arXiv API User's Manual

Please review the Terms of Use for arXiv APIs before using the arXiv API.

Table of Contents

1. Preface

The arXiv API allows programmatic access to the hundreds of thousands of e-prints hosted on arXiv.org.

This manual is meant to provide an introduction to using the API, as well as documentation describing its details, and as such is meant to be read by both beginning and works, see the API Quickstart. For more detailed information, see Structure of the API.

For examples of using the API from several popular programming languages including perl, python and ruby, see the Examples section.

Finally, the Appendices contain an explanation of all input parameters to the API, as well as the output format.

2. API QuickStart

The easiest place to start with the API is by accessing it through a web browser. For examples of accessing the API through common programming languages, see the Ex

Most everyone that has read or submitted e-prints on the arXiv is familiar with the arXiv human web interface. These HTML pages can be accessed by opening up your w web browser

http://arxiv.org

From there, the article listings can be browsed by clicking on one of the many links, or you can search for articles using the search box in the upper right hand side of the that contain the word `electron` in the title or abstract, I would type `electron` in the search box, and click `Go`. If you follow my example, you will see something like tl result, with links to the abstract page, pdf, etc.

In its simplest form, the API can be used in exactly the same way. However, it uses a few shortcuts so there is less clicking involved. For example, you can see the same s

http://export.arxiv.org/api/query?search_query=all:electron.

Alternatively, you can search for articles that contain `electron` *AND* `proton` with the API by entering

http://export.arxiv.org/api/query?search_query=all:electron+AND+all:proton

What you see will look different from the HTML interface, but it contains the same information as the search done with the human interface. The reason why the results Atom 1.0 format, and not HTML. Since Atom is defined as an XML grammar, it is much easier to digest for programs than HTML. The API is not intended to be used inside simple way to debug a program that does use the API.

You might notice that your web browser has asked you if you want to "subscribe to this feed" after you enter the API url. This is because Atom is one of the formats used are usually read with feed reader software, and are what is generated by the existing arXiv rss feeds. The current arXiv feeds only give you updates on new papers withir thing to do with the API then is to generate your own feed, based on a custom query!

To learn more about how to construct custom search queries with the API, see the appendix on the details of query construction. To learn about what information is ret To learn more about writing programs to call the API, and digest the responses, we suggest starting with the section on Structure of the API.

3. Structure of the API

In this section, we'll go over some of the details of interacting with the API. A diagram of a typical API call is shown below:

Example: A typical API call

```
Request from url: http://export.arxiv.org/api/query  (1)
 with parameters: search_query=all:electron
                .
                .
                .
API server processes the request and sends the response
                .
                .
                .
Response received by client.  (2)
```

1. The request can be made via HTTP GET, in which the parameters are encoded in the url, or via an HTTP POST in which the parameters are encoded in the HTTP requ methods.
2. If all goes well, the HTTP header will show a 200 OK status, and the response body will contain the Atom response content as shown in the example response.

3.1. Calling the API

As mentioned above, the API can be called with an HTTP request of type GET or POST. For our purposes, the main difference is that the parameters are included in the u Thus if the parameters list is unusually long, a POST request might be preferred.

The parameters for each of the API methods are explained below. For each method, the base url is

```
http://export.arxiv.org/api/{method_name}?{parameters}
```

3.1.1. Query Interface

The API query interface has `method_name=query`. The table below outlines the parameters that can be passed to the query interface. Parameters are separated with the

query				
	parameters	**type**	**defaults**	**required**
	`search_query`	string	None	No
	`id_list`	comma-delimited string	None	No
	`start`	int	0	No
	`max_results`	int	10	No

3.1.1.1. search_query and id_list logic

We have already seen the use of `search_query` in the quickstart section. The `search_query` takes a string that represents a search query used to find articles. The cor query construction appendix. The `id_list` contains a comma-delimited list of arXiv id's.

The logic of these two parameters is as follows:

- If only `search_query` is given (`id_list` is blank or not given), then the API will return results for each article that matches the search query.
- If only `id_list` is given (`search_query` is blank or not given), then the API will return results for each article in `id_list` .
- If *BOTH* `search_query` and `id_list` are given, then the API will return each article in `id_list` that matches `search_query` . This allows the API to act as a results

This is summarized in the following table:

`search_query` present	`id_list` present	API returns
yes	no	articles that match `search_query`
no	yes	articles that are in `id_list`
yes	yes	articles in `id_list` that also match `search_query`

3.1.1.2. start and max_results paging

Many times there are hundreds of results for an API query. Rather than download information about all the results at once, the API offers a paging mechanism through download chucks of the result set at a time. Within the total results set, `start` defines the index of the first returned result, *using 0-based indexing*. `max_results` is the example, if wanted to step through the results of a `search_query` of `all:electron` , we would construct the urls:

```
http://export.arxiv.org/api/query?search_query=all:electron&start=0&max_results=10 (1)
http://export.arxiv.org/api/query?search_query=all:electron&start=10&max_results=10 (2)
http://export.arxiv.org/api/query?search_query=all:electron&start=20&max_results=10 (3)
```

1. Get results 0-9
2. Get results 10-19
3. Get results 20-29

Detailed examples of how to perform paging in a variety of programming languages can be found in the examples section.

