
parselyapi Documentation

Release 0.1

Parsely, Inc.

April 23, 2012

CONTENTS

1 Parse.ly Developer Manual	3
1.1 Why use the Parse.ly API?	3
1.2 Application Programming Interface (API)	3
1.3 Contact the Parse.ly Team	27
2 Indices and tables	29
Index	31

Parse.ly provides an API built specifically for the needs of online content publishers.

If you run a content site of any kind – online news site, magazine, vertical content, or blog – and want to better engage your users with your content, Parse.ly can help.

We parse and analyze your content using sophisticated natural language processing technology and analyze reader behavior and engagement with that content to expose trends in real-time.

We also expose this data and various sitewide insights through our real-time analytics product, Parse.ly Dash.

PARSE.LY DEVELOPER MANUAL

Note

You can read this document by navigating the contents listed below. However, if you prefer, we also have a [single HTML page with the entire manual](#) and a [gorgeous, printable PDF](#) you can use, instead.

1.1 Why use the Parse.ly API?

Publisher technology teams want simple, data-driven ways to increase engagement on their sites.

Parse.ly's platform already analyzes your content and groups all of it by post, author, section, and topic. The API provides a way to get access to the same data available in *Parse.ly Dash* – namely, detailed, real-time analytics of how content is performing on your site. The API also provides a mechanism for providing content search and recommendation functionality to your users.

Parse.ly API access is bundled into the Plan and Promote tiers of the Parse.ly Dash product. Simply integrating Dash provides access to this API, and you may use the control panel in Dash to manage your API keys.

[Sign up for Dash](#) to gain access.

1.2 Application Programming Interface (API)

1.2.1 API Specification

API Tutorial

The root URL for all Parse.ly API requests is:

`http://api.parse.ly.com/v2/`

Authenticating requests

Parse.ly's APIs support three authentication schemes which are configurable in your Dash account. These are:

- Public
- Shared Secret

- OAuth

Public is obviously the simplest authentication scheme: there is none! You can temporarily turn on Public requests to play with the API in your browser. However, then any of the below requests are available to any web clients, so this is not enabled by default and must be enabled explicitly by a staff member on your Dash account. Some publishers do not mind pageview information about their site being publicly available (indeed, some publishers like Gawker and Forbes display this information proudly on their pages), and for some publishers, being able to access our HTTP/JSON API via its JSON-P interface simplifies development and integration.

Shared Secret is the next simplest authentication scheme: there is a secret “string” you must pass to all requests to the API. This string can be regenerated from the Dash interface. This is good for customers who want to use the API for internal reporting and systems but do not want to deal with the sometimes onerous setup involved for authentication via OAuth.

Note

Shared Secret is essentially “security by obscurity”. If you can keep your secret hidden from the public *and* you know that any requests you make to the API will be behind a secure network, then this is a good option to hide your data from others. But note that merely *transmitting* the HTTP request with your secret embedded is a security risk, and that technically-savvy hackers who use wire sniffing technology could determine your secret key. The OAuth scheme should be used if the discovery of this data by the public would be a cause for great concern.

OAuth is our “full security” option for customers who consider their pageview data to be highly sensitive information. In this case, you must use a consumer key and consumer secret to access the API, and you must use an OAuth client library to integrate with it.

HTTP and JSON API conventions

To keep things nice and organized, we have identified all the URLs exposed by our API and listed all of their query parameters and status codes in this reference document.

Note

This API is implemented as HTTP endpoints with JSON responses. There are some parameters related to authentication that are common to all requests, however other endpoints have custom query parameters documented below. Where possible, we have tried to provide representative examples of request and response patterns.

Request Templates Most of our requests are HTTP GET requests. All requests support the following parameters:

- `apikey` – which should be aligned with the API key specified in your Parse.ly JavaScript tracker; this is also viewable in your Dash account. This is required for all requests.
- `secret` – if the simple shared secret authentication scheme is enabled, this will be required for every request.
- `callback` – to support the JSON-P standard, all GET requests can take a callback parameter which will wrap the JSON response in the specified callback function, allowing easy integration into web frontends via JavaScript.

No special headers are required for your requests in the Public or Shared Secret authentication schemes; thus you can do testing using standard tools like cURL or Chrome’s JSONView plugin.

When you are using OAuth authentication scheme, HTTP headers required by the OAuth standard will be required for all of your requests.

Many requests also support pagination via the `limit` and `page` parameters. Some requests also support sorting via the `sort` parameter, but its valid values depend on the endpoint.

Response Templates All Parse.ly API requests are expected to return one or more matching records. If requests fail, they return failure HTTP status codes.

Successful responses (code 200) are JSON documents using a “data envelope” format, which has this form:

```
{
  "data": [ {}, {}, {}, ... ]
}
```

Therefore, after you use a JSON library to parse a response from the API, access the `response.data` field to get a list of matching records. Here is simple jQuery code that makes a Parse.ly API request and iterates each of the resulting documents:

```
jQuery.getJSON(
  "http://api.parsely.com/analytics/posts?apikey=apikey.com",
  function printDocuments(response) {
    jQuery.each(response.data, function(i, record) {
      console.log(record.title + " | " + record.url);
    });
  }
);
```

Similar code could be written using standard HTTP libraries for programming languages.

Note

We are using this envelope to future-proof the API – in the future, we may use the envelope for storing fields such as response version numbers, pagination information, or detailed status/error reporting.

Inside the `data` list, the Parse.ly API typically returns one of the two following content types:

- **Posts** (typically article pages with associated url)
- **Metas** (Authors, Sections, Topics)

The Analytics API will return a `_hits` field on both Posts and Metas, indicating the number of pageviews in the selected time period. The Search and Recommendation APIs will not include this `_hits` field.

The Shares API will return a `_shares` field on both Posts and Metas, indicating the number of social shares in the selected time period. The Search and Recommendation APIs will not include this `_shares` field.

A **Post** is the Parse.ly API’s primary data type, and it has the following fields:

- title (required)
- url (required)
- author
- section
- pub_date
- thumb_url_medium

Fields not marked required may be missing in the response documents.

A **Meta** is a grouping mechanism for reporting analytics from several posts. Metas have only one field, the meta itself. So, the “author” meta includes the field “author”, with the string identifying the author. In the Analytics and Shares APIs, the Metas will include `_hits` or `_shares` counts. Metas cannot be retrieved in the Search and Recommendation APIs.

Analytics

Here is an example response document from an `GET /analytics/{type}` request, specifying `posts` as the type:

```
{
  "data": [
    {
      "url": "http://apikey.com/1234",
      "title": "Growing need for concert composers found in NYC",
      "section": "Music",
      "author": "John Smith",
      "pub_date": "2012-05-01T00:00:00",
      "_hits": 123354
    },
    // other similar JSON documents...
  ]
}
```

Let us assume this is the top post on the site. In this case, the top author on the site might also be the author of this post. A query to the `/analytics/authors` would thus reveal:

```
{
  "data": [
    {
      "author": "John Smith",
      "_hits": 183358
    },
    // other similar JSON documents...
  ]
}
```

The `_hits` value for the author meta is bigger than the `_hits` value in the individual post because the meta is grouping traffic information from multiple posts by John Smith.

