Package 'fulltext'

August 29, 2016

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 used in and outside of R.

Version 0.1.8

License MIT + file LICENSE

URL https://github.com/ropensci/fulltext

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

LazyLoad yes

VignetteBuilder knitr

Imports methods, utils, stats, httr (>= 1.1.0), magrittr, xml2 (>= 1.0.0), jsonlite, rplos (>= 0.6.0), rcrossref (>= 0.5.4), aRxiv, rentrez, tm, rredis, R.cache, digest, whisker

Suggests testthat, knitr

RoxygenNote 5.0.1

NeedsCompilation no

Author Scott Chamberlain [aut, cre]

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2016-07-23 08:32:28

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.

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`
- extract text from pdfs - `ft_extract`
- serialize to different data formats - `ft_serialize`
- extract certain article sections (e.g., authors) - `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.

Feedback

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

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

---

cache *Cache blobs of json, xml or pdfs of text from ft_get() function*

Description

Cache blobs of json, xml or pdfs of text from ft_get() function

Usage

```
cache_options_set(cache = TRUE, backend = "rds", path = "~/.fulltext")

cache_options_get()

cache_clear(cachetype = NULL)
```

Arguments

cache	(logical) If TRUE, cache results, if not objects saved within R session.
backend	(character) One of rds, rcache, redis
path	path to local storage. used only if backend="rds"
cachetype	The cache type

Examples

```
## Not run:
ft_get('10.1371/journal.pone.0086169', from='plos', cache=FALSE)
ft_get('10.1371/journal.pone.0086169', from='plos', cache=TRUE)

cache_options_set(backend="redis")
cache_options_get()
(x <- ft_get('10.1371/journal.pone.0086169', from='plos', cache=TRUE, backend="redis"))
```

```
x %>% collect()

cache_options_set(backend="rcache")
cache_options_get()
(x <- ft_get('10.1371/journal.pone.0086169', from='plos'))
x %>% collect()

# Many different sources
(res <- ft_search(query='ecology', from='entrez'))
cache_options_set(backend="rds")
out <- ft_get(res)
out$entrez
out %>% collect() %>% chunks("title")

## End(Not run)
```

---

chunks *Extract chunks of data from articles*

Description

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 vizualization and analysis.

Usage

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

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, and Entrez 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')
chunks(x, what="authors")

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

# Coerce list output to a data.frame, where possible
(dois <- searchplos(q="*:*", fl='id',
   fq=list('doc_type:full',"article_type:\"research article\""), limit=5)$data$id)
x <- ft_get(dois, from="plos")
x %>% chunks("publisher") %>% tabularize()
x %>% chunks("refs") %>% tabularize()
x %>% chunks(c("doi","publisher")) %>% tabularize()
```

```
x %>% chunks(c("doi","publisher","permissions")) %>% tabularize()

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

x <- ft_get('10.3389/fnagi.2014.00130', from='entrez')
x %>% 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") %>%
 chunks("publisher")

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

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

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

## End(Not run)
```

`collect` *Collect data from a remote source in fulltext*

Description

`collect` grabs full text data from a remote storage device. `get_text` is a convenience function to grab the nested text data and bring it up in the list for easier access

Usage

```
collect(x, ...)

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

get_text(x, ...)

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

Arguments

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

Examples

```
## Not run:
# Get some data, stash in rds file
x <- ft_get('10.1371/journal.pone.0086169', from='plos', cache=TRUE, backend="rds")

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

# Collect data from the rds file
y <- x %>% collect()

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

# Let's get the actual
x %>% collect() %>% get_text()

## End(Not run)
```

---

extract_tools *PDF extraction tools*

Description

If you want to use ft_extract function, it currently has two options for how to extract text from PDFs: xpdf and ghostscript.

xpdf installation

See http://www.foolabs.com/xpdf/download.html for instructions on how to download and install 'xpdf'. For OSX, you an also get 'xpdf' via Homebrew (https://github.com/homebrew/homebrew-x11/blob/master/xpdf.rb) with `brew install xpdf`. Apparently, you can optionally install Poppler, which is built on xpdf. Get it at http://poppler.freedesktop.org/

ghostscript installation

See http://www.ghostscript.com/doc/9.16/Install.htm for instructions on how to download and install 'ghostscript'. For OSX, you an also get 'ghostscript' via Homebrew (https://github.com/Homebrew/homebrew/blob/mast with `brew install gs`

