

Thursday, September 1, 2005

### Flickr API Fun

I like stuff I can pick up and do something useful with in an hour or two. Perhaps my attention span is too short, but if I have to read a 300 page spec before I get to Hello World, then it's not for me. Or you would at least have to pay me a lot of money to suffer through it. I think people refer to this as "immediacy". For me I think it is mostly lazyness. If I can't figure it out in an hour, it's broken as far as I am concerned.

Flickr's REST API is not broken. You can read all about it at <http://flickr.com/services/api>.

There are links there to various wrappers for the API, but I ended up writing my own. I have a bad habit of doing that. This entry will focus on my PHP wrapper for the Flickr API. It is based on Cal's version and is compatible with it, but it expands on it and puts some PHP 5.1 features to good use. You can see it here:

[http://lerdorf.com/php/flickr\\_api.phps](http://lerdorf.com/php/flickr_api.phps)

Before you get started, in case you want to follow along, go get yourself an API key at

<http://flickr.com/services/api/key.gne>

You will need two pieces of information to fully use the API. An API key and an API secret. And if you are going to do anything that requires authentication, you need to set a callback url as well. More on that later. To get your secret after applying for and getting your API key, go to

[http://www.flickr.com/services/api/registered\\_keys.gne](http://www.flickr.com/services/api/registered_keys.gne)  
and click on "Edit Configuration".

Many functions in the API do not require authentication. Getting a list of someone's public photos, for example, is something anybody can do by just browsing Flickr, or by just going to this URL:

[http://flickr.com/services/rest/?method=flickr.people.getPublicPhotos&user\\_id=56053642@N00&api\\_key=3aba8184848f9263b80795c95529bcd1](http://flickr.com/services/rest/?method=flickr.people.getPublicPhotos&user_id=56053642@N00&api_key=3aba8184848f9263b80795c95529bcd1)

Guess what, you just sent a REST Web Services query.

Or, slightly cooler. A list of tags related to the tag you provide based on Flickr's clustering code.

[http://flickr.com/services/rest/?method=flickr.tags.getRelated&tag=monkey&api\\_key=3aba8184848f9263b80795c95529bcd1](http://flickr.com/services/rest/?method=flickr.tags.getRelated&tag=monkey&api_key=3aba8184848f9263b80795c95529bcd1)

The whole point of web services is to provide data in a machine-readable way so you can do something more interesting with it. That's where the API wrapper comes in. You can of course also use Flickr's feed mechanism to do this.

But back to the API and the PHP wrapper. Getting a list of someone's public photos is done like this:

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

```
$secrets = array('api_key'=>'your_key_here','api_secret'=>'your_secret'); $flickr = new Flickr($secrets); $photos = $flickr->peopleGetPublicPhotos('56053642@N00');
```

This will give you an array of photos. Or to be precise, an array of information about the photos. Note the mapping of the method name. `$flickr->peopleGetPublicPhotos` maps to `flickr.people.getPublicPhotos` in the documentation. And the returned XML is converted to a more useful (and more memory-cacheable - I'll write something up soon on that) PHP array. The example result XML for 2 photos looks like this:

Which gets mapped to this PHP array (in `print_r` format):

```
Array (
  [page] => 1
  [pages] => 1
  [perpage] => 100
  [total] => 2
  [photos] => Array (
    [39006009] => Array (
      [id] => 39006009
      [owner] => 56053642@N00
      [secret] => f2086066d5
      [server] => 33
      [title] => IMG_7564.JPG
      [ispublic] => 1
      [isfriend] => 0
      [isfamily] => 0
    )

    [39006000] => Array (
      [id] => 39006000
      [owner] => 56053642@N00
      [secret] => 4ec57bd51f
      [server] => 28
      [title] => IMG_7551.JPG
      [ispublic] => 1
      [isfriend] => 0
      [isfamily] => 0
    )
  )
)
```

To turn a photo into a URL you can use in an IMG tag you would call the `$flickr->getPhotoURL()` method. It isn't very complex. Here is what it does:

```
function getPhotoURL($p, $size='s', $ext='jpg') { return "http://photos{$p['server']}.flickr.com/{$p['id']}{$p['secret']}{$size}.$ext"; }
```

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

The default size is the small 75x75 square thumbnail. See the URL Documentation for further info. So here is the full code to put up the first 50 thumbnails from someone's photostream:

The contents of secrets.inc is just that \$secrets array I referred to above. You can see the output of this script at flickr\_demo1.php.