Likewise, let us assume that this breakout post has made the Music section the top section on the site. If that's the case, then this will be the result of a query to the `/analytics/sections` endpoint:

```
{
  "data": [
    {
      "section": "Music",
      "_hits": 254808
    },
    // other similar JSON documents...
  ]
}
```

In this case, the “Music” Section Meta has much more traffic than either the Post or the “John Smith” Author meta. This is because it is aggregating traffic from multiple posts **by multiple authors** within the section.

A Meta is special because it can also be used to get a listing of posts that fall within that Meta – in other words, you can “peek inside the group” contributing to the summed `_hits` value.

This is also useful for e.g. showing “top posts by an author” or “top posts within a section”. The endpoint for this is simple: `GET /analytics/{meta}/{value}/detail`. So, following the above example, we could do `/analytics/author/John%20Smith/detail` and get a response like this:

```
{
  "data": [
    {
      "url": "http://apikey.com/1234",
      "title": "Growing need for concert composers found in NYC",
      "author": "John Smith",
      "section": "Music",
      "pub_date": "2012-05-01T00:00:00",
      "_hits": 123354
    },
    {
      "url": "http://apikey.com/255401/",
      "title": "Kurt Vonnegut's 8 Tips On How To Write A Great Story",
      "author": "John Smith",
      "section": "Entertainment",
      "pub_date": "2012-04-03T13:11:00",
      "_hits": 27123
    },
    {
      "url": "http://apikey.com/255394/",
      "title": "10 of Film History's Meanest Bullies",
      "author": "John Smith",
      "section": "Entertainment",
      "pub_date": "2012-04-03T14:40:00",
      "_hits": 13255
    },
    // other similar JSON documents...
  ]
}
```

Reference:

- GET /analytics/{type}
- GET /analytics/{meta}/{value}/detail

Shares

Beta. The Shares API provides number of shares across all top social networks for each Post. Shares can also be aggregated by the “author” Meta, similarly to the Analytics API. At this time, “section” and “topic” Meta grouping for Shares is not supported.

To find the “top shared Posts” on your site, hit the GET /shares/{type} endpoint using posts as the type. This will return a response like this:

```
{
  "data": [
    {
      "title": "The 11 Most Important Apple Execs Now That Steve Jobs is Gone"
      "url": "http://apikey.com/244049/",
      "author": "Peter White",
      "section": "Technology",
      "pub_date": "2012-03-30T08:26:00Z",
      "_shares": 505
    },
    // other similar JSON documents...
  ]
}
```

Notice the `_shares` field, which contains total shares aggregated across these social networks:

- Twitter
- Facebook
- LinkedIn
- StumbleUpon
- Pinterest

You can also find the most shared authors using `/shares/authors`:

```
{
  "data": [
    {
      "author": "Peter White",
      "_shares": 823
    },
    // other similar JSON documents...
  ]
}
```

This API is still under heavy development. Coming soon:

- breakdown of shares in each social network; e.g. get most shared stories on Twitter or Facebook
- full support for author, section, and topic metas (like the Analytics API)

Reference:

- GET `/shares/{type}`

Search

Beta. The Search API provides arbitrary keyword searching for Posts indexed on your site.

It only has a single endpoint, GET `/search`, and the `q` parameter controls the search query.

Here is an example finding articles that mention Steve Jobs, `/search?q=steve+jobs`:

```
{
  "data": [
    {
      "title": "Ipods And Crime"
      "url": "http://apikey.com/219045/",
      "author": "Kevin Black",
      "section": "Technology",
      "pub_date": "2012-02-20T19:08:00Z",
    },
    {
      "title": "The 11 Most Important Apple Execs Now That Steve Jobs is Gone"
      "url": "http://apikey.com/244049/",
      "author": "Peter White",
      "section": "Technology",
      "pub_date": "2012-03-30T08:26:00Z",
    },
    // other similar JSON documents...
  ]
}
```

Reference:

- `GET /search`

Recommendation API

Parse.ly's Recommendation API provides a way to show users relevant Posts, either by using information about that user's interests or using attributes of the currently-viewed Post.

Contextual Contextual Recommendations use a URL as input and return a list of relevant Posts as output. The results are not personalized to the individual.

The endpoint for this is `GET /related` and the parameter required is `url`. Let's say the URL passed is about the Apple iPod. The returned "related" Posts might look like this:

```
{
  "data": [
    {
      "title": "Ipods And Crime"
      "url": "http://apikey.com/219045/",
      "author": "Kevin Black",
      "section": "Technology",
      "pub_date": "2012-02-20T19:08:00Z",
    },
    // other similar JSON documents...
  ]
}
```

When using this API, there is no guarantee that the user hasn't already visited the related articles.

Personalized Personalized Recommendations return Posts relevant to the user's interests, and does not include Posts that the user has already read in the past.

Integrating Personalized Recommendations is slightly more complicated than the above APIs because it requires two steps on behalf of the implementer.

1. Training. This logs that an individual with a specific UUID has visited a specific URL on your site.
2. Recommending. Using a saved UUID for the user, return relevant stories from the Parse.ly API.

To train a user profile for your site, use the `GET /profile` endpoint. Pass the `uuid` and `url` parameters to log that an individual with that UUID has visited the specified URL.

Generating UUIDs is sometimes a point of confusion for our customers. Parse.ly already generates a UUID for every user using a cookie-based approach. This is accessible on pages with the Parse.ly tracker installed in the field `PARSELY.uuid`. You are free to re-use this UUID for your application if the cookie-based approach is sufficient for you.

However, some customers may have better UUIDs available. If you have e.g. an internal integer ID which also identifies registered/premium users on your site as well as anonymous visitors, this may be a better choice. Parse.ly does not care what the UUID is – it is simply a string used for identification on further calls to the API.

Once a user is trained, you can view that user's history with `/history` endpoint. Pass the `uuid` and you will see the URLs currently associated with that user. This is mainly useful for debugging purposes.

Finally, call the `GET /related` endpoint with the `uuid` parameter to get personalized recommendations for that user. The default strategy uses a document similarity metric weighted toward most recently seen Posts, and a filter excluding all visited Posts; we plan to introduce additional strategies over time.

Personalized recommendations will appear as multiple posts, similarly to Contextual recommendations. For example, if you train a user on a story about 4G data access plans for mobile phones, you may end up with recommendations like this:

```
{
  "data": [
    {
      "title": "Verizon announces new 4G phone"
      "url": "http://apikey.com/244232/",
      "author": "John Peters",
      "section": "Technology",
      "pub_date": "2012-03-30T08:26:00Z",
    },
    // other similar JSON documents...
  ]
}
```

Reference:

- GET /profile
- GET /related
- GET /history

API Reference

The root URL for all Parse.ly API requests is:

`http://api.parsely.com/v2/`

Analytics

GET `/analytics/{type}`

Returns a list of posts, authors, sections, or topics depending on the specified type. This is typically used to generate front page or article page widgets featuring your “Most Popular” content. Use `GET /analytics/{meta}/{value}/detail` to retrieve drill-downs to posts for “authors”, “sections”, and “topics” listings.

