# Ning API Documentation

*Release 1.0*

**Ning Inc.**

January 27, 2012

# Contents

**1 Changelog** **3**
- 1.1 January 27th, 2012 Changelog . . . 3
- 1.2 November 18th, 2011 Changelog . . . 4

**2 API Overview** **7**
- 2.1 Introduction . . . 7
- 2.2 Authentication . . . 8
- 2.3 Request Overview . . . 9
- 2.4 Response Overview . . . 13
- 2.5 Visibility . . . 17
- 2.6 Moderation . . . 18

**3 API Reference** **23**
- 3.1 Activity Items . . . 23
- 3.2 Blog Posts . . . 25
- 3.3 Broadcast Message . . . 27
- 3.4 Comments . . . 28
- 3.5 Network . . . 30
- 3.6 Photos . . . 31
- 3.7 Tags . . . 34
- 3.8 OAuth Tokens . . . 34
- 3.9 Users . . . 35
- 3.10 Friend . . . 38
- 3.11 Property Type Definitions . . . 39
- 3.12 Error Codes . . . 40

**4 API Examples** **43**
- 4.1 Accessing the Ning API with cURL . . . 43
- 4.2 Accessing the Ning API with PHP . . . 46
- 4.3 Accessing the Ning API with Python . . . 49

**Index** **53**

This document is also available as a `PDF download`.

CHAPTER 1

# Changelog

## 1.1 January 27th, 2012 Changelog

### 1.1.1 New features

- *Blog Post endpoint*
  - New `featuredDate` property
  - New sort order `/BlogPost/featured`
- *Photo endpoint*
  - New `featuredDate` property
  - New sort order `/Photo/featured`
- *User endpoint*
  - New `featuredDate` property
  - New sort order `/User/featured`
- *Comments endpoint*
  - New `featuredDate` property
  - New sort order `/Comment/featured`

### 1.1.2 Details

We have added the `featuredDate` property to blog posts, photos, members, and comments. This field is a timestamp of when the Network Creator or an administrator clicked the feature button on the Ning Site.

Additionally we added the ability to sort on the `featuredDate` property. The most recently featured items will come first. Note that the `createdDate` property is not used as part of the sort order. This means that an a blog post from two years ago that was featured today will come before a blog post I featured yesterday. This can be used to highlight greatest hits from the past.

### 1.1.3 Examples

Get twenty recently featured Blog Posts:

```
GET /xn/rest/apiexample/1.0/BlogPost/featured?count=20&fields=featuredDate
```

Get twenty recently featured Photos:

```
GET /xn/rest/apiexample/1.0/Photo/featured?count=20&fields=featuredDate
```

Get twenty recently featured members:

```
GET /xn/rest/apiexample/1.0/User/featured?count=20&fields=featuredDate
```

Get twenty recently featured comments:

```
GET /xn/rest/apiexample/1.0/Comment/featured?count=20&fields=featuredDate
```

## 1.2 November 18th, 2011 Changelog

### 1.2.1 New features

- *New Tag endpoint*
  - Allows developers to access all the tags attached to a content item.
- *New Friend endpoint*
  - Get the list of your friends
  - Get the list of friends for another member
  - Add a friend
  - Remove a friend
- *Blog Post endpoint*
  - Add tags on creation
  - Update tags
  - Filter by tag
  - Renamed the `tags` field to `topTags`
- *Photo endpoint*
  - Add tags on creation
  - Update tags
  - Filter by tag
  - Renamed the `tags` field to `topTags`
- *Activity endpoint*
  - Include CreateTopic event in results
  - Include CreateTopicComment event in results

### 1.2.2 Details

We chose to rename the `tags` field to `topTags` because that's what it contains. Calling the field `tags` was misleading because one would expect to see all tags associated with a content item.

To access all tags attached to a content item, developers needed a way to paginate through long lists. This is why we ended up with a new *tag endpoint*.

### 1.2.3 Examples

Get twenty tags from a blog post:

```
GET /xn/rest/apiexample/1.0/Tag/list?attachedTo=12344:BlogPost:12314&count=20
```

Get the list of your friends:

```
GET /xn/rest/apiexample/1.0/Friend/recent?fields=friend,state&count=10
```

Get the list of friends for another member:

```
GET /xn/rest/apiexample/1.0/Friend/recent?author=3ixs6bzjxfkz6&fields=friend,state
```

Add a friend:

```
POST /xn/rest/apiexample/1.0/Friend?friend=3ixs6bzjxfkz6
```

Remove a friend:

```
DELETE /xn/rest/apiexample/1.0/Friend?friend=3ixs6bzjxfkz6
```

Add tags on blog post creation:

```
POST /xn/rest/apiexample/1.0/BlogPost

title=Hello+tags
description=Tags+are+fun
tag=cat
tag=dogs
tag=multi+word
```

Update tags on a photo:

```
PUT /xn/rest/apiexample/1.0/Photo?id=1234:Photo:3234

tag=cat
tag=dogs
tag=multi+word
```

Get three recent blog posts tagged *cats*:

```
GET /xn/rest/apiexample/1.0/BlogPost/recent?tag=cats&fields=title,description&count=3
```

CHAPTER 2

# API Overview

## 2.1 Introduction

The Ning API is a RESTful API that allows developers to access the content on their Ning Networks. This document is meant to provide an overview of the Ning API that can be used to reference different methods available to you. Be sure to checkout our examples and happy coding!

### 2.1.1 What you can do

You can CREATE, READ, UPDATE, and DELETE content for the following features:

- **Photos**
- **Blog Posts**
- **Comments**

In addition to the interacting with specific features on a site, you can:

- View profile information
- Update your status message
- Retrieve a list of Ning Networks you are a member of
- View the latest activity

### 2.1.2 What you cannot do

Currently, you cannot access the following features:

- **Chat**
- **Events**
- **Forum**
- **Groups**
- **Video**

Additionally you cannot:

- Update your profile information
- Create Ning Networks
- Join Ning Networks

## 2.2 Authentication

The Ning API uses the OAuth standard for authentication. An application developer is provisioned consumer credentials by the Network Creator. The API key management page is found via the “API Keys” link of the Tools section of the management page.

Once granted, the application can then request access to member resources by authenticating using the member’s email and password. All requests to the Ning API are required to be signed using the member’s token and the consumer’s key.

OAuth involves three parties:

**Resource Owner** A member of a Ning Network

**Client** An application accessing the member’s content on the Ning Network

**Server** The Ning API endpoint that accepts the OAuth requests

---

**Note:** Tokens currently do not expire, but an access management system is being developed.

---

### 2.2.1 Obtaining Credentials

Before the application makes any requests on behalf of the member, the member must first authorize the application to access their data. To do this, the application performs a POST request to the Token resource. The resource verifies the identity of the member using basic authentication.

The POST request must include the OAuth parameters passed as form parameters. The Content-Type should be `application/x-www-form-urlencoded`. The signature is generated using the consumer key and consumer secret provisioned by Ning.

Example of obtaining token and consumer key using cURL from the command line:

```
curl -u admin@example.com \
    -d 'oauth_signature_method=PLAINTEXT&oauth_consumer_key=0d716e57-5ada-4b29-a33c-2f4af1b26837&oaut
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Token?xn_pretty=true'
```

Once you run the command above and entered your password, the server will respond with an access token, access token secret, and your consumer key as seen below:

```
{
  "success" : true,
  "entry" : {
    "author" : "cpor74jnszaj",
    "oauthConsumerKey" : "0d716e57-5ada-4b29-a33c-2f4af1b26837",
    "oauthToken" : "4e31acbd-baee-4b1d-b788-9232a8778e8f",
    "oauthTokenSecret" : "1c5dee59-d3a0-4128-8581-488c236e6bfb"
  },
  "resources" : {
  }
}
```

The `oauthConsumerKey` is equivalent to a username and is used to identify the application when making requests. It is the same value as the `oauth_consumer_key` in the `authorization` field.

The `oauthToken` is used to identify the member that the application is making requests for. It cannot be used with other consumer keys. The `oauthTokenSecret` is used when signing requests to verifiy that your application has the right to use the `oauthToken`.

### 2.2.2 Making a request

The HTTP "Authorization" header is used to transmit the application's credentials (i.e. contains a valid OAuth token and consumer key combination for a request). Signing requests ensures that only valid applications are allowed to make requests to your site.

**Note:** If you are unable to set the Authorization header, you can use a X-Authorization header instead and it will be interpreted the same the Authorization header.

The authorization header begins with:

```
Authorization: OAuth
```

It is followed with a comma separated list of 4 key-value parameters:

**oauth_consumer_key** The `oauthConsumerKey` given by Ning from the token request

**oauth_token** The `oauthToken` given by Ning from the token request

**oauth_signature_method** The name of the signature method. Currently `PLAINTEXT` and `HMAC-SHA1` are supported

**oauth_signature** Used to prove ownership of the consumer key and token. See the OAuth specification for information on how to compute the **oauth_signature**

**oauth_timestamp** The POSIX timestamp when the request was made

**oauth_nonce** A random string that when used with the **oauth_timestamp** ensures that the request has never been made before

**oauth_version** The OAuth version used for this request, it must be `1.0`

**Note:** The `PLAINTEXT` signature method requires a HTTPS connection

An authorization header using `PLAINTEXT` would look like:

```
Authorization: OAuth oauth_version="1.0",oauth_timestamp="1276205581",oauth_nonce="df0a1260cb2dcd536c
```

Example of making a signed request in cURL from the command line:

```
curl -H 'Authorization: OAuth oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Photo/recent?xn_pretty=true&fields=image.ur
```

## 2.3 Request Overview

### 2.3.1 Rate Limits

The Ning API has a throttle rate of 20 requests per second, per Ning Network. When requests go over this limit, the API will return a 503 HTTP response.

