



Cornell University

We gratefully acknowledge support from the Simons Foundation,  
member institutions, and all contributors.

# arXiv API User's Manual



Please review the [Terms of Use for arXiv APIs](#) before using the arXiv API.

## Table of Contents

### [1. Preface](#)

### [2. API QuickStart](#)

### [3. Structure of the API](#)

#### [3.1. Calling the API](#)

##### [3.1.1. Query Interface](#)

###### [3.1.1.1. search\\_query and id\\_list logic](#)

###### [3.1.1.2. start and max\\_results paging](#)

###### [3.1.1.3. sort order for return results](#)

#### [3.2. The API Response](#)

#### [3.3. Outline of an Atom feed](#)

##### [3.3.1. Feed Metadata](#)

###### [3.3.1.1. <title>, <id>, <link> and <updated>](#)

###### [3.3.1.2. OpenSearch Extension Elements](#)

##### [3.3.2. Entry Metadata](#)

###### [3.3.2.1. <title>, <id>, <published>, and <updated>](#)

###### [3.3.2.1. <summary>, <author> and <category>](#)

###### [3.3.2.3. <link>'s](#)

###### [3.3.2.4. <arxiv> extension elements](#)

#### [3.4. Errors](#)

## [4. Examples](#)

### [4.1. Simple Examples](#)

#### [4.1.1. Perl](#)

#### [4.1.2. Python](#)

#### [4.1.3. Ruby](#)

#### [4.1.4. PHP](#)

### [4.2. Detailed Parsing Examples](#)

## [5. Appendices](#)

### [5.1. Details of Query Construction](#)

#### [5.1.1. A Note on Article Versions](#)

### [5.2. Details of Atom Results Returned](#)

### [5.3. Subject Classifications](#)

# 1. Preface

The arXiv API allows programmatic access to the hundreds of thousands of e-prints hosted on [arXiv.org](https://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 advanced users. To get a flavor for how the API 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

[Examples](#) section.

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 web browser, and entering the following url in your 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 page. For example, if I wanted to search for articles 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 this](#): a web page listing the title and authors of each 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 search results for `electron` by entering the url

[http://export.arxiv.org/api/query?search\\_query=all:electron](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](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 look different is that the API returns results in the 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 a web browser by itself, but this is a particularly 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

by web sites to syndicate their content. These feeds 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 within the category you specify. One immediately useful 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 returned by the API, see the section on [the API response](#). 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 request header. Most client libraries support both 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 url for a GET request, but not for the POST request. 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 `&` sign in the constructed url's.

query				
parameters	type	defaults	required	
<code>search_query</code>	string	None	No	

<code>id_list</code>	comma-delimited string	None	No
<code>start</code>	int	0	No
<code>max_results</code>	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 construction of `search_query` is described in the [search 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 filter.

This is summarized in the following table:

<code>search_query</code> <b>present</b>	<code>id_list</code> <b>present</b>	<b>API returns</b>
yes	no	articles that match <code>search_query</code>
no	yes	articles that are in <code>id_list</code>
yes	yes	articles in <code>id_list</code> that also match <code>search_query</code>

### 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 `start` and `max_results` that allows you to download chunks 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 number of results returned by the query. For 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 variety of languages.

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 `max_results` and `start` query parameters. For example to retrieve matches 6001-8000: `http://export.arxiv.org/api/query?search_query=all:electron&start=6000&max_results=2000`

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 least request smaller slices. For bulk metadata 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 explanation. A request for 30000 results will typically 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" (Apache Lucene's default [RELEVANCE](#) ordering), "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 syndication, and is very similar to RSS for this purpose. 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 the concept of a list of items, and thus is well-suited 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 specification](#).

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 metadata for each of the returned results. Below we 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 than the results given by the [HTML arXiv search interface](#). The HTML interface automatically sorts results in descending order based on the date of their submission, while the API returns results 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 sorting by date, you can always do this within your programs by 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 extensions to Atom that we describe below.

#### 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&id_list=&start=0&max_results=
</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, and always puts them in the order

`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 requested in the past. This id can then be used as a key in a database.

```
<id xmlns="http://www.w3.org/2005/Atom">  
  http://arxiv.org/api/cHxbi0dZaP560DnBPIenZhgz5f8  
</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&id_list=&start=0&max_results=  
  rel="self" type="application/atom+xml"/>
```

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 POST.

