Package: uniprotREST (via r-universe)

August 23, 2024

Title R wrapper for the UniProt website REST API

Version 1.0.0

Description Simple functions to access the UniProt website REST API. Wraps httr2 functions to easily map IDs and query all available databases.

License MIT + file LICENSE

URL https://csdaw.github.io/uniprotREST/, https://github.com/csdaw/uniprotREST

BugReports https://github.com/csdaw/uniprotREST/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends R (>= 2.10)

Imports checkmate, httr2, magrittr, utils

Suggests Biostrings, covr, httptest2, httpuv, knitr, rmarkdown, testthat (>= 3.0.0)

LazyData true

Config/testthat/edition 3

VignetteBuilder knitr

Repository https://csdaw.r-universe.dev

RemoteUrl https://github.com/csdaw/uniprotREST

RemoteRef HEAD

RemoteSha 3da0fee571c76b6ec96ee3a87ff22db1c78161b8

Contents

`fetch_paged` *Fetch results via pagination*

Description

This function performs a request for data from the UniProt REST API, fetches the results using pagination, and saves them to a file or into memory.

You likely won't use this function directly, but rather one of the wrapper functions: `uniprot_map()`, `uniprot_search()`, or `uniprot_single()`.

Things to note::

1. The pagination endpoint is less expensive for the API infrastructure to deal with versus `fetch_stream()`, as the memory demand is distributed over a longer period of time.
2. If the connection is interrupted while fetching results with pagination, only the request for the current page needs to be reattempted. With the stream endpoint, the entire request needs to be completely restarted.

Usage

```
fetch_paged(req, n_pages, format = "tsv", path = NULL, verbosity = NULL)
```

Arguments

`req`	`httr2_request` object, generated by e.g. `httr2::request()` or `uniprot_request()`.
`n_pages`	`integer`, the number of pages to be fetched. This can be calculated by dividing the number of total results by the page size e.g. `resp$headers$x-total-results / page_size`.
`format`	`string`, data format to fetch. Can one of `"tsv"`, or `"fasta"`.
`path`	`string` (optional), file path to save the results, e.g. `"path/to/results.tsv"`. The file must not already exist, otherwise an error is thrown.
`verbosity`	`integer` (optional), how much information to print? • 0: no output • NULL (default): minimal output • 1: show request headers • 2: show request headers and bodies • 3: show request headers, bodies, and curl status messages

Value

By default, returns an object whose type depends on `format`:

- `tsv`: `data.frame`
- `json`: `list`
- `fasta`: Biostrings::AAStringSet (or `named character` if Biostrings not installed)

If `parse = FALSE`, returns an `httr2_response`. If `path` is specified, saves the parsed results to the file path indicated, and returns `NULL` invisibly.

Examples

```
## Not run:
  req <- uniprot_request(
    "https://rest.uniprot.org/uniref/search",
    query = "P99999",
    format = "tsv",
    fields = "id,name,count",
    size = 1
  )

  fetch_paged(req, n_pages = 3)

## End(Not run)
```

---

fetch_stream *Fetch results via stream*

Description

This function performs a request for data from the UniProt REST API, fetches the results using the stream endpoint, and saves them to a file or into memory.

You likely won't use this function directly, but rather one of the wrapper functions: `uniprot_map()`, `uniprot_search()`, or `uniprot_single()`.

Things to note::

1. The stream endpoint is expensive for the API to process. If this endpoint has too many requests a `429` status error will occur. In this case use `fetch_paged()` or try again later.
2. Up to 10,000,000 results can be fetched via stream. If you want to get more results you should use `fetch_paged()`, or consider downloading some datasets from UniProt's FTP website.

Usage

```
fetch_stream(req, format = "tsv", parse = TRUE, path = NULL, verbosity = NULL)
```

Arguments

req	httr2_request object, generated by e.g. httr2::request() or uniprot_request().
format	string, data format to fetch. Can be one of "tsv", "json" or "fasta".
parse	logical, should the response body be parsed e.g. into a data.frame or should the httr2_response object be returned instead? Default is TRUE. Does nothing if path is provided.
path	string (optional), file path to save the results, e.g. "path/to/results.tsv". The file must not already exist, otherwise an error is thrown.
verbosity	integer (optional), how much information to print? • 0: no output • NULL (default): minimal output • 1: show request headers • 2: show request headers and bodies • 3: show request headers, bodies, and curl status messages

Value

By default, returns an object whose type depends on format:

- tsv: data.frame
- json: list
- fasta: Biostrings::AAStringSet (or named character if Biostrings not installed)

If parse = FALSE, returns an httr2_response. If path is specified, saves the parsed results to the file path indicated, and returns NULL invisibly.