### 2.3.2 Resource URL structure

A typical resource URL looks like:

```
http://external.ningapis.com/xn/rest/apiexample/1.0/BlogPost/recent
```

A resource URL consists of several parts:

**Ning API Domain `http://external.ningapis.com`** The base URL for the Ning API

**Namespace Prefix `/xn/rest`** This defines that we are using the Ning API.

**Ning Subdomain `/apiexample`** Specifies the Ning Ning Network that we are making requests to. You must use the Ning subdomain, you cannot use a custom domain.

**API Version `/1.0`** Currently Ning is on version 1.0 of it's Ning API and doesn't support any other version.

**Resource Type `/BlogPost`** The data we are working with.

**Sort Ordering `/recent`** The order to return the results. This is an optional in the URL. Requests without a sort order do not guarantee any sort order. Sort order only applies to GET requests and recent is the only supported sort type at this time.

### 2.3.3 Resource URL parameters

You can add parameters to your resources requests to return more specific results or to page through large result sets.

#### Fields parameter

Every request URL supports the optional `fields` parameter. The value of the `fields` parameter is a comma-separated list of resource properties that should be returned in the response. The ''field'' parameter tells the server to return a certain set of fields. (This may be more or less fields that what is returned by default)

Example: To return the titles and creation dates of the most recent blog posts, the request URL would be:

```
http://external.ningapis.com/xn/rest/apiexample/1.0/BlogPost/recent?fields=title,createdDate
```

Most requests have the option to request additional information about the resources returned. For example, when you request a Blog Post, it returns the screen name of the member who created it. Normally you would need to perform a second request to retrieve the name and profile photo of the author. With sub-resources you specify that you want fields from related content like the author's name and profile photo returned as well.

Example: Request for the all recent blog posts with the author's full name and icon for each item:

```
GET http://external.ningapis.com/xn/rest/1.0/BlogPost/recent?fields=title,descirption,author.fullName
```

There are two main advantages to using sub-resources:

1. If multiple resources share the same sub-resources (e.g., several Photos with the same author), sending the sub-resource once reduces the size of the response.
2. Parsing of responses by client libraries is simpler since a parser can be written to handle each sub-resource type and re-used with multiple resources.

#### Pagination parameters

All requests that specify a sort order support 2 parameters used for paging:

**anchor** Unique identifier for the last-returned page of results

**count** The number of results to return, with a maximum of 100. This parameter also supports negative values down to -100. Negative `count` parameters returns results before the `anchor` parameter.

Example: Request that retrieves the twenty most recent blog posts would be:

```
http://external.ningapis.com/xn/rest/apiexample/1.0/BlogPost/recent?count=20
```

Example: Request that retrieves the next twenty blog posts would be:

```
http://external.ningapis.com/xn/rest/apiexample/1.0/BlogPost/recent?anchor=UxbFfoc1dq0wX1t,count=20
```

### Filter parameters

To retrieve more specific results, there are several parameters that can be used to filter the kind of data you're returning.

**Note:** Some resources don't support every filter parameter. Additionally, some filters can be used more than once. Check the resource reference section for a full list of filter parameters that a particular resource supports.

**approved** Only include posts that have been approved by the Network Creator, Administrators, or content moderators

**author** Only include content from a specific member of the site

**id** Retrieve only a specific content item

**private** If true the request will only include content that has it's visibility marked as `me` or `friends`. If request is not made by the Network Creator, the server will return 403 Forbidden.

### Other parameters

There are a handful of query parameters that tweak behavior to work around special edge cases.

**xn_pretty** If set to true, then JSON is pretty printed, making it much easier to read. This is useful when debugging.

**xn_always200** XMLHttpRequests have a tendency to not provide a body for non-200 responses. Setting this parameter to true causes all responses to be 200. The caller should use the status field of the response to get the real status code.

**xn_method** IE 8's support of cross-origin resource sharing doesn't preflight requests to allow non-GET or POST methods. Setting this parameter overrides the actual method of the request.

**xn_authInBody** IE 8's support of cross-origin resource sharing doesn't preflight requests to allow custom headers. If this parameter is set to true, then the first line (i.e., everything up to ASCII 10, aka, "n") of the body is treated as the value of the Authorization header.

**Warning:** Unknown parameters, duplicate parameters, or invalid parameter values will result in 400 error responses

## 2.3.4 Request Types

### Reading Content

The **GET** command is used to retrieve specific content or sets of content.

Example: Retrieve the 2 most recent photos on a site:

```
GET /xn/rest/apiexample/1.0/Photo/recent?fields=image.url,title&count=2 HTTP/1.1
Host: external.ningapis.com
Authorization: oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b29-a33c-2f4af1k
```

Example: Retrieve the 2 most recent photos from a site

```
curl -H 'Authorization: OAuth oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4k
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Photo/recent?xn_pretty=true&fields=image.ur
```

### Creating Content

The **POST** command is used for creating new content. The fields are specified as form-encoded parameters using either `application/x-www-form-urlencoded` or `multipart/form-data`. When uploading a file, the parameter should be named `file`.

Example: Create a blog post:

```
POST /xn/rest/apiexample/1.0/BlogPost?fields=title HTTP/1.1
Host: external.ningapis.com
Authorization: oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b29-a33c-2f4af1k
Content-Length: 47
Content-Type: application/x-www-form-urlencoded
title=BlogPost&description=BlogPost description
```

Example: Create a blog post using cURL

```
curl -H 'Authorization: OAuth oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4k
    -d 'title=BlogPost' \
    -d 'description=BlogPost description'
    'https://external.ningapis.com/xn/rest/apiexample/1.0/BlogPost?fields=title'
```

**Note:** Content creation respects the approval settings of the Ning Network. If the moderation is on then posts by members must be approved first.

### Updating Content

The **PUT** command is used to update existing content on a Ning Network. The fields are specified as form-encoded parameters using `application/x-www-form-urlencoded`. Unknown or read-only fields will be ignored.

Example: Update a photo:

```
PUT /xn/rest/apiexample/1.0/Photo?id=3011345:Photo:1077 HTTP/1.1
Host: external.ningapis.com
Authorization: oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b29-a33c-2f4af1k
Content-Length: 85
Content-Type: application/x-www-form-urlencoded

title=Updated Photo Title&description=Updated Photo Description&id=3011345:Photo:1099
```

```
curl -X PUT \
    -H 'Authorization: OAuth oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b2
    -d 'id=3011345:Photo:1099' \
    -d 'title=Updated Photo Title' \
    -d 'description=Updated Photo Description' \
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Photo?xn_pretty=true'
```

#### Deleting Content

The **DELETE** command is used to delete content on a Ning Network. The ID is specified as a parameter and you can only delete one content item at a time.

Example: Delete a photo:

```
DELETE http://external.ningapis.com/xn/rest/examplenetwork/1.0/Photo?id=1220998:Photo:5581
```

Example: Delete a photo using cURL

```
curl -X DELETE \
    -H 'Authorization: OAuth oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b2
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Photo?xn_pretty=true&id=3011345:Photo:1099'
```

## 2.4 Response Overview

### 2.4.1 Response Structure

Responses from resources are formatted as JSON objects. The exact structure of the response object will vary depending on the resource, parameters, and request type. Additional details on the exact structure of a response can be found in the sections below and on the *Request Types* page.

### 2.4.2 Caching

The data returned by queries is not cached, metadata used for determining access rights and site properties is cached for less than ten seconds.

This means that if a photo has it's visibility changed from `me` to `all`, it is possible that a request made by another member within ten seconds of the change will not include the photo in the response.

On the other hand, an update to a blog post's title will be see immediately by the next request.

### 2.4.3 Successful Response

#### Read responses

There are 3 types of responses you can get from a successful READ request: **single item**, **multi-item**, and **list**

##### Single Item Response

Single item requests are requests for a specific content item by it's ID. The response contains two properties: success and entry. Success will be true and entry will contain a JSON representation of the requested item. If the requested item cannot be found the server will respond with a 404 error.

Example: If you made a request for a blog post on a site:

```
GET http://external.ningapis.com/xn/rest/apiexample/1.0/BlogPost?id=2570916:BlogPost:322
```

Your response would be structured like the following:

```
{
    "success": true,
    "entry": [{
        "id": "2570916:BlogPost:322",
        "author": "3ixs6bzjxfkv6",
        "title": "Cycling Las Vegas",
        "description": "Cycling is Las Vegas is better than you would think"
    }],
}
```

### Multi-Item Response

The request is made for multiple content items using multiple `id` parameters. The resulting `entry` resource will be an array of JSON representations or an empty array of if none of the items could be found.

Example: If you made a request for 2 photos on a site:

```
GET http://external.ningapis.com/xn/rest/examplenetwork/1.0/Photo?&id=1220999:Photo:5605&id=1220999:
```

Your response would be structured like the following:

```
{
  "success" : true,
  "entry" : [ {
    "id" : "1220999:Photo:5591",
    "author" : "exampleuser",
    "createdDate" : "2010-01-26T23:20:13.591Z"
  }, {
    "id" : "1220999:Photo:5605",
    "author" : "qrwc2g2iqjhi",
    "createdDate" : "2010-02-26T20:49:06.605Z"
  } ],
  "resources" : {
  }
}
```

### List Response

Requests for lists of items (such as /BlogPost/recent) will return an array of JSON representations or an empty array. It will also contain information for paginating through the result set.