Flickr users are uniquely identified by a very cryptic-looking nsid. You don't see this id anywhere when you are clicking around on Flickr. But you can look up a user's nsid if you know their photo url, their user name or their email address. flickr\_demo2.php shows you how to do that. Change the u= parameter in the URL to look up other users.

Playing with the non-authenticated functions of the API can get you far, but Flickr also lets you authenticate, and it will let the users using your application authenticate themselves. That lets you do a whole class of cool things that something like the RSS feed mechanism doesn't provide. For example, I wrote a Gallery to Flickr migration tool that can take my Gallery photo albums and copy the pictures to Flickr and put them in a Flickr set with the same name as the Gallery album they came from. You could also write alternative frontends for it and integrate your Flickr photos with your own web site. Or perhaps write a Gallery plugin that uses Flickr as the backend. All sorts of possibilities here.

But in order to do any sort of reading of non-public information or writing to your Flickr account via the API, you have to authenticate. Flickr uses a token-based authentication system where you make a roundtrip to flickr.com for the user to log into his flickr account and choose whether or not to grant your application the requested level of access. That means that your application never sees the user's credentials, but instead gets a token with the appropriate rights associated with it that it can then use. Each API call then includes this token, the application's key and all the arguments for whatever method you are calling and a signature using your application-specific secret across all the arguments. That means that even if someone sniffs your traffic, all they can do is replay the exact API call. They can't use it to execute arbitrary things against your account. Users can also remove an application's access later by going to <http://www.flickr.com/services/auth/list.gne>. The system is described at <http://www.flickr.com/services/api/auth.spec.html> and is worth a read if you are interested, but you don't really need to understand it. Just use the wrapper. Here is how:

This probably looks a bit cryptic. This says that if you already have a token, we simply create a \$flickr object and we are ready to go. If there is no token on the request and there is no 'frob', then redirect the user to the authentication URL which is generated by the call to \$flickr->getAuthUrl with the desired permission level as an argument. The user will then get sent back to your callback url, which you would set to this same script most likely, and on that callback we still don't have a token, but you will be called with a frob parameter. A call to \$flickr->getFrobToken turns the frob into a token. You actually get back an auth array containing not just the token but also the user's nsid, permission level for the token, username and fullname. The idea is then that you include the above

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

blurb on your pages, as flickr\_auth.inc, for example and pass the token along from page to page in your web application.

Now we can write our first full little authenticated example. Not much to it. We just call `$flickr->authCheckToken` on our token to see what Flickr thinks of the token we are using.

You can see the output by clicking on flickr\_demos.php and selecting flickr\_demo3.

So, by having those 3 includes at the top, by the time we get control we have a fully authenticated `$flickr` object that we can start using.

So, an all-out demo. In flickr\_demo4 we upload a photo:

```
$photo_id = $flickr->upload($fname,$title,$desc,$tags,$perms,0);
```

Check to see if you already have a set named "Sample Set". If you don't, create it (adding the uploaded photo at the same time):

```
$set = $flickr->photosetsCreate("Sample Set", $photo_id);
```

If you do already have that set, add the uploaded photo to it:

```
$flickr->photosetsAddPhoto($set_id, $photo_id);
```

Then we can add a note:

```
$note_id = $flickr->photosNotesAdd($photo_id,342,70,50,50,"This is Carl");
```

Get info on the photo:

```
$photo = $flickr->photosGetInfo($photo_id);
```

And get a direct URL to it:

```
$url = $flickr->getPhotoURL($photo,'m');
```

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Have a look at the full source code and try it by going to [http://lerdorf.com/php/flickr\\_demos.php](http://lerdorf.com/php/flickr_demos.php) and clicking on demo4.

