Package: rPDBapi (via r-universe)

May 19, 2026

Type Package

Title A Comprehensive Interface for Accessing the Protein Data Bank

Version 3.0.1

Maintainer Selcuk Korkmaz <selcukorkmaz@gmail.com>

Description Provides an R interface to the 'RCSB' Protein Data Bank ('PDB') Search and Data APIs (<https://www.rcsb.org/>). Supports full-text, attribute, sequence, motif, structure, and chemical searches; retrieval of entry-, assembly-, polymer-entity-, and chemical-component-level metadata; and conversion of API responses into analysis-ready tables and typed R objects for reproducible structural bioinformatics workflows.

License MIT + file LICENSE

Encoding UTF-8

Imports dplyr, purrr, httr, jsonlite, xml2, bio3d, magrittr, methods

Suggests knitr, r3dmol, rmarkdown, shiny, testthat

VignetteBuilder knitr

RoxygenNote 7.3.3

NeedsCompilation no

Author Selcuk Korkmaz [aut, cre] (ORCID: <https://orcid.org/0000-0003-4632-6850>), Bilge Eren Yamasan [aut] (ORCID: <https://orcid.org/0000-0002-6525-2503>)

Config/pak/sysreqs libxml2-dev libssl-dev

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

Date/Publication 2026-03-08 03:14:33 UTC

RemoteUrl https://github.com/cran/rPDBapi

RemoteRef HEAD

RemoteSha 69974100930d163c93e7db60b3173be9fa8767dc

Contents

`add_property` *Add or Merge Properties for RCSB PDB Data Fetching*

Description

This function facilitates the management of properties and subproperties required for data retrieval from the Protein Data Bank (PDB). It accepts a list of properties where each key represents a property category (e.g., 'cell', 'exptl'), and the corresponding value is a character vector of subproperties (e.g., 'volume', 'method'). The function ensures that if a property already exists, its subproperties are merged without duplication, guaranteeing that each subproperty remains unique.

Usage

```
add_property(property)
```

Arguments

`property` A list where each element corresponds to a property category. The names of the list elements are the properties, and their values are character vectors containing the subproperties. Each subproperty should be provided as a character vector. The full list of available properties and their descriptions can be found at `https://data.rcsb.org/#data-schema`.

For example, a 'property' list might look like:

cell A character vector of subproperties like `"length_a"`, `"length_b"`, `"length_c"`.

exptl A character vector of subproperties like `"method"`.

Details

The ‘add_property‘ function is particularly useful when users need to dynamically build or update a list of properties required for complex queries in the PDB. By automatically handling duplicate entries, this function streamlines the process of constructing property lists, which can then be used in subsequent data retrieval operations.

The function operates as follows:

1. Checks if the input ‘property‘ is a list. If not, it throws an error.
2. Iterates through each property in the list, ensuring that subproperties are unique and in character vector format.
3. If a property already exists in the list, it merges the subproperties while eliminating duplicates.

Value

A modified list that consolidates the input properties. If a property already exists in the input list, its subproperties are merged, removing any duplicates.

Note

It is important to ensure that the subproperties are correctly formatted as character vectors. The function does not modify the format of the subproperties.

See Also

`fetch_data`, `query_search` for related functions that utilize properties in querying the PDB.

Examples

```
# Example usage:
properties <- list(cell = c("length_a", "length_b", "length_c"), exptl = c("method"))
# Add new properties or merge existing ones
updated_properties <- add_property(properties)
print(updated_properties)
```

---

as_rpdb_assembly *Convert Data to an rPDBapi Assembly Object*

Description

Convert Data to an rPDBapi Assembly Object

Usage

```
as_rpdb_assembly(x, metadata = list())
```

Arguments

x	Input data.
metadata	Optional metadata list.

Value

Object of class rPDBapi_assembly.

---

as_rpdb_chemical_component *Convert Data to an rPDBapi Chemical Component Object*

Description

Convert Data to an rPDBapi Chemical Component Object

Usage

```
as_rpdb_chemical_component(x, metadata = list())
```

Arguments

x	Input data.
metadata	Optional metadata list.

Value

Object of class rPDBapi_chemical_component.

---

as_rpdb_entry *Convert Data to an rPDBapi Entry Object*

Description

Convert Data to an rPDBapi Entry Object

Usage

```
as_rpdb_entry(x, metadata = list())
```

Arguments

x	Input data.
metadata	Optional metadata list.

Value

Object of class rPDBapi_entry.

as_rpdb_polymer_entity *Convert Data to an rPDBapi Polymer Entity Object*

Description

Convert Data to an rPDBapi Polymer Entity Object

Usage

```
as_rpdb_polymer_entity(x, metadata = list())
```

Arguments

x	Input data.
metadata	Optional metadata list.

Value

Object of class `rPDBapi_polymer_entity`.

as_rpdb_structure *Convert Data to an rPDBapi Structure Object*

Description

Convert Data to an rPDBapi Structure Object

Usage

```
as_rpdb_structure(x, metadata = list())
```

Arguments

x	Input data.
metadata	Optional metadata list.

Value

Object of class `rPDBapi_structure`.

autoresolve_sequence_type *Automatically Determine the Sequence Type*

Description

The 'autoresolve_sequence_type' function analyzes the characters in a given sequence to determine whether it is a DNA, RNA, or protein sequence. The function uses standard IUPAC nucleotide and amino acid codes to classify the sequence based on its composition.

Usage

```
autoresolve_sequence_type(sequence)
```

Arguments

sequence	A string representing the nucleotide or protein sequence to be analyzed. The sequence should be composed of characters corresponding to standard IUPAC codes.

Value

A string indicating the resolved sequence type: 'DNA', 'RNA', or 'PROTEIN'. If the sequence contains ambiguous characters or does not fit clearly into one of these categories, an error is thrown.