**anchor** A token used to identify the current page of results relative to the entire set

**firstPage** If true, this is the first page of the result set

**lastPage** If true, this is the last page of the result set

Example: If you made a request for the 5 most recent photos on a site:

```
GET http://external.ningapis.com/xn/rest/examplenetwork/1.0/Photo/recent?count=5
```

Your response would be structured like the following:

```
{
  "success" : true,
  "anchor" : "sBLHSuPH8xNPdBxyRHy3pw",
  "firstPage" : true,
  "lastPage" : false,
  "entry" : [ {
    "id" : "1220999:Photo:5868",
    "author" : "exampleuser",
    "createdDate" : "2010-06-02T10:54:41.868Z"
  }, {
    "id" : "1220999:Photo:5605",
    "author" : "qrwc2g2iqjhi",
    "createdDate" : "2010-02-26T20:49:06.605Z"
  }, {
    "id" : "1220999:Photo:5591",
    "author" : "exampleuser",
    "createdDate" : "2010-01-26T23:20:13.591Z"
  }, {
    "id" : "1220999:Photo:5590",
    "author" : "exampleuser",
    "createdDate" : "2010-01-26T23:09:02.590Z"
  }, {
    "id" : "1220999:Photo:5589",
    "author" : "exampleuser",
    "createdDate" : "2010-01-26T21:32:44.589Z"
  } ],
  "resources" : {
  }
}
```

### Count Response

Count requests return the number of items created after a user specified date. For a successful count request, your response will contain two properties: `success` and `count`. `success` will be `true` and `count` will be the number of items in the result set.

```
{
    "success": true,
    "count": 23
}
```

### Create responses

For a successful CREATE request, your response will contain two properties: `success` and `id`. `success` will be `true` and `id` will be the ID of the item you just created.

Example: If you made a request to create a blog post:

```
POST /xn/rest/apiexample/1.0/BlogPost?fields=title HTTP/1.1
Host: external.ningapis.com
Authorization: oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b29-a33c-2f4af1
Content-Length: 47
Content-Type: application/x-www-form-urlencoded
title=BlogPost&description=BlogPost description
```

The response would look like:

```
{
    "success": true,
    "id": "123:Photo:456"
}
```

### Update responses

For a successful UPDATE request, your response will contain 1 property: `success` and it's value will be `true`

Example: if you made a request to update a photo:

```
PUT /xn/rest/apiexample/1.0/Photo?id=3011345:Photo:1077 HTTP/1.1
Host: external.ningapis.com
Authorization: oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b29-a33c-2f4af1b
Content-Length: 85
Content-Type: application/x-www-form-urlencoded

title=Updated Photo Title&description=Updated Photo Description
```

The response would look like:

```
{
    "success": true
}
```

### Delete responses

For a successful DELETE request, your response will contain 1 property: `success` and it's value will be `true`

Example: if you made a request to delete a photo:

```
DELETE http://external.ningapis.com/xn/rest/examplenetwork/1.0/Photo?id=1220998:Photo:5581
```

The response would look like:

```
{
    "success": true
}
```

## 2.4.4 Failed Response

If a request fails, there are several pieces of info that will help clue you into what's going on:

**success** For failed requests, this will always be false

**reason** A human readable reason for the failure

**status** The HTTP status code for the request

**code** A numeric value indicating the category this error belongs to. This is meant to be consumed programmatically.

**subcode** A numeric value that indicates the specific error that was triggered. This is meant to be consumed programmatically.

**trace** An identifier that is used by Ning to debug issues. Include this number in any bug reports.

Here is an example of a failed response:

```
{
    "success": false,
    "reason": "Count must be less than 100, but got: 500",
    "status": 400,
    "code": 1,
    "subcode": 7,
    "trace": "4a076440-f132-4fbf-81bf-8f4d2eefa4c7"
}
```

## 2.5 Visibility

The Ning API respects the privacy settings of your content. There are 3 different visibility settings for content items:

**All** All members of your site can view the content item

**Friends** Only members you have added as friends on the site can view the content item

**Me** Only the creator of the content item can view it

**Note:** The privacy settings for a site does not apply to the Ning API (only the visibility settings of content items applies). This is done because you must be a member of the site to use the API. Members of a Ning Network can can already see the content on a private site, so custom functionality and code they interact with should be able to do the same.

### 2.5.1 Additional notes about visibility settings

Currently the **friends** visibility setting is treated the same as the **me** visibility setting.

An exception to the visibility rules is made for the Network Creator. The Network Creator can see all content, regardless of visibility settings. *At this time. Network administrators and content moderators are unable to see private content.*

### 2.5.2 Surfacing your own private content

There are 2 ways to retrieve your own private content.

#### Fetch items by `ID`

Example: To view one of your private blog posts you would use this request:

```
GET /xn/rest/apiexample/1.0/BlogPost?id=<Blog Post ID>
```

#### Filtering a list of results by the `author`

Example: To see a list of all your recent blog posts, including private posts, you would make the following request:

```
GET /xn/rest/apiexample/1.0/BlogPost/recent?author=<my screen name>
```

**Note:** A request made for a list of items without an author filter will only return content with visibility *all*.

## 2.6 Moderation

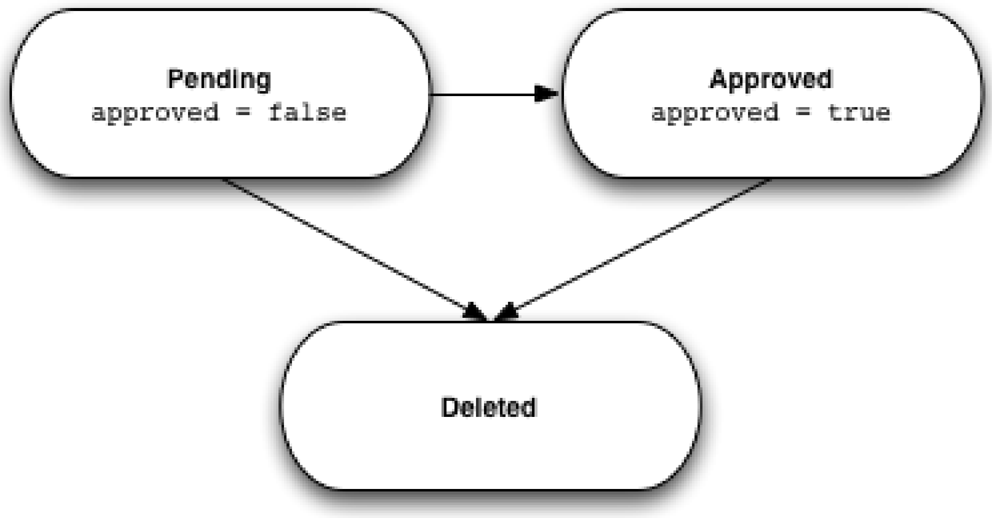
You can configure your site to only allow content to be posted after an administrator approves it. There are two types of moderation: content and member. Both are explained in detail bellow.

### 2.6.1 Content Moderation

Content moderation applies to the following features:

- *Photos*
- *Blog Posts*

There are three states for content: `pending`, `approved`, and `deleted`. You can see the flow of states here:



**Note:** Content cannot return to the pending state, it must be approved or deleted.

#### Checking for Pending Content

By default both approved and unapproved items are returned in a list request, you must specifically filter out the items you don't want to see. To filter out the unwanted content items, use the `approved` query parameter.

```
GET /BlogPost/recent?approved=false&fields=approved&count=10
```

The above request will return the ten most recent blog posts waiting to be approved. In this case there is only one blog post waiting moderation. Notice that the `approved` field is currently *false*:

```
{
    "anchor": "b1XpktNQ5xHwLpHrdkcDn6dkyRdcgZR0",
    "entry": [
        {
            "approved": false,
            "author": "0fhpmsd4e9ep2",
            "createdDate": "2011-04-12T21:46:11.771Z",
            "id": "3011345:BlogPost:14144"
        }
    ],
    "firstPage": true,
    "lastPage": true,
    "resources": {},
    "success": true
}
```

**Note:** I have included the `approved` field in the `fields` parameter to demonstrate that we received the correct result. This parameter is not necessary to make the `approved` filter function correctly.

### Approving Content

You can approve content by setting the `approved` field to *true* using a PUT request. In the example bellow the PUT request is made to the *blog post* endpoint. The body of the PUT request is shown on the second line.

```
PUT /BlogPost
id=3011345%3ABlogPost%3A14144&approved=true
```

If the member making the request is an administrator the server will respond with a success message:

```
{
    "success": true
}
```

Once approved you cannot return to the pending state. If you want to remove the content, you must delete it. If you want to temporarily hide content from the rest of the site you can set the *visibility field* to *me*.

### Rejecting Content

To reject content you must send a DELETE request with the ID of the content as a query parameter. You cannot send a PUT request to set the `approved` field to *false*. If you do, the server will respond with an error telling you to delete the content instead.

To delete a blog post with the ID *011345:BlogPost:14144*, the request would look like:

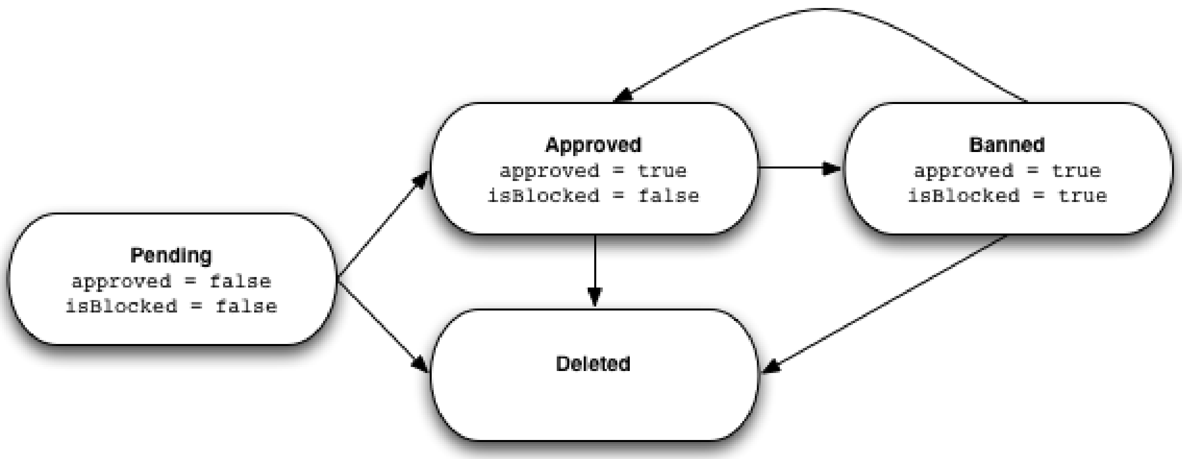
```
DELETE /BlogPost?id=3011345:BlogPost:14144
```

If the member making the request is an administrator the server will respond with a success message:

```
{
    "success": true
}
```

## 2.6.2 Member Moderation

Member moderation is similar to content moderation. The only difference is the introduction of a *banned* member state. Here is the flow of states:



When a member joins the site they are placed in the pending state until an administrator approves them. If they are approved, the member can start using the site. If they are rejected the member is deleted.