Path arguments type – One of “posts”, “authors”, “sections”, “topics”

Opt. params

- **days** – Number of days since today to consider for `_hits` value; defaults to 14 (2 weeks). Use `days=1` or `days=3` to only consider traffic from last several days.
- **pub_date_start** – Publication date in `YYYY-MM-DD` format, e.g. `2012-01-01` for January 1, 2012. Must be specified with `pub_date_end`.
- **pub_date_end** – Publication end date; must be specified with `pub_date_start`, must be greater than that value, and must not be in the future.
- **sort** – Sort value; default is `_hits` (popularity). *momentum* calculates a change in hits over the time period and sorts by the fastest-growing posts.
- **limit** – Number of records to retrieve; defaults to “10”.
- **page** – Page number to retrieve if multiple pages are available; defaults to `1`. Retrieving a page that is unavailable returns an empty record list.

- **callback** – JSON-P callback, a JavaScript function name that will be used to wrap the JSON response.

GET `/analytics/{meta}/{value}/detail`

Returns a list of posts falling under the specified author, section or topic. This is typically used in one of two scenarios. On its own, it can be used to enrich article pages with “Most Popular Posts by this Author” and section pages with “Most Popular Posts in this Section”. In combination with `GET /analytics/{type}`, it can be used to implement a drill-down – first, you retrieve a list of most popular authors with that endpoint, then you drill into the posts for one specific author with this endpoint.

Path arguments `meta` – one of *author, section, topic*

Opt. params

- **days** – Number of days since today to consider for `_hits` value; defaults to 14 (2 weeks). Use `days=1` or `days=3` to only consider traffic from last several days.
- **pub_date_start** – Publication date in `YYYY-MM-DD` format, e.g. `2012-01-01` for January 1, 2012. Must be specified with `pub_date_end`.
- **pub_date_end** – Publication end date; must be specified with `pub_date_start`, must be greater than that value, and must not be in the future.
- **sort** – Sort value; default is `_hits` (popularity). *momentum* calculates a change in hits over the time period and sorts by the fastest-growing posts.
- **limit** – Number of records to retrieve; defaults to `10`.
- **page** – Page number to retrieve if multiple pages are available; defaults to `1`. Retrieving a page that is unavailable returns an empty record list.
- **callback** – JSON-P callback, a JavaScript function name that will be used to wrap the JSON response.

Shares

GET `/shares/{type}`

Beta. Retrieve a listing of top posts or authors by social shares across top social networks.

Path arguments `type` – one of “posts”, “authors”

Opt. params

- **days** – Number of days since today to consider for `_shares` value; defaults to 14 (2 weeks). Use `days=1` or `days=3` to only consider shares from last several days.
- **limit** – Number of records to retrieve; defaults to “10”.
- **page** – Page number to retrieve if multiple pages are available; defaults to `1`. Retrieving a page that is unavailable returns an empty record list.
- **callback** – JSON-P callback, a JavaScript function name that will be used to wrap the JSON response.

GET `/shares/post/detail`

Beta. For a given canonical URL, return share counts across the top social networks. The following table shows what each integer value returned in the response contains.

Key	Social Network
tw	Twitter
fb	Facebook
li	Linkedin
su	StumbleUpon
pi	Pinterest

Response document will contain a key for each social network. There is also a `total` key summing them for convenience. Note that some keys may not be present for older posts.

Path arguments `url` – Canonical URL for which share counts should be retrieved.

Opt. params `callback` – JSON-P callback, a JavaScript function name that will be used to wrap the JSON response.

Search

GET `/search`

Beta. Search for Posts by keyword. These can match against title or full content.

Query params `q` – Search query keywords; quoting is supported, e.g. `q="steve jobs"` instead of `q=steve+jobs`.

Opt. params

- **limit** – Number of records to retrieve; defaults to “10”.
- **page** – Page number to retrieve if multiple pages are available; defaults to *1*. Retrieving a page that is unavailable returns an empty record list.
- **callback** – JSON-P callback, a JavaScript function name that will be used to wrap the JSON response.

Recommendations

GET `/profile`

Train a user profile for the purpose of personalized recommendations. Use `GET /related` to return recommendations using the same UUID as specified here.

Query params

- **uuid** – UUID for user profile being trained.
- **url** – Canonical URL for page being trained against profile.

GET `/related`

Retrieve a list of Post recommendations for a specified profile UUID (trained in `GET /profile`) or a specified canonical URL.

Query params

- **url** – Canonical URL to use as the basis for contextual recommendations. Cannot be specified with `uuid`.
- **uuid** – User profile ID to use as the basis for personalized recommendations. Cannot be specified with `url`.

Opt. params

- **days** – Number of days since today to consider for `_hits` value; defaults to 14 (2 weeks). Use `days=1` or `days=3` to only consider traffic from last several days.

- **limit** – Number of records to retrieve; defaults to “10”.
- **page** – Page number to retrieve if multiple pages are available; defaults to 1. Retrieving a page that is unavailable returns an empty record list.
- **callback** – JSON-P callback, a JavaScript function name that will be used to wrap the JSON response.

GET /history

Retrieve a list of URLs visited by a user by UUID. This is the training profile being used for personalized recommendations.

Query params **uuid** – User profile ID to look up history on.

1.2.2 Integrations

JavaScript Tracker

What does the Tracker do?

The Parse.ly Tracker (aka Ptrack) is a small piece of JavaScript code that monitors user actions taken on your site to build up rich user profiles for recommendations and expose trending topic information in our analytics product, Parse.ly Dash.

Inserting the tracker on your site

You will want to insert the Parse.ly Tracker in the footer of your website template. Ideal placement is as the last code block before the closing of the *body* tag. If your website uses a templating system, this is usually in the “footer” template.

```
<!-- START Parse.ly Include: Standard -->
<div id="parsely-root" style="display: none">
  <a id="parsely-cfg" data-parsely-site="#####"
    href="http://parsely.com">Powered by the Parse.ly Publisher Platform (P3).</a>
</div>
<script>
(function(s, p, d) {
  var h=d.location.protocol, i=p+"-"+s,
      e=d.getElementById(i), r=d.getElementById(p+"-root"),
      u=h=="https:"?"dlz2jff7j1zjs58.cloudfront.net"
        : "static."+p+".com";
  if (e) return;
  e = d.createElement(s); e.id = i; e.async = true;
  e.src = h+"//"+u+"/p.js"; r.appendChild(e);
})( "script", "parsely", document);
</script>
<!-- END Parse.ly Include: Standard -->
```

Simply copy/paste the code block above. Notice that there is a configuration tag (*parsely-cfg*) which has a data property, *data-parsely-site="#####"*. You must replace the ##### with your own site domain name, e.g. *nytimes.com*. This ensures that no matter what host name your Parse.ly tag appears on, all the traffic data gets aggregated into a single account.