Examples

```
# Example of determining the sequence type for a DNA sequence
seq_type_dna <- autoresolve_sequence_type("ATGCGTACGTAGC")
print(seq_type_dna)  # Should return "DNA"

# Example of determining the sequence type for a protein sequence
seq_type_protein <- autoresolve_sequence_type("MVLSPADKTNVKAAW")
print(seq_type_protein)  # Should return "PROTEIN"

# Example of an ambiguous sequence that causes an error
# autoresolve_sequence_type("ATGB")  # Should throw an error due to ambiguity
```

build_assembly_id *Build an Assembly Identifier*

Description

Build an Assembly Identifier

Usage

```
build_assembly_id(entry_id, assembly_id)
```

Arguments

entry_id	Entry identifier.
assembly_id	Assembly index or identifier.

Value

Character scalar assembly ID formatted as ENTRY-ASSEMBLY.

---

build_entity_id *Build an Entity Identifier*

Description

Build an Entity Identifier

Usage

```
build_entity_id(entry_id, entity_id)
```

Arguments

entry_id	Entry identifier.
entity_id	Entity index or identifier.

Value

Character scalar entity ID formatted as ENTRY_ENTITY.

---

build_entry_id *Build an Entry Identifier*

Description

Build an Entry Identifier

Usage

```
build_entry_id(entry_id)
```

Arguments

entry_id	Entry identifier.

Value

Character scalar entry ID.

`build_instance_id` *Build an Instance Identifier*

Description

Build an Instance Identifier

Usage

```
build_instance_id(entry_id, instance_id, separator = ".")
```

Arguments

`entry_id`	Entry identifier.
`instance_id`	Instance/chain identifier.
`separator`	Separator between entry and instance. Defaults to `"."`.

Value

Character scalar instance ID.

`cache_info` *Inspect rPDBapi Cache Contents*

Description

Inspect rPDBapi Cache Contents

Usage

```
cache_info(cache_dir = NULL)
```

Arguments

`cache_dir`	Optional cache directory.

Value

A list with cache summary and entry table.

`ChemicalOperator` *Create a Chemical Search Operator for SMILES/InChI Descriptors*

Description

The 'ChemicalOperator' function constructs an operator object used for chemical searches within the RCSB Protein Data Bank (PDB). This function is particularly useful for querying the PDB database using chemical structure descriptors, such as SMILES (Simplified Molecular Input Line Entry System) or InChI (International Chemical Identifier) strings. The function supports various matching criteria to tailor the search results according to specific needs.

Usage

```
ChemicalOperator(descriptor, matching_criterion = "graph-strict")
```

Arguments

`descriptor`	A string representing the chemical structure in either SMILES or InChI format. The function automatically detects the format based on the input string. If the descriptor starts with "InChI=", it is treated as an InChI string; otherwise, it is assumed to be a SMILES string.
`matching_criterion`	A string specifying the criterion for matching the chemical structure. The matching criterion determines how closely the input descriptor should match the structures in the PDB database. The possible values are predefined in the 'DescriptorMatchingCriterion' list, with "graph-strict" being the default. Other options may include "graph-exact," "graph-relaxed," and "fingerprint-similarity," among others.

Details

The 'ChemicalOperator' function is designed for advanced users who need to search for chemical structures in the PDB using specific descriptors. The function allows flexibility in defining the level of matching precision, making it suitable for both exact and fuzzy searches.

The matching criteria provided by the 'matching_criterion' argument allow users to control the strictness of the search. For example:

graph-strict Matches chemical structures based on atom type, bond order, and chirality, with strict graph matching.

graph-relaxed Allows for a more relaxed matching by ignoring certain structural details.

fingerprint-similarity Uses molecular fingerprints to find similar structures based on a similarity threshold.

Value

The function returns a list structured as a 'ChemicalOperator' object. This object contains the input descriptor, the type of descriptor (SMILES or InChI), and the specified matching criterion. The resulting 'ChemicalOperator' object can be used in subsequent functions that perform chemical searches in the PDB database.

See Also

'perform_search' for executing a search using the created 'ChemicalOperator'.

Examples

```
# Example 1: Search for a chemical using a SMILES string
smiles_operator <- ChemicalOperator(descriptor = "C1=CC=CC=C1", matching_criterion = "graph-strict")
smiles_operator

# Example 2: Search using an InChI string with a relaxed matching criterion
inchi_operator <- ChemicalOperator(descriptor = "InChI=1S/C7H8O/c1-6-2-4-7(9)5-3-6/h2-5,9H,1H3",
                                   matching_criterion = "graph-relaxed")
inchi_operator
```

---

clear_rpdbapi_cache *Clear rPDBapi Cache Directory*

Description

Clear rPDBapi Cache Directory

Usage

```
clear_rpdbapi_cache(cache_dir = NULL)
```

Arguments

cache_dir	Optional cache directory.

Value

Invisible list with removal summary.

`ComparisonOperator` *Create a Comparison Search Operator*

Description

Constructs a 'ComparisonOperator' object for search operations that perform comparison checks on attribute values. This operator allows for evaluating attributes using comparison operators such as 'equal', 'greater_than', or 'less_than', making it suitable for numerical and date-based searches.

Usage

```
ComparisonOperator(attribute, value, comparison_type)
```

Arguments

`attribute`	The attribute to be compared. This should be the field within the RCSB PDB that you want to evaluate.
`value`	The value to compare against. This is the reference value for the comparison.
`comparison_type`	A string specifying the type of comparison (e.g., 'equal', 'greater_than', 'less_than'). Supported comparison types are 'equal', 'not_equal', 'greater_than', 'less_than', etc.

Value

An object of class 'ComparisonOperator' that can be used in search queries to retrieve entries where the attribute meets the specified comparison criteria.