Even if you don't use Flickr, I think there are a lot of interesting things here. An interesting web service authentication mechanism, a nice and clean REST API that allows for complex operations, and some PHP 5.1 XML and stream handling if you look closely at the flickr\_api code.

Posted by Rasmus in Software at 09:00

You may be interested in looking at Phlickr, an object oriented PHP 5, Flickr API kit. It also uses CURL and SimpleXML. It's pretty easy to use and well documented.

Anonymous on Sep 1 2005, 15:53

Yeah, I looked at Phlickr. It returns raw SimpleXML objects though. In order to cache responses in APC's shared memory segment without having to serialize them they need to be parsed further down into arrays.

Anonymous on Sep 1 2005, 15:59

Why don't you use `require_once`?

Anonymous on Sep 1 2005, 22:59

If you know what you are including and when, you shouldn't ever need to use `include_once/require_once`. They are also slower than doing a straight `include/require`.

Anonymous on Sep 1 2005, 23:12

But `_once` avoids possible multiple inclusions, and `require` prevents further execution (and whatever various error messages) if the `.inc` files get moved, `chmod'd`, etc.

How much slower are they? I never heard of any performance difference. Can't be much.

BTW, nice article - I checked out flickr's API page a while ago and didn't really understand what was possible with it. Your demos help.

Anonymous on Sep 1 2005, 23:20

Why take a performance hit when none is needed. If the include files get moved around, a `require_once` won't fix that anyway. Always using `require_once` calls is just a bad habit as far as I am concerned. There may be a few common files that are prone to multiple inclusion in a complex application, but most shouldn't be included multiple times and if you are including them multiple times you'd probably want to know about it and not just ignore it the way the `_once` functions do.

As far as stopping execution goes, that has nothing to do with `_once`. That's a `require` vs. `include` thing.

Anonymous on Sep 2 2005, 00:46

I hate to argue with you about PHP ;), get all OT here, and about something pretty trivial, but I gotta disagree - other languages won't let you get past including files that can't be included. I always use `require_once`, except when `include_once` is needed (avoids loading code that may or may not be used). This prevents scripts from trying to do who-knows-what after a failed include (which happens sometimes after upgrading, etc.), and spitting out who-knows-what to users.

There'd probably be less of a performance hit by avoiding OOP too, but the other benefits outweigh that.

Anonymous on Sep 2 2005, 01:07

Again, `_once` has nothing to do with what happens on a failed include. You can make a case for using `require` over `include` so you get a hard failure, but you don't always want a hard failure and in this particular example you get your hard failure right away anyway as you would end up calling a function that hasn't been defined. You also get a big fat warning that you can choose to make fatal in your error handler, so the difference is pretty minor.

Anonymous on Sep 2 2005, 01:17

> `_once` has nothing to do with what happens on a failed include

I know!

I can't think of a case where I wouldn't want a hard failure if required (heh) files weren't included. And I would rather not rely on some other hopefully resulting error terminating the script (sometimes they can get pretty far and print out lots of warnings, errors, etc.).

Anonymous on Sep 2 2005, 01:24

Sure, if you are not using your own error handler, that is probably true. But when you do have your own error handler, it is more convenient to get a warning because you then have full control over what you can do. With an `E_FATAL` it doesn't matter what you do in your error handler, you are done.

I should probably write up something on error handling at some point. Perhaps something for my next long flight.