DOM-free version

Some sites are using tag management systems, such as [UberTags](#), to manage the JavaScript snippets loading on their sites. Some older tag management systems do not support JavaScript snippets with DOM dependencies – which is to say, JavaScript snippets that rely upon a certain HTML element being present in the pages.

Since the default Parse.ly tag uses an HTML DIV element for configuration and as a scratch area, these systems often have integration challenges with Parse.ly's JS.

The following code may work in these situations. It creates the DOM/HTML element using JavaScript and then loads the Parse.ly code in the standard way:

```
<!-- START Parse.ly Include: DOM-Free -->
<script>
(function(d) {
  var site = "#####",
      b = d.body,
      e = d.createElement("div");
  e.innerHTML = '<span id="parsely-cfg" data-parsely-site="'+site+'"/></span>';
  e.id = "parsely-root";
  e.style.display = "none";
  b.appendChild(e);
})(document);
(function(s, p, d) {
  var h=d.location.protocol, i=p+"-"+s,
      e=d.getElementById(i), r=d.getElementById(p+"-root"),
      u=h=="https:?"+"dlz2jf7j1zjs58.cloudfront.net"
      : "static."+p+".com";
  if (e) return;
  e = d.createElement(s); e.id = i; e.async = true;
  e.src = h+"//"+u+"/p.js"; r.appendChild(e);
})( "script", "parsely", document);
</script>
<!-- END Parse.ly Include: DOM-Free -->
```

Completely async loading version

If you happen to use [LABjs](#), a third-party library for asynchronous JavaScript loading, you can use this form, which will have better performance than any other option:

```
<!-- START Parse.ly Include: Async -->
<div id="parsely-root" style="display: none">
  <a id="parsely-cfg" data-parsely-site="#####"
    href="http://parsely.com">Powered by the Parse.ly Publisher Platform (P3).</a>
  <script>
    $LAB.script(document.location.protocol=="https:?"
      "https://dlz2jf7j1zjs58.cloudfront.net/p.js":
      "http://static.parsely.com/p.js");
  </script>
</div>
<!-- END Parse.ly Include: Async -->
```

The [LABjs](#) library is only about 3-4kb but can turn many third-party JavaScript integrations into non-blocking, async integrations. We therefore highly recommend it to online sites dealing with third-party JavaScript performance problems. Our scripts at Parse.ly are written to work fine with async loaders like [LABjs](#).

Will this tag break or slow down my site?

No way! Our JavaScript code is written in such a way to be the *last thing that loads* on your page.

We host our DNS servers (parsely.com) with a global, distributed DNS system, which has no downtime and extremely fast resolution times across the globe. Our JavaScript code itself is hosted in a global content delivery network (CDN) with edge locations in every global region.

Our code itself is optimized to not impact user experience in any unintended way. It also automatically leverages asynchronous JavaScript loading technologies in newer browsers. For older browsers, we use a sophisticated loading process that loads a few bytes of data from a high-speed CDN (a “bootstrap” file) and then uses an asynchronous JavaScript loading library (the LABjs library mentioned above) to ensure that none of our other assets block your page load.

Once our tracking code is loaded, we drop an extremely small cookie (just a user ID) and asynchronously beacon back information to our analytics server. If our analytics server is down, all that happens is that the actions are no longer tracked – there is no impact on the user otherwise.

Our team is staffed by JavaScript experts who know the pain and frustration with crappy third-party JavaScript integrations. We therefore have taken a lot of care to make our tracking code integration a no-brainer, safe decision.

Don't believe us? Check out our [public Pingdom report](#) showing our uptime and global response times.

Optimizing tracker placement

If you are curious about the optimal placement of tracker code on your website, you can take a look at this article on [how tracking scripts affect page loads](#).

Parse.ly Crawler

What does the Crawler do?

The Parse.ly Crawler is a backend system that makes HTTP requests to your web server in order to download content and metadata from your site pages. In its most typical use, news sites will have their article pages crawled by the crawler so that they may appear as “Posts” inside Parse.ly Dash, and so that other pieces of metadata such as author, section, publication date, image, and topics are automatically extracted.

Technical details about the Crawler

Parse.ly's Crawler will send out crawl requests to your pages as the pageviews stream in from those URLs. It will crawl your content using the user-agent: `Mozilla/5.0 (compatible; parse.ly scraper/0.14; +http://parsely.com)`. Note that the version field can change over time.

Here are the current set of IP addresses for our Crawler worker machines: 174.143.144.39, 174.143.252.206, 204.232.209.126.

You may be worried about additional load that this crawler might put on your server. This is generally not a concern. The Crawler takes the following steps to be polite to the servers it crawls:

- It limits the number of concurrent requests it opens to your server to ensure it doesn't affect your concurrency throughput.
- It caches articles it has already seen.
- It introduces a small delay between HTTP requests to ensure the load is spread out.

- It does not pro-actively spider your site; instead, pages are crawled only as they are visited by users. This way, archived articles that are not visited are not needlessly crawled.

In the first month of integration or so you will see more crawling activity than in future months, as Parse.ly will be loading in older articles that receive occasional visits. This will wane over time.

Finally, we must emphasize that crawling is an **entirely back-end operation**. That is, crawling in no way affects the pageload performance of your visitors coming to your site. It is done entirely asynchronously by Parse.ly's servers "after the fact".

Crawling through pay walls

Some Parse.ly customers do not have all of their content accessible to the public due to a "pay wall". In these cases, you must coordinate with our support team to arrange for a special login account, only accessible to our Crawler, which can gain access to your pay wall content. The credentials for this account will only be used by the Crawler and will not be shared with anyone else.

Parse.ly Dash

A Web Interface for Parse.ly Services

To manage Parse.ly services on your site, you use Parse.ly Dash. The only prerequisite for using Dash is to have *the tracker* enabled.

Once enabled, you can use Dash to get a picture of the data Parse.ly is collecting, and also add site widgets which drive additional pageviews on your site.

[Sign up for Dash](#) to gain access.

JavaScript Widget

Turning on the JavaScript Widget is an easy way to increase your pageviews by between 5-10% by utilizing Parse.ly's personalized recommendation technology.

Installation

If you already have the *JavaScript Tracker* enabled on your site, then you don't need to make any additional changes to your article templates. You may choose to customize the look-and-feel of the widget using standard CSS or by [contacting the Parse.ly team](#), as we regularly do custom theming work for our clients to reduce integration cost.

Popular Widget Styles

We have three default widget styles available that can be used by our customers. One is a simple link display, e.g.

- [As 'Obama Surge' stalls, are Dems filling 'enthusiasm gap' with hired help?](#)
- [Obama slinks away from Wisconsin union fight](#)
- [Mitt Romney Edges Past Obama In New Poll](#)
- [Obama rallies voters on DC radio show](#)
- [Poll: Obama fares well against most top Republican contenders for 2012 election](#)

A second is a subtle slide-out widget with one top article, and a way to cycle through recommendations.