Examples

```
## Not run:
  req <- uniprot_request(
    "https://rest.uniprot.org/uniref/stream",
    query = "P99999",
    format = "tsv",
    fields = "id,name,count"
  )

  fetch_stream(req)

## End(Not run)
```

`formats` *(Dataset) UniProt download formats*

Description

This dataframe contains the available download file types i.e. values for `format` for each UniProt database. See **Examples** below for how to use this object

Usage

```
formats
```

Format

An object of class `data.frame` with 43 rows and 3 columns.

Details

Columns:

func	`character`, `uniprotREST` function to be used.
database	`character`, UniProt database to be queried.
format	`character`, the different formats available to download i.e. the string you'll use with the `uniprotREST` function.

Source

UniProtKB download formats have been determined by hand by querying each REST API database with an unavailable format (usually `txt`) and determining the allowed formats from the resulting error response.

See Also

return_fields

Examples

```
# What UniProt databases are available to query?
levels(formats$database)[1:12]

# What formats are available for querying the `proteomes` database,
# using `uniprot_search()`
formats[formats$database == "proteomes" & formats$func == "search", "format"]
```

from_to_dbs *(Dataset) From/to databases for ID mapping*

Description

This dataframe contains details of the databases that can be mapped from/to with `uniprot_map()`. See from_to_rules for the rules on which database identifiers can be mapped to what other databases.

Usage

```
from_to_dbs
```

Format

An object of class `data.frame` with 100 rows and 5 columns.

Details

Columns:

name	`character`, database name.
from	`character`, is the database allowed to be mapped `from`?
to	`character`, is the database allowed to be mapped `to`?
url	`character`, database URL.
formats_db	`factor`, relevant `formats` database, see formats for which download formats are available

Source

From/to pairs have been downloaded according to the "Valid from and to databases pairs" section on the UniProt ID Mapping page.

See Also

from_to_rules

from_to_rules *(Dataset) From/to rules for ID mapping*

Description

This list contains the valid from/to pairings that can be used with `uniprot_map()`. See **Examples** below for how to use this object. Also see from_to_dbs for the more information on each database.

Usage

```
from_to_rules
```

Format

An object of class `list` of length 98.

Source

From/to pairs have been downloaded according to the "Valid from and to databases pairs" section on the UniProt ID Mapping page.

See Also

from_to_dbs

Examples

```
# Show valid `from` values
names(from_to_rules)

# Show valid `to` values for a given `from` e.g. SGD
from_to_rules[["SGD"]]
```

`resp_body_fasta` *Extract FASTA data from response body*

Description

This function gets FASTA data from an `httr2_response` and either saves it to a file or parses it in memory into a Biostrings::AAStringSet or `named character` vector.

Usage

```
resp_body_fasta(resp, con = NULL, encoding = NULL)
```

Arguments

`resp`	`httr2_response` object, generated by e.g. `httr2::req_perform()` or `fetch_stream()`/`fetch_paged()`
`con`	`string` or base::connection object (optional), the file in which to save the data.
`encoding`	`string` (optional), character encoding of the body text. If not specified, will use the encoding specified by the content-type, falling back to `"UTF8"` with a warning if it cannot be found. The resulting string is always re-encoded to UTF-8.

Value

By default, returns a Biostrings::AAStringSet object. If the `Biostrings` package is not installed, returns a `named character` vector. If `con` is not NULL, returns nothing and saves the FASTA sequences to the file specified by `con`.

Examples

```
resp <- structure(
  list(method = "GET", url = "https://example.com",
       body = charToRaw(">Protein1\nAAA\n>Protein2\nCCC")),
  class = "httr2_response"
)

resp_body_fasta(resp)
```

`resp_body_tsv` *Extract tab-delimited data from response body*

Description

This function gets tab-delimited data from an `httr2_response` and either saves it to a file or parses it into a `data.frame` in memory.

