

Package ‘fulltext’

January 22, 2019

Title Full Text of 'Scholarly' Articles Across Many Data Sources

Description Provides a single interface to many sources of full text 'scholarly' data, including 'Biomed Central', Public Library of Science, 'Pubmed Central', 'eLife', 'F1000Research', 'PeerJ', 'Pensoft', 'Hindawi', 'arXiv' 'preprints', and more. Functionality included for searching for articles, downloading full or partial text, downloading supplementary materials, converting to various data formats.

Version 1.2.0

License MIT + file LICENSE

URL <https://github.com/ropensci/fulltext/> (devel)
<https://ropensci.github.io/fulltext-book/> (user manual)

BugReports <https://github.com/ropensci/fulltext/issues>

LazyLoad yes

Encoding UTF-8

Language en-US

VignetteBuilder knitr

Imports crul (>= 0.7.0), magrittr, xml2 (>= 1.1.1), jsonlite, rplos (>= 0.8.0), rcrossref (>= 0.8.0), crminer (>= 0.2.0), microdemic (>= 0.2.0), aRxiv, rentrez (>= 1.1.0), data.table, hoardr (>= 0.5.0), pdftools, storr, tibble, digest

Suggests roxygen2 (>= 6.1.1), testthat, knitr, pubchunks

RoxygenNote 6.1.1

X-schema.org-applicationCategory Literature

X-schema.org-keywords text-mining, literature, pdf, xml, publications, citations, full-text, TDM

X-schema.org-isPartOf <https://ropensci.org>

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (<<https://orcid.org/0000-0003-1444-9135>>)

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2019-01-22 18:50:03 UTC

R topics documented:

fulltext-package	2
as.ft_data	5
cache	6
cache_file_info	7
ftxt_cache	8
ft_abstract	9
ft_browse	10
ft_collect	11
ft_extract	12
ft_get	13
ft_get-warnings	21
ft_links	22
ft_providers	25
ft_search	26
ft_serialize	29
ft_table	30
fulltext-defunct	31
Index	32

fulltext-package *Fulltext search and retrieval of scholarly texts.*

Description

fulltext is a single interface to many sources of scholarly texts. In practice, this means only ones that are legally useable. We will support sources that require authentication on a case by case basis - that is, if more than just a few people will use it, and it's not too burdensome to include, then we can include that source.

Manual

See <https://ropensci.github.io/fulltext-book/> for a longer form manual for using **fulltext**.

What's included

We currently include support for search and full text retrieval for a variety of publishers. See [ft_search\(\)](#) for what we include for search, and [ft_get\(\)](#) for what we include for full text retrieval.

Use cases

The following are tasks/use cases supported:

- search - `ft_search()`
- get texts - `ft_get()`
- get full text links - `ft_links()`
- get abstracts - `ft_abstract()`
- extract text from pdfs - `ft_extract()`
- serialize to different data formats - `ft_serialize()`
- extract certain article sections (e.g., authors) - moved to **pubchunks**

DOI delays

Beware that DOIs are not searchable via Crossref/Entrez immediately. The delay may be as much as a few days, though should be less than a day. This delay should become shorter as services improve. The point of this is that you may not find a match for a relatively new DOI (e.g., for an article published the same day). We've tried to account for this for some publishers. For example, for Crossref we search Crossref for a match for a DOI, and if none is found we attempt to retrieve the full text from the publisher directly.

Rate limits

Scopus: 20,000 per 7 days. See https://dev.elsevier.com/api_key_settings.html for rate limit information. To see what your personal rate limit details are, request verbose HTTP request output - this will vary on the function you are using - see the docs for the function. See the response headers `X-RateLimit-Limit`, `X-RateLimit-Remaining`, and `X-RateLimit-Reset` (your limit, those requests remaining, and UTC date/time it will reset)

Microsoft: 10,000 per month, and 1 per second. There are no rate limit headers, sorry :(

PLOS: There are no known rate limits for PLOS, though if you do hit something let us know.

Crossref: From time to time Crossref needs to impose rate limits to ensure that the free API is usable by all. Any rate limits that are in effect will be advertised in the `X-Rate-Limit-Limit` and `X-Rate-Limit-Interval` HTTP headers. This boils down to: they allow X number of requests per some time period. The numbers can change so we can't give a rate limit that will always be in effect. If you're curious pass in `verbose = TRUE` to your function call, and you'll get headers that will display these rate limits. See also **Authentication**.

Authentication

BMC: BMC is integrated into Springer Publishers now, and that API requires an API key. Get your key by signing up at <https://dev.springer.com/>, then you'll get a key. Pass the key to a named parameter `key` to `bmcopts`. Or, save your key in your `.Renvirom` file as `SPRINGER_KEY`, and we'll read it in for you, and you don't have to pass in anything.

Scopus: Scopus requires two things: an API key and your institution must have access. For the API key, go to <https://dev.elsevier.com/index.html>, register for an account, then when you're in your account, create an API key. Pass in as variable `key` to `scopusopts`, or store your key under the name `ELSEVIER_SCOPUS_KEY` as an environment variable in `.Renvirom`, and we'll read it in for

you. See [Startup](#) for help. For the institution access go to a browser and see if you have access to the journal(s) you want. If you don't have access in a browser you probably won't have access via this package. If you aren't physically at your institution you will likely need to be on a VPN or similar so that your IP address is in the range that the two publishers are accepting for that institution.

Microsoft: Get a key by creating an Azure account at <https://www.microsoft.com/cognitive-services/en-us/subscriptions>, then requesting a key for **Academic Knowledge API** within **Cognitive Services**. Store it as an environment variable in your `.Renvi` file - see [Startup](#) for help. Pass your API key into `maopts` as a named element in a list like `list(key = Sys.getenv('MICROSOFT_ACADEMIC_KEY'))`

Crossref: Crossref encourages requests with contact information (an email address) and will forward you to a dedicated API cluster for improved performance when you share your email address with them. <https://github.com/CrossRef/rest-api-doc#good-manners-more-reliable-service> To pass your email address to Crossref via this client, store it as an environment variable in `.Renvi` like `crossref_email = name@example.com`