An approved member can leave the site in one of two ways: banning or deletion. A deleted member is completely removed from the system and may rejoin the site using the normal sign up flow.

A banned member has all of their content removed, but they remain in the system. If a banned member tries to join again, they receive a message letting them know that they are banned and cannot join again. An administrator can reinstate a banned member, however their content that was deleted when they were banned is not restored.

### Check for pending members

By default both approved and unapproved members are returned in a list request, you must specifically filter out the members you don't want to see. To filter out the unwanted members, use the `approved` query parameter.

```
GET /User/recent?approved=false&fields=approved&count=10
```

The above request will return the ten most recent pending members. In this case there is only one pending member. Notice that the `approved` field is currently *false*:

```
{
    "anchor": "PLPJF2UPMmrHklgsMzQbzRoGpVGmeg8s",
    "entry": [
        {
            "approved": false,
            "author": "13y7imet0czgr",
            "createdDate": "2011-04-12T23:51:28.759Z",
            "id": "3011345:User:14232"
        }
    ],
    "firstPage": true,
    "lastPage": true,
    "resources": {},
    "success": true
}
```

**Note:** I have included the `approved` field in the `fields` parameter to demonstrate that we received the correct result. This parameter is not necessary to make the `approved` filter function correctly.

### Approve a Pending Member

To approve a pending member, set the `approved` property to *true* using a PUT request. In the example bellow, the member with the ID *3011345:User:14232* is approved. The second line represents the body of the request:

```
PUT /User
id=3011345%3AUser%3A14232&approved=True
```

If the member making the request is an administrator the server will respond with a success message:

```
{
    "success": true
}
```

### Reject a Pending Member

To reject a pending member you must delete them by sending a DELETE request and specifying their ID. You cannot reject a member by setting the `approved` field to *false*. If you try, you will receive an error message. The following example rejects the member with the ID *3011345:User:14232*.

```
DELETE /User?id=3011345%3AUser%3A14232
```

If the member making the request is an administrator the server will respond with a success message:

```
{
    "success": true
}
```

### Ban Existing Member

To ban an existing member, send a PUT request and set the `isBlocked` field to *true*. In the example bellow, the member with the ID *3011345:User:14232* is banned. The second line represents the body of the request:

**Note:** When a member is banned, all of their content is deleted and cannot be recovered by unbanning.

```
PUT /User
isBlocked=True&id=3011345%3AUser%3A14232
```

If the member making the request is an administrator the server will respond with a success message:

```
{
    "success": true
}
```

### Reinstate a Banned Member

To reinstate a banned member, send a PUT request and set the `isBlocked` field to *false*. In the example bellow, the member with the ID *3011345:User:14232* is unbanned. The second line represents the body of the request:

**Note:** When a member is banned, all of their content is deleted and cannot be recovered by unbanning.

```
PUT /User
isBlocked=False&id=3011345%3AUser%3A14232
```

If the member making the request is an administrator the server will respond with a success message:

```
{
    "success": true
}
```

CHAPTER 3

# API Reference

## 3.1 Activity Items

### 3.1.1 Properties

**id** (*id*) ***[Read only]*** Unique identifier for this activity item. For example: `urn:uuid:009d0e07c93`

**author** (*screenname*) ***[Read only]*** The screen name of the member who generated the activity item

**type** (*activity*) ***[Read only]*** The specific action that generated the activity item

**createdDate** (*date*) ***[Read only]*** When the activity item was generated

**contentId** (*id*) ***[Read only]*** The ID for content that this activity item is about. For example: `94301:Photo:75451`. For JoinNetwork and StatusUpdate `acitivty types` the type is `screenname`.

**url** (*url*) ***[Read/Write]*** the URL to the result of an activity. For example, if a photo was created, this will be a link to the photo.

**title** (*string*) ***[Read/Write]*** The title of the content item that was generated. This only applies to content that has a title such as blog posts and photos.

**description** (*string*) ***[Read/Write]*** The text of the content item or a generic description of the activity. For `CreateBlogPost`, this would be the contents of the blog post. For `JoinNetwork` it would be "Jane has joined the The Cycling Network"

**attachedTo** (*id*) ***[Read only]*** **For comment activity only.** ID of the content the comment for. For example if a member commented on a photo, the `contentID` will be `94301:Comment:75451` and `attachedTo` would be `94301:Photo:8675`

**attachedToType** (*type*) ***[Read only]*** **For comment activity only.** The owner of the content that the comment is attached to. For example if a member commented on a photo, the `contentID` will be `94301:Comment:75451` and `attachedToType` would be `Photo`

**attachedToAuthor** (*id*) ***[Read only]*** **For comment activity only.** The author of the content that was commented on.

**image** (*id*) ***[Read only]*** **For photo activity only.** ID of the image referenced in the activity. To be used with the `image.url`, `image.width`, and `image.height` sub-properties.

### 3.1.2 Sub-Properties

**author.fullName** (*string*) ***[Read only]*** The full name of the author

**author.iconUrl** (*url*) ***[Read only]*** The author's profile photo

**author.url** (*url*) ***[Read only]*** The author's profile page on the Ning Network

**image.url** (*url*) ***[Read only]*** **Applies only to CreatePhoto and CreatePhotoComment activity items**. The URL of for the photo

**image.width** (*int*) ***[Read only]*** **Applies only to CreatePhoto and CreatePhotoComment activity items**. The width of the photo

**image.height** (*int*) ***[Read only]*** **Applies only to CreatePhoto and CreatePhotoComment activity items**. The height of the photo

### 3.1.3 HTTP GET

#### `/xn/rest/apiexample/1.0/Activity/recent`

Retrieve the latest activity from the Ning Network

##### Optional Query Parameters:

**author** (*screenname*) Retrieve activity items for a specific member

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one activity item by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving activity items before the `startIndex`.

#### `/xn/rest/apiexample/1.0/Activity/count`

Retrieve the number of activity items created after the given date

##### Required Query Parameters:

**createdAfter** (*date*) The ISO-8601 date to start counting after. If the date is more than a week old, a 400 response will be returned

##### Optional Query Parameters:

**author** (*screenname*) Retrieve the count of activity items for a specific member

### 3.1.4 HTTP DELETE

#### `/xn/rest/apiexample/1.0/Activity`

Retrieve the latest activity from the Ning Network

#### Required Query Parameters:

**id** (*id*) Delete a specific activity item

## 3.2 Blog Posts

Manage blog posts on your Ning Network

### 3.2.1 Properties

**id** (*id*) ***[Read only]*** The unique ID for a blog post

**author** (*screenname*) ***[Read only]*** The screen name of the member who created the blog post

**createdDate** (*date*) ***[Read only]*** The time stamp from the creation date

**publishTime** (*date*) ***[Read/Write]*** The time stamp for when to publish the blog post, can be a time in the future or the past. If you want to use the current time, use the value `now`.

**publishStatus** (*string*) ***[Read/Write]*** The status of the blog post, can be either `draft`, `queued`, or `publish`

**updatedDate** (*date*) ***[Read only]*** The time stamp from the last modification

**featuredDate** (*date*) ***[Read only]*** When the blog post was featured

**title** (*string*) ***[Read/Write]*** The title of the blog post

**description** (*string*) ***[Read/Write]*** The content of the blog post

**visibility** (*visibility*) ***[Read/Write]*** Can friends see or all members see the blog post

**approved** (*boolean*) ***[Read/Write]*** True if the administrator has approved the blog post

**commentCount** (*int*) ***[Read only]*** The number of comments on the blog post

**url** (*url*) ***[Read only]*** URL of the blog post

**topTags** (*string*) ***[Read only]*** A array of the most popular tags: `[ "food", "apple" ]`

**tag** (*string*) ***[Write only]*** Used during create and update requests to set tags on the blog post. Reading from this field is **depreciated**, instead use the `topTags` attribute.

### 3.2.2 Sub-Properties

**author.fullName** (*string*) ***[Read only]*** The full name of the author

**author.iconUrl** (*url*) ***[Read only]*** The author's profile photo

**author.url** (*url*) ***[Read only]*** The author's profile page on the Ning Network

### 3.2.3 HTTP GET

#### `/xn/rest/apiexample/1.0/BlogPost`

Retrieve a specific blog post or a specific set of blog posts, up to 100.

#### Required Query Parameters:

**id** (*id*) The ID of the blog post, multiple `id` parameters are allowed, up to 100

#### Optional Query Parameters:

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

### `/xn/rest/apiexample/1.0/BlogPost/recent`

Retrieve blog posts sorted by `createdDate`

#### Optional Query Parameters:

**author** (*screenname*) Retrieve blog posts for a specific user

**private** (*boolean*) If true, only retrieve private blog posts. If you are not the Network Creator, using this parameter will result in a 403 Forbidden error.

**approved** (*boolean*) If true, only retrieve approved blog posts

**tag** (*string*) Only include blog posts with the specified tag

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one blog post by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving blog posts before the `anchor`.

### `/xn/rest/apiexample/1.0/BlogPost/featured`

Retrieve blog posts sorted by `featuredDate`

#### Optional Query Parameters:

**author** (*screenname*) Retrieve blog posts for a specific user

**approved** (*boolean*) If true, only retrieve approved blog posts

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one blog post by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving blog posts before the `anchor`.