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 the midnight of the day that you are calling the API. **This is very 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 applies to production systems, and of course you are free to play 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 OpenSearch enabled clients to digest our results. Such clients 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 many results are in the result set for the query:

```
<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 analogous to `start`, and `max_results` [discussed above](#).

```
<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 one or more child `<entry>` elements with each `<entry>` representing an article in the returned results set. As explained in the [errors](#) section, if there are errors, a single `<entry>` element representing the error is returned. Below the element description describes the elements for `<entry>`'s representing arXiv articles. For a general discussion of arXiv metadata, see the [arXiv metadata 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

the retrieved article was submitted and processed. If 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>` are different (see the [original query](#)).

```
<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 the author.

```
<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. The `<category>` element has two attributes, `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/query?id\\_list=cs/9901002v1](http://export.arxiv.org/api/query?id_list=cs/9901002v1)

```
<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 [<arxiv> extension elements](#) below for information on 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 which live in the `arxiv` namespace

```
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, except it has the name `<arxiv:primary_category>`. For example, from the query [http://export.arxiv.org/api/query?id\\_list=cs/9901002v1](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, from the query [http://export.arxiv.org/api/query?id\\_list=0710.5765v1](http://export.arxiv.org/api/query?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 contains a url to a more detailed explanation of the message.

For example, the API call [http://export.arxiv.org/api/query?id\\_list=1234.12345](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=&id_list=1234.12345" rel="self"
    type="application/atom+xml"/>
  <title xmlns="http://www.w3.org/2005/Atom">ArXiv Query:
    search_query=&id_list=1234.12345</title>
  <id
    xmlns="http://www.w3.org/2005/Atom">http://arxiv.org/api/kvuntZ8c9a
    <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>
  <opensearch:startIndex
xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">0</opensearch>

  <opensearch:itemsPerPage
xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">1</opensearch>
  <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_1234.1234
rel="alternate" type="text/html"/>
    <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
<a href="http://export.arxiv.org/api/query?start=not_an_int">http://export.arxiv.org/api/query?start=not_an_int</a>	<code>start</code> must be an integer
<a href="http://export.arxiv.org/api/query?start=-1">http://export.arxiv.org/api/query?start=-1</a>	<code>start</code> must be $\geq 0$

<a href="http://export.arxiv.org/api/query?max_results=not_an_int">http://export.arxiv.org/api/query? max_results=not_an_int</a>	<code>max_results</code> must be an integer
<a href="http://export.arxiv.org/api/query?max_results=-1">http://export.arxiv.org/api/query? max_results=-1</a>	<code>max_results</code> must be $\geq 0$
<a href="http://export.arxiv.org/api/query?id_list=1234.1234">http://export.arxiv.org/api/query? id_list=1234.1234</a>	malformed id - see <a href="#">arxiv identifier explanation</a>
<a href="http://export.arxiv.org/api/query?id_list=cond—mat/0709123">http://export.arxiv.org/api/query?id_list=cond —mat/0709123</a>	malformed id - see <a href="#">arxiv identifier explanation</a>