---

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)

ft_browse_sections(x, what = "abstract", output = NULL, browse = TRUE)
```

Arguments

x	An object of class `ft_data` - the output from a call to `ft_get`
what	(character) One of macrodocs (default), publisher, or whisker.
browse	(logical) Whether to browse (default) or not. If `FALSE`, return the url.
output	A file path, if not given, uses a temporary file, deleted up on leaving the R session.

Details

`what=whisker` not operational yet. When operational, will use whisker to open html page from XML content, each section parsed into separate section.

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")

# Browse sections
x <- ft_get(c('10.1371/journal.pone.0086169',"10.1371/journal.pone.0110535"), from='plos')
ft_browse_sections(x, "abstract")
ft_browse_sections(x, "categories")

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

## 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 a variety of extraction tools. Inputs can be either paths to PDF files, or the output of `ft_get` (class `ft_data`).

Usage

```
ft_extract(x, which = "xpdf", ...)

## S3 method for class 'gs_char'
print(x, ...)

## S3 method for class 'xpdf_char'
print(x, ...)
```

Arguments

`x`	Path to a pdf file, or an object of class `ft_data`, the output from `ft_get`
`which`	One of gs or xpdf (default).
`...`	further args passed on

Details

For xpdf, you can pass on addition options via flags. See Examples. Right now, you can't pass options to Ghostscript if you're using the gs option.

xpdf installation: See `http://www.foolabs.com/xpdf/download.html` for instructions on how to download and install xpdf. For OSX, you an also get xpdf via homebrew.

ghostscript installation: See `http://www.ghostscript.com/doc/9.16/Install.htm` for instructions on how to download and install ghostscript

Value

An object of class gs_char, xpdf_char

Examples

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

(res_xpdf <- ft_extract(path)) # xpdf is the default
(res_xpdf <- ft_extract(path, "xpdf"))
(res_gs <- ft_extract(path, "gs"))

# pass on options to xpdf
## preserve layout from pdf
ft_extract(path, "xpdf", "-layout")
## preserve table structure as much as possible
ft_extract(path, "xpdf", "-table")
## last page to convert is page 2
ft_extract(path, "xpdf", "-l 2")
## first page to convert is page 3
ft_extract(path, "xpdf", "-f 3")

# 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
res2$arxiv$data$data[[1]]$data

## biorxiv
res <- ft_get('10.1101/012476')
res2 <- ft_extract(res)
res$biorxiv$data
res2$biorxiv$data
res2$biorxiv$data$data[[1]]$data

## End(Not run)
```

---

ft_extract_corpus *Extract text from one to many pdf documents into a tm Corpus or Vcorpus.*

Description

Extract text from one to many pdf documents into a tm Corpus or Vcorpus.

Usage

```
ft_extract_corpus(paths, which = "xpdf", ...)
```

Arguments

paths	Path to one or more pdfs
which	One of gs or xpdf.
...	further args passed on to readerControl parameter in Corpus

Value

A tm Corpus (or VCorpus, later that is)

See Also

ft_extract

Examples

```
## Not run:
path <- system.file("examples", "example1.pdf", package = "fulltext")
(res <- ft_extract_corpus(path, "xpdf"))
tm::TermDocumentMatrix(res$data)

(res_gs <- ft_extract_corpus(path, "gs"))

## End(Not run)
```

---

ft_get *Get full text*

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. See Details for help on how to use this function.