Usage

```
resp_body_tsv(resp, page = NULL, con = NULL, encoding = NULL)
```

Arguments

`resp`	`httr2_response` object, generated by e.g. `httr2::req_perform()` or `fetch_stream()`/`fetch_paged()`
`page`	`integer` (optional), response page number. If `page` > 1 then the table header is removed before saving to file. Only used when `con` is specified.
`con`	`string` or base::connection object (optional), the file in which to save the data.
`encoding`	`string` (optional), character encoding of the body text. If not specified, will use the encoding specified by the content-type, falling back to `"UTF8"` with a warning if it cannot be found. The resulting string is always re-encoded to UTF-8.

Value

By default, returns a `data.frame`. If `con` is not NULL, returns nothing and saves tab-delimited text to the file specified by `con`.

Examples

```
resp <- structure(
  list(method = "GET", url = "https://example.com",
       body = charToRaw("Entry\tGene Names (primary)\nP99999\tCYCS\n")),
  class = "httr2_response"
)

resp_body_tsv(resp)
```

`return_fields` *(Dataset) UniProt return fields*

Description

This dataframe contains the valid fields (i.e. columns) of data you can request from UniProt. The strings in the `field` column is what you'll use with the functions in this package. See **Examples** below for how to use this object.

Usage

```
return_fields
```

Format

An object of class `data.frame` with 389 rows and 4 columns.

Details

Columns:

database	`character`, UniProt database to be queried.
section	`character`, similar return fields are grouped together in sections.
field	`character`, the return field i.e. the string used to request the desired column of information.
label	`character`, human readable column name that will be returned by the API.

Source

UniProtKB return fields have been scraped from the UniProtKB return fields page. The return fields from other Uniprot databases have been determined by hand using Web Developer Tools (F12) to inspect the GET request made when searching the different database on the UniProt website.

See Also

formats

Examples

```
# What UniProt databases are available to query?
levels(return_fields$database)

# What fields are available for the `proteomes` database?
return_fields[return_fields$database == "proteomes", "field"]
```

uniprot_map *Map from/to UniProt IDs*

Description

This function wraps the UniProt ID Mapping service which maps between the identifiers used in one database, to the identifiers of another. By default it maps UniProtKB accessions to UniProt, and returns a `data.frame` with metadata about the mapped protein accessions. You can also map IDs from/to other databases e.g. `from = "Ensembl", to = "UniProtKB"`.

Things to note:

This service has limits on the number of IDs allowed. Very large mapping requests are likely to fail. Try to split your queries into smaller chunks in case of problems.

- 100,000 = maximum number of input `ids` allowed
- 500,000 = maximum number of entries that will be output

Usage

```
uniprot_map(
  ids,
  from = "UniProtKB_AC-ID",
  to = "UniProtKB",
  format = "tsv",
  path = NULL,
  fields = NULL,
  isoform = NULL,
  method = "paged",
  page_size = 500,
  compressed = NULL,
  verbosity = NULL,
  dry_run = FALSE
)
```

Arguments

`ids`	`character`, vector of identifiers to map from. Should not contain duplicates. Maximum length = 100,000 ids.
`from`	`string`, database to map from. Default is `"UniProtKB_AC-ID"`. See from_to_dbs possible databases whose identifiers you can map from.
`to`	`string`, database to map to. Default is `"UniProtKB"`. See from_to_rules for the possible databases you can map to, depending on the `from` database.
`format`	`string`, data format to fetch. Default is `"tsv"`. Can be one of `"tsv"` or `"fasta"`.
`path`	`string` (optional), file path to save the results, e.g. `"path/to/results.tsv"`.

`fields` — `character` (optional), fields (i.e. columns) of data to get. Only used if `to` is a UniProtKB, UniRef, or UniParc database. See return_fields for all available fields.

`isoform` — `logical` (optional), should protein isoforms be included in the results? Not necessarily relevant for all formats and databases.

`method` — `string`, download method to use. Either `"paged"` (default) or `"stream"`. Paged is more robust to connection issues and takes less memory. Stream may be faster, but uses more memory and is more sensitive to connection issues.

`page_size` — `integer` (optional), how many entries per page to request? Only relevant if `method = "paged"`. It's best to leave this at `500`.

`compressed` — `logical` (optional), should gzipped data be requested? Only relevant if `method = "stream"` and `path` is specified.