## 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, have libraries that allow you to make HTTP requests. 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 languages that we know of that can easily handle 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 language is not up here, write us with an example, and 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&id_list=&start=0&max_results=
rel="self" type="application/atom+xml"/>
  <title xmlns="http://www.w3.org/2005/Atom">ArXiv Query:
search_query=all:electron&id_list=&start=0&max_results=
  <id
xmlns="http://www.w3.org/2005/Atom">http://arxiv.org/api/cHxbi0dZaP
  <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</opens
  <opensearch:startIndex
xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">0</opensear
  <opensearch:itemsPerPage
xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">1</opensear
  <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
```

whereas 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 advanced programming of the API to perform such tasks 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 the libraries used to parse Atom.

Language	Library	Parsing Example	Paging Example
Perl	<a href="#">XML::Atom</a>	<a href="#">parsing</a>	<a href="#">paging</a>
Python	<a href="#">feedparser</a>	<a href="#">parsing</a>	<a href="#">paging</a>
Ruby	<a href="#">feedtools</a>	<a href="#">parsing</a>	<a href="#">paging</a>
PHP	<a href="#">SimplePie</a>	<a href="#">parsing</a>	<a href="#">paging</a>

## 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 through `id_list` makes the API a powerful tool for 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 the `id_list` parameter to filter results sets in [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 well as the author list, abstracts, comments and journal 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 by the author `Adrian Del Maestro`. We could construct the following query

[http://export.arxiv.org/api/query?search\\_query=au:del\\_maestro](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 <code>id_list</code> 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 provides one date filter, `submittedDate`, that allow you to select data within a given date range of when the data was submitted to arXiv. The expected format is `[YYYYMMDDTTTT+TO+YYYYMMDDTTTT]` were the `TTTT` is provided in 24 hour time to the minute, in GMT. We could construct the following query using `submittedDate`.

[https://export.arxiv.org/api/query?search\\_query=au:del\\_maestro+AND+submittedDate:\[202301010600+TO+202401010600\]](https://export.arxiv.org/api/query?search_query=au:del_maestro+AND+submittedDate:[202301010600+TO+202401010600])

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 `Adrian DelMaestro` that also contain the word `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](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 `+` sign encodes a space, which is useful since spaces 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 that the `<title>` of the returned feed has spaces 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 author `Adrian DelMaestro` with titles that *did not* contain the word `checkerboard`, we could construct the following query:

[http://export.arxiv.org/api/query?  
search\\_query=au:del\\_maestro+ANDNOT+ti:checkerboard](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 a url, use `%28` for a left-paren `(`, and `%29` for a right-paren `)`. For example, if we wanted all of the articles by the author `Adrian DelMaestro` with titles that *did not* contain the words `checkerboard`, OR `Pyrochlore`, we could construct the following query:

[http://export.arxiv.org/api/query?  
search\\_query=au:del\\_maestro+ANDNOT+%28ti:checkerboard+OR+ti:Pyrochlore%29](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 example, if we wanted all of the articles by the author `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](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 <code>search_query</code> 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 article, it is resubmitted, and the version number is 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 information about a specific version, you can do this by 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](http://export.arxiv.org/api/query?id_list=cond-mat/0207270). To retrieve the very first version of this article, you could use the query [http://export.arxiv.org/api/query?id\\_list=cond-mat/0207270v1](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	
<code>&lt;title&gt;</code>	The title of the feed containing a canonicalized query string.
<code>&lt;id&gt;</code>	A unique id assigned to this query.
<code>&lt;updated&gt;</code>	The last time search results for this query were updated. Set to midnight of the current day.
<code>&lt;link&gt;</code>	A url that will retrieve this feed via a GET request.
<code>&lt;opensearch:totalResults&gt;</code>	The total number of search results for this query.

<code>&lt;opensearch:startIndex&gt;</code>	The o-based index of the first returned result in the total results list.
<code>&lt;opensearch:itemsPerPage&gt;</code>	The number of results returned.
<b>entry elements</b>	
<code>&lt;title&gt;</code>	The title of the article.
<code>&lt;id&gt;</code>	A url <code>http://arxiv.org/abs/id</code>
<code>&lt;published&gt;</code>	The date that <code>version 1</code> of the article was submitted.
<code>&lt;updated&gt;</code>	The date that the retrieved version of the article was submitted. Same as <code>&lt;published&gt;</code> if the retrieved version is version 1.
<code>&lt;summary&gt;</code>	The article abstract.
<code>&lt;author&gt;</code>	One for each author. Has child element <code>&lt;name&gt;</code> containing the author name.
<code>&lt;link&gt;</code>	Can be up to 3 given url's associated with this article.


<code>&lt;category&gt;</code>	The arXiv or ACM or MSC category for an article if present.
<code>&lt;arxiv:primary_category&gt;</code>	The primary arXiv category.
<code>&lt;arxiv:comment&gt;</code>	The authors comment if present.
<code>&lt;arxiv:affiliation&gt;</code>	The author's affiliation included as a subelement of <code>&lt;author&gt;</code> if present.
<code>&lt;arxiv:journal_ref&gt;</code>	A journal reference if present.
<code>&lt;arxiv:doi&gt;</code>	A url for the resolved DOI to an external resource if present.

5.3. Subject Classifications

For the complete list of arXiv subject classifications, please visit the [taxonomy](#) page.

About


Help


 Contact


Copyright



Privacy Policy

Web Accessibility Assistance

 [Subscribe](#)

 [Report a documentation issue](#)

[arXiv Operational Status](#) 

Get status notifications via  email or  slack