Crossref TDM: TDM = "Text and Data Mining". This applies to the few publishers - Wiley and Elsevier - that are part of this program (TDM). When using `ft_get()`, and you want to get papers from these two publishers, you'll need two things: (1) an API key for the Crossref TDM. Go to <https://apps.crossref.org/clickthrough/researchers> and you'll be asked to login with your ORCID. If you don't have an ORCID go to <https://orcid.org/> and get one. After logging in with your ORCID, click on the "API token" tag and grab your API key. Put your API key in `.Renvi` file or similar (e.g. `.zshrc` or `.bash_profile`, etc.) with the entry `CROSSREF_TDM=yourkey`. We'll look for the environment variable `CROSSREF_TDM` within this package. See <http://tdmsupport.crossref.org/> for more information on the Crossref TDM program. (2) Your institution needs to have access to the journal you're requesting papers from. If you're not sure about this just go to a browser and see if you have access to the journal(s) you want. If you don't have access in a browser you probably won't have access via this package. If you aren't physically at your institution you will likely need to be on a VPN or similar so that your IP address is in the range that the two publishers are accepting for that institution. Also talk to your librarian if you aren't sure about access or have questions about it. In some cases, you may also need to request that Elsevier removes a "fence" for your institution - that is, your institution has access to XYZ journal(s), but they don't yet allow programmatic access. This has happened at least a few times that I know of.

Entrez: NCBI limits users to making only 3 requests per second. But, users who register for an API key are able to make up to ten requests per second. Getting a key is simple; register for a "my ncbi" account then click on a button in the account settings page. Once you have an API key, you can pass it as the argument `api_key` to `entrezopts` in both `ft_get()` and `ft_search()`. However, we advise you use environment variables instead as they are more secure. To do that you can set an environment variable for the current R session like `Sys.setenv(ENTREZ_KEY="yourkey")` OR better yet set it in your `.Renvi` or equivalent file with an entry like `ENTREZ_KEY=yourkey` so that it is used across R sessions.

No authentication needed for **PLOS**, **eLife**, **arxiv**, **biorxiv**, **Euro PMC**

Let us know if you run into trouble with authentication.

Feedback

Let us know what you think at <https://github.com/ropensci/fulltext/issues>

Author(s)

Scott Chamberlain myrmecocystus@gmail.com

Will Pearse

as.ft_data

Coerce directory of papers to ft_data object

Description

create the same object that `ft_get()` outputs from your cached files - without having to run `ft_get()` again

Usage

```
as.ft_data(path = NULL)
```

Arguments

path cache path. if not given, we use the default cache path. Default: NULL

Details

We use an internal store of identifiers to keep track of files. These identifiers are in the output of `ft_get()` and you can see them in that output. If a file does not have a matching entry in our index of files (e.g., if you drop a file into the cache location as in the example below), then we assign it an index based on the file path; we'd ideally use an article DOI or similar but we can not safely retrieve it with just a file path.

Value

an object of class `ft_data`

See Also

[ft_get\(\)](#)

Examples

```
# put a file in the cache in case there aren't any
dir <- file.path(tempdir(), "testing")
dir.create(dir)
file <- system.file("examples", "elif.xml", package = "fulltext")
writeLines(readLines(file), tempfile(tmpdir = dir, fileext = ".xml"))

# call as.ft_data
x <- as.ft_data(path = dir)

# output lives underneath a special list index "cached"
```

```

# representing already present files
x$cached

## Not run:
# collect chunks
if (requireNamespace("pubchunks")) {
  library(pubchunks)
  res <- ft_collect(x)
  pub_chunks(res, c("doi", "title")) %>% pub_tabularize()
}

## End(Not run)

```

cache	<i>Set or get cache options</i>
-------	---------------------------------

Description

Set or get cache options

Usage

```
cache_options_set(path = "fulltext", backend = "ext",
  overwrite = FALSE, full_path = NULL)
```

```
cache_options_get()
```

Arguments

path	(character) End of directory path. Default: "fulltext". See Details.
backend	(character) Only "ext" supported for now.
overwrite	(logical) overwrite cached file or not. Default: FALSE
full_path	(character) Full cache path. If given path is ignored.

Value

both functions return the cache options in a named list

Managing cached files

The default cache directory is `paste0(rappdirs::user_cache_dir(), "/R/fulltext")`, but you can set your own path using `cache_path_set()`

You can alternatively set the entire cache path with the `full_path` parameter.

You can only pass `path` or `full_path` - but not both.

`cache_delete` only accepts 1 file name, while `cache_delete_all` doesn't accept any names, but deletes all files. For deleting many specific files, use `cache_delete` in a `lapply()` type call

See Also

[ftxt_cache](#), [cache_file_info\(\)](#)

Other caching-functions: [cache_file_info](#), [ftxt_cache](#)

Examples

```
## Not run:
cache_options_get()
cache_options_set(path = "foobar")
cache_options_get()

# set full path
path <- tempdir()
cache_options_set(full_path = path)
cache_options_get()

## End(Not run)
```

cache_file_info

Get information on possibly bad files in your cache

Description

Get information on possibly bad files in your cache

Usage

```
cache_file_info()
```

Details

This function only identifies possibly bad files. You have to remove/delete them yourself. See example for how to do so. You can also open up your cache folder and delete them that way as well.

Value

list, with three elements:

- `xml_not_valid`: xml files that could not be read in with [xml2::read_xml\(\)](#)
- `xml_abstract_only`: xml files that only have abstracts. you can of choose to retain these if you like
- `pdf_not_valid`: pdf files that could not be read in with [pdftools::pdf_info\(\)](#)

See Also

Other caching-functions: [cache](#), [ftxt_cache](#)

Examples

```
# identify likely bad files
res <- cache_file_info()

# you can remove them yourself, e.g.,
# invisible(lapply(res$xml_abstract_only, unlink))
```

ftxt_cache

Inspect and manage cached files

Description

Inspect and manage cached files

Useful user functions for managing cached files

- `ftxt_cache$list()` returns a character vector of full path file names
- `ftxt_cache$files()` returns file objects with metadata
- `ftxt_cache$details()` returns files with details
- `ftxt_cache$delete()` delete specific files
- `ftxt_cache$delete_all()` delete all files, returns nothing

See Also

[cache](#), [cache_file_info\(\)](#)

Other caching-functions: [cache_file_info](#), [cache](#)