In cases where the API needs to be called multiple times in a row, we encourage you to play nice and incorporate a 3 second delay in your code. The detailed examples below illustrate how to do this in a

Because of speed limitations in our implementation of the API, the maximum number of results returned from a single call (`max_results`) is limited to 30000 in slices of at most 2000 at a time, using the retrieve matches 6001-8000: http://export.arxiv.org/api/query?search_query=all:electron&start=6000&max_results=8000

Large result sets put considerable load on the server and also take a long time to render. We recommend to refine queries which return more than 1,000 results, or at le harvesting or set information, etc., the OAI-PMH interface is more suitable. A request with `max_results` >30,000 will result in an HTTP 400 error code with appropriate take a little over 2 minutes to return a response of over 15MB. Requests for fewer results are much faster and correspondingly smaller.

3.1.1.3. sort order for return results

There are two options for for the result set to the API search, `sortBy` and `sortOrder` .

`sortBy` can be "relevance", "lastUpdatedDate", "submittedDate"

`sortOrder` can be either "ascending" or "descending"

A sample query using these new parameters looks like:

```
http://export.arxiv.org/api/query?search_query=ti:"electron thermal conductivity"&sortBy=lastUpdatedDate&sortOrder=ascending
```

3.2. The API Response

Everything returned by the API in the body of the HTTP responses is Atom 1.0, including errors. Atom is a grammar of XML that is popular in the world of content syndica Typically web sites with dynamic content such as news sites and blogs will publish their content as Atom or RSS feeds. However, Atom is a general format that embodies to returning the arXiv search results.

3.3. Outline of an Atom feed

In this section we will discuss the contents of the Atom documents returned by the API. To see the full explanation of the Atom 1.0 format, please see the Atom specifica

An API response consists of an Atom `<feed>` element which contains metadata about the API call performed, as well as child `<entry>` elements which embody the me explain each of the elements and attributes. We will base our discussion on the sample results feed discussed in the examples section.

You may notice that the results from the API are ordered differently that the results given by the HTML arXiv search interface. The HTML interface automatically sorts results in descending order based o according to relevancy from the internal search engine. Thus when debugging a search query, we encourage you to use the API within a web browser, rather than the HTML search interface. If you want reading the `<published>` tag for each entry as explained below.

3.3.1. Feed Metadata

Every response will contain the line:

```
<?xml version="1.0" encoding="utf-8"?>
```

to signify that we are receiving XML 1.0 with a UTF-8 encoding. Following that line will be a line indicating that we are receiving an Atom feed:

```
<feed xmlns="http://www.w3.org/2005/Atom"
xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/"
xmlns:arxiv="http://arxiv.org/schemas/atom">
```

You will notice that three XML namespaces are defined. The default namespace signifies that we are dealing with Atom 1.0. The other two namespaces define extension

3.3.1.1. <title>, <id>, <link> and <updated>

The `<title>` element gives the title for the feed:

```
<title xmlns="http://www.w3.org/2005/Atom">
    ArXiv Query:  search_query=all:electron&amp;id_list=&amp;start=0&amp;max_results=1
</title>
```

The title contains a canonicalized version of the query used to call the API. The canonicalization includes all parameters, using their defaults if they were not included, an `search_query` , `id_list` , `start` , `max_results` , even if they were specified in a different order in the actual query.

The `<id>` element serves as a unique id for this query, and is useful if you are writing a program such as a feed reader that wants to keep track of all the feeds requeste database.

```
<id xmlns="http://www.w3.org/2005/Atom">
    http://arxiv.org/api/cHxbiOdZaP56ODnBPIenZhzg5f8
</id>
```

The id is guaranteed to be unique for each query.