### `/xn/rest/apiexample/1.0/BlogPost/count`

Retrieve the number of blog posts created after the given date

#### Required Query Parameters:

**createdAfter** (*date*) The ISO-8601 date to start counting after. If the date is more than a week old, a 400 response will be returned

#### Optional Query Parameters:

**author** (*screenname*) Retrieve the number of blog posts for a specific user

**private** (*boolean*) If true, only retrieve the number of private blog posts. If you are not the Network Creator, using this parameter will result in a 403 Forbidden error.

**approved** (*boolean*) If true, only retrieve the number of approved blog posts

**tag** (*string*) Only count blog posts with the specified tag

### 3.2.4 HTTP POST

**`/xn/rest/apiexample/1.0/BlogPost`**

Create a new blog post

### 3.2.5 HTTP PUT

**`/xn/rest/apiexample/1.0/BlogPost`**

Update the blog post with the given ID

#### Required Query Parameters:

**id** (*id*) The ID of the blog post

### 3.2.6 HTTP DELETE

**`/xn/rest/apiexample/1.0/BlogPost`**

Delete the blog post with the given ID

#### Required Query Parameters:

**id** (*id*) The ID of the blog

## 3.3 Broadcast Message

Send broadcast messages to the members of your Ning Network

### 3.3.1 Properties

**subject** (*string*) ***[Read/Write]*** Subject of the broadcast message

**body** (*string*) ***[Read/Write]*** Body of the broadcast message

**messageId** (*string*) ***[Read/Write]*** Unique string within the target site. The purpose of this property is to prevent the accidental sending of duplicate messages. Currently the API checks for duplicates within the past five minutes. If a duplicate is detected a 4xx error response is returned.

### 3.3.2 HTTP POST

**`/xn/rest/apiexample/1.0/BroadcastMessage`**

Send a broadcast message

## 3.4 Comments

### 3.4.1 Properties

**id** (*id*) ***[Read only]*** Unique identifier for this comment. For example: `94301:Comment:8675`

**author** (*screenname*) ***[Read only]*** The screen name of the member who created the comment

**createdDate** (*date*) ***[Read only]*** When the comment was created

**updatedDate** (*date*) ***[Read only]*** When the comment was last updated

**featuredDate** (*date*) ***[Read only]*** When the comment was featured

**description** (*string*) ***[Read/Write]*** The text of the comment

**attachedTo** (*id*) ***[Read only]*** The ID of content the comment is for

**attachedToType** (*type*) ***[Read only]*** The type of content that the comment is attached to

**attachedToAuthor** (*id*) ***[Read only]*** The owner of the content that the comment is attached to

**approved** (*boolean*) ***[Read/Write]*** True if the administer has approved this comment

### 3.4.2 Sub-Properties

**author.fullName** (*string*) ***[Read only]*** The full name of the author

**author.iconUrl** (*url*) ***[Read only]*** The author's profile photo

**author.url** (*url*) ***[Read only]*** The author's profile page on the Ning Network

### 3.4.3 HTTP GET

**`/xn/rest/apiexample/1.0/Comment/recent`**

Retrieve comments sorted by `createdDate`

#### Required Query Parameters:

**attachedTo** (*id*) Retrieve comments for a specific content item

#### Optional Query Parameters:

**approved** (*boolean*) If true, only retrieve approved comments

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one comment by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving comments before the `anchor`.

#### `/xn/rest/apiexample/1.0/Comment/featured`

Retrieve comments sorted by `featuredDate`

#### Required Query Parameters:

**attachedTo** (*id*) Retrieve comments for a specific content item

#### Optional Query Parameters:

**approved** (*boolean*) If true, only retrieve approved comments

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one comment by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving comments before the `anchor`.

### 3.4.4 HTTP POST

#### `/xn/rest/apiexample/1.0/Comment`

Create a new comment. The `attachedTo` and `description` properties are required when creating a comment.

### 3.4.5 HTTP DELETE

#### `/xn/rest/apiexample/1.0/Comment`

Delete the comment with the given ID

**Required Query Parameters:**

**id** (*id*) The ID of the comment

## 3.5 Network

### 3.5.1 Properties

**id** (*id*) ***[Read only]*** Unique identifier for the Ning Network

**author** (*screenname*) ***[Read only]*** The screen name of the Network Creator

**createdDate** (*date*) ***[Read only]*** The date the Ning Network was created

**subdomain** (*string*) ***[Read only]*** The sub-domain of the Ning Network. For example, the subdomain of `http://apiexample.ning.com/` is `apiexample`

**name** (*string*) ***[Read only]*** The full name of the Ning Network

**iconUrl** (*url*) ***[Read only]*** The URL for the Ning Network's icon

**defaultUserIconUrl** (*url*) ***[Read only]*** The URL for the default profile photo for members who have not uploaded their own.

**blogPostModeration** (*boolean*) ***[Read only]*** True if blog posts must be approved by an administrator before being published

**userModeration** (*boolean*) ***[Read only]*** True if members must be approved by an administrator before joining the Ning Network

**photoModeration** (*boolean*) ***[Read only]*** True if photos must be approved by an administrator before being published

**eventModeration** (*boolean*) ***[Read only]*** True if events must be approved by an administrator before being published

**groupModeration** (*boolean*) ***[Read only]*** True if groups must be approved by an administrator before being published

**videoModeration** (*boolean*) ***[Read only]*** True if videos must be approved by an administrator before being published

### 3.5.2 Sub-Properties

**author.fullName** (*string*) ***[Read only]*** The full name of the author

**author.iconUrl** (*url*) ***[Read only]*** The author's profile photo

**author.url** (*url*) ***[Read only]*** The author's profile page on the Ning Network

### 3.5.3 HTTP GET

#### `/xn/rest/apiexample/1.0/Network`

Retrieve information about the Ning Network the request was made against

#### Optional Query Parameters:

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

### `/xn/rest/apiexample/1.0/Network/alpha`

Retrieve the Ning Networks your have created, sorted alphabetically by subdomain.

#### Optional Query Parameters:

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

## 3.6 Photos

Manage photos on your Ning Network

### 3.6.1 Properties

**id** (*id*) ***[Read only]*** The unique ID for a photo

**author** (*screenname*) ***[Read only]*** The screen name of the member who created the photo

**createdDate** (*date*) ***[Read only]*** The time stamp from the creation date

**updatedDate** (*date*) ***[Read only]*** The time stamp from the last modification

**featuredDate** (*date*) ***[Read only]*** When the photo was featured

**title** (*string*) ***[Read/Write]*** The title of the photo

**description** (*string*) ***[Read/Write]*** The content of the photo

**visibility** (*visibility*) ***[Read/Write]*** Can friends see or all members see the photo

**approved** (*boolean*) ***[Read/Write]*** True if the administer has approved the photo

**commentCount** (*int*) ***[Read only]*** The number of comments on the photo

**url** (*url*) ***[Read only]*** URL of the photo's detail page

**topTags** (*string*) ***[Read only]*** A array of the most popular tags: `[ "food", "apple" ]`

**tag** (*string*) ***[Write only]*** Used during create and update requests to set tags on the photo. Reading from this field is **depreciated**, instead use the `topTags` attribute.

### 3.6.2 Sub-Properties

**author.fullName** (*string*) ***[Read only]*** The full name of the author

**author.iconUrl** (*url*) ***[Read only]*** The author's profile photo

**author.url** (*url*) ***[Read only]*** The author's profile page on the Ning Network

**image.url** (*url*) ***[Read only]*** The URL of for the photo

**image.width** (*int*) ***[Read only]*** The width of the photo

**image.height** (*int*) ***[Read only]*** The height of the photo

### 3.6.3 HTTP GET

#### `/xn/rest/apiexample/1.0/Photo`

Retrieve a specific photo or a specific set of photos, up to 100

##### Required Query Parameters:

**id** (*id*) The ID of the photo, multiple `id` parameters are allowed, up to 100

##### Optional Query Parameters:

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

#### `/xn/rest/apiexample/1.0/Photo/recent`

Retrieve photos sorted by `createdDate`

##### Optional Query Parameters:

**author** (*screenname*) Retrieve photos for a specific user

**private** (*boolean*) If true, only retrieve private photos. If you are not the Network Creator, using this parameter will result in a 403 Forbidden error.

**approved** (*boolean*) If true, only retrieve approved photos

**tag** (*string*) Only include photos with the specified tag

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one photo by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving photos before the `anchor`.

#### `/xn/rest/apiexample/1.0/Photo/featured`

Retrieve photos sorted by `featuredDate`

##### Optional Query Parameters:

**author** (*screenname*) Retrieve photos for a specific user

**approved** (*boolean*) If true, only retrieve approved photos

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one photo by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving photos before the `anchor`.

#### `/xn/rest/apiexample/1.0/Photo/count`

Retrieve the number of photos created after the given date

##### Required Query Parameters:

**createdAfter** (*date*) The ISO-8601 date to start counting after. If the date is more than a week old, a 400 response will be returned

##### Optional Query Parameters:

**author** (*screenname*) Retrieve the number of photos for a specific user

**private** (*boolean*) If true, only retrieve the number of private photos. If you are not the Network Creator, using this parameter will result in a 403 Forbidden error.

**approved** (*boolean*) If true, only retrieve the number of approved photos

**tag** (*string*) Only count photos with the specified tag

### 3.6.4 HTTP POST

#### `/xn/rest/apiexample/1.0/Photo`

Create a new photo. The request must be sent as `multipart/form-data`, where the image data is sent as under a part nammed `file`.

### 3.6.5 HTTP PUT

#### `/xn/rest/apiexample/1.0/Photo`

Update the photo with the given ID. If you are updating the image data, the request must be sent as `multipart/form-data`, where the image data is sent as under a part nammed `file`.