Examples

```
# Search for entries where an attribute equals a specific value
operator <- ComparisonOperator(attribute = "rcsb_entry_info.resolution_combined",
                               value = 2.0, comparison_type = "EQUAL")
operator
```

`ContainsPhraseOperator`
Create a Contains Phrase Search Operator

Description

Constructs a 'ContainsPhraseOperator' object for search operations that look for attributes containing a specific phrase. This operator is ideal for scenarios where the search needs to be more precise than just individual words, such as finding an exact phrase within a text attribute.

Usage

```
ContainsPhraseOperator(attribute, value)
```

Arguments

attribute	The attribute to be evaluated. This should be the text field within the RCSB PDB that you want to search against.
value	The phrase to search for in the attribute. The search will look for this exact sequence of words within the specified attribute.

Value

An object of class 'ContainsPhraseOperator' that can be used in search queries to retrieve entries where the attribute contains the specified phrase.

Examples

```
# Search for entries containing a specific phrase in an attribute
operator <- ContainsPhraseOperator(attribute = "rcsb_primary_citation.title",
                                   value = "molecular dynamics")
print(operator)
```

ContainsWordsOperator *Create a Contains Words Search Operator*

Description

Constructs a 'ContainsWordsOperator' object for search operations that look for attributes containing specific words. This operator is particularly useful for text-based searches where the goal is to find entries that include particular keywords or phrases within a specified attribute.

Usage

```
ContainsWordsOperator(attribute, value)
```

Arguments

attribute	The attribute to be evaluated. This should be the text field within the RCSB PDB that you want to search against.
value	The words to search for in the attribute. This can be a single word or a set of words, and the search will return entries containing any of these words in the specified attribute.

Value

An object of class 'ContainsWordsOperator' that can be used in search queries to retrieve entries where the attribute contains the specified words.

Examples

```
# Search for entries containing specific words in an attribute
operator <- ContainsWordsOperator(attribute = "rcsb_primary_citation.title",
                                  value = "crystal structure")
print(operator)
```

`data_fetcher` *Fetch RCSB PDB Data Based on Specified Criteria*

Description

The 'data_fetcher' function provides a flexible way to access data from the RCSB Protein Data Bank (PDB). By specifying an identifier, data type, and a set of properties, users can tailor the data retrieval process to meet their specific research needs. The function integrates several steps, including validating IDs, generating a JSON query, fetching the data, and formatting the response.

Usage

```
data_fetcher(
  id = NULL,
  data_type = "ENTRY",
  properties = NULL,
  return_as_dataframe = TRUE,
  verbosity = FALSE
)
```

Arguments

`id`	A single identifier or a list of identifiers for the data to be fetched. These IDs correspond to the entries, assemblies, polymer entities, or other entities within the RCSB PDB. The ID must match the data type you are querying (e.g., PDB ID for entries, assembly ID for assemblies).
`data_type`	A string specifying the type of data to fetch. The available options for `data_type` include: `"ENTRY"` Fetches data related to PDB entries. This is the default option and returns basic entry-level information. `"ASSEMBLY"` Fetches data related to biological assemblies. The IDs should be formatted as `"PDB_ID-ASSEMBLY_ID"`. `"POLYMER_ENTITY"` Fetches data related to polymeric molecular entities, such as protein chains. `"BRANCHED_ENTITY"` Fetches data related to branched entities, such as carbohydrates or other branched molecules. `"NONPOLYMER_ENTITY"` Fetches data related to non-polymeric entities, such as ligands, cofactors, or small molecules. `"POLYMER_ENTITY_INSTANCE"` Fetches data related to specific instances of polymeric entities, also known as chains.

"BRANCHED_ENTITY_INSTANCE" Fetches data related to specific instances of branched entities.

"NONPOLYMER_ENTITY_INSTANCE" Fetches data related to specific instances of non-polymeric entities.

"CHEMICAL_COMPONENT" Fetches data related to chemical components, such as individual atoms or molecular fragments.

Each data_type corresponds to a specific hierarchy level within the PDB data structure, and selecting the appropriate type ensures that you retrieve relevant and accurate data.

properties A list or dictionary of properties to be included in the data fetching process. The properties should match the data type you are querying. For example, if you are fetching POLYMER_ENTITY data, you might specify properties such as "rcsb_entity_source_organism" or "rcsb_cluster_membership". The properties argument allows users to customize the data they retrieve by specifying exactly which attributes of the data type they are interested in. It is important to match the properties to the correct data type to ensure accurate and meaningful results. The RCSB PDB Data Schema provides detailed documentation on the properties available for each data type. Users are encouraged to refer to this resource when selecting properties for their queries.

return_as_dataframe

A boolean indicating whether to return the response as a dataframe. If TRUE, the data is returned in a structured dataframe format, which is convenient for analysis and manipulation in R. If FALSE, the data is returned in its original format, which may be a nested list or JSON-like structure. Default is TRUE.

verbosity A boolean flag indicating whether to print status messages during the function execution. When set to TRUE, the function will output messages detailing the progress and any issues encountered.

Details

The 'data_fetcher' function is particularly useful for researchers who need to access and analyze specific subsets of PDB data. By providing a list of IDs and the corresponding data type, users can retrieve only the information relevant to their study, reducing the need to manually filter or process large datasets. The function also supports fetching multiple properties simultaneously, allowing for a more comprehensive data retrieval process.

Value

If return_as_dataframe = TRUE, returns a data frame tagged with class "rPDBapi_dataframe". Otherwise returns a validated raw payload tagged with class "rPDBapi_fetch_response".