`verbosity` — `integer` (optional), how much information to print?

- 0: no output
- NULL (default): minimal output
- 1: show request headers
- 2: show request headers and bodies
- 3: show request headers, bodies, and curl status messages

`dry_run` — `logical`, perform request with `httr2::req_dry_run()`? Requires the `httpuv` package to be installed. Default is `FALSE`.

Value

By default, returns an object whose type depends on `format`:

- `tsv`: `data.frame`
- `fasta`: Biostrings::AAStringSet (or `named character` if Biostrings not installed)

If `path` is specified, saves the results to the file path indicated, and returns `NULL` invisibly. If `dry_run = TRUE`, returns a list containing information about the request, including the request `method`, `path`, and `headers`.

See Also

Other API wrapper functions: `uniprot_search()`, `uniprot_single()`

Examples

```
## Not run:
  # Default, get info about UniProt IDs
  uniprot_map(
    "P99999",
    format = "tsv",
    fields = c("accession", "gene_primary", "feature_count")
  )

  # Other common use, mapping other IDs to UniProt
  # (or vice-versa)
```

```
  uniprot_map(
    c("ENSG00000088247", "ENSG00000162613"),
    from = "Ensembl",
    to = "UniProtKB"
  )


## End(Not run)
```

`uniprot_request` *Create a UniProt HTTP request*

Description

This function creates an `httr2::request` object. You likely won't use this function directly, but rather one of the wrapper functions: `uniprot_map()`, `uniprot_search()`, or `uniprot_single()`.

Usage

```
uniprot_request(url, method = "GET", ..., max_tries = 5, rate = 1/1)
```

Arguments

`url`	`string`, the URL to make the request.
`method`	`string`, the HTTP request method. One of `"GET"` (default), `"POST"`, or `"HEAD"`.
`...`	Name-value pairs that provide query parameters. Each value must be either a length-1 atomic vector (which is automatically escaped) or `NULL` (which is silently dropped).
`max_tries`	`integer`, the number of maximum attempts to perform the HTTP request. Default is `5`.
`rate`	`numeric`, the maximum number of requests per second. Default is `1 / 1` i.e. 1 request per 1 second.

Value

Returns an `httr2_request` object, (which is essentially a list).

Examples

```
# Construct a request for the accession and
# gene name for human Cytochrome C
uniprot_request(
  url = "https://rest.uniprot.org/uniprotkb/P99999",
  fields = "accession,gene_primary",
  format = "tsv"
)
```

`uniprot_search` *Search UniProt*

Description

Search Uniprot via REST API. By default it searches the supplied query against UniProtKB and returns a `data.frame` of matching proteins. It is a wrapper for this UniProt API endpoint.

Usage

```
uniprot_search(
  query,
  database = "uniprotkb",
  format = "tsv",
  path = NULL,
  fields = NULL,
  isoform = NULL,
  method = "paged",
  page_size = 500,
  compressed = NULL,
  verbosity = NULL,
  dry_run = FALSE
)
```

Arguments

`query`	`string`, the search query. See this page for helping constructing search queries.
`database`	`string`, database to look up. Default is `"uniprotkb"`. See the **Databases** section below for all available databases.
`format`	`string`, data format to fetch. Default is `"tsv"`. Can be one of `"tsv"` or `"fasta"`.
`path`	`string` (optional), file path to save the results, e.g. `"path/to/results.tsv"`.
`fields`	`character` (optional), fields (i.e. columns) of data to get. The fields available depends on the database used, see return_fields for all available fields.
`isoform`	`logical` (optional), should protein isoforms be included in the results? Not necessarily relevant for all formats and databases.
`method`	`string`, download method to use. Either `"paged"` (default) or `"stream"`. Paged is more robust to connection issues and takes less memory. Stream may be faster, but uses more memory and is more sensitive to connection issues.
`page_size`	`integer` (optional), how many entries per page to request? Only relevant if `method = "paged"`. It's best to leave this at `500`.
`compressed`	`logical` (optional), should gzipped data be requested? Only relevant if `method = "stream"` and `path` is specified.
`verbosity`	`integer` (optional), how much information to print?