The `<link>` element provides a URL that can be used to retrieve this feed again.

```
<link xmlns="http://www.w3.org/2005/Atom" href="http://arxiv.org/api/query?search_query=all:electron&amp;id_list=&amp;start=0&amp;max_results=1" r
```

Note that the url in the link represents the canonicalized version of the query. The `<link>` provides a GET requestable url, even if the original request was done via POS

The `<updated>` element provides the last time the contents of the feed were last updated:

```
<updated xmlns="http://www.w3.org/2005/Atom">2007-10-08T00:00:00-04:00</updated>
```

Because the arXiv submission process works on a 24 hour submission cycle, new articles are only available to the API on the midnight *after* the articles were processed. The `<updated>` tag thus reflects t **important** - search results do not change until new articles are added. Therefore there is no need to call the API more than once in a day for the same query. Please cache your results. This primarily app around with the API while you are developing your program!

3.3.1.2. OpenSearch Extension Elements

There are several extension elements defined in the OpenSearch namespace

```
http://a9.com/-/spec/opensearch/1.1/
```

OpenSearch is a lightweight technology that acts in a similar way as the Web Services Description Language. The OpenSearch elements we have included allow OpenSea often include search result aggregators and browser pluggins that allow searching from a variety of sources.

The OpenSearch extension elements can still be useful to you even if you are not writing one of these applications. The `<opensearch:totalResults>` element lists how

```
<opensearch:totalResults xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">
   1000
</opensearch:totalResults>
```

This can be very useful when implementing paging of search results. The other two elements `<opensearch:startIndex>` , and `<opensearch:itemsPerPage>` are analog

```
<opensearch:startIndex xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">
   0
</opensearch:startIndex>
<opensearch:itemsPerPage xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">
```

```
    1
</opensearch:itemsPerPage>
```

3.3.2. Entry Metadata

If there are no errors, the `<feed>` element contains 0 or more child `<entry>` elements with each `<entry>` representing an article in the returned results set. As expla `<entry>` element representing the error is returned. Below the element description describes the elements for `<entry>` 's representing arXiv articles. For a general dis explanation.

3.3.2.1. <title>, <id>, <published>, and <updated>

The `<title>` element contains the title of the article returned:

```
<title xmlns="http://www.w3.org/2005/Atom">
    Multi-Electron Production at High Transverse Momenta in ep Collisions at HERA
</title>
```

The `<id>` element contains a url that resolves to the abstract page for that article:

```
<id xmlns="http://www.w3.org/2005/Atom">
    http://arxiv.org/abs/hep-ex/0307015
</id>
```

If you want only the arXiv id for the article, you can remove the leading `http://arxiv.org/abs/` in the `<id>` .

The `<published>` tag contains the date in which the `first` version of this article was submitted and processed. The `<updated>` element contains the date on which t the version is version 1, then `<published> == <updated>` , otherwise they are different. In the example below, the article retrieved was version 2, so `<updated>` and `<`

```
<published xmlns="http://www.w3.org/2005/Atom">
    2007-02-27T16:02:02-05:00
</published>
<updated xmlns="http://www.w3.org/2005/Atom">
    2007-06-25T17:09:59-04:00
</updated>
```

3.3.2.2. <summary>, <author> and <category>

The `<summary>` element contains the abstract for the article:

```
<summary xmlns="http://www.w3.org/2005/Atom">
    Multi-electron production is studied at high electron transverse momentum
    in positron- and electron-proton collisions using the H1 detector at HERA.
    The data correspond to an integrated luminosity of 115 pb-1. Di-electron
    and tri-electron event yields are measured. Cross sections are derived in
    a restricted phase space region dominated by photon-photon collisions. In
    general good agreement is found with the Standard Model predictions.
    However, for electron pair invariant masses above 100 GeV, three
    di-electron events and three tri-electron events are observed, compared to
    Standard Model expectations of 0.30 \pm 0.04 and 0.23 \pm 0.04,
    respectively.
</summary>
```

There is one `<author>` element for each author of the paper in order of authorship. Each `<author>` element has a `<name>` sub-element which contains the name of t

```
<author xmlns="http://www.w3.org/2005/Atom">
      <name xmlns="http://www.w3.org/2005/Atom">H1 Collaboration</name>
</author>
```

If author affiliation is present, it is included as an `<arxiv:affiliation>` subelement of the `<author>` element as discussed below.

The `<category>` element is used to describe either an arXiv, ACM, or MSC classification. See the arXiv metadata explanation for more details about these classifications `scheme` , which is the categorization scheme, and `term` which is the term used in the categorization. Here is an example from the query http://export.arxiv.org/api/que

```
<category xmlns="http://www.w3.org/2005/Atom" term="cs.LG" scheme="http://arxiv.org/schemas/atom"/>
<category xmlns="http://www.w3.org/2005/Atom" term="cs.AI" scheme="http://arxiv.org/schemas/atom"/>
<category xmlns="http://www.w3.org/2005/Atom" term="I.2.6" scheme="http://arxiv.org/schemas/atom"/>
```

Note that in this example, there are 3 category elements, one for each category. The first two correspond to arXiv categories, and the last one to an ACM category. See < how to identify the arXiv primary category.