Examples

```
# Example 1: Fetching basic entry information
properties <- list(cell = c("length_a", "length_b", "length_c"), exptl = c("method"))
data_fetcher(
  id = c("4HHB"),
  data_type = "ENTRY",
```

```
  properties = properties,
  return_as_dataframe = TRUE
)

# Example 2: Fetching polymer entity data
properties <- list(
  rcsb_entity_source_organism = c("ncbi_taxonomy_id", "ncbi_scientific_name"),
  rcsb_cluster_membership = c("cluster_id", "identity")
)
data_fetcher(
  id = c("4HHB_1", "12CA_1"),
  data_type = "POLYMER_ENTITY",
  properties = properties,
  return_as_dataframe = TRUE
)

# Example 3: Fetching non-polymer entity data
properties <- list(
  rcsb_nonpolymer_entity = c("details", "formula_weight", "pdbx_description"),
  rcsb_nonpolymer_entity_container_identifiers = c("chem_ref_def_id")
)
data_fetcher(
  id = c("3PQR_5", "3PQR_6"),
  data_type = "NONPOLYMER_ENTITY",
  properties = properties,
  return_as_dataframe = TRUE
)

# Example 4: Fetching chemical component data
properties <- list(
  rcsb_id = list(),
  chem_comp = list("type", "formula_weight", "name", "formula"),
  rcsb_chem_comp_info = list("initial_release_date")
)
data_fetcher(
  id = c("NAG", "EBW"),
  data_type = "CHEMICAL_COMPONENT",
  properties = properties,
  return_as_dataframe = TRUE
)
```

`data_fetcher_batch` *Batch Fetch RCSB PDB Data with Optional Retry and Cache*

Description

This additive helper retrieves data in batches and can retry failed batches. Caching is optional and disabled by default, preserving existing package behavior.

Usage

```
data_fetcher_batch(
  id,
  data_type = "ENTRY",
  properties,
  return_as_dataframe = TRUE,
  batch_size = 50,
  retry_attempts = 3,
  retry_backoff = 0.5,
  progress = FALSE,
  cache = FALSE,
  cache_dir = NULL,
  verbosity = FALSE
)
```

Arguments

id	Character vector of identifiers.
data_type	Data type passed to data_fetcher().
properties	Property list passed to data_fetcher().
return_as_dataframe	Logical; passed through to data_fetcher().
batch_size	Number of IDs per batch.
retry_attempts	Number of retry attempts per batch.
retry_backoff	Backoff multiplier in seconds.
progress	Logical; show progress messages when TRUE.
cache	Logical; enable on-disk cache.
cache_dir	Optional cache directory.
verbosity	Logical; passed to data_fetcher().

Value

A combined object matching the chosen return mode with provenance metadata.

DefaultOperator *Create a Default Search Operator*

Description

Constructs a 'DefaultOperator' object for use in general search operations within the RCSB PDB. This operator is used when a simple, non-specific search operation is needed, based on a single value. The 'DefaultOperator' can be employed in scenarios where the search criteria do not require complex conditions or additional logic.

Usage

```
DefaultOperator(value)
```

Arguments

value	The value to be used in the search operation. This is the key criterion for the search and should be a valid string or numeric value that the search will use as the matching term.

Value

An object of class 'DefaultOperator' representing the default search operator. This object can be used directly in query formulations or further manipulated within complex search logic.

Examples

```
# Create a basic search operator with a specific value
operator <- DefaultOperator("4HHB")
print(operator)
```

---

describe_chemical *Describe Chemical Compound from RCSB PDB*

Description

Retrieves detailed information about a chemical compound from the RCSB Protein Data Bank (PDB) based on its chemical ID.

Usage

```
describe_chemical(chem_id, url_root = URL_ROOT)
```

Arguments

chem_id	A string representing the 3-character chemical ID. This ID is typically an alphanumeric string used to uniquely identify ligands, cofactors, or other small molecules within macromolecular structures. The string must not exceed 3 characters.
url_root	A string representing the URL for retrieving information about chemical compounds. By default, this is set to the global constant URL_ROOT, but users can specify a different URL if needed.

Value

A list containing detailed information about the chemical compound. This list includes various fields such as:

rcsb_chem_comp_descriptor A sublist containing chemical descriptors like SMILES, InChI strings, molecular weight, and other chemical properties.

rcsb_chem_comp_info Information regarding the compound's classification, formula, and additional relevant data.

other Other fields may also be included, depending on the specific compound and the data available from the RCSB PDB.

Examples

```
## Not run:
# Retrieve chemical information for N-Acetyl-D-Glucosamine (NAG)
chem_desc <- describe_chemical('NAG')
chem_desc

# Access the SMILES string of the compound
smiles_string <- chem_desc$rcsb_chem_comp_descriptor$smiles
smiles_string


## End(Not run)
```

ExactMatchOperator *Create an Exact Match Search Operator*

Description

Constructs an 'ExactMatchOperator' object for precise search operations within the RCSB PDB. This operator is designed to match an exact attribute value, making it ideal for searches where specificity is required. For example, if you need to find all entries that exactly match a certain attribute value, this operator will ensure only those precise matches are returned.

Usage

```
ExactMatchOperator(attribute, value)
```

Arguments

`attribute`	The attribute to match. This should be the name of the field within the RCSB PDB that you want to search against.
`value`	The exact value to search for. This is the specific value of the attribute you are interested in. The search will return only those records where the attribute exactly matches this value.

Value

An object of class 'ExactMatchOperator'. This object can be used in search queries to retrieve precise matches within the RCSB PDB database.

Examples

```
# Search for entries with an exact match to a given attribute
operator <- ExactMatchOperator(attribute = "rcsb_entry_info.resolution_combined",
                               value = "2.0")
print(operator)
```

`ExistsOperator` *Create an Existence Search Operator*

Description

Constructs an 'ExistsOperator' object for search operations to check the existence of an attribute. This operator is useful in queries where you need to ensure that a certain attribute is present within the entries being searched, regardless of its value.