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Anonymous on Sep 2 2005, 01:27

Ah, okay, you have a point there. But doesn't having your own error handler reduce performance? ;P  
Anonymous on Sep 2 2005, 01:34

Sure, on an error things get slower. But I don't care about performance outside of my common codepath. On my common codepath I do however care a lot and go out of my way to make sure things are fast.  
Anonymous on Sep 2 2005, 01:38

Can you elaborate on how `require_once` (and `require` too?) hits performance? That's the first I've heard of it. I can't imagine it's much. I didn't see anything in the docs, except for a comment suggesting using a wrapper class(!) instead.  
Anonymous on Sep 2 2005, 01:50

The `_once` functions have more work to do. I never said `require` was slower than `include`. The two are pretty much identical in the code. But the `_once` functions are slower. The beauty of open source is that it doesn't matter what the docs say, nor what you hear, you have the source code and the definitive answer at your fingertips:

[http://cvs.php.net/co.php/ZendEngine2/zend\\_vm\\_def.h?r=1.59.2.5#2666](http://cvs.php.net/co.php/ZendEngine2/zend_vm_def.h?r=1.59.2.5#2666)

`include/require` is a one-liner. `include_once/require_once` need to fiddle around with the `included_files` hash. Easy enough to benchmark. Take a little script that just does:

```
< ?php include 'inc.inc';? >
```

And in `inc.inc` we have:

```
This is a test  
< ?php echo "Hello World"? >  
This is a test
```

On one of my boxes I get:

```
10000 fetches, 5 max parallel, 410000 bytes, in 8.60147 seconds  
41 mean bytes/connection  
1162.59 fetches/sec, 47666.2 bytes/sec  
msecs/connect: 0.263676 mean, 20.707 max, 0.109 min  
msecs/first-response: 3.88822 mean, 1987.42 max, 0.722 min  
HTTP response codes:  
code 200 -- 10000
```

Then change `inc.php` to use `require_once` instead and I get:

```
10000 fetches, 5 max parallel, 410000 bytes, in 9.1419 seconds  
41 mean bytes/connection  
1093.86 fetches/sec, 44848.4 bytes/sec  
msecs/connect: 0.27873 mean, 33.92 max, 0.11 min  
msecs/first-response: 4.12044 mean, 369.463 max, 0.741 min  
HTTP response codes:  
code 200 -- 10000
```

Running it a few times back and forth the `require_once` is always 55 to 75 requests per second slower than a straight `require`. Of course, in a test that does nothing else, the effect is magnified and pales in comparison to an SQL query or anything real. It is unlikely to ever be your bottleneck, but that doesn't change the fact that it is slower. Whether it is measurable in your system is something you will have to test for yourself.

Anonymous on Sep 2 2005, 02:23

> and pales in comparison to an SQL query or anything real

Absolutely. Does the performance difference of `include` vs. `require_once` make any difference to most everybody in real world situations? I doubt anybody, but perhaps Yahoo, would have traffic at those levels where it might make a difference. But they also have money to just add more hardware too ;). There are many many other factors that tweaking would bring result in performance improvements. I'm reminded of the last slide of your "Why PHP?" presentation (which I can't seem to find online anymore): Don't spend all your time optimizing every last thing - there's a point of diminishing returns (paraphrasing). I also mostly use double quotes instead of single quotes, even when vars aren't inserted or strings could be concatenated together, even though I've heard single quotes are marginally faster. I find double quotes more convenient and readable. I mostly just worry about optimizing database queries - they're the lion's share of performance bottlenecks. And REST/SOAP calls. Beyond that, almost nobody will notice any benefit from additional performance tweaks.

Anonymous on Sep 2 2005, 11:23

Hi, any chance of you sharing your Gallery to Flickr migration tool with me/us/the world? I'm about to do exactly that and I'd love to have it automated!! I'm good with code/php/whatever so it'd be fine if you didn't want to support it or anything.

Thanks!

Anonymous on Sep 4 2005, 19:03

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

I've thought about this, and will continue to use `include_once` and `require_once`, and think the arguments against it are really weak. Any performance impact is negligible. And it's really really bad form to let your code continue to execute past errors.  
Anonymous on Sep 23 2005, 13:45

Depends on your scale and your OS. An extra `open()` syscall (when using an opcode cache) for every include can add up quickly.  
Anonymous on Sep 23 2005, 14:01

Hi, maybe I'm an idiot but have you made the dependencies for your demo files (aka `secrets.inc` and `flickr_api.inc`) available to download?