3.3.2.3. <link>'s

For each entry, there are up to three `<link>` elements, distinguished by their `rel` and `title` attributes. The table below summarizes what these links refer to

rel	title	refers to	always present
alternate	-	abstract page	yes
related	pdf	pdf	yes
related	doi	resolved doi	no

For example:

```
<link xmlns="http://www.w3.org/2005/Atom" href="http://arxiv.org/abs/hep-ex/0307015v1" rel="alternate" type="text/html"/>
<link xmlns="http://www.w3.org/2005/Atom" title="pdf" href="http://arxiv.org/pdf/hep-ex/0307015v1" rel="related" type="application/pdf"/>
<link xmlns="http://www.w3.org/2005/Atom" title="doi" href="http://dx.doi.org/10.1529/biophysj.104.047340" rel="related"/>
```

3.3.2.4. <arxiv> extension elements

There are several pieces of arXiv metadata that are not able to be mapped onto the standard Atom specification. We have therefore defined several extension elements

```
http://arxiv.org/schemas/atom
```

The arXiv classification system supports multiple <category> tags, as well as a primary classification. The primary classification is a replica of an Atom <category> tag, exc example, from the query http://export.arxiv.org/api/query?id_list=cs/9901002v1, we have

```
<arxiv:primary_category xmlns:arxiv="http://arxiv.org/schemas/atom" term="cs.LG" scheme="http://arxiv.org/schemas/atom"/>
```

signifying that `cs.LG` is the primary arXiv classification for this e-print.

The `<arxiv:comment>` element contains the typical author comments found on most arXiv articles:

```
<arxiv:comment xmlns:arxiv="http://arxiv.org/schemas/atom">
   23 pages, 8 figures and 4 tables
</arxiv:comment>
```

If the author has supplied affiliation information, then this is included as an `<arxiv:affiliation>` subelement of the standard Atom `<author>` element. For example, id_list=0710.5765v1, we have

```
<author>
   <name>G. G. Kacprzak</name>
   <arxiv:affiliation xmlns:arxiv="http://arxiv.org/schemas/atom">NMSU</arxiv:affiliation>
</author>
```

If the author has provided a journal reference for the article, then there will be a `<arxiv:journal_ref>` element with this information:

```
<arxiv:journal_ref xmlns:arxiv="http://arxiv.org/schemas/atom">
   Eur.Phys.J. C31 (2003) 17-29
</arxiv:journal_ref>
```

If the author has provided a DOI for the article, then there will be a `<arxiv:doi>` element with this information:

```
<arxiv:doi xmlns:arxiv="http://arxiv.org/schemas/atom">
   10.1529/biophysj.104.047340
</arxiv:doi>
```

3.4. Errors

Errors are returned as Atom feeds with a single entry representing the error. The `<summary>` for the error contains a helpful error message, and the `<link>` element c message.

For example, the API call http://export.arxiv.org/api/query?id_list=1234.12345 contains a malformed id, and results in the error

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">
  <link xmlns="http://www.w3.org/2005/Atom" href="http://arxiv.org/api/query?search_query=&amp;id_list=1234.12345" rel="self" type="application/at
  <title xmlns="http://www.w3.org/2005/Atom">ArXiv Query: search_query=&amp;id_list=1234.12345</title>
  <id xmlns="http://www.w3.org/2005/Atom">http://arxiv.org/api/kvuntZ8c9a4Eq5CF7KY03nMug+Q</id>
  <updated xmlns="http://www.w3.org/2005/Atom">2007-10-12T00:00:00-04:00</updated>
  <opensearch:totalResults xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">1</opensearch:totalResults>
  <opensearch:startIndex xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">0</opensearch:startIndex>
```

```
  <opensearch:itemsPerPage xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">1</opensearch:itemsPerPage>
  <entry xmlns="http://www.w3.org/2005/Atom">
    <id xmlns="http://www.w3.org/2005/Atom">http://arxiv.org/api/errors#incorrect_id_format_for_1234.12345</id>
    <title xmlns="http://www.w3.org/2005/Atom">Error</title>
    <summary xmlns="http://www.w3.org/2005/Atom">incorrect id format for 1234.12345</summary>
    <updated xmlns="http://www.w3.org/2005/Atom">2007-10-12T00:00:00-04:00</updated>