Usage

```
ExistsOperator(attribute)
```

Arguments

`attribute`	The attribute whose existence is to be checked. This should be the field within the RCSB PDB that you want to ensure is present in the search results.

Value

An object of class 'ExistsOperator' that can be used in search queries to retrieve entries where the specified attribute exists.

Examples

```
# Search for entries where a specific attribute exists
operator <- ExistsOperator(attribute = "rcsb_primary_citation.doi")
print(operator)
```

extract_calpha_coordinates *Extract C-alpha Coordinates*

Description

Extract C-alpha Coordinates

Usage

```
extract_calpha_coordinates(structure)
```

Arguments

structure	Structure object returned by `get_pdb_file()` for CIF/PDB.

Value

Tibble with chain, residue, and coordinate columns for C-alpha atoms.

extract_ligand_table *Extract Ligand Table*

Description

Extract Ligand Table

Usage

```
extract_ligand_table(x)
```

Arguments

x	Chemical-component-like table or object.

Value

Tibble with ligand descriptor columns when available.

extract_taxonomy_table *Extract Taxonomy Table*

Description

Extract Taxonomy Table

Usage

```
extract_taxonomy_table(x)
```

Arguments

x	Polymer-entity-like table or object.

Value

Tibble with taxonomy columns when available.

fetch_data *Fetch Data from RCSB PDB Using a JSON Query*

Description

This function sends a GraphQL JSON query to the RCSB Protein Data Bank (PDB) to fetch data corresponding to a specified set of IDs. It is designed to handle the complexities of interacting with the PDB's GraphQL API, ensuring that errors in the query process are handled gracefully and that users are informed of any discrepancies in the expected and returned data.

Usage

```
fetch_data(json_query, data_type, ids)
```

Arguments

json_query	A JSON string representing the query to be sent to the PDB. This query must be well-formed and adhere to the GraphQL query structure required by the PDB's API. It typically includes details such as the fields to be retrieved and the conditions for data selection.
data_type	A string indicating the type of data to be fetched. While this parameter is not directly used in the function, it can provide context for the data being retrieved, such as "ENTRY", "ASSEMBLY", "POLYMER_ENTITY", etc.
ids	A character vector of identifiers for which data is being requested. These IDs should correspond to valid entries in the PDB and should match the data structure expected by the PDB's API.

Details

The function performs several checks and operations:

* It validates the ‘json_query‘ to ensure that it is neither ‘NULL‘ nor empty. * It attempts to send the query to the PDB’s GraphQL endpoint using a helper function (assumed to be ‘search_graphql‘). * It checks the server’s response to determine if the query was successful or if any errors were encountered. * If the data returned does not match the expected IDs, the function issues warnings and stops execution, providing details on the missing IDs. * The function ensures that the returned data is correctly named according to the provided IDs.

The function is particularly useful for developers and researchers who need to programmatically access and manipulate large datasets from the PDB. It abstracts away some of the complexity of directly interacting with the PDB’s API, providing a more user-friendly interface for data retrieval.

Value

A list with class `"rPDBapi_fetch_response"` containing the fetched payload under `$data`. The list entries are validated and normalized to the requested `ids`. Malformed responses raise typed errors.

`find_papers` *Search for and Retrieve Paper Titles from PDB*

Description

This function searches the Protein Data Bank (PDB) for scholarly articles related to a specified search term. It retrieves the titles of up to a specified maximum number of papers associated with PDB entries. The function relies on ‘query_search‘ to perform the initial search and ‘get_info‘ to fetch detailed information for each PDB entry, including the citation titles.

Usage

```
find_papers(search_term, max_results = 10)
```

Arguments

`search_term`	A string specifying the term to search for in the PDB. This term can relate to any aspect of the PDB entries, such as keywords, molecular functions, or specific proteins (e.g., "CRISPR").
`max_results`	An integer indicating the maximum number of paper titles to retrieve. Defaults to 10. The function will retrieve the titles for the first ‘max_results‘ PDB entries returned by the search.

Details

This function is useful for researchers who want to quickly find relevant literature associated with specific PDB entries. The process involves two main steps:

1. **Search Query**: The function uses ‘query_search‘ to find PDB entries matching the search term.
2. **Fetching Paper Titles**: For each PDB ID returned by the search, ‘get_info‘ is used to retrieve detailed information, including the titles of any associated citations.

The function includes robust error handling to manage cases where the search term does not return any results, or where there are issues retrieving details for specific PDB entries. Warnings are provided if no citations are found for a given PDB ID or if other issues are encountered.

Value

A named list where each element's name is a PDB ID and its value is the title of the corresponding paper. If no papers are found or an error occurs, the function returns an empty list with warnings or error messages to help diagnose the issue.

Examples

```
# Retrieve the primary citation title for a known entry term
entry_papers <- find_papers("4HHB", max_results = 1)
print(entry_papers)
```

`find_results` *Retrieve Specific Fields for Search Results from RCSB PDB*

Description

This function searches the Protein Data Bank (PDB) for entries related to a specified search term and retrieves specific information from those entries. It is useful for extracting targeted data from search results, such as citations, experimental methods, or structural details. The function leverages ‘query_search‘ to perform the initial search and ‘get_info‘ to fetch detailed data for each PDB entry.

Usage

```
find_results(search_term, field = "citation")
```

Arguments

`search_term`	A string specifying the term to search for in the PDB. This term can relate to various aspects of the PDB entries, such as keywords, molecular functions, protein names, or specific research areas.
`field`	A string indicating the specific field to retrieve for each search result. The default is "citation". The field should correspond to one of the following valid options:

"citation" Information about the primary citation associated with the PDB entry.

"audit_author" Details about the authors who contributed to the PDB entry.

"cell" Cell dimensions and related crystallographic information.