##### Required Query Parameters:

**id** (*id*) The ID of the photo

### 3.6.6 HTTP DELETE

#### `/xn/rest/apiexample/1.0/Photo`

Delete the photo with the given ID

#### Required Query Parameters:

**id** (*id*) The ID of the photo

## 3.7 Tags

Retrieve the list of tags attached to a content item

### 3.7.1 HTTP GET

#### `/xn/rest/apiexample/1.0/Tag/list`

Retrieve tags attached to a content item, sorted in no particular order.

#### Required Query Parameters:

**attachedTo** (*id*) Retrieve tags for the specified content item

#### Optional Query Parameters:

**author** (*screenname*) Only retrieve tags added by a particular member

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one tag by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving tags before the `anchor`.

## 3.8 OAuth Tokens

### 3.8.1 Properties

**oauthToken** (*string*) ***[Read only]*** OAuth access token that allows you to access a member's data on the Ning Network

**oauthTokenSecret** (*string*) ***[Read only]*** OAuth access token secret that is used to sign OAuth requests

**oauthConsumerKey** (*string*) ***[Read only]*** A unique string that that identifies the consumer

**author** (*string*) ***[Read only]*** The screenname of the member that the `oauthToken` is to be used for.

### 3.8.2 HTTP POST

#### `/xn/rest/apiexample/1.0/Token`

Creates a new OAuth access token for the current user. The user is verified using *basic authentication* This is to be used for 2-legged OAuth. Currently tokens do not expire. See *Authentication* for details on using this resource.

## 3.9 Users

Manage members on your Ning Network

### 3.9.1 Properties

**id** (*id*) ***[Read only]*** The unique ID the user object

**author** (*screenname*) ***[Read only]*** The screen name of the member who ones the profile

**createdDate** (*date*) ***[Read only]*** The time stamp from when member joined the Ning Network

**updatedDate** (*date*) ***[Read only]*** The time stamp from when the member last updated their profile

**featuredDate** (*date*) ***[Read only]*** When the member was featured

**approved** (*boolean*) ***[Read/Write]*** True if the administrator has approved the member. Note that setting this value to false is not supported. To reject a pending member you must send a DELETE request.

**visibility** (*visibility*) ***[Read only]*** Who can view the members profile page

**email** (*string*) ***[Read only]*** The members email address

**url** (*url*) ***[Read only]*** URL for the member's profile page on the Ning Network

**fullName** (*string*) ***[Read only]*** The members name on the Ning Network

**iconUrl** (*url*) ***[Read only]*** The URL for the member's profile photo

**birthdate** (*date*) ***[Read only]*** The day the member was born, for example 1980-07-18

**commentCount** (*int*) ***[Read only]*** The number of comments on the member's profile page

**gender** (*string*) ***[Read only]*** The gender of the member.

**location** (*string*) ***[Read only]*** A string containing the member's city, country and zip code

**isOwner** (*boolean*) ***[Read only]*** True if this member is the Network Creator

**isAdmin** (*boolean*) ***[Read only]*** True if this member is the Network Creator or an administrator for this Ning Network. This value does not include members who have moderation permissions (roles).

**isMember** (*boolean*) ***[Read only]*** True if this member is a member of the Ning Network

**isBlocked** (*boolean*) ***[Read/Write]*** True if the member has been banned from the Ning Network

**state** (*string*) ***[Read only]*** The precedence of the states is `owner > admin > (member|pending|unfinished|unknown)`

**statusMessage** (*string*) ***[Read/Write]*** The current status message for this member

**profileQuestions** (*string*) ***[Read only]*** An array of profile questions and their answers. For example: `[ {"question" : "Your favorite color", "answer" : "blue" } ]`

**Note:** The `email` and `profileQuestions` properties are only accessible by the Network Creator and the owner of the profile.

### 3.9.2 Sub-Properties

**author.fullName** (*string*) ***[Read only]*** The full name of the author

**author.iconUrl** (*url*) ***[Read only]*** The author's profile photo

**author.url** (*url*) ***[Read only]*** The author's profile page on the Ning Network

### 3.9.3 HTTP GET

#### `/xn/rest/apiexample/1.0/User`

Retrieve specific member or a specific set of members. You can use any combination of `id` and `author` parameters, up to 100. If you make a request without the `id` or `author` parameters, the result will be the User resource for the member who made the request.

##### Optional Query Parameters:

**id** (*id*) The ID of the member, multiple `id` parameters are allowed, up to 100.

**author** (*screenname*) The screen name of the member, multiple `author` parameters are allowed, up to 100.

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

#### `/xn/rest/apiexample/1.0/User/recent`

Retrieve members sorted by `createdDate`

##### Optional Query Parameters:

**isMember** (*boolean*) If true, only returns members in the `admin`, `member`, or `owner` states

**approved** (*boolean*) If true, only retrieve approved members

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one member by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving members before the `anchor`.

#### `/xn/rest/apiexample/1.0/User/alpha`

Retrieve members sorted by `fullName`

##### Optional Query Parameters:

**isMember** (*boolean*) If true, only returns members in the `admin`, `member`, or `owner` states

**approved** (*boolean*) If true, only retrieve approved members

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one member by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving members before the `anchor`.

#### `/xn/rest/apiexample/1.0/User/featured`

Retrieve members sorted by `featuredDate`

##### Optional Query Parameters:

**isMember** (*boolean*) If true, only returns members in the `admin`, `member`, or `owner` states

**approved** (*boolean*) If true, only retrieve approved members

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one member by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving members before the `anchor`.

#### `/xn/rest/apiexample/1.0/User/count`

Retrieve the number of members joined after the given date

##### Required Query Parameters:

**createdAfter** (*date*) The ISO-8601 date to start counting after. If the date is more than a week old, a 400 response will be returned

##### Optional Query Parameters:

**isMember** (*boolean*) If true, only returns the number of members in the `admin`, `member`, or `owner` states

**approved** (*boolean*) If true, only retrieve the number of approved members

### 3.9.4 HTTP PUT

#### `/xn/rest/apiexample/1.0/User`

Update the status of the member

##### Optional Query Parameters:

**id** (*id*) The ID of the member such as `94301:User:8675`

### 3.9.5 HTTP DELETE

#### `/xn/rest/apiexample/1.0/User`

Rejects a pending user. This does not ban an existing member. To ban an existing member, you need to send a PUT request with `isBlocked` set to true.

##### Required Query Parameters:

**id** (*id*) The ID of the member such as `94301:User:8675`

## 3.10 Friend

Manage your friends and access list of friends of another member on your Ning Network.

### 3.10.1 Properties

**author** (*screenname*) ***[Read only]*** The screen name of the member who made the friend request

**friend** (*screenname*) ***[Read/Write]*** The screen name of the member who received the friend request

**state** (*string*) ***[Read/Write]*** The state of the relationship between the author and friend. Can be either `pending`, `friend`, or `blocked`.

### 3.10.2 HTTP GET

#### `/xn/rest/apiexample/1.0/Friend/recent`

Retrieve your list of friend or the friends of another member.

##### Optional Query Parameters:

**author** (*screenname*) Retrieve the list of friends for a specific member

**state** (*string*) Only return friends in a certain state. Can be either `friend`, `friend_request`, `follower`, `pending`, `blocked_friend`, `blocked_follower`

**fields** (*field*) Comma-separated list of desired properties, this is only a hint to the server

**anchor** (*string*) An opaque token encodes the page of results returned, used in conjunction with `count` to page through a result set

**count** (*int*) Returns one friend by default and supports a maximum of 100. Also supports negative values down to -100 for retrieving friends before the `anchor`

### 3.10.3 HTTP POST

#### `/xn/rest/apiexample/1.0/Friend`

Add another member as a friend. The `state` of the relationship will be `pending` until the member accepts the request.

### 3.10.4 HTTP DELETE

**`/xn/rest/apiexample/1.0/Friend`**

Remove the friend with the given screenname.

#### Required Query Parameters:

**friend** (*screenname*) The screenname of the friend to remove

**state** (*string*) Type of relationship to remove, supports `friend` or `follower`

## 3.11 Property Type Definitions

**activity** The type of activity. We currently support:

- CreateBlogPost
- CreateBlogPostComment
- CreatePhoto
- CreatePhotoComment
- CreateUserComment
- CreateTopic
- CreateTopicComment
- JoinNetwork
- StatusUpdate

**boolean** Accepted values: `true` or `false`

**date** Dates are specified as ISO-8601 strings. The timezone will always be Z (UTC). For example: 2009-03-06T23:01:41.049Z

**field** A property of the content type

**fields** Comma-separated list of desired `fields`, this is only a hint to the server

**id** A unique ID that identifies a specific content item

**int** An integer

**screenname** A unique string that identifies a member, such as `3ixs6bzjxfkz6`

**string** A plain string

**type** The name of a Ning content type. We currently support:

- BlogPost
- Photo
- Activity

**url** A standard URL such as http://www.ning.com/

**visibility** Who can see the content item. Supported values are:

- `all`

- `me`
- `friends`

## 3.12 Error Codes

When the Ning API is unable to return a *succesful response*, it will return an error code. The table bellow is a reference for every error code, subcode, and HTTP status.