Examples

```
## Not run:
ftxt_cache

# list files in cache
ftxt_cache$list()

# list details of files in cache
ftxt_cache$details()

# delete certain database files
# ftxt_cache$delete("file path")
# ftxt_cache$list()

# delete all files in cache
# ftxt_cache$delete_all()
# ftxt_cache$list()

## End(Not run)
```

ft_abstract	<i>Get abstracts</i>
-------------	----------------------

Description

Get abstracts

Usage

```
ft_abstract(x, from = "plos", plosopts = list(), scopusopts = list(),  
           maopts = list(), crossrefopts = list(), ...)
```

```
ft_abstract_ls()
```

Arguments

x	(character) DOIs as a character vector
from	Source to query. One or more of plos (default), scopus, microsoft, or crossref
plosopts	PLOS options, a named list.
scopusopts	Scopus options, a named list.
maopts	Microsoft Academic options, a named list.
crossrefopts	Crossref options, a named list.
...	curl options passed on to <code>httr::GET()</code> or crul::HttpClient

Details

See **Rate Limits** and **Authentication** in [fulltext-package](#) for rate limiting and authentication information, respectively

Value

An object of class `ft_abstract`

Examples

```
# List publishers included  
ft_abstract_ls()  
  
## Not run:  
# PLOS  
## search  
(res <- ft_search(query = 'biology', from = 'plos', limit = 25,  
                 plosopts = list(fq = list('doc_type:full', '-article_type:correction',  
                                           '-article_type:viewpoints'))))  
## get abstracts  
dois <- res$plos$data$id  
(out <- ft_abstract(x = dois, from = "plos"))
```

```

out$plos

# Scopus
opts <- list(key = Sys.getenv('ELSEVIER_SCOPUS_KEY'))

## search
(res <- ft_search(query = 'biology', from = 'scopus', scopusopts = opts,
  limit = 25))
## get abstract
dois <- na.omit(res$scopus$data$`prism:doi`)
out <- ft_abstract(x = dois[1], from = "scopus", scopusopts = opts)
out
out$scopus

(out <- ft_abstract(x = dois[1:5], from = "scopus", scopusopts = opts))

# use scopus Ids
(res <- ft_search(query = 'biology', from = 'scopus', scopusopts = opts,
  limit = 50))
ids <- fulltext::strextract(res$scopus$data$`dc:identifier`, "[0-9]+")
(out <- ft_abstract(x = ids[1:4], from = 'scopus',
  scopusopts = list(
    key = Sys.getenv('ELSEVIER_SCOPUS_KEY'),
    id_type = "scopus_id"
  )
))

# Microsoft
key <- Sys.getenv("MICROSOFT_ACADEMIC_KEY")
(res <- ft_search("Y=[2010, 2012]", from = "microsoft",
  maopts = list(key = key)))
ids <- res$ma$data$Id
(out <- ft_abstract(x = ids, from = "microsoft",
  maopts = list(
    key = Sys.getenv('MICROSOFT_ACADEMIC_KEY')
  )
))
out$ma
cat(unlist(lapply(out$ma, "[[", "abstract")), sep = "\n\n")

# Crossref
(res <- ft_search("ecology", from = "crossref",
  crossrefopts = list(filter = c(has_abstract = TRUE))))
ids <- res$crossref$data$doi
(out <- ft_abstract(x = ids, from = "crossref"))
out$crossref

## End(Not run)

```

Description

Browse an article in your default browser

Usage

```
ft_browse(x, what = "macrodocs", browse = TRUE)
```

Arguments

x An object of class ft_data - the output from a call to [ft_get\(\)](#)
 what (character) One of macrodocs (default) or publisher
 browse (logical) Whether to browse (default) or not. If FALSE, return the url.

Examples

```
## Not run:
x <- ft_get('10.7554/eLife.04300', from='elife')
ft_browse(x)
ft_browse(x, browse=FALSE)

ft_browse( ft_get('10.3389/fphar.2014.00109', from="entrez") )

# open to publisher site
ft_browse(x, "publisher")

## End(Not run)
```

ft_collect	<i>Collect article text from local files</i>
------------	--

Description

ft_collect grabs full text data from file paths in your ft_data object (result of call to ft_get()). ft_text is a convenience function to grab the nested text data and bring it up in the list for easier access

Usage

```
ft_collect(x, ...)

ft_text(x, ...)

## Default S3 method:
ft_text(x, ...)

## S3 method for class 'ft_data'
ft_text(x, ...)
```

Arguments

x Input. An object of class ft_data
 ... Further args, ignored.

Details

The result of this call is actual text you can read

Value

an object of class ft_data, but the data slot should have character string of text from the XML/plain text/PDF file

Examples

```
## Not run:
# Get some data
x <- ft_get('10.1371/journal.pone.0086169', from='plos')

# note that the data is not in the object, gives NULL
x$plos$data$data

# Collect data from the .xml file
y <- x %>% ft_collect()

# note how the data is now in the object
y$plos$data$data

# Let's get the actual
## ft_collect() alone, replaces file pointers with parsed text,
## maintaining object structure
x %>% ft_collect()
## pulls the text out of the object
x %>% ft_collect() %>% ft_text()

## End(Not run)
```

ft_extract

Extract text from a single pdf document

Description

ft_extract attempts to make it easy to extract text from PDFs, using **pdftools**. Inputs can be either paths to PDF files, or the output of `ft_get()` (class ft_data).

Usage

```
ft_extract(x)
```

Arguments

x Path to a pdf file, or an object of class ft_data, the output from `ft_get()`

Value

An object of class pdf_t_char in the case of character input, or of class ft_data in the case of ft_data input

Examples

```
## Not run:
path <- system.file("examples", "example1.pdf", package = "fulltext")
(res <- ft_extract(path))

# use on output of ft_get() to extract pdf to text
## arxiv
res <- ft_get('cond-mat/9309029', from = "arxiv")
res2 <- ft_extract(res)
res$arxiv$data
res2$arxiv$data

## biorxiv
res <- ft_get('10.1101/012476')
res2 <- ft_extract(res)
res$biorxiv$data
res2$biorxiv$data

## End(Not run)
```

ft_get

Download full text articles

Description

ft_get is a one stop shop to fetch full text of articles, either XML or PDFs. We have specific support for PLOS via the **rplos** package, Entrez via the **rentrez** package, and arXiv via the **arXiv** package. For other publishers, we have helpers to ft_get to sort out links for full text based on user input. Articles are saved on disk. See Details for help on how to use this function.