"diffrn" Information about the diffraction experiment.

"diffrn_detector" Details about the detector used in the diffraction experiment.

"diffrn_radiation" Radiation source details used in the diffraction experiment.

"diffrn_source" Source of the radiation used in the experiment.

"entry" Basic information about the PDB entry, including its identifier.

"exptl" Details about the experimental methods used to determine the structure.

"exptl_crystal" Information about the crystals used in the experiment.

"exptl_crystal_grow" Details on the crystal growth conditions.

"pdbx_sgproject" Information on the Structural Genomics Project.

"pdbx_audit_revision_details" Details of any revisions made to the PDB entry.

"pdbx_audit_revision_history" History of the revisions for the PDB entry.

"pdbx_database_related" Related database entries.

"pdbx_database_status" Current status of the PDB entry in the database.

"rcsb_accession_info" Accession information for the PDB entry.

"rcsb_entry_container_identifiers" Identifiers associated with the entry container.

"rcsb_entry_info" General information about the PDB entry.

"rcsb_primary_citation" Details of the primary citation for the PDB entry.

"refine" Information about the refinement of the structure.

"refine_hist" History of the refinement process.

"refine_ls_restr" Details about the least-squares restraints used in refinement.

"reflns" Information about the reflections used in the crystallographic experiment.

"reflns_shell" Details about the shell reflections used in the experiment.

"software" Software used in the structure determination process.

"struct" Structural information about the PDB entry.

"struct_keywords" Keywords associated with the structure.

"symmetry" Symmetry information of the crystal structure.

"rcsb_id" The RCSB ID of the PDB entry.

Details

This function is ideal for researchers who need to extract specific data fields from multiple PDB entries efficiently. The process involves two main steps:

1. **Search Query**: The function uses ‘query_search‘ to find PDB entries that match the provided search term.

2. **Field Retrieval**: For each PDB ID returned by the search, ‘get_info‘ is used to retrieve the specified field.

Error handling is robust, with informative messages provided when the search term yields no results, when an individual PDB entry cannot be retrieved, or when the specified field is not found in the retrieved data.

Value

A named list where each element's name is a PDB ID and its value is the information for the specified field from the corresponding search result. If no results are found, or if an error occurs during data retrieval, the function returns an empty list with appropriate warnings or error messages.

Examples

```
# Retrieve keywords for a single known entry term
entry_keywords <- find_results("4HHB", field = "struct_keywords")
utils::head(entry_keywords, 1)
```

---

generate_json_query *Generate a JSON Query for RCSB PDB Data Retrieval*

Description

This function constructs a JSON query string tailored for retrieving data from the RCSB Protein Data Bank (PDB). It is designed to accommodate a variety of data types, such as entries, polymer entities, assemblies, and chemical components. The function is particularly useful when specific properties need to be queried across multiple PDB identifiers.

Usage

```
generate_json_query(ids, data_type, properties)
```

Arguments

ids	A character vector of identifiers corresponding to the data you wish to retrieve. These identifiers should match the type of data specified in the ‘data_type‘ argument. For example, if ‘data_type‘ is 'ENTRY', the identifiers should be PDB entry IDs like "1XYZ" or "2XYZ".
data_type	A string indicating the type of data to query. The function supports the following types: **"ENTRY"** Refers to the entire PDB entry, typically associated with the full structural dataset. **"POLYMER_ENTITY"** Represents specific polymer entities, such as proteins or nucleic acids, within an entry. **"BRANCHED_ENTITY"** Refers to branched entities, often used to describe glycans or other complex molecules.

"NONPOLYMER_ENTITY" Describes non-polymeric entities, such as ligands or cofactors.

"POLYMER_ENTITY_INSTANCE" Specific instances of polymer entities, usually chains within a structure.

"BRANCHED_ENTITY_INSTANCE" Instances of branched entities within a structure.

"NONPOLYMER_ENTITY_INSTANCE" Instances of non-polymeric entities within a structure.

"ASSEMBLY" Refers to macromolecular assemblies, which can include multiple chains and entities.

"CHEMICAL_COMPONENT" Specific chemical components, often small molecules or ligands, within the PDB.

`properties` A named list where each element represents a property to be included in the query for the corresponding ‘data_type‘. Each element of the list should be a character vector containing the specific properties you wish to retrieve. For example, for ‘data_type = "ENTRY"‘, properties might include ‘cell = c("volume", "angle_beta")‘ to retrieve cell volume and angle beta of the unit cell.

Details

The function is designed to be flexible and extensible, allowing users to construct complex queries with multiple properties and data types. The function internally maps the ‘data_type‘ to the appropriate query format and ensures that the JSON string is properly constructed. The ‘properties‘ argument should align with the data type; otherwise, the query might not return the desired results.

The ‘generate_json_query‘ function is particularly useful for researchers and bioinformaticians who need to retrieve specific datasets from the PDB, especially when dealing with large-scale structural biology data. The generated JSON query can be used in subsequent API calls to fetch the required data in a structured and efficient manner.

Value

A string representing the generated JSON query. This string is formatted according to the requirements of the RCSB PDB API and can be used directly in a GraphQL query to retrieve the specified data.

