Package 'fulltext'

October 19, 2018

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 1.1.0

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 (>= 0.4.0), httr (>= 1.3.1), 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, pdftools, storr, tibble, digest

Suggests roxygen2 (>= 6.1.0), testthat, knitr

RoxygenNote 6.1.0

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>), Will Pearse [ctb]

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2018-10-19 18:10:03 UTC

R topics documented:

`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) - `ft_chunks()`
- grab supplementary materials for (re-)analysis of data - `ft_get_si()` accepts article identifiers, and output from `ft_search()` and `ft_get()`

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 man 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 `.Renviron` file as `SPRINGER_KEY`, and we'll read it in for you, and you don't have to pass in anything.

Scopus: Scopus requires an API key to search their service. 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 .Renviron, and we'll read it in for you. See Startup for help.

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 .Renviron 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 .Renviron like `crossref_email = name@example.com`

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 .Renviron 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/elife.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

# collect chunks
# x %>% ft_collect %>% ft_chunks(c("doi", "title"))
```

---

cache *Set or get cache options*

Description

Set or get cache options

Usage

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

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

Managing cached files

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

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()

## 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` *Get abstracts*

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
`scopusopts`	Scopus options
`maopts`	Microsoft Academic options
`crossrefopts`	Crossref options
`...`	curl options passed on to `httr::GET()` 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)
```

`ft_browse` *Browse an article in your default browser*

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_chunks *Extract chunks of data from articles*

Description

ft_chunks makes it easy to extract sections of an article. You can extract just authors across all articles, or all references sections, or the complete text of each article. Then you can pass the output downstream for visualization and analysis.

Usage

```
ft_chunks(x, what = "all")

ft_tabularize(x)
```

Arguments

x	An object of class ft_data, the output from a call to ft_get()
what	What to get, can be one or more in a vector or list. See Details.

Details

Options for the what parameter:

- front - Publisher, journal and article metadata elements
- body - Body of the article
- back - Back of the article, acknowledgments, author contributions, references
- title - Article title
- doi - Article DOI
- categories - Publisher's categories, if any
- authors - Authors
- keywords - Keywords
- abstract - Article abstract
- executive_summary - Article executive summary
- refs - References
- refs_dois - References DOIs - if available
- publisher - Publisher name
- journal_meta - Journal metadata
- article_meta - Article metadata
- acknowledgments - Acknowledgments
- permissions - Article permissions
- history - Dates, recieved, published, accepted, etc.

Note that we currently only support PLOS, eLife, Entrez, and Elsevier right now; more to come.

Value

A list of output, one for each thing requested

Examples

```
## Not run:
x <- ft_get('10.1371/journal.pone.0086169', from='plos')
x %>% ft_collect %>% ft_chunks(what="authors")

library("rplos")
(dois <- searchplos(q="*:*", fl='id',
   fq=list('doc_type:full',"article_type:\"research article\""),
     limit=5)$data$id)
x <- ft_collect(ft_get(dois, from="plos"))
x %>% ft_chunks("front")
x %>% ft_chunks("body")
x %>% ft_chunks("back")
x %>% ft_chunks("history")
x %>% ft_chunks(c("doi","history")) %>% ft_tabularize()
x %>% ft_chunks("authors")
x %>% ft_chunks(c("doi","categories"))
x %>% ft_chunks("all")
x %>% ft_chunks("publisher")
x %>% ft_chunks("acknowledgments")
x %>% ft_chunks("permissions")
x %>% ft_chunks("journal_meta")
x %>% ft_chunks("article_meta")

# Coerce list output to a data.frame, where possible
dois <- c('10.7554/elife.28589', '10.7554/elife.14009', '10.7554/elife.13941',
  '10.7554/elife.22170', '10.7554/elife.29285')
x <- ft_get(dois)
x <- x %>% ft_collect()
x$elife
x %>% ft_chunks("publisher") %>% ft_tabularize()
x %>% ft_chunks("refs") %>% ft_tabularize()
x %>% ft_chunks(c("doi","publisher")) %>% ft_tabularize()
x %>% ft_chunks(c("doi","publisher","permissions")) %>% ft_tabularize()

x <- ft_get(c("10.3389/fnagi.2014.00130",'10.1155/2014/249309',
  '10.1155/2014/162024'), from='entrez')
x <- x %>% ft_collect()
x %>% ft_chunks("doi") %>% ft_tabularize()
x %>% ft_chunks("authors") %>% ft_tabularize()
x %>% ft_chunks(c("doi","publisher","permissions")) %>% ft_tabularize()
x %>% ft_chunks("history") %>% ft_tabularize()

x <- ft_get('10.3389/fnagi.2014.00130', from='entrez')
x <- x %>% ft_collect()
x %>% ft_chunks("keywords")

# Piping workflow
```

```
opts <- list(fq=list('doc_type:full',"article_type:\"research article\""))
ft_search(query='ecology', from='plos', plosopts = opts)$plos$data$id %>%
 ft_get(from = "plos") %>%
 ft_chunks("publisher")

# Via entrez
res <- ft_get(c("10.3389/fnagi.2014.00130",'10.1155/2014/249309',
   '10.1155/2014/162024'), from='entrez')
res <- res %>% ft_collect()
ft_chunks(res, what="abstract")
ft_chunks(res, what="title")
ft_chunks(res, what="keywords")
ft_chunks(res, what="publisher")

(res <- ft_search(query='ecology', from='entrez'))
ft_get(res$entrez$data$doi, from='entrez') %>% ft_collect() %>% ft_chunks("title")
ft_get(res$entrez$data$doi[1:4], from='entrez') %>%
 ft_collect() %>%
 ft_chunks("acknowledgments")
ft_get(res$entrez$data$doi[1:4], from='entrez') %>%
 ft_collect() %>%
 ft_chunks(c('title','keywords'))

# From eLife
x <- ft_get(c('10.7554/eLife.04251', '10.7554/eLife.04986'), from='elife')
x %>% ft_chunks("abstract")
x %>% ft_chunks("publisher")
x %>% ft_chunks("journal_meta")
x %>% ft_chunks("acknowledgments")
x %>% ft_chunks("refs_dois")
x %>% ft_chunks(c("abstract", "executive_summary"))

## End(Not run)
```