Usage

```
ft_get(x, from = NULL, type = "xml", try_unknown = TRUE,
       plosopts = list(), bmcopts = list(), entrezopts = list(),
       elifeopts = list(), elsevieropts = list(), wileyopts = list(),
       crossrefopts = list(), progress = FALSE, ...)
```

```
ft_get_ls()
```

Arguments

x	Either identifiers for papers, either DOIs (or other ids) as a list of character strings, or a character vector, OR an object of class <code>ft</code> , as returned from ft_search()
from	Source to query. Optional.
type	(character) one of <code>xml</code> (default), <code>pdf</code> , or <code>plain</code> (Elsevier only). We choose to go with <code>xml</code> as the default as it has structure that a machine can reason about, but you are of course free to try to get <code>xml</code> , <code>pdf</code> , or <code>plain</code> (in the case of Elsevier).
try_unknown	(logical) if publisher plugin not already known, we try to fetch full text link either from <code>ftdoi.org</code> or from Crossref. If not found at <code>ftdoi.org</code> or at Crossref we skip with a warning. If found with <code>ftdoi.org</code> or Crossref we attempt to download. Only applicable in <code>character</code> and <code>list</code> S3 methods. Default: <code>TRUE</code>
plosopts	PLOS options, a named list. See rplos::plos_fulltext()
bmcopts	BMC options. parameter DEPRECATED
entrezopts	Entrez options, a named list. See rentrez::entrez_search() and entrez_fetch()
elifeoports	eLife options, a named list.
elsevieropts	Elsevier options, a named list.
wileyopts	Wiley options, a named list.
crossrefopts	Crossref options, a named list.
progress	(logical) whether to show progress bar or not. default: <code>FALSE</code> . if <code>TRUE</code> , we use utils::txtProgressBar() and utils::setTxtProgressBar to create the progress bar; and each progress bar connection is closed on function exit. A progress bar is run for each data source. Works for all S3 methods except <code>ft_get.links</code> . When articles are not already downloaded you see the progress bar. If articles are already downloaded/cached, normally we throw messages saying so, but if a progress bar is requested, then the messages are suppressed to not interrupt the progress bar.
...	Further args passed on to crul::HttpClient

Details

There are various ways to use `ft_get`:

- Pass in only DOIs - leave `from` parameter `NULL`. This route will first query Crossref API for the publisher of the DOI, then we'll use the appropriate method to fetch full text from the publisher. If a publisher is not found for the DOI, then we'll throw back a message telling you a publisher was not found.
- Pass in DOIs (or other pub IDs) and use the `from` parameter. This route means we don't have to make an extra API call to Crossref (thus, this route is faster) to determine the publisher for each DOI. We go straight to getting full text based on the publisher.
- Use [ft_search\(\)](#) to search for articles. Then pass that output to this function, which will use info in that object. This behaves the same as the previous option in that each DOI has publisher info so we know how to get full text for each DOI.

Note that some publishers are available via Entrez, but often not recent articles, where "recent" may be a few months to a year or so. In that case, make sure to specify the publisher, or else you'll get back no data.

Value

An object of class `ft_data` (of type `S3`) with slots for each of the publishers. The returned object is split up by publishers because the full text format is the same within publisher - which should facilitate text mining downstream as different steps may be needed for each publisher's content.

Note that we have a print method for `ft_data` so you see something like this:

```
<fulltext text>
[Docs] 4
[Source] ext - /Users/foobar/Library/Caches/R/fulltext
[IDs] 10.2307/1592482 10.2307/1119209 10.1037/11755-024 ...
```

Within each publisher there is a list with the elements:

- found: number of full text articles found
- dois: the DOIs given and searched for
- data
 - backend: the backend. right now only ext for "by file extension", we may add other backends in the future, thus we retain this
 - cache_path: the base directory path for file caching
 - path: if file retrieved the full path to the file. if file not retrieved this is NULL
 - data: if text extracted (see `ft_collect()`) the text will be here, but until then this is NULL
- opts: the options given like article type, dois
- errors: data.frame of errors, with two columns for article id and error

Important Access Notes

See **Rate Limits** and **Authentication** in [fulltext-package](#) for rate limiting and authentication information, respectively.

In particular, take note that when fetching full text from Wiley and Elsevier, the only way that's done (unless it's one of their OA papers) is through the Crossref TDM flow in which you need a Crossref TDM API key and your institution needs to have access to the exact journal you are trying to fetch a paper from. If your institution doesn't have access you may still get a result, but likely its only the abstract. See **Authentication** in [fulltext-package](#) for details.

Notes on the type parameter

Type is sometimes ignored, sometimes used. For certain data sources, they only accept one type. By data source/publisher:

- PLOS: pdf and xml
- Entrez: only xml
- eLife: pdf and xml
- Pensoft: pdf and xml
- arXiv: only pdf

- BiorXiv: only pdf
- Elsevier: xml and plain
- Wiley: only pdf
- Peerj: pdf and xml
- Informa: only pdf
- FrontiersIn: pdf and xml
- Copernicus: pdf and xml
- Scientific Societies: only pdf
- Crossref: depends on the publisher
- other data sources/publishers: there are too many to cover here - will try to make a helper in the future for what is covered by different publishers

How data is stored

ft_get used to have many options for "backends". We have simplified this to one option. That one option is that all full text files are written to disk on your machine. You can choose where these files are stored.

In addition, files are named by their IDs (usually DOIs), and the file extension for the full text type (pdf or xml usually). This makes inspecting the files easy.

Data formats

xml full text is stored in .xml files. pdf is stored in .pdf files. And plain text is stored in .txt files.

Reusing cached articles

All files are written to disk and we check for a file matching the given DOI/ID on each request - if found we use it and throw message saying so.

Caching

Previously, you could set caching options in each ft_get function call. We've simplified this to only setting caching options through the function `cache_options_set()` - and you can get your cache options using `cache_options_get()`. See those docs for help on caching.

Notes on specific publishers

- arXiv: The IDs passed are not actually DOIs, though they look similar. Thus, there's no way to not pass in the from parameter as we can't determine unambiguously that the IDs passed in are from arXiv.org.
- bmc: Is a hot mess since the Springer acquisition. It's been removed as an officially supported plugin, some DOIs from them may still work when passed in here, who knows, it's a mess.

Warnings

You will see warnings thrown in the R shell or in the resulting object. See [ft_get-warnings](#) for more information on what warnings mean.