- 0: no output
- NULL (default): minimal output
- 1: show request headers
- 2: show request headers and bodies
- 3: show request headers, bodies, and curl status messages

`dry_run` `logical`, perform request with `httr2::req_dry_run()`? Requires the `httpuv` package to be installed. Default is `FALSE`.

Value

By default, returns an object whose type depends on `format`:

- `tsv`: `data.frame`
- `fasta`: Biostrings::AAStringSet (or `named character` if Biostrings not installed)

If `path` is specified, saves the results to the file path indicated, and returns `NULL` invisibly. If `dry_run = TRUE`, returns a list containing information about the request, including the request `method`, `path`, and `headers`.

Databases

The following databases are available to query:

- `uniprotkb`: UniProt Knowledge Base
- `uniref`: UniProt Reference Clusters
- `uniparc`: UniProt Archive
- `proteomes`: Reference proteomes
- `taxonomy`: Taxonomy
- `keywords`: Keywords
- `citations`: Literature references
- `diseases`: Disease queries
- `database`: Cross references
- `locations`: Subcellular location
- `unirule`: UniRule
- `arba`: ARBA (Association-Rule-Based Annotator)

See Also

Other API wrapper functions: `uniprot_map()`, `uniprot_single()`

Examples

```
## Not run:
  # Search for all human glycoproteins from SwissProt
  res <- uniprot_search(
    query = "(proteome:UP000005640) AND (keyword:KW-0325) AND (reviewed:true)",
    database = "uniprotkb",
    format = "tsv",
    fields = c("accession", "gene_primary", "feature_count")
  )

  # Look at the resulting dataframe
  head(res)

## End(Not run)
```

---

uniprot_single *Download a single UniProt entry*

Description

Get a single entry from UniProt. By default it fetches the webpage using `format = "json"` and outputs a list of information, but different formats are available from different databases. It is a wrapper for this UniProt API endpoint.

Usage

```
uniprot_single(
  id,
  database = "uniprotkb",
  format = "json",
  path = NULL,
  fields = NULL,
  isoform = NULL,
  verbosity = NULL,
  dry_run = FALSE
)
```

Arguments

`id`	`string`, entry ID. Form depends on database e.g. `"P12345"` for UniProtKB, `"UPI0000128BBF"` for UniParc, etc.
`database`	`string`, database to look up. Default is `"uniprotkb"`. See the **Databases** section below for all available databases.
`format`	`string`, data format to fetch. Default is `"json"`. Can be one of `"tsv"`, `"json"`, or `"fasta"`.
`path`	`string` (optional), file path to save the results, e.g. `"path/to/results.tsv"`.

`fields`	`character` (optional), fields (i.e. columns) of data to get. The fields available depends on the database used, see return_fields for all available fields.
`isoform`	`logical` (optional), should protein isoforms be included in the results? Not necessarily relevant for all formats and databases.
`verbosity`	`integer` (optional), how much information to print? • 0: no output • NULL (default): minimal output • 1: show request headers • 2: show request headers and bodies • 3: show request headers, bodies, and curl status messages
`dry_run`	`logical`, perform request with `httr2::req_dry_run()`? Requires the `httpuv` package to be installed. Default is `FALSE`.

Value

By default, returns an object whose type depends on `format`:

- `tsv`: `data.frame`
- `json`: `list`
- `fasta`: Biostrings::AAStringSet (or `named character` if Biostrings not installed)

If `path` is specified, saves the results to the file path indicated, and returns `NULL` invisibly. If `dry_run = TRUE`, returns a list containing information about the request, including the request `method`, `path`, and `headers`.

Databases

The following databases are available to query:

- `uniprotkb`: UniProt Knowledge Base
- `uniref`: UniProt Reference Clusters
- `uniparc`: UniProt Archive
- `proteomes`: Reference proteomes
- `taxonomy`: Taxonomy
- `keywords`: Keywords
- `citations`: Literature references
- `diseases`: Disease queries
- `database`: Cross references
- `locations`: Subcellular location
- `unirule`: UniRule
- `arba`: ARBA (Association-Rule-Based Annotator)

See Also

Other API wrapper functions: `uniprot_map()`, `uniprot_search()`

Examples

```
## Not run:
  # Download the entry for human Cytochrome C
  res <- uniprot_single("P99999", format = "json")

  # Look at the structure of the resulting list
  str(res, max.level = 1)

## End(Not run)
```

Index