A third is a more aggressive slide-out widget with image thumbnails and relevance scores. This design has the highest clickthrough rate and is especially recommended for users who have expressed interest in related content.



We are also considering other widget designs and are regularly testing new design ideas to get the highest possible CTRs for our customers. We also have an easy-to-customize widget framework that allows you to build your own styles and designs by using our `parselyjs`.

Enabling Widget and Switching Styles

To switch between the widget styles, you need to use the Parse.ly Administration interface, which is called *Parse.ly Dash*. Go to the “Site Plugins” area and choose “Optimization Widget” to enable the widget and configure style settings.

1.2.3 Licensing Information

Premium Licensing

Our API is only available to paying customers of Parse.ly Dash.

Our API access is segmented into three tiers, tied to our three tiers of Parse.ly Dash access. These tiers are:

- Preview
- Basic
- Advanced

Preview is available to customers in the Dash “Track” tier. You have access to our APIs, but no access to the OAuth authentication scheme. Further, you are limited in the number of API calls you can make per month and are not allowed to depend on the API in production environment. The API is available only for development and testing services, and to run one-off reports.

Basic is available to customers in the Dash “Plan” tier. This provides access to our Analytics API, but not our Shares, Search, or Recommendation APIs.

Advanced is available to customers in the Dash “Promote” tier and for customers using custom pricing. This provides full access to all Parse.ly APIs and top-tier support from our engineering staff.

Legal Summary

The two documents you must be aware of from a legal standpoint before you start using Parse.ly are:

- *Parse.ly API Terms of Use*
- *Parse.ly Privacy Policy*

Parse.ly API Terms of Use

Thank you for using the Parsely, Inc. (“Parse.ly”) application programming interfaces (“Parse.ly APIs”). By using Parse.ly APIs, you agree to these terms of use (the “Terms of Use”) and the Parse.ly Terms of Service (“TOS”). The TOS can be found at: <http://blog.cogtree.com/terms-of-service/>

BY ACCEPTING THESE TERMS OF USE, OR BY USING OR ACCESSING ANY PORTION OF THE PARSE.LY APIS, YOU IRREVOCABLY AGREE TO THESE TERMS OF USE, AND YOU REPRESENT AND WARRANT THAT YOU HAVE ALL AUTHORITY NECESSARY TO BIND YOURSELF (AND, IF YOU ARE EMPLOYED BY OR OTHERWISE REPRESENT ANY CORPORATION OR OTHER LEGAL ENTITY THAT WISHES TO USE THE PARSE.LY APIS, THAT ENTITY) TO THESE TERMS OF USE. IF YOU DO NOT AGREE TO THESE TERMS OF USE, YOU MAY NOT USE THE PARSE.LY APIS.

“You” means the individual person accessing or using the Parse.ly APIs on his or her own behalf; or, if the APIs or Documentation are accessed or used on behalf of an organization, corporation or other legal entity, “you” means such legal entity.

Parse.ly reserves the right to update and change, from time to time, these Terms of Use and all documents incorporated by reference. You can always find the most recent version of these Terms of Use at <http://parse.ly/api/termsfuse.html>. Parse.ly may change these Terms of Use by posting a new version without notice to you. Use of the Parse.ly APIs after such change constitutes acceptance of such changes.

In the event of any inconsistency between these Terms of Use and the TOS, these Terms of Use control.

1. Licensed Uses and Restrictions

Parse.ly APIs are owned by Parse.ly and are licensed to you on a worldwide (except as limited below), non-exclusive, non-sublicenseable basis on the terms and conditions set forth herein. These Terms of Use define legal use of the Parse.ly APIs, all updates, revisions, substitutions, and any copies of the Parse.ly APIs made by or for you. All rights not expressly granted to you are reserved by Parse.ly.

1. Subject to the restrictions set forth in these Terms of Use, you may use the Parse.ly APIs and any updates provided by Parse.ly (in its sole discretion) solely to interface with Parse.ly’s proprietary content recommendation service (the “Parse.ly Service”). Your license to the Parse.ly APIs under these Terms of Use continues until it is terminated by either party. You may terminate the license by discontinuing use of all or any of the Parse.ly APIs. Parse.ly may terminate the license at any time for any reason. Parse.ly may make changes, upgrades, or discontinue all or any portion of any Parse.ly API at any time for any reason. These Terms of Use terminate automatically if (i) you violate any term of these Terms of Use, (ii) Parse.ly publicly posts a written notice of termination on Parse.ly’s Web site, (iii) Parse.ly sends a written notice of termination to you, or (iv) Parse.ly ceases providing access to the Parse.ly APIs to you. Upon the expiration or any termination of these Terms of Service, the license granted to you will terminate and you, at your expense, will promptly return all copies of the API and all Confidential Information in your possession to Parse.ly. The provisions

of Sections 2, 3, 7, 8, 10, and 12 shall survive termination or expiration of this Agreement for any reason.

2. Your application may make automated calls or other data requests to or through the Parse.ly network (“Calls”). Parse.ly may at any time, and over any given period of time, limit the number of Calls you may send to the Parse.ly network, or prohibit any application created by you from sending Calls to the Parse.ly network, as Parse.ly deems appropriate in its sole discretion.
3. If your product or service uses or is based upon the Parse.ly APIs, then you will comply with any Parse.ly attribution policy as in effect from time to time.
4. You will use the APIs in compliance with all applicable laws, statutes, regulations, ordinances or other rules promulgated by governing authorities having jurisdiction over the parties,
5. You shall use instructions provided in the Parse.ly APIs to place application identification information (“API Key”) into any application or service you develop that incorporates or makes any use of the Parse.ly APIs. You can sign up for an API Key at <http://parse.ly/p3>. You must provide accurate identification, contact, and other information required as part of the registration process. You will NOT create any script or other automated tool that attempts to create multiple API Keys. You may not allow any third party to use your API Key for their own benefit.
6. You shall NOT:
 1. use the Parse.ly APIs in any manner or for any purpose that violates any law or regulation, promotes illegal activities, violates third party rights or terms of service, violates any right of any person, including but not limited to intellectual property rights, rights of privacy, or rights of personality, or in any manner inconsistent with the Parse.ly TOS or these Terms of Use;
 2. modify, adapt, alter, translate the API;
 3. use the Parse.ly APIs to engage in spamming or other advertising or marketing activities that violate any applicable laws, regulations or generally-accepted advertising industry guidelines;
 4. sell, lease, share, transfer, or sublicense the Parse.ly APIs, or access or access codes thereto, or derive income from the use or provision of the Parse.ly APIs, whether for direct commercial or monetary gain or otherwise, without Parse.ly’s prior, express, written permission;
 5. use the Parse.ly APIs in a manner that adversely effects Parse.ly and/or the Parse.ly Service or exceeds: (a) reasonable request volume, as set by Parse.ly from time to time, (b) constitutes excessive or abusive usage, or (c) otherwise fails to comply or is inconsistent with any part of the Parse.ly API documentation, as determined by Parse.ly in its sole discretion;
 6. otherwise exercise rights to the API except as expressly allowed by these Terms of Service;
 7. reverse engineer or attempt to reconstruct, identify or discover any underlying ideas, underlying user interface techniques or algorithms related to the Parse.ly Service;
 8. remove, obscure or alter any Parse.ly’s copyright notices, trademarks or other proprietary rights notices affixed to or contained within the API; or
 9. use the Parse.ly APIs in a product or service that competes with products or services offered by Parse.ly, without Parse.ly’s prior, express, written permission.