I'm lost--please help me!  
Anonymous on Sep 25 2005, 05:44

It's one of the first links in the article:

[http://lerdorf.com/php/flickr\\_api.phps](http://lerdorf.com/php/flickr_api.phps)

And I explain what should be in the secrets file:

```
$secrets = array('api_key'=>'your_key_here','api_secret'=>'your_secret');
```

Anonymous on Sep 25 2005, 07:08

hi,

i just downloaded the api and played around with it - very nice, thanks! i now use it to display a random selection of my flickr favorites on my site.

check it out here:  
<http://me.phillipoertel.com>

the source:  
[http://me.phillipoertel.com/downloads/flickr\\_favorites.zip](http://me.phillipoertel.com/downloads/flickr_favorites.zip)  
Anonymous on Oct 3 2005, 13:13

Hey Rasmus - You don't have to prefix your private variables with underscores anymore! ;)  
Anonymous on Oct 3 2005, 17:01

It's habit. That way I know they are internal as I use them without having to hunt for the declaration.  
Anonymous on Oct 3 2005, 17:06

Well, I finally got a chance to try this out. Mega kudos for making it crystal clear. Your wrapper classes work great so far. Only issue I came across is the `getPhotoURL` function - no `"_{size}"` suffix is added for medium (  
Anonymous on Oct 3 2005, 18:43

Whoa, looks like last part of my comment got stripped. It was:

&lt; 500 pixels) images.

BTW, great to see that APC has been updated to work with PHP 5. Last I checked, none of the other caches had been updated to work with PHP 5.  
Anonymous on Oct 3 2005, 18:45

I changed that function to:

```
function getPhotoURL($p, $size='s', $ext='jpg') {
    if(empty($size)) {
        return "http://photos{$p['server']}.flickr.com/{$p['id']}_{$p['secret']}.{$ext}";
    } else {
        return "http://photos{$p['server']}.flickr.com/{$p['id']}{$p['secret']}{$size}.{$ext}";
    }
}
```

Anonymous on Oct 3 2005, 18:49

Hmm, my underscores got munged in that else bit.  
Anonymous on Oct 3 2005, 18:51

I added a couple function I found useful:

```
// A really powerful and useful function
function photosSearch($user_id="", $tags="", $tag_mode="", $text="", $min_upload_date="", $max_upload_date=",
```

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

```
$min_taken_date=", $max_taken_date=", $license=", $extras=", $per_page=", $page_sort=") {
    $params = array(
        'user_id' => $user_id,
        'tags' => $tags,
        'tag_mode' => $tag_mode,
        'text' => $text,
        'min_upload_date' => $min_upload_date,
        'max_upload_date' => $max_upload_date,
        'min_taken_date' => $min_taken_date,
        'max_taken_date' => $max_taken_date,
        'license' => $license,
        'extras' => $extras,
        'per_page' => $per_page,
        'page_sort' => $page_sort,
    );

    $xml = $this->callMethod('flickr.photos.Search',$params);
    if(!$xml) { return FALSE; }
    foreach($xml->photos->attributes() as $k=>$v) {
        $ret[$k] = (string)$v;
    }
    $i=0;
    foreach($xml->photos->photo as $photo) {
        foreach($photo->attributes() as $k=>$v) {
            $ret['photos'][$(string)$photo['id']][$k] = (string)$v;
        }
        $i++;
    }
    return $ret;
}

function urlsGetUserProfile($user_id) {
    $params = array('user_id'=>$user_id);
    $xml = $this->callMethod('flickr.urls.GetUserProfile',$params);
    if(!$xml) { return FALSE; }
    $attr = $xml->user->attributes();
    return (string)$attr['url'];
}
Anonymous on Oct 6 2005, 16:52
```

Hey Rasmus, thanks for creating this - its a cool toy! Also gave me an excuse to install PHP5 on my server.