Usage

```
ft_get(x, from = NULL, plosopts = list(), bmcopts = list(),
  entrezopts = list(), elifeopts = list(), cache = FALSE,
  backend = "rds", path = "~/.fulltext", ...)

## S3 method for class 'character'
ft_get(x, from = NULL, plosopts = list(),
  bmcopts = list(), entrezopts = list(), elifeopts = list(),
  cache = FALSE, backend = "rds", path = "~/.fulltext", ...)

## S3 method for class 'list'
ft_get(x, from = NULL, plosopts = list(), bmcopts = list(),
```

```
  entrezopts = list(), elifeopts = list(), cache = FALSE,
  backend = "rds", path = "~/.fulltext", ...)

## S3 method for class 'ft'
ft_get(x, from = NULL, plosopts = list(), bmcopts = list(),
  entrezopts = list(), elifeopts = list(), cache = FALSE,
  backend = "rds", path = "~/.fulltext", ...)
```

Arguments

x	Either identifiers for papers, either DOIs (or other ids) as a list of charcter strings, or a character vector, OR an object of class ft, as returned from ft_search
from	Source to query. Optional.
plosopts	PLOS options. See plos_fulltext
bmcopts	BMC options. parameter DEPRECATED
entrezopts	Entrez options. See entrez_search and entrez_fetch
elifeopts	eLife options
cache	(logical) To cache results or not. If cache=TRUE, raw XML, or other format that article is in is written to disk, then pulled from disk when further manipulations are done on the data. See also cache
backend	(character) One of rds, rcache, or redis
path	(character) Path to local folder. If the folder doesn't exist, we create it for you.
...	Further args passed on to GET

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.

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.

Examples

```
## Not run:
# If you just have DOIs and don't know the publisher
## PLOS
ft_get('10.1371/journal.pone.0086169')
## PeerJ
ft_get('10.7717/peerj.228')
## eLife
ft_get('10.7554/eLife.03032')
## 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'))
## FrontiersIn
res <- ft_get(c('10.3389/fphar.2014.00109', '10.3389/feart.2015.00009'))
## Hindawi - via Entrez
res <- ft_get(c('10.1155/2014/292109','10.1155/2014/162024','10.1155/2014/249309'))
## F1000Research - via Entrez
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/zookeys.499.8360')
### you'll need to specify the publisher for a DOI from a recent publication
ft_get('10.3897/zookeys.515.9332', from = "pensoft")
## Copernicus
out <- ft_get(c('10.5194/angeo-31-2157-2013', '10.5194/bg-12-4577-2015'))
out$copernicus
## arXiv - only pdf, you have to pass in the from parameter
res <- ft_get(x='cond-mat/9309029', from = "arxiv", cache=TRUE, backend="rds")
res %>% ft_extract
## bioRxiv - only pdf
res <- ft_get(x='10.1101/012476')
res$biorxiv
## Karger Publisher
ft_get('10.1159/000369331')
## CogentOA Publisher
ft_get('10.1080/23311916.2014.938430')
## MDPI Publisher
ft_get('10.3390/nu3010063')
ft_get('10.3390/nu7085279')
ft_get(c('10.3390/nu3010063', '10.3390/nu7085279')) # not working, only getting 1
```

```
# If you know the publisher, give DOI and publisher
## by default, PLOS gives back XML
ft_get('10.1371/journal.pone.0086169', from='plos')
## you can instead get json
ft_get('10.1371/journal.pone.0086169', from='plos', plosopts=list(wt="json"))

(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')

# Caching
res <- ft_get('10.1371/journal.pone.0086169', from='plos', cache=TRUE, backend="rds")

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

## End(Not run)
```

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, ...)

## S3 method for class 'character'
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, ...)

## S3 method for class 'ft_data'
ft_get_si(x, si, from = NA, save.name = NA, dir = NA,
  cache = TRUE, vol = NA, issue = NA, list = FALSE, timeout = 10, ...)

## S3 method for class 'ft'
ft_get_si(x, si, from = NA, 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 supplment (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(), ...)

## S3 method for class 'ft'
ft_links(x, from = NULL, plosopts = list(),
  crossrefopts = list(), entrezopts = list(), bmcopts = list(), ...)

## S3 method for class 'ft_ind'
ft_links(x, from = NULL, plosopts = list(),
  crossrefopts = list(), entrezopts = list(), bmcopts = list(), ...)