| Group | Error | HTTP Status | Code | Subcode |
|---|---|---|---|---|
| Unknown | Internal error | 500 Internal Server Error | 0 | 0 |
| Authorization | Internal error | 500 Internal Server Error | 1 | 0 |
| Authorization | Bogus authorization header | 400 Bad Request | 1 | 1 |
| Authorization | Invalid username or password | 401 Unauthorized | 1 | 2 |
| Authorization | Invalid credentials | 401 Unauthorized | 1 | 3 |
| Authorization | Additional authorization required | 401 Unauthorized | 1 | 4 |
| Authorization | Consumer key refused | 401 Unauthorized | 1 | 5 |
| Authorization | Consumer key rejected | 401 Unauthorized | 1 | 6 |
| Authorization | Consumer key unknown | 401 Unauthorized | 1 | 7 |
| Authorization | Nonce used | 401 Unauthorized | 1 | 8 |
| Authorization | Parameter absent | 400 Bad Request | 1 | 9 |
| Authorization | Permission denied | 401 Unauthorized | 1 | 10 |
| Authorization | Permission unknown | 401 Unauthorized | 1 | 11 |
| Authorization | Signature invalid | 401 Unauthorized | 1 | 12 |
| Authorization | Method rejected | 400 Bad Request | 1 | 13 |
| Authorization | Timestamp refused | 401 Unauthorized | 1 | 14 |
| Authorization | Token expired | 401 Unauthorized | 1 | 15 |
| Authorization | Token rejected | 401 Unauthorized | 1 | 16 |
| Authorization | Token revoked | 401 Unauthorized | 1 | 17 |
| Authorization | Token used | 401 Unauthorized | 1 | 18 |
| Authorization | User refused | 401 Unauthorized | 1 | 19 |
| Authorization | Version rejected | 400 Bad Request | 1 | 20 |
| Authorization | Parameter rejected | 400 Bad Request | 1 | 21 |
| Authorization | Missing basic authentication | 401 Unauthorized | 1 | 22 |
| Authorization | Invalid user name | 401 Unauthorized | 1 | 23 |
| Authorization | Invalid password | 401 Unauthorized | 1 | 24 |
| RequestMetadata | Internal error | 500 Internal Server Error | 2 | 0 |
| RequestMetadata | Network not found | 404 Not Found | 2 | 1 |
| RequestMetadata | Resource not found | 404 Not Found | 2 | 2 |
| RequestMetadata | Bogus params | 400 Bad Request | 2 | 3 |
| RequestMetadata | User not allowed | 403 Forbidden | 2 | 4 |
| Content | Internal error | 500 Internal Server Error | 3 | 0 |
| Content | Content not found | 404 Not Found | 3 | 1 |
| Content | Bogus value | 400 Bad Request | 3 | 2 |
| Content | Unavailable | 503 Service Unavailable | 3 | 3 |
| Query | Bogus field | 400 Bad Request | 4 | 1 |
| Query | Bad parameter | 400 Bad Request | 4 | 2 |
| Update | Internal error | 500 Internal Server Error | 5 | 0 |
| Update | Disabled | 403 Forbidden | 5 | 1 |
| Update | Forbidden | 403 Forbidden | 5 | 2 |

Continued on next page

**Table 3.1 – continued from previous page**

| Group | Error | HTTP Status | Code | Subcode |
|---|---|---|---|---|
| Update | Bad key | 400 Bad Request | 5 | 3 |
| Update | Needs captcha | 403 Forbidden | 5 | 4 |
| Update | Spammy | 400 Bad Request | 5 | 5 |
| Update | Soft blocked | 503 Service Unavailable | 5 | 6 |
| Update | Bad content id | 400 Bad Request | 5 | 7 |
| Update | Bad title | 400 Bad Request | 5 | 8 |
| Update | Bad description | 400 Bad Request | 5 | 9 |
| Update | Bad timestamp | 400 Bad Request | 5 | 10 |
| Update | Bad type | 400 Bad Request | 5 | 11 |
| Update | Bad object | 400 Bad Request | 5 | 12 |
| Update | Bad comment | 400 Bad Request | 5 | 13 |
| Update | Bad mime type | 400 Bad Request | 5 | 14 |
| Update | Cant add | 403 Forbidden | 5 | 15 |
| Update | Too big | 400 Bad Request | 5 | 16 |
| Update | File missing | 400 Bad Request | 5 | 17 |
| Update | Feature failure | 500 Internal Server Error | 5 | 18 |
| Update | Not moderator | 403 Forbidden | 5 | 19 |
| Update | Not logged in | 403 Forbidden | 5 | 20 |
| Update | Not admin | 403 Forbidden | 5 | 21 |
| Update | Unknow user | 400 Bad Request | 5 | 22 |
| Update | Not found | 404 Not Found | 5 | 23 |
| Update | Not member | 403 Forbidden | 5 | 24 |

CHAPTER 4

# API Examples

## 4.1 Accessing the Ning API with cURL

cURL is a command line utility that can be used to interact with the Ning API. It is mostly used for playing around with the Ning API. In this example, all requests use the `xn_pretty=true` parameter to make responses easier to read.

All of the requests using the following consumer credentials:

**Consumer Key** 0d716e57-5ada-4b29-a33c-2f4af1b26837

**Consumer Secret** f0963fa5-1259-434f-86fc-8a17d14b16ca

The photo requests use the following token:

**Access Key** a2f85402-f16c-4677-91e2-a334d362ad47

**Access Secret** b42a0833-e1e2-4b02-a906-258a157bc702

### 4.1.1 Retrieving tokens

```
#!/bin/bash

# Request the access token for the given member using the Ning API

curl -u test@example.com \
    -d 'oauth_signature_method=PLAINTEXT&oauth_consumer_key=0d716e57-5ada-4b29-a33c-2f4af1b26837&oaut
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Token?xn_pretty=true'
```

Response:

```
{
  "success" : true,
  "entry" : {
    "author" : "2cpor74jnszaj",
    "oauthConsumerKey" : "0d716e57-5ada-4b29-a33c-2f4af1b26837",
    "oauthToken" : "f450a524-652b-4700-9877-b2e8868fc0ca",
    "oauthTokenSecret" : "0cc82e2f-1a6b-4747-b846-01681ac5e25d"
  },
  "resources" : {
  }
}
```

## 4.1.2 Retrieving photos

```
#!/bin/bash

# Query the Ning API for the five most recent photos, returning the photo's
# title and the URL of the image

curl -H 'Authorization: OAuth oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Photo/recent?xn_pretty=true&fields=title,in
```

Response:

```
{
  "success" : true,
  "anchor" : "uxxWYGU-F4KrKZhlAq2toZ4o2xoGphLM",
  "firstPage" : true,
  "lastPage" : false,
  "entry" : [ {
    "id" : "3011345:Photo:3120",
    "author" : "2cpor74jnszaj",
    "createdDate" : "2010-09-29T20:21:27.120Z",
    "title" : "Updated Photo Title",
    "image" : "3011345:Photo:3120"
  }, {
    "id" : "3011345:Photo:3112",
    "author" : "2cpor74jnszaj",
    "createdDate" : "2010-09-29T20:07:16.112Z",
    "title" : "Updated Photo Title",
    "image" : "3011345:Photo:3112"
  }, {
    "id" : "3011345:Photo:3110",
    "author" : "2cpor74jnszaj",
    "createdDate" : "2010-09-29T20:06:36.110Z",
    "title" : "Photo Title",
    "image" : "3011345:Photo:3110"
  }, {
    "id" : "3011345:Photo:3108",
    "author" : "2cpor74jnszaj",
    "createdDate" : "2010-09-29T19:09:39.108Z",
    "title" : "Expedition 24 Soyuz Landing (201009250033HQ)",
    "image" : "3011345:Photo:3108"
  }, {
    "id" : "3011345:Photo:3105",
    "author" : "2cpor74jnszaj",
    "createdDate" : "2010-09-29T19:09:34.105Z",
    "title" : "Here Comes Comet Hartley 2! (NASA, Comets, 09/28/10)",
    "image" : "3011345:Photo:3105"
  } ],
  "resources" : {
    "3011345:Photo:3120" : {
      "url" : "http://api.ning.com:80/files/K4J5EOVUeUGMYc-jmIhbesRX1hcoxFDq-QXet5OGyAlNSxCCNGe1XPsAN
    },
    "3011345:Photo:3112" : {
      "url" : "http://api.ning.com:80/files/CNnwhHNSBj5HWGOq9jcI6c2a**IaIFlQyGTOBd-HUELNDIZ2LENqMvO3-
    },
    "3011345:Photo:3110" : {
      "url" : "http://api.ning.com:80/files/fB839UrKcK*Z*jIxTfir9tzRgUfkUW1Dv73E2arr6k9j-LpKNF3x5owOI
    },
    "3011345:Photo:3108" : {
      "url" : "http://api.ning.com:80/files/O2p*fNpRXae4ESKRO6d0jPVkUSnB8hfGecHIO9osYxTHGEY1Ae5m*K2U
    },
    "3011345:Photo:3105" : {
      "url" : "http://api.ning.com:80/files/RCys80y56jsyOuoP3mhZKEsDXa93J3wcol3hv44VN2nVoLkhyc8ejloV
    }
  }
}
```

### 4.1.3 Add a new photo

```
#!/bin/bash

# Upload a photo using the Ning API

curl -H 'Authorization: OAuth oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4
    -F 'title=Photo Title' \
    -F 'description=Photo Description' \
    -F 'file=@/home/test/img/cat.jpg' \
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Photo?xn_pretty=true'
```

Response:

```
{
  "success" : true,
  "id" : "3011345:Photo:3120"
}
```

### 4.1.4 Update a photo

```
#!/bin/bash

# Update a photo using the Ning API

curl -X PUT \
    -H 'Authorization: OAuth oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b
    -d 'id=3011345:Photo:3120' \
    -d 'title=Updated Photo Title' \
    -d 'description=Updated Photo Description' \
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Photo?xn_pretty=true'
```

Response:

```
{
  "success" : true
}
```

### 4.1.5 Delete a photo

```
#!/bin/bash

# Delete a photo using the Ning API

curl -X DELETE \
    -H 'Authorization: OAuth oauth_signature_method="PLAINTEXT",oauth_consumer_key="0d716e57-5ada-4b2
    'https://external.ningapis.com/xn/rest/apiexample/1.0/Photo?xn_pretty=true&id=3011345:Photo:3120'
```

Response:

```
{
  "success" : true
}
```

## 4.2 Accessing the Ning API with PHP

PHP is a popular scripting language used for making web applications. Here are some examples of how to use the Ning API with your own PHP scripts. The examples use the Ning API PHP library.

