Require HTTPS here**

PARAMETERS:

- apikey: (string) The applications API key
- uid: (integer) The currently logged in frontend-users identifier (will be verified)
- network: (string) The name of the module handling the social network.
- v: (string) md5 checksum of the concatenated apikey,uid and network

## Checkin Approvals:

The CGSocialApp module allows the ability for an authorized person to approve a checkin before it is available for public view, or before it is counted in any leaderboards. There are two ways to do checkin approvals:

A) A text field in the CompanyDirectory module can be used to specify a unique password (per location) that would be supplied in order to approve one or more checkins. To implement this follow these steps:

1. Add a text field into the CompanyDirectory module. This field should **not** be marked as “public”. Name it something like “approval\_password”
2. Edit each company (location) that will require checkin approval (that is in one or more

of the categories specified in the “Initial Checkin Status” step above

3. In the “approval\_password” field you created above enter a text password. Note this password and the associated company for future reference. You may share this with the people who will be approving checkins.

4. In the CGSocialApp module's admin panel select the “approval\_password” field as the “CompanyDirectory Password Field”

B) A text field containing a comma separated list of usernames or user ids can be used to specify the unique users that can approve checkins for each location. For this method you will need to implement the following steps:

1. Add a text field into the CompanyDirectory module. This field should **not** be marked as “public”. Name it something like “approval\_users”

2. Edit each company (location) that will require checkin approval (that is in one or more of the categories specified in the “Initial Checkin Status” step above

3. In the “approval\_users” field you created above, enter a comma separated list of user id's or usernames of each person that will have the authority to approve checkins for that location

4. In the CGSocialApp module's admin panel select the “approval\_users” field as the “CompanyDirectory Users Field”

**Note 1:** If you specify both an “approval\_password” and “approval\_users” field, and enter text into them then both methods will be required to approve checkins. The password must match, AND the user approving the checkin must be in the list.

**Note 2:** If some location categories are selected to require checkin approval, then either the “approval\_users” or “approval\_password” method must be used for each company. Not enabling one of these methods will mean that there is no way to approve posts for that location.

## **Sample Client:**

### 1. Requirements

The sample client is a PHP app designed to be used from the console (I've only tested with a linux console, so if you are using a windoze machine your mileage may vary).

- Requires PHP Curl
- Requires PHP Readline Capabilities
- Requires PHP 5.3+

### 2. Sample Session

#### 1. Register new user

USERS

REGISTER [somebody@somewhere.com](mailto:somebody@somewhere.com) thepassword

2. Login

LOGIN [somebody@somewhere.com](#) thepassword

3. List Companies

LOCATIONS

LIST

4. Checkin

CHECKIN loc=5 msg="this is my checkin" lat=xx.xxxx lng=yy.yyyy  
file1=/path/to/file1.jpg file2="/path/to/file with spaces.jpg"