See Also

[as.ft_data\(\)](#)

Examples

```
# List publishers included
ft_get_ls()

## Not run:
# If you just have DOIs and don't know the publisher
## PLOS
ft_get('10.1371/journal.pone.0086169')

# Collect all errors from across papers
# similarly can combine from different publishers as well
res <- ft_get(c('10.7554/eLife.03032', '10.7554/eLife.aaaa'), from = "elife")
res$elife$errors

## PeerJ
ft_get('10.7717/peerj.228')
ft_get('10.7717/peerj.228', type = "pdf")

## eLife
### xml
ft_get('10.7554/eLife.03032')
res <- ft_get(c('10.7554/eLife.03032', '10.7554/eLife.32763'), from = "elife")
res$elife
respdf <- ft_get(c('10.7554/eLife.03032', '10.7554/eLife.32763'),
  from = "elife", type = "pdf")
respdf$elife

elife_xml <- ft_get('10.7554/eLife.03032', from = "elife")
library(magrittr)
elife_xml %<>% ft_collect()
elife_xml$elife
### pdf
elife_pdf <- ft_get(c('10.7554/eLife.03032', '10.7554/eLife.32763'),
  from = "elife", type = "pdf")
elife_pdf$elife
elife_pdf %<>% ft_collect()
elife_pdf %>% ft_extract()

## some BMC DOIs will work, but some may not, who knows
ft_get(c('10.1186/2049-2618-2-7', '10.1186/2193-1801-3-7'), from = "entrez")

## FrontiersIn
res <- ft_get(c('10.3389/fphar.2014.00109', '10.3389/feart.2015.00009'))
res
res$frontiersin

## Hindawi - via Entrez
res <- ft_get(c('10.1155/2014/292109', '10.1155/2014/162024', '10.1155/2014/249309'))
```

```

res
res$hindawi
res$hindawi$data$path
res$hindawi$data$data
res %>% ft_collect() %>% .$hindawi

## F1000Research - via Entrez
x <- ft_get('10.12688/f1000research.6522.1')
## Two different publishers via Entrez - retains publisher names
res <- ft_get(c('10.1155/2014/292109', '10.12688/f1000research.6522.1'))
res$hindawi
res$f1000research

## Pensoft
ft_get('10.3897/mycokeys.22.12528')

## Copernicus
out <- ft_get(c('10.5194/angeo-31-2157-2013', '10.5194/bg-12-4577-2015'))
out$copernicus

## arXiv - only pdf, you have to pass in the from parameter
res <- ft_get(x='cond-mat/9309029', from = "arxiv")
res$arxiv
res %>% ft_extract %>% .$arxiv

## bioRxiv - only pdf
res <- ft_get(x='10.1101/012476')
res$biorxiv

## Karger Publisher
(x <- ft_get('10.1159/000369331'))
x$karger

## MDPI Publisher
(x <- ft_get('10.3390/nu3010063'))
x$mdpi
ft_get('10.3390/nu7085279')
ft_get(c('10.3390/nu3010063', '10.3390/nu7085279')) # not working, only getting 1

# Scientific Societies
## this is a paywall article, you may not have access or you may
x <- ft_get("10.1094/PHYTO-04-17-0144-R")
x$scientificsocieties

# Informa
x <- ft_get("10.1080/03088839.2014.926032")
ft_get("10.1080/03088839.2013.863435")

## CogentOA - part of Inform/Taylor Francis now
ft_get('10.1080/23311916.2014.938430')

library(rplos)
(dois <- searchplos(q="*:*", fl='id',

```

```

    fq=list('doc_type:full',"article_type:\research article\"), limit=5)$data$id)
ft_get(dois, from='plos')
ft_get(c('10.7717/peerj.228','10.7717/peerj.234'), from='entrez')

# elife
ft_get('10.7554/eLife.04300', from='elife')
ft_get(c('10.7554/eLife.04300', '10.7554/eLife.03032'), from='elife')
## search for elife papers via Entrez
dois <- ft_search("elife[journal]", from = "entrez")
ft_get(dois)

# Frontiers in Pharmacology (publisher: Frontiers)
doi <- '10.3389/fphar.2014.00109'
ft_get(doi, from="entrez")

# Hindawi Journals
ft_get(c('10.1155/2014/292109','10.1155/2014/162024','10.1155/2014/249309'), from='entrez')
res <- ft_search(query='ecology', from='crossref', limit=50,
                crossrefopts = list(filter=list(has_full_text = TRUE,
                                                member=98,
                                                type='journal-article'))))

out <- ft_get(res$crossref$data$DOI[1:20], from='entrez')

# Frontiers Publisher - Frontiers in Aging Nueroscience
res <- ft_get("10.3389/fnagi.2014.00130", from='entrez')
res$entrez

# Search entrez, get some DOIs
(res <- ft_search(query='ecology', from='entrez'))
res$entrez$data$doi
ft_get(res$entrez$data$doi[1], from='entrez')
ft_get(res$entrez$data$doi[1:3], from='entrez')

# Search entrez, and pass to ft_get()
(res <- ft_search(query='ecology', from='entrez'))
ft_get(res)

# elsevier, ugh
## set an environment variable like Sys.setenv(CROSSREF_TDM = "your key")
ft_get(x = "10.1016/j.trac.2016.01.027", from = "elsevier")

# wiley, ugh
## Wiley has only PDF, so type parameter doesn't do anything
ft_get(x = "10.1006/asle.2001.0035", from = "wiley")

# IEEE, ugh
ft_get('10.1109/TCSVT.2012.2221191', type = "pdf")

# AIP Publishing
ft_get('10.1063/1.4967823', try_unknown = TRUE)

# PNAS