    <link xmlns="http://www.w3.org/2005/Atom" href="http://arxiv.org/api/errors#incorrect_id_format_for_1234.12345" rel="alternate" type="text/htm
    <author xmlns="http://www.w3.org/2005/Atom">
      <name xmlns="http://www.w3.org/2005/Atom">arXiv api core</name>
    </author>
  </entry>
</feed>
```


The following table gives information on errors that might occur.

Sample query	Error Explanation
http://export.arxiv.org/api/query?start=not_an_int	`start` must be an integer
http://export.arxiv.org/api/query?start=-1	`start` must be >= 0
http://export.arxiv.org/api/query?max_results=not_an_int	`max_results` must be an integer
http://export.arxiv.org/api/query?max_results=-1	`max_results` must be >= 0
http://export.arxiv.org/api/query?id_list=1234.1234	malformed id - see arxiv identifier explanation
http://export.arxiv.org/api/query?id_list=cond—mat/0709123	malformed id - see arxiv identifier explanation

4. Examples

Once you have familiarized yourself with the API, you should be able to easily write programs that call the API automatically. Most programming languages, if not all, hav Since Atom is growing, not all languages have libraries that support Atom parsing, so most of the programming effort will be in digesting the responses you receive. The calling the api via HTTP and parsing the results include:

- Perl (via LWP) (example)
- Python (via urllib) (example)
- Ruby (via uri and net/http) (example)
- PHP (via file_get_contents()) (example)

4.1. Simple Examples

Below we include code snippets for these languages that perform the bare minimum functionality - calling the api and printing the raw Atom results. If your favorite lang we'll be glad to post it!

All of the simple examples produce an output which looks like:

Example: A Typical Atom Response


```
    <?xml version="1.0" encoding="utf-8"?>
    <feed xmlns="http://www.w3.org/2005/Atom" xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/" xmlns:arxiv="http://arxiv.org/schemas/atom">
      <link xmlns="http://www.w3.org/2005/Atom" href="http://arxiv.org/api/query?search_query=all:electron&amp;id_list=&amp;start=0&amp;max_result
      <title xmlns="http://www.w3.org/2005/Atom">ArXiv Query: search_query=all:electron&amp;id_list=&amp;start=0&amp;max_results=1</title>
      <id xmlns="http://www.w3.org/2005/Atom">http://arxiv.org/api/cHxbiOdZaP56ODnBPIenZhzg5f8</id>
      <updated xmlns="http://www.w3.org/2005/Atom">2007-10-08T00:00:00-04:00</updated>
      <opensearch:totalResults xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">1000</opensearch:totalResults>
      <opensearch:startIndex xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">0</opensearch:startIndex>
      <opensearch:itemsPerPage xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">1</opensearch:itemsPerPage>
      <entry xmlns="http://www.w3.org/2005/Atom" xmlns:arxiv="http://arxiv.org/schemas/atom">
        <id xmlns="http://www.w3.org/2005/Atom">http://arxiv.org/abs/hep-ex/0307015</id>
        <published xmlns="http://www.w3.org/2005/Atom">2003-07-07T13:46:39-04:00</published>
        <updated xmlns="http://www.w3.org/2005/Atom">2003-07-07T13:46:39-04:00</updated>
        <title xmlns="http://www.w3.org/2005/Atom">Multi-Electron Production at High Transverse Momenta in ep Collisions at
      HERA</title>
        <summary xmlns="http://www.w3.org/2005/Atom">  Multi-electron production is studied at high electron transverse momentum in
    positron- and electron-proton collisions using the H1 detector at HERA. The
    data correspond to an integrated luminosity of 115 pb-1. Di-electron and
    tri-electron event yields are measured. Cross sections are derived in a
    restricted phase space region dominated by photon-photon collisions. In general
    good agreement is found with the Standard Model predictions. However, for
    electron pair invariant masses above 100 GeV, three di-electron events and
    three tri-electron events are observed, compared to Standard Model expectations
    of 0.30 \pm 0.04 and 0.23 \pm 0.04, respectively.
    </summary>
        <author xmlns="http://www.w3.org/2005/Atom">
```

```
      <name xmlns="http://www.w3.org/2005/Atom">H1 Collaboration</name>
    </author>
    <arxiv:comment xmlns:arxiv="http://arxiv.org/schemas/atom">23 pages, 8 figures and 4 tables</arxiv:comment>
    <arxiv:journal_ref xmlns:arxiv="http://arxiv.org/schemas/atom">Eur.Phys.J. C31 (2003) 17-29</arxiv:journal_ref>
    <link xmlns="http://www.w3.org/2005/Atom" href="http://arxiv.org/abs/hep-ex/0307015v1" rel="alternate" type="text/html"/>
    <link xmlns="http://www.w3.org/2005/Atom" title="pdf" href="http://arxiv.org/pdf/hep-ex/0307015v1" rel="related" type="application/pdf"/>
    <arxiv:primary_category xmlns:arxiv="http://arxiv.org/schemas/atom" term="hep-ex" scheme="http://arxiv.org/schemas/atom"/>
    <category term="hep-ex" scheme="http://arxiv.org/schemas/atom"/>
  </entry>
</feed>
```

4.1.1. Perl

LWP is in the default perl installation on most platforms. It can be downloaded and installed from CPAN. Sample code to produce the above output is:

```
use LWP;
use strict;

my $url = 'http://export.arxiv.org/api/query?search_query=all:electron&start=0&max_results=1';
my $browser = LWP::UserAgent->new();
my $response = $browser->get($url);
print $response->content();
```

4.1.2. Python

The urllib module is part of the python standard library, and is included in any default installation of python. Sample code to produce the above output in Python 2.7 is:

```
import urllib
url = 'http://export.arxiv.org/api/query?search_query=all:electron&start=0&max_results=1'
data = urllib.urlopen(url).read()
print data
```

wheras in Python 3 an example would be:

```
import urllib.request as libreq
with libreq.urlopen('http://export.arxiv.org/api/query?search_query=all:electron&start=0&max_results=1') as url:
  r = url.read()
print(r)
```

4.1.3. Ruby

The net/http and uri modules are part of the ruby standard library, and are included in any default installation of ruby. Sample code to produce the above output is:

```
require 'net/http'
require 'uri'
url = URI.parse('http://export.arxiv.org/api/query?search_query=all:electron&start=0&max_results=1')
res = Net::HTTP.get_response(url)
print res.body
```

4.1.4. PHP

The file_get_contents() function is part of the PHP core language:

```
<?php
$url = 'http://export.arxiv.org/api/query?search_query=all:electron&start=0&max_results=1';
$response = file_get_contents($url);
print_r($response);
?>
```

4.2. Detailed Parsing Examples

The examples above don't cover how to parse the Atom results returned to extract the information you might be interested in. They also don't cover how to do more ad as downloading chunks of the full results list one page at a time. The table below contains links to more detailed examples for each of the languages above, as well as to

Language	Library	Parsing Example	Paging Example
Perl	XML::Atom	parsing	paging
Python	feedparser	parsing	paging

Ruby	feedtools	parsing	paging
PHP	SimplePie	parsing	paging

5. Appendices

5.1. Details of Query Construction

As outlined in the Structure of the API section, the interface to the API is quite simple. This simplicity, combined with `search_query` construction, and result set filtering harvesting data from the arXiv. In this section, we outline the possibilities for constructing `search_query`'s to retrieve our desired article lists. We outlined how to use tl search_query and id_list logic.

In the arXiv search engine, each article is divided up into a number of fields that can individually be searched. For example, the titles of an article can be searched, as we reference. To search one of these fields, we simply prepend the field prefix followed by a colon to our search term. For example, suppose we wanted to find all articles b construct the following query

http://export.arxiv.org/api/query?search_query=au:del_maestro

This returns nine results. The following table lists the field prefixes for all the fields that can be searched.

prefix	explanation
ti	Title
au	Author
abs	Abstract
co	Comment
jr	Journal Reference
cat	Subject Category
rn	Report Number
id	Id (use `id_list` instead)
all	All of the above

Note: The `id_list` parameter should be used rather than `search_query=id:xxx` to properly handle article versions. In addition, note that `all:` searches in each of the fields simultaneously.

The API allows advanced query construction by combining these search fields with Boolean operators. For example, suppose we want to find all articles by the author `A` `checkerboard` in the title. We could construct the following query, using the `AND` operator:

http://export.arxiv.org/api/query?search_query=au:del_maestro+AND+ti:checkerboard

As expected, this query picked out the one of the nine previous results with `checkerboard` in the title. Note that we included `+` signs in the urls to the API. In a url, a `+` are not allowed in url's. It is always a good idea to escape the characters in your url's, which is a common feature in most programming libraries that deal with url's. Note in the query constructed. It is a good idea to look at <title> to see if you have escaped your url correctly.

The following table lists the three possible Boolean operators.

`AND`
`OR`
`ANDNOT`

The `ANDNOT` Boolean operator is particularly useful, as it allows us to filter search results based on certain fields. For example, if we wanted all of the articles by the auth contain the word `checkerboard`, we could construct the following query:

http://export.arxiv.org/api/query?search_query=au:del_maestro+ANDNOT+ti:checkerboard

As expected, this query returns eight results.

Finally, even more complex queries can be used by using parentheses for grouping the Boolean expressions. To include parentheses in in a url, use `%28` for a left-parer we wanted all of the articles by the author `Adrian DelMaestro` with titles that *did not* contain the words `checkerboard`, OR `Pyrochore`, we could construct the followi

http://export.arxiv.org/api/query?search_query=au:del_maestro+ANDNOT+%28ti:checkerboard+OR+ti:Pyrochlore%29

This query returns three results. Notice that the <title> element displays the parenthesis correctly meaning that we used the correct url escaping.

So far we have only used single words as the field terms to search for. You can include entire phrases by enclosing the phrase in double quotes, escaped by `%22`. For ex `Adrian DelMaestro` with titles that contain `quantum criticality`, we could construct the following query:

http://export.arxiv.org/api/query?search_query=au:del_maestro+AND+ti:%22quantum+criticality%22

This query returns one result, and notice that the feed `<title>` contains double quotes as expected. The table below lists the two grouping operators used in the API.

symbol	encoding	explanation
( )	%28 %29	Used to group Boolean expressions for Boolean operator precedence.
double quotes	%22 %22	Used to group multiple words into phrases to search a particular field.
space	+	Used to extend a `search_query` to include multiple fields.

5.1.1. A Note on Article Versions

Each arXiv article has a version associated with it. The first time an article is posted, it is given a version number of 1. When subsequent corrections are made to an articl incremented. At any time, any version of an article may be retrieved.

When using the API, if you want to retrieve the latest version of an article, you may simply enter the arxiv id in the `id_list` parameter. If you want to retrieve informati appending `vn` to the id, where `n` is the version number you are interested in.

For example, to retrieve the latest version of `cond-mat/0207270`, you could use the query http://export.arxiv.org/api/query?id_list=cond-mat/0207270. To retrieve the ve http://export.arxiv.org/api/query?id_list=cond-mat/0207270v1

5.2. Details of Atom Results Returned

The following table lists each element of the returned Atom results. For a more detailed explanation see Outline of an Atom Feed.

element	explanation
feed elements	
`<title>`	The title of the feed containing a canonicalized query string.
`<id>`	A unique id assigned to this query.
`<updated>`	The last time search results for this query were updated. Set to midnight of the current day.
`<link>`	A url that will retrieve this feed via a GET request.
`<opensearch:totalResults>`	The total number of search results for this query.
`<opensearch:startIndex>`	The 0-based index of the first returned result in the total results list.
`<opensearch:itemsPerPage>`	The number of results returned.
entry elements	
`<title>`	The title of the article.
`<id>`	A url `http://arxiv.org/abs/id`
`<published>`	The date that `version 1` of the article was submitted.
`<updated>`	The date that the retrieved version of the article was submitted. Same as `<published>` if the retrieved version is version 1.
`<summary>`	The article abstract.
`<author>`	One for each author. Has child element `<name>` containing the author name.
`<link>`	Can be up to 3 given url's associated with this article.
`<category>`	The arXiv or ACM or MSC category for an article if present.
`<arxiv:primary_category>`	The primary arXiv category.
`<arxiv:comment>`	The authors comment if present.
`<arxiv:affiliation>`	The author's affiliation included as a subelement of `<author>` if present.
`<arxiv:journal_ref>`	A journal reference if present.
`<arxiv:doi>`	A url for the resolved DOI to an external resource if present.

5.3. Subject Classifications

astro-ph	Astrophysics
astro-ph.CO	Cosmology and Nongalactic Astrophysics
astro-ph.EP	Earth and Planetary Astrophysics
astro-ph.GA	Astrophysics of Galaxies
astro-ph.HE	High Energy Astrophysical Phenomena
astro-ph.IM	Instrumentation and Methods for Astrophysics
astro-ph.SR	Solar and Stellar Astrophysics
cond-mat.dis-nn	Disordered Systems and Neural Networks
cond-mat.mes-hall	Mesoscale and Nanoscale Physics
cond-mat.mtrl-sci	Materials Science