Examples

```
# Example 1: Generate a query for PDB entries with specific properties
ids <- c("1XYZ", "2XYZ")
properties <- list(cell = c("volume", "angle_beta"), exptl = c("method"))
json_query <- generate_json_query(ids, "ENTRY", properties)
print(json_query)

# Example 2: Generate a query for chemical components with specified properties
ids <- c("ATP", "NAD")
properties <- list(chem_comp = c("formula_weight", "type"))
json_query <- generate_json_query(ids, "CHEMICAL_COMPONENT", properties)
print(json_query)
```

```
# Example 3: Generate a query for polymer entities within a PDB entry
ids <- c("1XYZ_1", "2XYZ_2")
properties <- list(entity_src_gen = c("organism_scientific", "gene_src_common"))
json_query <- generate_json_query(ids, "POLYMER_ENTITY", properties)
print(json_query)
```

`get_fasta_from_rcsb_entry` *Retrieve FASTA Sequence from PDB Entry or Specific Chain*

Description

This function retrieves FASTA sequences from the RCSB Protein Data Bank (PDB) for a specified entry ID (`rcsb_id`). It can return either the full set of sequences associated with the entry or, if specified, the sequence corresponding to a particular chain within that entry. This flexibility makes it a useful tool for bioinformaticians and structural biologists needing access to protein or nucleic acid sequences.

Usage

```
get_fasta_from_rcsb_entry(
  rcsb_id,
  chain_id = NULL,
  verbosity = TRUE,
  fasta_base_url = FASTA_BASE_URL
)
```

Arguments

`rcsb_id`	A string representing the PDB ID for which the FASTA sequence is to be retrieved. This is the primary identifier of the entry in the PDB database.
`chain_id`	A string representing the specific chain ID within the PDB entry for which the FASTA sequence is to be retrieved. If `chain_id` is NULL (the default), the function will return all sequences associated with the entry. The chain ID should match one of the chain identifiers in the PDB entry (e.g., "A", "B").
`verbosity`	A boolean flag indicating whether to print status messages during the function execution. When set to `TRUE` (the default), the function will output messages detailing the progress and any issues encountered.
`fasta_base_url`	A string representing the base URL for the FASTA retrieval. By default, this is set to the global constant `FASTA_BASE_URL`, but users can specify a different URL if needed.

Details

The function queries the RCSB PDB database using the provided entry ID (`rcsb_id`) and optionally a chain ID (`chain_id`). It sends an HTTP GET request to retrieve the corresponding FASTA file. The response is then parsed into a list of sequences. If a chain ID is provided, the function will return only the sequence corresponding to that chain. If no chain ID is provided, all sequences are returned.

If a request fails, the function provides informative error messages. In the case of a network failure, the function will stop execution with a clear error message. Additionally, if the chain ID does not exist within the entry, the function will return an appropriate error message indicating that the chain was not found.

The function also supports passing a custom base URL for the FASTA file retrieval, providing flexibility for users working with different PDB mirrors or services.

Value

* If `chain_id` is NULL, the function returns a list of FASTA sequences associated with the provided `rcsb_id`, where organism names or chain descriptions are used as keys. * If `chain_id` is specified, the function returns a character string representing the FASTA sequence for that specific chain. * If the specified `chain_id` is not found in the PDB entry, the function will stop execution with an informative error message.

Examples

```
# Example 1: Retrieve all FASTA sequences for the entry 4HHB
all_sequences <- get_fasta_from_rcsb_entry("4HHB", verbosity = TRUE)
print(all_sequences)

# Example 2: Retrieve the FASTA sequence for chain A of entry 4HHB
chain_a_sequence <- get_fasta_from_rcsb_entry("4HHB", chain_id = "A", verbosity = TRUE)
print(chain_a_sequence)
```

`get_info` *Retrieve Information for a Given PDB ID*

Description

This function retrieves comprehensive information for a specified PDB (Protein Data Bank) entry by querying the RCSB PDB RESTful API. The function handles HTTP requests, processes JSON responses, and can manage legacy PDB identifiers. It is particularly useful for obtaining all available data related to a specific PDB entry, which can include metadata, structural details, experimental methods, and more.

Usage

```
get_info(pdb_id)
```

Arguments

pdb_id	A string specifying the PDB entry of interest. The ‘pdb_id‘ should be a 4-character alphanumeric code, representing the unique identifier of a PDB entry (e.g., "1XYZ"). If a legacy PDB identifier is provided in the format ‘PDB_ID:CHAIN_ID‘, it will be automatically converted to the new format for querying.

Details

The ‘get_info‘ function is versatile and designed for researchers who need to extract detailed structural and experimental information from the RCSB PDB. The function is robust, providing error handling for various scenarios, including network failures, incorrect PDB IDs, and API errors. It automatically manages legacy PDB IDs, ensuring compatibility with the latest API standards.

The output is a structured list that can be easily parsed or manipulated for further analysis, making it an essential tool for bioinformaticians and structural biologists working with PDB data.

The function also offers flexibility in querying different parts of the RCSB PDB API by adjusting the ‘url_root‘ parameter, allowing users to target specific datasets within the PDB.

Value

A list object (an ordered dictionary in R) containing detailed information about the specified PDB entry. The returned list includes various data fields, depending on the content available for the entry. For example, it may contain information about the structure's authors, resolution, experiment type, macromolecules, ligands, etc. If the data retrieval fails at any stage (e.g., network issues, invalid PDB ID, API downtime), the function will return ‘NULL‘ and provide an informative error message.

Examples

```
pdb_info <- get_info(pdb_id = "1XYZ")
print(pdb_info)
```

---

get_pdb_api_url *Generate a PDB API URL*

Description

This function constructs a full PDB API URL by concatenating the base URL, an API endpoint, and an identifier.

Usage

```
get_pdb_api_url(endpoint, id, base_url = BASE_URL)
```

Arguments

endpoint	A character string representing the specific API endpoint to be accessed (e.g., "/pdb/entry/").
id	A character string representing the identifier for the resource (e.g., a PDB ID or other relevant ID).
base_url	A string representing the base URL to generate PDB API url. By default, this is set to the global constant BASE_URL, but users can specify a different URL if needed.

Value

A character string containing the full URL for accessing the PDB API resource.