Here's a simple update to get data on photosizes -

```
function getPhotoSizes($id) {
    $params = array('photo_id'=>$id);
    $xml = $this->callMethod('flickr.photos.getSizes',$params);

    if(!$xml) { return FALSE; }

    $i=0;
    foreach($xml->sizes->size as $size) {
        foreach($size->attributes() as $k=>$v) {
            $ret[$i][$k] = (string)$v;
        }
        $i++;
    }
    return $ret;
}
```

Your output loop could do something like this -

```
foreach($photos['photos'] as $photo) {
    $sizes = $flickr->getPhotoSizes($photo['id']);
    if (is_array($sizes)) {
        foreach($sizes as $size) {
            print " ";
        }
    }
}
```

Anonymous on Oct 24 2005, 04:58

Hey Rasmus,

I paid a guy to do me a gallery2flickr type-o-script to transfer all my 3500+ pics from Gallery to Flickr. If your interested, please check it out here:

[http://www.in-duce.net/archives/migration\\_from\\_gallery\\_to\\_flickr.php](http://www.in-duce.net/archives/migration_from_gallery_to_flickr.php)

cheers.

Anonymous on Nov 14 2005, 05:44

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

I've been enjoying your beautiful adaptation.

Q: How can I edit that will random display from my own Flickr photos?

Thankx in advance,

Arthur

Anonymous on Dec 14 2005, 09:46

Hi,

This is everso slightly off-topic - as I am not using this API php, but parsing the basic flickr RSS. eg:  
[http://www.flickr.com/services/feeds/photos\\_public.gne?tags=cats&format=rss\\_200](http://www.flickr.com/services/feeds/photos_public.gne?tags=cats&format=rss_200)

But, I'm going completely mad here, trying to use SimpleXML to extract the media:thumbnail url attribute of the Flickr RSS feed, which uses the 'media' namespace located at

<http://search.yahoo.com/mrss/> (in the xmlns tag)

I can parse the feed OK, but when it comes to getting the thumbnail url, I tried this:

...where \$feeditem is the name assigned while looping the items with a foreach:

```
$media = $feeditem->children("http://search.yahoo.com/mrss/");  
$thumb = $media->thumbnail['url'];
```

but I don't get anything out??? :(

I was finding all this SimpleXML really easy to grasp and very, very welcome with alot of work I'm doing with OPML and RSS, but this has me foxed.

If anyone could be of any assistance at all, I would be truly grateful - and I'll doc it too ;)

eg:

```
< rss >  
< channel >  
< item >  
< media:thumbnail url="http://etc.etc.etc" / >  
< /item >  
< channel >  
< /rss >
```

many thanks for a wonderful new version of PHP!

Cheers,

Kosso

Anonymous on Jan 5 2006, 20:12

You could just use my RSS parser:

[http://www.lerdorf.com/php/simple\\_rss.phps](http://www.lerdorf.com/php/simple_rss.phps)

Or look through the code to see how I solved the namespace problem.

Anonymous on Jan 5 2006, 20:50

Hi,

Thanks for that, but it kind of does alot more than I need ;)

However, I did manage to get the url attribute I need while looping the channel->item as \$feeditem with

```
$thumburl = $feeditem->children('http://search.yahoo.com/mrss')->content->attributes();
```

though, oddly the attributes() only throws back one value, the url, but not the other two attributes, height and width from:

```
< media:thumbnail url="http://flickr.com/etc/etc/yada/yada.jpg" height="75" width="75" / >
```

surely there should be 3?

great parser you have there though ;) thx.

Kosso

Anonymous on Jan 6 2006, 15:06

sorry, that was meant to be:

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

```
$thumbnail = $feeditem->children('http://search.yahoo.com/mrss')->thumbnail->attributes();  
Anonymous on Jan 6 2006, 15:16
```

Really a great page, thanks for taking the time to write this. Damn, another 2 hours spent having fun with images APIs instead of sleeping because of you ;-)  
About "flickr\_demo2.php shows you how to do that." I like the output and could not trace the required function, would you care to share it or show us how ? Thanks again.  
Anonymous on Mar 19 2009, 15:04

Hello u

How to get my Flickr API key ?  
Anonymous on Dec 22 2009, 02:30