cond-mat.other	Other Condensed Matter
cond-mat.quant-gas	Quantum Gases
cond-mat.soft	Soft Condensed Matter
cond-mat.stat-mech	Statistical Mechanics
cond-mat.str-el	Strongly Correlated Electrons
cond-mat.supr-con	Superconductivity
cs.AI	Artificial Intelligence
cs.AR	Hardware Architecture
cs.CC	Computational Complexity
cs.CE	Computational Engineering, Finance, and Science
cs.CG	Computational Geometry
cs.CL	Computation and Language
cs.CR	Cryptography and Security
cs.CV	Computer Vision and Pattern Recognition
cs.CY	Computers and Society
cs.DB	Databases
cs.DC	Distributed, Parallel, and Cluster Computing
cs.DL	Digital Libraries
cs.DM	Discrete Mathematics
cs.DS	Data Structures and Algorithms
cs.ET	Emerging Technologies
cs.FL	Formal Languages and Automata Theory
cs.GL	General Literature
cs.GR	Graphics
cs.GT	Computer Science and Game Theory
cs.HC	Human-Computer Interaction
cs.IR	Information Retrieval
cs.IT	Information Theory
cs.LG	Learning

cs.LO	Logic in Computer Science
cs.MA	Multiagent Systems
cs.MM	Multimedia
cs.MS	Mathematical Software
cs.NA	Numerical Analysis
cs.NE	Neural and Evolutionary Computing
cs.NI	Networking and Internet Architecture
cs.OH	Other Computer Science
cs.OS	Operating Systems
cs.PF	Performance
cs.PL	Programming Languages
cs.RO	Robotics
cs.SC	Symbolic Computation
cs.SD	Sound
cs.SE	Software Engineering
cs.SI	Social and Information Networks
cs.SY	Systems and Control
econ.EM	Econometrics
eess.AS	Audio and Speech Processing
eess.IV	Image and Video Processing
eess.SP	Signal Processing
gr-qc	General Relativity and Quantum Cosmology
hep-ex	High Energy Physics - Experiment
hep-lat	High Energy Physics - Lattice
hep-ph	High Energy Physics - Phenomenology
hep-th	High Energy Physics - Theory
math.AC	Commutative Algebra
math.AG	Algebraic Geometry
math.AP	Analysis of PDEs
math.AT	Algebraic Topology
math.CA	Classical Analysis and ODEs
math.CO	Combinatorics
math.CT	Category Theory
math.CV	Complex Variables
math.DG	Differential Geometry
math.DS	Dynamical Systems
math.FA	Functional Analysis
math.GM	General Mathematics
math.GN	General Topology
math.GR	Group Theory

math.GT	Geometric Topology
math.HO	History and Overview
math.IT	Information Theory
math.KT	K-Theory and Homology
math.LO	Logic
math.MG	Metric Geometry
math.MP	Mathematical Physics
math.NA	Numerical Analysis
math.NT	Number Theory
math.OA	Operator Algebras
math.OC	Optimization and Control
math.PR	Probability
math.QA	Quantum Algebra
math.RA	Rings and Algebras
math.RT	Representation Theory
math.SG	Symplectic Geometry
math.SP	Spectral Theory
math.ST	Statistics Theory
math-ph	Mathematical Physics
nlin.AO	Adaptation and Self-Organizing Systems
nlin.CD	Chaotic Dynamics
nlin.CG	Cellular Automata and Lattice Gases
nlin.PS	Pattern Formation and Solitons
nlin.SI	Exactly Solvable and Integrable Systems
nucl-ex	Nuclear Experiment
nucl-th	Nuclear Theory
physics.acc-ph	Accelerator Physics
physics.ao-ph	Atmospheric and Oceanic Physics
physics.app-ph	Applied Physics
physics.atm-clus	Atomic and Molecular Clusters
physics.atom-ph	Atomic Physics
physics.bio-ph	Biological Physics
physics.chem-ph	Chemical Physics
physics.class-ph	Classical Physics
physics.comp-ph	Computational Physics
physics.data-an	Data Analysis, Statistics and Probability
physics.ed-ph	Physics Education
physics.flu-dyn	Fluid Dynamics
physics.gen-ph	General Physics
physics.geo-ph	Geophysics

physics.hist-ph	History and Philosophy of Physics
physics.ins-det	Instrumentation and Detectors
physics.med-ph	Medical Physics
physics.optics	Optics
physics.plasm-ph	Plasma Physics
physics.pop-ph	Popular Physics
physics.soc-ph	Physics and Society
physics.space-ph	Space Physics
q-bio.BM	Biomolecules
q-bio.CB	Cell Behavior
q-bio.GN	Genomics
q-bio.MN	Molecular Networks
q-bio.NC	Neurons and Cognition
q-bio.OT	Other Quantitative Biology
q-bio.PE	Populations and Evolution
q-bio.QM	Quantitative Methods
q-bio.SC	Subcellular Processes
q-bio.TO	Tissues and Organs
q-fin.CP	Computational Finance
q-fin.EC	Economics
q-fin.GN	General Finance
q-fin.MF	Mathematical Finance
q-fin.PM	Portfolio Management
q-fin.PR	Pricing of Securities
q-fin.RM	Risk Management
q-fin.ST	Statistical Finance
q-fin.TR	Trading and Market Microstructure
quant-ph	Quantum Physics
stat.AP	Applications
stat.CO	Computation
stat.ME	Methodology
stat.ML	Machine Learning
stat.OT	Other Statistics
stat.TH	Statistics Theory

Last updated 2018-02-05

"arXiv API User's Manual" revision 0.5.2. Last modified 2020-01-14.