Parse.ly reserves the right to immediately suspend access to the API if you breach any terms of these Terms.

2. Ownership and Relationship of Parties

The Parse.ly APIs may be protected by copyrights, trademarks, service marks, international treaties, and/or other proprietary rights and laws of the U.S. and other countries. Parse.ly’s rights apply to the Parse.ly APIs and all output and executables of the Parse.ly APIs, excluding any software components developed by you which do not themselves

incorporate the Parse.ly APIs or any output or executables of the Parse.ly APIs. You agree to abide by all applicable proprietary rights laws and other laws, as well as any additional copyright notices or restrictions contained in these Terms of Use and in the TOS. Parse.ly owns all rights, title, and interest in and to the Parse.ly APIs. These Terms of Use grant you no right, title, or interest in any intellectual property owned or licensed by Parse.ly, including (but not limited to) the Parse.ly APIs and Parse.ly trademarks.

3. Confidentiality.

You acknowledge that the API is a commercially valuable asset which constitutes a trade secret of Parse.ly and that it contains Confidential Information proprietary to Parse.ly. For purposes of these Terms of Service, “Confidential Information” means the API, all information provided by Parse.ly about the API, and all information provided by Parse.ly that is clearly marked or identified as confidential or that a reasonable person would understand that such information is confidential. You shall not disclose Confidential Information to any third party or use Confidential Information for any purpose other than as expressly permitted by these terms of Service. You agree that you, including your employees, officers, and agents, if any, shall treat all Confidential Information with the same degree of care as it accords to your own confidential information but which, in no event, shall be less than reasonable care and shall not disclose, sell, transfer, publish, copy, display or otherwise make the API available, or any part thereof in any form, to any person or entity not a party to these Terms of Service. You shall take all reasonable precautions to ensure fulfillment of this confidentiality and nondisclosure obligations and agrees to immediately notify Parse.ly of any unauthorized use, copying and/or disclosure of the API, as well as, take such actions as are necessary to cease and prevent any further unauthorized use, copying and/or disclosure.

You acknowledge that any use or disclosure of the API by you in violation of or in any manner inconsistent with these Terms of Service would cause substantial and irreparable injury to Parse.ly, and that Parse.ly’s remedies at law will not be adequate. Accordingly, you agree that Parse.ly shall be entitled to injunctive relief with respect to any breach, or threatened breach of this Section, and that such right shall be in addition to, and not in limitation of, any other rights or remedies to which Parse.ly is or may be entitled at law or equity.

4. Support.

Parse.ly may elect to provide you with support or modifications for the Parse.ly APIs (collectively, “Support”), in its sole discretion, and may terminate such Support at any time without notice to you. Parse.ly may change, suspend, or discontinue any aspect of the Parse.ly APIs at any time, including the availability of any Parse.ly APIs. Parse.ly may also impose limits on certain features and services or restrict your access to parts or all of the Parse.ly APIs or the Parse.ly website without notice or liability.

5. Fees and Payments.

If more than a single individual user will be using or accessing the interface to the Parse.ly Services, you will pay Parse.ly a monthly license fee per user, as set forth on the order form accepted by Parse.ly. Parse.ly reserves the right to charge additional or different fees for future use of, support for, or access to the Parse.ly APIs or the Parse.ly Services in Parse.ly’s sole discretion provided that such charges will be disclosed in advance.

6. Copyright Complaints; Repeat Infringer Policy.

You agree to take whatever actions are necessary or are requested by Parse.ly to enable Parse.ly to comply with Parse.ly’s copyright policy and the take-down and other provisions of the Digital Millennium Copyright Act (“DMCA”) or other applicable laws and regulations with respect to your Parse.ly API. In addition, you acknowledge that in accordance with the DMCA and other applicable law, Parse.ly has adopted a policy of terminating, in appropriate circumstances and at its sole discretion, users and developers who are deemed to be repeat infringers, and that

you agree that you will, if requested by Parse.ly, take reasonable steps to terminate access to your Parse.ly API for any user who Parse.ly identifies to you as a repeat infringer.

7. Disclaimer of Any Warranty.

SOME OF THE PARSE.LY APIS ARE EXPERIMENTAL AND HAVE NOT BEEN TESTED IN ANY MANNER. PARSE.LY DOES NOT REPRESENT OR WARRANT THAT ANY PARSE.LY APIS ARE FREE OF INACCURACIES, ERRORS, BUGS, OR INTERRUPTIONS, OR ARE RELIABLE, ACCURATE, COMPLETE, OR OTHERWISE VALID.

THE PARSE.LY APIS ARE PROVIDED “AS IS” WITH NO WARRANTY, EXPRESS OR IMPLIED, OF ANY KIND AND PARSE.LY EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES AND CONDITIONS, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AVAILABILITY, SECURITY, TITLE AND/OR NON-INFRINGEMENT. PARSE.LY DOES NOT WARRANT THAT THE API WILL OPERATE WITHOUT INTERRUPTION OR ERROR. NO WARRANTY IS MADE BY PARSE.LY ON THE BASIS OF TRADE USAGE, COURSE OF DEALING OR COURSE OF TRADE. PARSE.LY DOES NOT WARRANT THAT THE PARSE.LY PLATFORM OR PARSE.LY SERVICES WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE PARSE.LY PLATFORM WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT ALL ERRORS WILL BE CORRECTED.

YOUR USE OF PARSE.LY APIS IS AT YOUR OWN DISCRETION AND RISK, AND YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGE THAT RESULTS FROM THE USE OF ANY PARSE.LY APIS INCLUDING, BUT NOT LIMITED TO, ANY DAMAGE TO YOUR COMPUTER SYSTEM OR LOSS OF DATA.