```

```

ft_get('10.1073/pnas.1708584115', try_unknown = TRUE)

# American Society for Microbiology
ft_get('10.1128/cvi.00178-17')

# American Society of Clinical Oncology
ft_get('10.1200/JCO.18.00454')

# American Institute of Physics
ft_get('10.1063/1.4895527')

# American Chemical Society
ft_get(c('10.1021/la903074z', '10.1021/jp048806z'))

# From ft_links output
## Crossref
(res2 <- ft_search(query = 'ecology', from = 'crossref', limit = 3))
(out <- ft_links(res2))
(ress <- ft_get(x = out, type = "pdf"))
ress$crossref

(x <- ft_links("10.1111/2041-210X.12656", "crossref"))
(y <- ft_get(x))

## PLOS
(res2 <- ft_search(query = 'ecology', from = 'plos', limit = 4))
(out <- ft_links(res2))
out$plos
(ress <- ft_get(x = out, type = "pdf"))
ress$plos
ress$plos$dois
ress$plos$data
ress$plos$data$path$`10.1371/journal.pone.0059813`

## No publisher plugin provided yet
ft_get('10.1037/10740-005')
### no link available for this DOI
res <- ft_get('10.1037/10740-005', try_unknown = TRUE)
res$crossref

# Get a progress bar - off by default
library(rplos)
(dois <- searchplos(q="*:*", fl='id',
  fq=list('doc_type:full',"article_type:\`research article\`"), limit=5)$data$id)
## when articles not already downloaded you see the progress bar
b <- ft_get(dois, from='plos', progress = TRUE)
## if articles already downloaded/cached, normally we through messages saying so
b <- ft_get(dois, from='plos', progress = FALSE)
## but if a progress bar is requested, then the messages are suppressed
b <- ft_get(dois, from='plos', progress = TRUE)

## End(Not run)

```

`ft_get-warnings`*fulltext warnings details*

Description

What can you do about the various warnings?

Details

This document is in relation to the function `ft_get()`

No plugin

For the warning "no plugin for Crossref ...", this is what happened internally:

This happens when we don't have a hard coded plugin for that specific publisher within this package (use `ft_get_ls()` to see what hard coded publisher plugins we have), but we do have generic functions for Crossref and fdoi.org that are also tried and may get a result. You are welcome to open up an issue at <https://github.com/ropensci/fulltext/issues> to discuss publisher specific plugins.

Access or an error

For the warning "you may not have access to ... or an error occurred" we've likely tried to get the full text but either an error occurred (which can be a lot of things), or you don't have access to the full text.

If you think the problem may be that you don't have access, check whether you are on an IP address that has access to the full text, and if you're not, get on one that does - most likely by being on campus/etc. or through a VPN.

Part of an article

For the warning "... was not found or may be a DOI for a part of an article" this happens for certain publishers (e.g., PLOS) that issue DOIs for parts of articles (e.g., abstract, body, supplements) - in which case it doesn't make sense to get full text of, for example, supplements.

No Crossref link

For the warning "no link found from Crossref", this happens when we've gone through the route of searching for a full text URL from the Crossref API, and there wasn't one, so we stop searching and give that warning.

ft_links	<i>Get full text links</i>
----------	----------------------------

Description

Get full text links

Usage

```
ft_links(x, from = NULL, plosopts = list(), crossrefopts = list(),
        entrezopts = list(), bmcopts = list(), ...)
```

```
ft_links_ls()
```

Arguments

x	One of ft, ft_ind, or a character string of DOIs.
from	Source to query. Ignored when ft_ind class passed.
plosopts	PLOS options, a named list. See ?searchplos
crossrefopts	Crossref options, a named list. See ?cr_works
entrezopts	Entrez options, a named list. See ?entrez_search
bmcopts	BMC options, a named list. See ?bmc_search
...	ignored right now

Details

Inputs can be an object of class ft, ft_ind, or a character string of DOIs. You can specify a specific source for four sources (PLOS, BMC, Crossref, and Entrez), but any other publishers we guess the publisher from the input DOI(s), then attempt to generate full text links based on the publisher (if found). Of course, guessing the publisher makes things slower as it requires an HTTP request.

Strategy varies by publisher. For some we can construct XML and PDF links only from the DOI. For others, we need to make an HTTP request to the publisher to get additional information - this of course makes things slower.

See **Rate Limits** and **Authentication** in [fulltext-package](#) for rate limiting and authentication information, respectively

Value

An object of class ft_links, with either a list or data.frame for each DOI, with links for XML and PDF links (typically).

Examples

```

# List publishers included
ft_links_ls()

## Not run:
# Entrez
(res1 <- ft_search(query='ecology', from='entrez'))
res1$entrez$data$doi
## directly from ft_search output
(out <- ft_links(res1))
out$entrez
out$entrez$data[[1]]
## directly individual elements of ft_search output
(out <- ft_links(res1$entrez))
out$entrez
## from character vector of DOIs
x <- c("10.1371/journal.pone.0086169", "10.1016/j.ympcv.2010.07.013")
(out2 <- ft_links(x, from = "entrez"))
out2$entrez

# Crossref
(res2 <- ft_search(query='ecology', from='crossref'))
res2$crossref$data$doi
## directly from ft_search output
(out <- ft_links(res2))
out$crossref
out$crossref$data[[1]]
## directly individual elements of ft_search output
(out <- ft_links(res2$crossref))
out$crossref
## from character vector of DOIs
x <- c("10.1016/S1754-5048(14)00139-1",
      "10.1016/B978-0-12-378260-1.50017-8")
(out2 <- ft_links(x, from = "crossref"))
out2$crossref

# PLOS
(res3 <- ft_search(query='ecology', from='plos', plosopts=list(
  fl=c('id','author','eissn','journal','counter_total_all',
      'alm_twitterCount'))))
res3$plos$data$id
## directly from ft_search output
(out <- ft_links(res3))
out$plos
out$plos$data[[1]]
## directly individual elements of ft_search output
(out <- ft_links(res3$plos))
out$plos
## from character vector of DOIs
x <- c("10.1371/journal.pone.0017342", "10.1371/journal.pone.0091497")
out3 <- ft_links(x, from = "plos")
out3$plos

```

```
# BMC
(res <- ft_search(query='ecology', from='bmc'))
res$bmc
## directly from ft_search output
(out <- ft_links(res))
out$bmc
out$bmc$data[[1]]
## directly individual elements of ft_search output
(out <- ft_links(res$bmc))
out$bmc

# Character input
out4 <- ft_links('10.1371/journal.pone.0086169')
out4$plos

# other publishers
## elife
res <- ft_links(c('10.7554/eLife.03032', '10.7554/eLife.02747'))
res$elife

## peerj
ft_links('10.7717/peerj.228')
ft_links(c('10.7717/peerj.228', '10.7717/peerj.1200'))

## wiley
res <- ft_links('10.1006/asle.2001.0035', from = "crossref")
res$crossref$data[[1]]$url

## informa
res <- ft_links('10.1174/02134749660569378', from = "crossref")
res$crossref$data[[1]]$url

## frontiersin
(res <- ft_links('10.3389/fphar.2014.00109'))
res$frontiersin

## copernicus
(res <- ft_links('10.5194/angeo-31-2157-2013'))
res$copernicus

## cogent
(res <- ft_links('10.1080/23311916.2014.938430'))
res$cogent

## bmc
(res <- ft_links('10.1186/2049-2618-2-7'))
res$bmc
(res <- ft_links('10.1186/2049-2618-2-7', from = "bmc"))

## Many publishers, elife and peerj
res <- ft_links(c('10.7554/eLife.03032', '10.7717/peerj.228'))
res$elife
```



```
res$peerj
## End(Not run)
```

ft_providers	<i>Search for information on journals or publishers.</i>
--------------	--

Description

Search for information on journals or publishers.