To use the Ning API PHP library, replace the following values in `NingApi.php`:

- `$subdomain`
- `$email`
- `$password`
- `$consumerKey`
- `$consumerSecret`

### 4.2.1 Retrieving tokens

The Ning API PHP library automatically requests a token when it is needed. The developer adds their email and password to the `NingApi.php` configuration file.

### 4.2.2 Retrieving photos

```
<?php
require_once('ning-api-php/NingApi.php');

// Get the most recent photo
$result = NingApi::instance()->photo->fetchNRecent();

print_r($result);
```

Outputs:

```
Array
(
    [success] => 1
    [anchor] => JSEwMVyGby2GnKyw-yJEDs03z9v-8loI
    [firstPage] => 1
    [lastPage] =>
    [entry] => Array
        (
            [0] => Array
                (
                    [id] => 3843070:Photo:525
                    [author] => 2cpor74jnszaj
                    [url] => http://apiexample.ning.com/xn/detail/3843070:Photo:525
                    [createdDate] => 2011-03-29T20:54:48.130Z
                    [updatedDate] => 2011-03-29T20:54:48.156Z
                    [title] => Photo Title
                    [description] => Photo Description
                    [visibility] => all
                    [tags] => Array
                        (
                        )

                    [image] => 3843070:Photo:525
                )

        )

    [resources] => Array
        (
            [3843070:Photo:525] => Array
                (
                    [height] => 600
                    [width] => 600
                    [url] => http://api.ning.com:80/files/1LE8YIrT3NQScp05KW4olxtlpPowyQl1Jui*8yJUrr
                )

            [2cpor74jnszaj] => Array
                (
                    [fullName] => John Quest
                    [iconUrl] => http://api.ning.com/files/RDvnKXegw0e*W997gwuLIb-wjvgV--aoqOtPha6tx5
                    [url] => http://apiexample.ning.com/profile/JohnQuest
                )

        )

)
```

### 4.2.3 Add a new photo

```php
<?php
require_once('ning-api-php/NingApi.php');

$parts = array(
    "title" => "Photo Title",
    "description" => "Photo Description",
    "file" => "@/Users/devin/Pictures/nasa/NASA-01.jpg"
);

// Create a new photo
$result = NingApi::instance()->photo->create($parts);

print_r($result);
```

Outputs:

```
Array
(
    [success] => 1
    [id] => 3843070:Photo:525
)
```

### 4.2.4 Update a photo

```php
<?php
require_once('ning-api-php/NingApi.php');

$parts = array(
    "title" => "Photo Title",
    "description" => "Photo Description",
    "file" => "@/Users/devin/Pictures/nasa/NASA-01.jpg"
);

// Create a new photo
$newPhoto = NingApi::instance()->photo->create($parts);

$parts = array(
    "title" => "Updated Photo Title",
    "description" => "Updated Photo Description",
    "id" => $newPhoto['id']
);

// Update the photo we created
$result = NingApi::instance()->photo->updateById($parts);

print_r($result);
```

Outputs:

```
Array
(
    [success] => 1
)
```

### 4.2.5 Delete a photo

```php
<?php
require_once('ning-api-php/NingApi.php');

$parts = array(
    "title" => "Photo Title",
    "description" => "Photo Description",
    "file" => "@/Users/devin/Pictures/nasa/NASA-01.jpg"
);

// Create a new photo
$newPhoto = NingApi::instance()->photo->create($parts);

// Delete the photo we created
$result = NingApi::instance()->photo->delete($newPhoto);
```

```
print_r($result);
```

Outputs:

```
Array
(
    [success] => 1
)
```

## 4.3 Accessing the Ning API with Python

The Ning API Python library simplifies the code required to interact with the Ning API.

### 4.3.1 Retrieving tokens

```python
"""
Request the access token for the given member using the Ning API
"""

import oauth2 as oauth
import ningapi

consumer = oauth.Consumer(
        key="0d716e57-5ada-4b29-a33c-2f4af1b26837",
        secret="f0963fa5-1259-434f-86fc-8a17d14b16ca"
        )

host = "external.ningapis.com"
network = "apiexample"

ning_api = ningapi.Client(host, network, consumer)


email = "test@example.com"
password = "I<3Ning"

token = ning_api.login(email, password)

print "Access Key: %s" % token.key
print "Access Secret: %s" % token.secret
```

Outputs:

```
Access Key: 30a29805-1fa0-4498-a832-96b46f00bf07
Access Secret: ab67211f-f14f-43b4-971b-81431722a593
```

### 4.3.2 Retrieving photos

```python
"""
Query the Ning API for the five most recent photos, returning the photo's
title and the URL of the image
"""

```

```python
import oauth2 as oauth
import ningapi

consumer = oauth.Consumer(
        key="0d716e57-5ada-4b29-a33c-2f4af1b26837",
        secret="f0963fa5-1259-434f-86fc-8a17d14b16ca"
        )
token = oauth.Token(
        key="07aa5613-6783-4735-b8f1-4c69642ad438",
        secret="e2c528ec-8b81-402c-8f88-8bf17ba8751f"
        )

host = "external.ningapis.com"
network = "apiexample"

ning_api = ningapi.Client(host, network, consumer, token)


fields = ["title", "image.url"]
attrs = {
    "fields": ",".join(fields),
    "count": 5
}

content = ning_api.get("Photo/recent", attrs)
for photo in content["entry"]:
    photo_id = photo["id"]
    photo_resource = content["resources"][photo_id]

    print "%s\n\t%s" % (photo["title"], photo_resource["url"])
```

Outputs:

```
Photo Title 1
        http://api.ning.com:80/files/v-cOIk8putiKMSg0Bi8m3zM5oLTKyMJ*sB5Mza2zijWo7ko9qrPwA38cx0evq*LS
Photo Title 2
        http://api.ning.com:80/files/FA1Wz3cE6XK9cOnGAzbckgX8DbbF-D-S*mqwEEljZZkE9snrAS0tiXoLnYC8-n3
Photo Title 3
        http://api.ning.com:80/files/5jv2LhKnz09*0EUKWctnCAPJTtRbALYpYqvIEzUcq3xfadrB5EMtLL7ieMslr7y
Photo Title 4
        http://api.ning.com:80/files/-OpCvgGDYR3T6wvL-uDQr3dXHR*TwJv-wNIrmKE1A2kDF7c7rODetxNgmpT1hXU
Photo Title 5
        http://api.ning.com:80/files/0xSW8BNOLVbZBmP8m86Grid3-8nAx0iUJKgH-qHbx37eB6jj-QhbZdgegtpvbgd
```

### 4.3.3 Add a new photo

```python
"""
Upload a photo using the Ning API
"""

import oauth2 as oauth
import ningapi

consumer = oauth.Consumer(
        key="0d716e57-5ada-4b29-a33c-2f4af1b26837",
        secret="f0963fa5-1259-434f-86fc-8a17d14b16ca"
        )
```

```
token = oauth.Token(
        key="07aa5613-6783-4735-b8f1-4c69642ad438",
        secret="e2c528ec-8b81-402c-8f88-8bf17ba8751f"
        )

host = "external.ningapis.com"
network = "apiexample"

ning_api = ningapi.Client(host, network, consumer, token)


photo_title = "Photo Title"
photo_desc = "Photo Description"
photo_path = "/Users/devin/Pictures/nasa/NASA-23.jpg"
photo_content_type = "image/jpeg"

photo_fields = {
        "title": photo_title,
        "description": photo_desc,
        "file": photo_path,
        "content_type": photo_content_type
        }

content = ning_api.post("Photo", photo_fields)

if content["success"]:
    print "Photo uploaded: %s" % content["id"]
```

Outputs:

```
Photo uploaded: 3011345:Photo:3932
```

### 4.3.4 Update a photo

```
"""
Update a photo using the Ning API
"""

import oauth2 as oauth
import ningapi

consumer = oauth.Consumer(
        key="0d716e57-5ada-4b29-a33c-2f4af1b26837",
        secret="f0963fa5-1259-434f-86fc-8a17d14b16ca"
        )
token = oauth.Token(
        key="07aa5613-6783-4735-b8f1-4c69642ad438",
        secret="e2c528ec-8b81-402c-8f88-8bf17ba8751f"
        )

host = "external.ningapis.com"
network = "apiexample"

ning_api = ningapi.Client(host, network, consumer, token)


photo_title = "Updated Photo Title"
```

```
photo_desc = "Updated Photo Description"

photo_fields = {
        "title": photo_title,
        "description": photo_desc,
        "id": "3011345:Photo:3930"
        }

content = ning_api.put("Photo", photo_fields)

if content["success"]:
    print "Photo updated"
```

Outputs:

```
Photo updated
```

### 4.3.5 Delete a photo

```
"""
Delete a photo using the Ning API
"""

import oauth2 as oauth
import ningapi

consumer = oauth.Consumer(
        key="0d716e57-5ada-4b29-a33c-2f4af1b26837",
        secret="f0963fa5-1259-434f-86fc-8a17d14b16ca"
        )
token = oauth.Token(
        key="07aa5613-6783-4735-b8f1-4c69642ad438",
        secret="e2c528ec-8b81-402c-8f88-8bf17ba8751f"
        )

host = "external.ningapis.com"
network = "apiexample"

ning_api = ningapi.Client(host, network, consumer, token)



photo_fields = {
        "id": "3011345:Photo:3928"
        }

content = ning_api.delete("Photo", photo_fields)

if content["success"]:
    print "Photo deleted"
```

Outputs:

```
Photo deleted
```

# Index

A

activity, **39**

B

boolean, **39**

D

date, **39**

F

field, **39**
fields, **39**

I

id, **39**
int, **39**

S

screenname, **39**
string, **39**

T

type, **39**

U

url, **39**

V

visibility, **39**