`ft_collect` *Collect article text from local files*

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 attemps 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 pdft_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(), ...)

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 `ft`, as returned from `ft_search()`
from	Source to query. Optional.
type	(character) one of xml (default), pdf, or plain (Elsevier only). We choose to go with xml as the default as it has structure that a machine can reason about, but you are of course free to try to get xml, pdf, or plain (in the case of Elsevier).
try_unknown	(logical) if publisher plugin not already known, we try to fetch full text link either from ftdoi.org or from Crossref. If not found at ftdoi.org or at Crossref we skip with a warning. If found with ftdoi.org or Crossref we attempt to download. Only applicable in `character` and `list` S3 methods. Default: `TRUE`
plosopts	PLOS options. See `rplos::plos_fulltext()`
bmcopts	BMC options. parameter DEPRECATED
entrezopts	Entrez options. See `rentrez::entrez_search()` and `entrez_fetch()`
elifeopts	eLife options
elsevieropts	Elsevier options
wileyopts	Wiley options
crossrefopts	Crossref options
...	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.

See **Rate Limits** and **Authentication** in fulltext-package for rate limiting and authentication information, respectively

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 retrived 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

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

## 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 packge (use `ft_get_ls()` to see what hard coded publisher plugins we have), but we do have generic functions for Crossref and ftdoi.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_get_si` *Download supplementary materials from journals*

Description

Put a call to this function where you would put a file-path - everything is cached by default, so you don't have to worry about multiple downloads in the same session.

Usage

```
ft_get_si(x, si, from = c("auto", "plos", "wiley", "science",
  "proceedings", "figshare", "esa_data_archives", "esa_archives",
  "biorxiv", "epmc"), save.name = NA, dir = NA, cache = TRUE,
  vol = NA, issue = NA, list = FALSE, timeout = 10, ...)
```

Arguments

`x`	One of: vector of DOI(s) of article(s) (a `character`), output from `ft_get`, or output from `ft_search`. Note: if using ESA journal, you can *only* use the ESA-specific article code (e.g., E092-201).
`si`	number of the supplement to be downloaded (1, 2, 3, etc.), or (for ESA and Science journals) the name of the supplement (e.g., "S1_data.csv"). Can be a `character` or `numeric`.

from	Publisher of article (character). The default (auto) uses crossref (cr_works) to detect the journal's publisher. Specifying the journal can somewhat speed up your download, or be used to force a download from EPMC (see details). You *must* specify if downloading from an ESA journal (esa_data_archives, esa_archives). You can only use this argument if x is a vector of DOI(s). Must be one of: auto (i.e., auto-detect journal; default), plos, wiley, science, proceedings, figshare, esa_data_archives, esa_archives, biorxiv, or epmc.
save.name	a name for the file to download (character). If NULL (default) this will be a combination of the DOI and SI number
dir	directory to save file to (character). If NULL (default) this will be a temporary directory created for your files
cache	if TRUE (default), the file won't be downloaded again if it already exists (in a temporary directory creates, or your chosen dir)
vol	Article volume (Proceedings journals only; numeric)
issue	Article issue (Proceedings journals only; numeric)
list	if TRUE, print all files within a zip-file downloaded from EPMC (default: FALSE). This is *very* useful if using EPMC (see notes)
timeout	how long to wait for successful download (default 10 seconds)
...	Further args passed on to GET