Usage

```
ft_providers(journal = NULL, publisher = NULL, limit = 10, ...)
```

Arguments

journal	Query terms
publisher	Source to query
limit	Number of records to return.
...	Further args passed on to crul::HttpClient

Value

An object of class ft_p

Examples

```
## Not run:
ft_providers(journal="Stem Cells International")
ft_providers(publisher="hindawi")
ft_providers(publisher="journal")

## End(Not run)
```

ft_search

*Search for full text***Description**

ft_search is a one stop shop for searching for articles across many publishers and repositories. We currently support search for PLOS via the **rplos** package, Crossref via the **rcrossref** package, Entrez via the **rentrez** package, arXiv via the **aRxiv** package, and BMC, Biorxiv, EuropePMC, and Scopus via internal helper functions in this package.

Many publishers' content is searchable via Crossref and Entrez - of course this doesn't mean that we can get full text for those articles. In the output objects of this function, we attempt to help by indicating what license is used for articles.

Usage

```
ft_search(query, from = "plos", limit = 10, start = 0,
          plosopts = list(), bmcopts = list(), crossrefopts = list(),
          entrezopts = list(), arxivopts = list(), biorxivopts = list(),
          euroopts = list(), scopusopts = list(), maopts = list(), ...)
```

```
ft_search_ls()
```

Arguments

query	(character) Query terms
from	(character) Source to query, one or more of "plos", "bmc", "crossref", "entrez", "arxiv", "biorxiv", "europmc", "scopus", or "ma"
limit	(integer) Number of records to return. default: 10
start	(integer) Record number to start at. Only used for 'scopus' right now. default: 0. Note that with some data sources we loop internally to get all the results you want with the limit parameter, so start in those cases will be ignored. See Looping section below.
plosopts	(list) PLOS options, a named list. See rplos::searchplos()
bmcopts	(list) BMC options, a named list. See bmc_search()
crossrefopts	(list) Crossref options, a named list. See rcrossref::cr_works()
entrezopts	(list) Entrez options, a named list. See rentrez::entrez_search()
arxivopts	(list) arxiv options, a named list. See aRxiv::arxiv_search()
biorxivopts	(list) biorxiv options, a named list. See biorxiv_search()
euroopts	(list) Euro PMC options, a named list. See eupmc_search()
scopusopts	(list) Scopus options, a named list. See scopus_search()
maopts	(list) Microsoft Academic options, a named list. See microsoft_search()
...	ignored right now

Details

Each of plosopts, scopusopts, etc. expect a named list.

See **Rate Limits** and **Authentication** in [fulltext-package](#) for rate limiting and authentication information, respectively

See <https://dev.elsevier.com/tips/ScopusSearchTips.htm> for help/tips on searching with Scopus

Value

An object of class ft, and objects of class ft_ind within each source. You can access each data source with \$

Looping

Note that we necessarily have to treat different sources/publishers differently internally in this function. Some we can search and get back as many results as desired automatically, while with others you'd have to manually iterate through to get all your results. Notes on different sources:

- PLOS: `rplos::searchplos()` used and includes internal looping of requests
- BMC: using internal function `bmc_search` that does not loop, so you have to iterate through requests manually
- Crossref: `rcrossref::cr_works()` used, but does not include internal looping of requests, but the max limit for one request is relatively high at 1000
- Entrez: `rentrez::entrez_search()` used, but does not include internal looping of requests
- arXiv: `arxiv::arxiv_search()` used and includes internal looping of requests
- BiorXiv: using internal function `biorxiv_search` that does not loop, so you have to iterate through requests manually
- Europe BMC: using internal function `eupmc_search` that does not loop, so you have to iterate through requests manually
- Scopus: using internal function `scopus_search_loop` that does include internal looping of requests
- Microsoft AR: using internal function `microsoft_search` that does not loop, so you have to iterate through requests manually

Note

for all *opts parameters, ee the function linked to in the parameter definition for what you can pass to it.

Examples

```
# List publishers included
ft_search_ls()

## Not run:
# Plos
```

```

(res1 <- ft_search(query='ecology', from='plos'))
res1$plos
ft_search(query='climate change', from='plos', limit=500,
  plosopts=list(
    fl=c('id','author','eissn','journal','counter_total_all',
      'alm_twitterCount'))

# Crossref
(res2 <- ft_search(query='ecology', from='crossref'))
res2$crossref

# BioRxiv
(res <- ft_search(query='owls', from='biorxiv'))
res$biorxiv

# Entrez
(res <- ft_search(query='ecology', from='entrez'))
res$entrez

# arXiv
(res <- ft_search(query='ecology', from='arxiv'))
res$arxiv

# BMC - can be very slow
(res <- ft_search(query='ecology', from='bmc'))
res$bmc

# Europe PMC
(res <- ft_search(query='ecology', from='europmc'))
res$europmc
## get the next batch of results, using the cursorMark result
ft_search(query='ecology', from='europmc',
  europts = list(cursorMark = res$europmc$cursorMark))

# Scopus
(res <- ft_search(query = 'ecology', from = 'scopus', limit = 100,
  scopusopts = list(key = Sys.getenv('ELSEVIER_SCOPUS_KEY'))))
res$scopus
## pagination
(res <- ft_search(query = 'ecology', from = 'scopus',
  scopusopts = list(key = Sys.getenv('ELSEVIER_SCOPUS_KEY')), limit = 5))
## lots of results
(res <- ft_search(query = "ecology community elk cow", from = 'scopus',
  limit = 100,
  scopusopts = list(key = Sys.getenv('ELSEVIER_SCOPUS_KEY'))))
res$scopus
## facets
(res <- ft_search(query = 'ecology', from = 'scopus',
  scopusopts = list(
    key = Sys.getenv('ELSEVIER_SCOPUS_KEY'),
    facets = "subjarea(count=5)"
  ), limit = 5))
res$scopus