8. Limitation of Liability.

PARSE.LY SHALL NOT, UNDER ANY CIRCUMSTANCES, BE LIABLE TO YOU FOR ANY INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL OR EXEMPLARY DAMAGES ARISING OUT OF OR IN CONNECTION WITH USE OF THE PARSE.LY APIS, WHETHER BASED ON BREACH OF CONTRACT, BREACH OF WARRANTY, TORT (INCLUDING NEGLIGENCE, PRODUCT LIABILITY OR OTHERWISE), OR ANY OTHER PECUNIARY LOSS, WHETHER OR NOT PARSE.LY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL PARSE.LY’S TOTAL AGGREGATE LIABILITY ARISING FROM OR RELATED TO THIS AGREEMENT, WHETHER IN CONTRACT OR IN TORT OR UNDER ANY OTHER LEGAL THEORY (INCLUDING STRICT LIABILITY AND NEGLIGENCE) EXCEED THE LESSER OF (A) THE TOTAL AMOUNT PAID TO PARSE.LY BY YOU DURING THE SIX (6) MONTH PERIOD IMMEDIATELY PRIOR TO THE EVENT GIVING RISE TO SUCH LIABILITY OR (B) ONE THOUSAND DOLLARS (\$1,000).

9. Essential Basis; Exclusions and Limitations.

The parties acknowledge and agree that the disclaimers, exclusions and limitations of liability set forth in these Terms of Service form an essential basis of these terms of Service, and that, absent any of such disclaimers, exclusions or limitations of liability, the terms of these Terms of Service, including, without limitation, the economic terms, would be substantially different.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF CERTAIN WARRANTIES OR THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES. ACCORDINGLY, SOME OF THE ABOVE LIMITATIONS OF SECTIONS 7 AND 8 MAY NOT APPLY TO YOU.

10. Release and Waiver.

To the maximum extent permitted by applicable law, you hereby release and waive all claims against Parse.ly, and its subsidiaries, affiliates, officers, agents, licensors, co-branders or other partners, and employees from any and all

liability for claims, damages (actual and/or consequential), costs and expenses (including litigation costs and attorneys' fees) of every kind and nature, arising from or in any way related to your use of Parse.ly APIs. If you are a California resident, you waive your rights under California Civil Code § 1542, which states, "A general release does not extend to claims which the creditor does not know or suspect to exist in his favor at the time of executing the release, which if known by him must have materially affected his settlement with the debtor." You understand that any fact relating to any matter covered by this release may be found to be other than now believed to be true and you accept and assume the risk of such possible differences in fact. In addition, you expressly waive and relinquish any and all rights and benefits which you may have under any other state or federal statute or common law principle of similar effect, to the fullest extent permitted by law.

11. Release, Hold Harmless and Indemnity.

You irrevocably and unconditionally release and covenant not to use or pursue any claim against Parse.ly and its subsidiaries, affiliates, officers, agents, licensors, co-branders or other partners, and employees, for any and all damages, liabilities, causes of action, judgments, and claims: (i) pertaining to your Parse.ly API, or (ii) which otherwise may arise in connection with your use of, reliance on, or reference to the Parse.ly platform, documentation, or Parse.ly API.

To the maximum extent permitted by applicable law, you agree to hold harmless and indemnify Parse.ly and its subsidiaries, affiliates, officers, agents, licensors, co-branders or other partners, and employees from and against any third party claim arising from or in any way related to your use of Parse.ly APIs or any violation of these Terms of Use or TOS, including any liability or expense arising from all claims, losses, damages (actual and/or consequential), suits, judgments, litigation costs and attorneys' fees, of every kind and nature. Parse.ly shall use good faith efforts to provide you with written notice of such claim, suit or action.

12. General Terms.

Notwithstanding any provision hereof, for all purposes of the Terms of Use, you and Parse.ly shall be and act independently and not as partner, joint venturer, agent, employee or employer of the other. You shall not have any authority to assume or create any obligation for or on behalf of Parse.ly, express or implied, and you shall not attempt to bind Parse.ly to any contract. If any provision of the Terms of Use is found by a court of competent jurisdiction to be invalid, the parties nevertheless agree that the court should endeavor to give effect to the parties' intentions as reflected in the provision and the other provisions of such documents remain in full force and effect. The Terms of Use and the relationship between you and Parse.ly shall be governed by the laws of the State of Delaware without regard to its conflict of law provisions. You and Parse.ly agree to submit to the personal jurisdiction of the courts located within the State of Delaware. Parse.ly's failure to exercise or enforce any right or provision of the Terms of Use shall not constitute a waiver of such right or provision. The section headings and subheadings contained in this agreement are included for convenience only, and shall not limit or otherwise affect the terms of the Terms of Use. Any construction or interpretation to be made of the Terms of Use shall not be construed against the drafter. The Terms of Use constitute the entire agreement between Parse.ly and you with respect to the subject matter hereof. You may not assign any of your rights or obligations under these Terms of Service without the prior written consent of Parse.ly. Any attempted assignment or transfer in violation of the foregoing will be void. No failure of either party to enforce any of its rights under these Terms of Service will act as a waiver of those rights.

Last updated: [January] 2010 / 428478 v2/RE

1.2.4 Appendix

OAuth

For production usage, Parse.ly's API uses 2-legged OAuth to authenticate API requests, so you will need a consumer key and secret to make these requests, as well as an OAuth client library.

We assume client and server both know a secret key that is only known to them.

Client side

Before making a request, the client must:

- extract the query arguments from this request and their values
- sort the query arguments alphabetically
- append the keys and values together to build a big string
- sign the string using HMAC-SHA1

To make a request, the client sends the request as-is, along with a special header, 'X-Parsely-Auth', that is to be populated with the signed string.

Here's an example:

- the client wants to make a request to is <http://api.parsey.com/v2/analytics/posts?apikey=key.com&days=1>
- the sorted query arguments are: days, apikey
- the string to sign is 'apikey=key.com&days=1'
- the client makes a request to <http://api.parsey.com/v2/analytics/posts?apikey=key.com&&days=1>, and adds to the header value 'X-Parsely-Auth' the signed string.

Server side

The server goes through the same process the client went through to calculate the signed string, using the secret key it has for this client. It then compares the value it found from its calculation to the value provided in the 'X-Parsely-Auth' header value. If they match, the client is authenticated.

References and notes:

- <http://www.thebuzzmedia.com/designing-a-secure-rest-api-without-oauth-authentication/>
- <http://philipsoutham.com/post/2172924723/two-legged-oauth-in-python>

Setting up oauth_proxy to test the secure API

In order to ergonomically test our API (e.g. using your browser and using command-line tools like *curl*), you can set up an OAuth Proxy. An OAuth proxy simply receives HTTP requests and forwards them to another HTTP service, while signing every request with the proper OAuth header entries.

Luckily, Seth Fitzsimmons has written a handy little tool that does just that. You can find the tool at the [oauth_proxy github repo](#) or read his [article about the tool](#).

Implementation

The *oauth_proxy* tool is implemented in Python as a plugin for the Twisted framework. Therefore, it requires that you have both a Python interpreter installed and a recent version of the Twisted framework, installable on most systems using *easy_install twisted*.

Once you have this tool, simply set it up with the consumer key and secret provided to you via Parse.ly Dash's control panel.

Since we use two-legged OAuth, you only need to run the following:

```
$ cd oauth-proxy
$ twistd -n oauth_proxy
  --consumer-key <consumer key>
  --consumer-secret <consumer secret>
```

Then, you can make requests to the Parse.ly API using *curl* as follows:

```
$ curl -x localhost:8001 http://api.parse.ly/p/user/items/current
```

which should return data in JSON format.