## S3 method for class 'character'
ft_links(x, from = NULL, plosopts = list(),
  crossrefopts = list(), entrezopts = list(), bmcopts = list(), ...)
```

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
...	Further args passed on to GET. Not working 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.

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

```
## 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'))

## 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 GET

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 and Biorxiv 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, plosopts = list(),
  bmcopts = list(), crossrefopts = list(), entrezopts = list(),
  arxivopts = list(), biorxivopts = list(), euroopts = list(),
  bmc_key = NULL, ...)
```

Arguments

query	Query terms
from	Source to query
limit	Number of records to return.
plosopts	PLOS options. See ?searchplos
bmcopts	BMC options. See ?bmc_search

crossrefopts	Crossref options. See ?cr_works
entrezopts	Entrez options. See ?entrez_search
arxivopts	arxiv options. See ?arxiv_search
biorxivopts	biorxiv options. See ?bx_search
euroopts	Euro PMC options. See ?eupmc_search
bmc_key	(character) A API key for Springer/BMC. See Details. Required when searching BMC - otherwise, not needed. Default: NULL
...	Further args passed on to GET. Not working right now...

Value

An object of class ft, and objects of class ft_ind within each source

BMC Authentication

BMC is integrated into Springer Publishers now, and that API requires an API key. Get your key by signing up here https://dev.springer.com/, then you'll get a key. Pass the key to the parameter bmc_key or to a param named key in the param bmcopts. However, the best option is to save the key in your .Renviron file like SPRINGER_KEY=yourkey, or in your .Rprofile file like springer_key="your key"

Examples

```
## 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='ecology', 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

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

## 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. In addition, this function allows you to save to various places, including Rds files, cached via `R.cache`, or to Redis.

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, 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 `read_xml` or `toJSON`

Value

An object of class `ft_parsed`

Examples

```
## Not run:
dois <- c('10.1371/journal.pone.0087376','10.1371%2Fjournal.pone.0086169',
'10.1371/journal.pone.0102976','10.1371/journal.pone.0105225',
'10.1371/journal.pone.0102722','10.1371/journal.pone.0033693')
res <- ft_get(dois, from='plos')
```

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

# From XML to JSON
(out <- ft_serialize(res, to='json'))
out$plos$data$data$`10.1371/journal.pone.0087376` # the json
jsonlite::fromJSON(out$plos$data$data$`10.1371/journal.pone.0087376`)

# To a list
out <- ft_serialize(res, to='list')
out$plos$data$data[[4]]
out$plos$data$data[[4]][[2]]$`article-meta`

# To various data stores on disk
## To an .Rds file
ft_serialize(res, to='file')

## To local files using R.cache package
res_rcache <- ft_serialize(res, to='rcache')

## To Redis
res_redis <- ft_serialize(res, to='redis')

# Chain together functions
doi <- '10.1371/journal.pone.0086169'
ft_get(doi, from='plos') %>%
   ft_serialize(to='xml') %>%
   ft_serialize(to='redis')

## End(Not run)
```

---

pdfx *PDF-to-XML conversion of scientific articles using pdfx*

Description

Uses a web service provided by Utopia at http://pdfx.cs.man.ac.uk/. Beware, this can be quite slow. pdfx posts the pdf from your machine to the web service, pdfx_html takes the output of pdfx and gives back a html version of extracted text, and pdfx_targz gives a tar.gz version of the extracted text. This will not work with PDFs that are scans of text, or mostly of images.

Usage

```
pdfx(file, what = "parsed", ...)

pdfx_html(input, ...)

pdfx_targz(input, write_path, ...)
```

Arguments

file	(character) Path to a file, or files on your machine. Required.
what	(character) One of parsed or text.
...	Further args passed to GET. These aren't named, so just do e.g. , verbose(), or timeout(3)
input	Output from pdfx function
write_path	Path to write tar ball to.

Value

pdfx gives XML parsed to xml_document, pdfx_html gives html, pdfx_targz writes a tar.gz file to disk.

Examples

```
## Not run:
path <- system.file("examples", "example2.pdf", package = "fulltext")
pdfx(file = path)

out <- pdfx(file = path)
pdfx_html(out)

out <- pdfx(file = path)
tarfile <- tempfile(fileext = "tar.gz")
pdfx_targz(input = out, write_path = tarfile)

## End(Not run)
```

Index