```

```

# PLOS, Crossref, and arxiv
(res <- ft_search(query='ecology', from=c('plos','crossref','arxiv')))
res$plos
res$arxiv
res$crossref

# Microsoft academic search
key <- Sys.getenv("MICROSOFT_ACADEMIC_KEY")
(res <- ft_search("Y='19'...", from = "microsoft", maopts = list(key = key)))
res$ma
res$ma$data$DOI

## End(Not run)

```

ft_serialize

Serialize raw text to other formats, including to disk

Description

ft_serialize helps you convert to various data formats. If your data is in unparsed XML (i.e., character class), you can convert to parsed XML. If in XML, you can convert to (ugly-ish) JSON, or a list.

Usage

```
ft_serialize(x, to = "xml", from = NULL, ...)
```

```
ft_get_keys(x)
```

Arguments

x	Input object, output from a call to ft_get. Required.
to	(character) Format to serialize to. One of list, xml, or json. Required. Output to xml returns object of class XMLInternalDocument.
from	(character) Format x is currently in. Function attempts to use metadata provided, or guess from data itself. Optional. CURRENTLY IGNORED.
...	Further args passed on to <code>xml2::read_xml()</code> or <code>jsonlite::toJSON()</code>

Value

An object of class ft_parsed

Examples

```
## Not run:
res <- ft_get('10.7717/peerj.228')

# if articles in xml format, parse the XML
(out <- ft_serialize(ft_collect(res), to='xml'))
out$peerj$data$data[[1]] # the xml

# From XML to JSON
(out <- ft_serialize(ft_collect(res), to='json'))
out$peerj$data$data$`10.7717/peerj.228` # the json
jsonlite::fromJSON(out$peerj$data$data$`10.7717/peerj.228`)

# To a list
out <- ft_serialize(ft_collect(res), to='list')
out$peerj$data$data
out$peerj$data$data[[1]]$body$sec$title

## End(Not run)
```

ft_table

*Collect metadata and text into a data.frame***Description**

Facilitates downstream processing with text mining packages by providing metadata and full text in a tidy data.frame format

Usage

```
ft_table(path = NULL, type = NULL, encoding = NULL,
         xml_extract_text = TRUE)
```

Arguments

path	a directory path, must exist
type	(character) type of files to get. Default is NULL which gets all types. Can be one of pdf, xml, or plain (file extensions: pdf, xml, and txt, respectively)
encoding	(character) encoding, if NULL we get it from getOption("encoding")
xml_extract_text	(logical) for XML, should we extract the text (TRUE) or return a string as XML (FALSE). Default: TRUE

Details

You can alternatively use `readtext::readtext()` or similar functions to achieve a similar outcome.

Examples

```
## Not run:
## from a directory path
x <- ft_table()
x

## only xml
ft_table(type = "xml")

## only pdf
ft_table(type = "pdf")

## don't pull text out of xml, just give back the xml please
x <- ft_table(xml_extract_text = FALSE)
x

## End(Not run)
```

fulltext-defunct *Defunct functions in fulltext*

Description

- [ft_extract_corpus](#): Function removed. As part of focusing scope of the package we're trying to limit dependencies, so downstream use of tm can still easily be done.
- [pdfx](#): Function removed. We're trying to focus the scope of the package - and this function is more out of scope now.
- [chunks](#): Function name changed to [ft_chunks\(\)](#)
- [tabularize](#): Function name changed to [ft_tabularize\(\)](#)
- [collect](#): Function name changed to [ft_collect\(\)](#)
- [get_text](#): Function name changed to [ft_text\(\)](#)
- [cache_clear](#) was never working anyway, and is now removed
- [ft_browse_sections](#): no sign that function used, and allows to remove a dependency
- [ft_get_si](#): moved to package `suppdata`
- [ft_chunks](#): moved to package `pubchunks`
- [ft_tabularize](#): moved to package `pubchunks`

Index

*Topic **package**

fulltext-package, 2

aRxiv::arxiv_search(), 26, 27

as.ft_data, 5

as.ft_data(), 17

biorxiv_search(), 26

bmc_search(), 26

cache, 6, 7, 8

cache_file_info, 7, 7, 8

cache_file_info(), 7, 8

cache_options_get(cache), 6

cache_options_get(), 16

cache_options_set(cache), 6

cache_options_set(), 16

chunks, 31

collect, 31

curl::HttpClient, 9, 14, 25

entrez_fetch(), 14

eupmc_search(), 26

ft_abstract, 9

ft_abstract(), 3

ft_abstract_ls(ft_abstract), 9

ft_browse, 10

ft_browse_sections, 31

ft_chunks, 31

ft_chunks(), 31

ft_collect, 11

ft_collect(), 15, 31

ft_extract, 12

ft_extract(), 3

ft_extract_corpus, 31

ft_get, 13

ft_get(), 2-5, 11-13, 21

ft_get-warnings, 16, 21

ft_get_keys(ft_serialize), 29

ft_get_ls(ft_get), 13

ft_get_si, 31

ft_links, 22

ft_links(), 3

ft_links_ls(ft_links), 22

ft_providers, 25

ft_search, 26

ft_search(), 2-4, 14

ft_search_ls(ft_search), 26

ft_serialize, 29

ft_serialize(), 3

ft_table, 30

ft_tabularize, 31

ft_tabularize(), 31

ft_text(ft_collect), 11

ft_text(), 31

ftxt_cache, 7, 8

fulltext(fulltext-package), 2

fulltext-defunct, 31

fulltext-package, 2, 9, 15, 22, 27

get_text, 31

jsonlite::toJSON(), 29

lapply(), 6

microsoft_search(), 26

pdftools::pdf_info(), 7

pdfx, 31

rcrossref::cr_works(), 26, 27

rentrez::entrez_search(), 14, 26, 27

rplos::plos_fulltext(), 14

rplos::searchplos(), 26, 27

scopus_search(), 26

Startup, 4

tabularize, 31

utils::setTxtProgressBar, 14

`utils::txtProgressBar()`, [14](#)

`xml2::read_xml()`, [7](#), [29](#)