If you run into issues getting a proper response from our server, feel free to [contact us](#) for help.

Using curlish to make command-line requests

...

RESTful APIs

Principles

These are the principles that guided our API design:

- **Aim to be RESTful:** This has a controversial definition within the API community, but we followed some of the guidelines set forth in some of the [REST resources](#) listed here.
- **Documented:** We should document our API and interchange formats in a way that makes it easy for our users to find what they're looking for and make the requests and interactions that they need to get their work done.
- **Meaningful Error States:** When there are problems finding requested data or if a service is down, we should have some well-established error states that can be handled gracefully by our clients.
- **JSON as an interchange format:** JSON is a lightweight and human-readable format. JSON's only real drawback vs. something like XML is that there is no good way to define a JSON "schema" to document the formats formally. There is a [jsonschema project](#) to do this, but it's still early days for that. So, we will just substitute a good schema for good documentation.
- **Tested:** The API should be tested, and, once it leaves "beta" state, it should be versioned and have a high degree of reliability and consistency.
- **Language Bindings Included:** RESTful API design really defines our wire format and mechanism for interacting with our server. But we should provide language bindings for various platforms to ease the ability to prototype applications atop the Parse.ly REST API.

Design Guidelines

Many of our design guidelines came from the excellent book by Leonard Richardson and Sam Ruby, *RESTful Web Services*, published by O'Reilly and the article [A Brief Intro to REST](#).

We highly recommend you read that article and [pick up a copy of the book](#) if you ever need to design a service of your own!

Give every “thing” an ID From “A Brief Intro to REST”:

Use URIs to identify everything that merits being identifiable, specifically, all of the “high-level” resources that your application provides, whether they represent individual items, collections of items, virtual and physical objects, or computation results.

Link things together This doesn’t just mean that you actually have links (which is a corollary of “everything has an ID”), but that when you return an object that references another object, you use the URL (link) to connect the two. “Use links to refer to identifiable things (resources) **wherever** possible.”

Use standard methods What REST does is provide a “metamodel” for all the “things” in your web application. You could think of this as a widely-used base interface called “Resource”. If you were to model this in Java. you’d have e.g.

```
interface Resource {
    Resource (URI u);
    Response get ();
    Response post (Request r);
    Response put (Request r);
    Response delete ();
}
```

Though things which implement the *Resource* interface can have a richer set of operations and behaviors, we should have reasonable actions that correspond to the above 4 operations. If we can map everything into those 4 (which sometimes correspond to the four actions in the CRUD model), all the better.

Communicate statelessly This quote summarizes this best:

“First of all, it’s important to stress that although REST includes the idea of statelessness, this does not mean that an application that exposes its functionality cannot have state – in fact, this would render the whole approach pretty useless in most scenarios. REST mandates that state be either turned into resource state, or kept on the client.”

The way the Parse.ly team interprets this guideline is to “prefer stateless communication, otherwise, idempotence, otherwise, *documentation!*”. Obviously, you can’t make things stateless, but making them idempotent is probably a good idea, and if you can’t at least make it idempotent, then we better have documented it!

Guarantee Idempotence Idempotence should be expected by our clients for many of our HTTP methods:

“GET is idempotent – if you issue a GET request and don’t get a result, you might not know whether your request never reached its destination or the response got lost on its way back to you. The idempotence guarantee means you can simply issue the request again. Idempotence is also guaranteed for PUT (which basically means “update this resource with this data, or create it at this URI if it’s not there already”) and for DELETE (which you can simply try again and again until you get a result – deleting something that’s not there is not a problem). POST, which usually means “create a new resource”, can also be used to invoke arbitrary processing and thus is neither safe nor idempotent.”

Multiple “Endpoints” We assign unique URLs to user profiles, articles, sources, and everything else in our system. This follows the practice described here:

“In a RESTful approach, an application might add a few million customer URIs to the Web; if it’s designed the same way applications have been designed in CORBA times, its contribution usually is a single “endpoint” – comparable to a very small door that provides entry to a universe of resource only for those who have the key.”

Bibliography

This API was designed taking the following documents into consideration:

- [A Brief Intro to REST](#)
- [Atlassian REST API Design Guidelines](#)
- [REST Anti-Patterns](#)
- [REST for the rest of us](#)
- [How I Explained REST to my Wife](#)

Parse.ly Technology

At its core, Parse.ly provides insights to the web’s best publishers. Our system marries real-time analytics with proprietary metadata/topic/concept extraction. Together, this allows seamless integration of timely insights on any content-rich website. Our technology not only solves a significant technical challenge (representing many man-years of research and development) but also solves user engagement and traffic retention problems for publishers. Publishers have told us these problems are at the core of the monetization of their businesses.

Data flow and architecture: To support the billions of page views and millions of pieces of content that are generated daily by our large publisher clients, Parse.ly is built atop a large-scale, distributed architecture. Two kinds of data flow among our servers and software. Content Analysis Data is necessary to understand a publisher’s content, and happens after the first bit of information about a page is beacons from our JavaScript tracker. Our servers first download the article content (HTML) and apply our custom crawler to analyze the page’s important attributes – author, section, image, publication date, title, and full text. We then apply proprietary topic/metadata extraction to the title and full text content, and persist the extracted metadata. Analytics Data allows us to measure content performance (such as pageviews/uniques), and this data is aggregated in a real-time data store for rapid queries.

To date, we have processed hundreds of gigabytes of content with our content engine extracting hundreds of thousands of unique topics. We have also processed terabytes of pageview data in our analytics data store.

Parse.ly’s content analysis system analyzes online content using two techniques: Natural Language Processing and Statistical Computational Linguistics. The former is a suite of custom algorithms for detecting things like topics (e.g. “technology startups”) and entities (e.g. “Steve Jobs”). The latter is used to generate statistical text profiles from each piece of content that can be used to discover latent relationships to other content on the customer’s site.

After being enriched by this analysis process, all the individual content pieces (e.g. news articles) are stored in a high-speed search index so they can be retrieved in milliseconds. Once this index is built, Parse.ly has enough information to begin aggregating analytics data into useful drilldown interfaces.

1.3 Contact the Parse.ly Team

We’re happy to help you use the Parse.ly API and provide you with an API key so you can get started.

1.3.1 Get an API Key

API keys are available in the “Settings” area of your Parse.ly Dash account. If you do not have a Dash account yet, you can [Sign up for Dash](#).

1.3.2 Contact Us Directly

To get this, you can contact us directly, either via e-mail (support@parse.ly) or [@parse.ly](https://twitter.com/parse.ly) on Twitter.

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

INDEX

G

GET (HTTP method)

- `/analytics/{meta}/{value}/detail`, **11**
- `/analytics/{type}`, **10**
- `/history`, **13**
- `/profile`, **12**
- `/related`, **12**
- `/search`, **12**
- `/shares/{type}`, **11**
- `/shares/post/detail`, **11**