Details

The examples probably give the best indication of how to use this function. In general, just specify the DOI of the article you want to download data from, and the number of the supplement you want to download (1, 5, etc.). ESA journals don't use DOIs (give the article code; see below), and Proceedings, Science, and ESA journals need you to give the filename of the supplement to download. For FigShare articles, you can give either the number or the name. The file extensions (suffixes) of files are returned as suffix attributes (see first example), which may be useful if you don't know the format of the file you're downloading.

For any DOIs not recognised (and if asked) the European PubMed Central API is used to look up articles. What this database calls a supplementary file varies by publisher; often they will simply be figures within articles, but we (obviously) have no way to check this at run-time. I strongly recommend you run any EPMC calls with list=TRUE the first time, to see the filenames that EPMC gives supplements, as these also often vary from what the authors gave them. This may actually be a 'feature', not a 'bug', if you're trying to automate some sort of meta-analysis.

Below is a list of all the publishers this supports, and examples of articles from them. I'm aware that there isn't perfect overlap between these publishers and the rest of the package; I plan to correct this in the near future.

auto Default. Use a cross-ref search (cr_works) on the DOI to determine the publisher.

plos Public Library of Science journals (e.g., PLoS One; http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0126524)

wiley Wiley journals, (e.g., http://onlinelibrary.wiley.com/doi/10.1111/ele.12289/abstract

science Science magazine (e.g., http://www.sciencemag.org/content/345/6200/1041.short)

proceedings Royal Society of London journals (e.g., http://rspb.royalsocietypublishing.org/content/282/1814/20151215). Requires vol and issue of the article.

figshare Figshare, (e.g., http://bit.ly/figshare-example)

esa_data_archives & esa_data You must give article codes, not DOIs, for these, which you can find on the article itself. An ESA Data Archive paper - not to be confused with an ESA Archive, which is the supplement to an ESA paper. The distinction seems less crazy once you're reading the paper - if it only describes a dataset, it's an esa_archive paper, else it's an esa_data_archive. For example, http://www.esapubs.org/archive/ecol/E092/201/default.htm is an esa_data_archive whose article code is E092-201-D1; http://esapubs.org/Archive/ecol/E093/059/default.htm is a esa_archive whose code is E093-059-D1.

biorxiv Load from bioRxiv (e.g., http://biorxiv.org/content/early/2015/09/11/026575)

epmc Look up an article on the Europe PubMed Central, and then download the file using their supplementary materials API (http://europepmc.org/restfulwebservice). See comments above in 'notes' about EPMC.

Note

Make sure that the article from which you're attempting to download supplementary materials *has* supplementary materials. 404 errors and 'file not found' errors can result from such cases.

Author(s)

Will Pearse (<will.pearse@gmail.com>)

Examples

```
## Not run:
#Put the function wherever you would put a file path
crabs <- read.csv(ft_get_si("10.6084/m9.figshare.979288", 2))

#View the suffix (file extension) of downloaded files
# - note that not all files are uploaded/stored with useful file extensions!
ft_get_si("10.6084/m9.figshare.979288", 2)
attr(ft_get_si("10.6084/m9.figshare.979288", 2), "suffix")

#ESA data papers and regular articles *must* be marked
fungi <- read.csv(ft_get_si("E093-059", "myco_db.csv",
                                        "esa_archives"))
mammals <- read.csv(ft_get_si("E092-201", "MCDB_communities.csv",
                                            "esa_data_archives"))
epmc.fig <- ft_get_si("10.1371/journal.pone.0126524", "pone.0126524.g005.jpg", "epmc")
#...note this 'SI' is not actually an SI, but rather an image from the paper.

# curl options
ft_get_si("E093-059", "myco_db.csv", "esa_archives")

## End(Not run)
```

`ft_links` *Get full text links*

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. See `?searchplos`
`crossrefopts`	Crossref options. See `?cr_works`
`entrezopts`	Entrez options. See `?entrez_search`
`bmcopts`	BMC options. 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 form 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.ympev.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` *Search for information on journals or publishers.*

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

# 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 `xml2::read_xml()` or `jsonlite::toJSON()`

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

---

`fulltext-deprecated` *Deprecated functions in fulltext*

Description

- ft_chunks: We're trying to focus the scope of the package - this function is moving to the package pubchunks
- ft_tabularize: We're trying to focus the scope of the package - this function is moving to the package pubchunks

Index