---

get_pdb_file *Download and Process PDB Files from the RCSB Database*

Description

The 'get_pdb_file' function is a versatile tool designed to download Protein Data Bank (PDB) files from the RCSB database. It supports various file formats such as 'pdb', 'cif', 'xml', and 'structfact', with options for file compression and handling alternate locations (ALT) and insertion codes (INSERT) in PDB files. This function also provides the flexibility to save the downloaded files to a specified directory or to a temporary directory for immediate use.

Usage

```
get_pdb_file(
  pdb_id,
  filetype = "cif",
  rm.insert = FALSE,
  rm.alt = TRUE,
  compression = TRUE,
  save = FALSE,
  path = NULL,
  verbosity = TRUE,
  download_base_url = DOWNLOAD_BASE_URL
)
```

Arguments

pdb_id	A 4-character string specifying the PDB entry of interest (e.g., "1XYZ"). This identifier uniquely represents a macromolecular structure within the PDB database.
filetype	A string specifying the format of the file to be downloaded. The default is 'cif'. Supported file types include: **'pdb'** The older PDB file format, which provides atomic coordinates and metadata.

'cif' The Crystallographic Information File (CIF) format, which is a newer standard replacing PDB files.

'xml' An XML format file, providing structured data that can be easily parsed for various applications.

'structfact' Structure factor files in CIF format, available for certain PDB entries, containing experimental data used to determine the structure.

`rm.insert` Logical flag indicating whether to ignore PDB insertion codes. Default is FALSE. If TRUE, records with insertion codes will be removed from the final data.

`rm.alt` Logical flag indicating whether to ignore alternate location indicators (ALT) in PDB files. Default is TRUE. If TRUE, only the first alternate location is kept, and others are removed.

`compression` Logical flag indicating whether to download the file in a compressed format (e.g., .gz). Default is TRUE, which is recommended for faster downloads, especially for CIF files.

`save` Logical flag indicating whether to save the downloaded file to a specified directory. Default is FALSE, which means the file is processed and optionally saved, but not retained after processing unless specified.

`path` A string specifying the directory where the downloaded file should be saved. If NULL, the file is saved in a temporary directory. If 'save' is TRUE, this path is required.

`verbosity` A boolean flag indicating whether to print status messages during the function execution.

`download_base_url`
A string representing the base URL for the PDB file retrieval. By default, this is set to the global constant `DOWNLOAD_BASE_URL`, but users can specify a different URL if needed.

Details

The 'get_pdb_file' function is an essential tool for structural biologists and bioinformaticians who need to download and process PDB files for further analysis. By providing options to handle alternate locations and insertion codes, this function ensures that the data is clean and ready for downstream applications. Additionally, the ability to save files locally or work with them in a temporary directory provides flexibility for various workflows. Error handling and informative messages are included to guide the user in case of issues with file retrieval or processing.

Value

A list of class `"pdb"` containing the following components:

`atom` A data frame containing atomic coordinate data (ATOM and HETATM records). Each row corresponds to an atom, and each column to a specific record type (e.g., element, residue, chain).

`xyz` A numeric matrix of class `"xyz"` containing the atomic coordinates from the ATOM and HETATM records.

`calpha` A logical vector indicating whether each atom is a C-alpha atom (TRUE) or not (FALSE).

`call` The matched call, storing the function call for reference.

`path` The file path where the file was saved, if 'save' was TRUE.

The function handles errors and warnings for various edge cases, such as unsupported file types, failed downloads, or issues with reading the file.

Examples

```
# Download a CIF file and process it without saving
pdb_file <- get_pdb_file(pdb_id = "4HHB", filetype = "cif")

# Download a PDB file, save it, and remove alternate location records
pdb_file <- get_pdb_file(pdb_id = "4HHB", filetype = "pdb", save = TRUE, path = tempdir())

# Understanding the tertiary structure of proteins is
# crucial for elucidating their functional mechanisms,
# especially in the context of ligand binding, enzyme catalysis,
# and protein-protein interactions.
# The tertiary structure refers to the three-dimensional arrangement
# of all atoms within a protein,
# including its secondary structure elements like alpha helices
# and beta sheets, and how these elements
# are organized in space. Using the get_pdb_file function
# to retrieve the PDB file and the r3dmol
# package for visualization, researchers can gain insights
# into the overall 3D structure of a protein.
# The following example demonstrates how to visualize the
# ltertiary structure of a protein using the
# PDB entry 4HHB:

if (any(rownames(installed.packages()) == "r3dmol")) {
  r3d <- asNamespace("r3dmol")

  # Retrieve and parse a PDB structure
  pdb_path <- get_pdb_file("4HHB", filetype = "pdb", save = TRUE, verbosity = FALSE)

  # Visualize the tertiary structure using r3dmol
  viewer <- r3d$r3dmol()
  viewer <- r3d$m_add_model(viewer, pdb_path$path, format = "pdb")
  viewer <- r3d$m_set_style(viewer, style = r3d$m_style_cartoon())
  viewer <- r3d$m_zoom_to(viewer)

  # Display the molecular viewer
  viewer
}
# In this example, the protein structure is represented
# in a cartoon style, which is particularly
# effective for visualizing the overall fold of the protein,
# including the orientation and interaction
# of its secondary structure elements.
#. To further enhance the analysis,
# it is often important to
# highlight specific regions of interest,
```

```
# such as potential ligand-binding sites.
# These sites can be identified based on prior knowledge,
# experimental data, or computational predictions.
# The following code snippet demonstrates
# how to highlight potential ligand-binding sites in the
# protein structure:

# Highlight potential ligand-binding sites
# Note: Manually define residues of interest based
# on prior knowledge or external analysis
binding_sites <- c(45, 60, 85)  # Example residue numbers

if (exists("viewer")) {
  viewer <- r3d$m_set_style(
    viewer,
    sel = r3d$m_sel(resi = binding_sites),
    style = r3d$m_style_sphere(color = "red", radius = 1.5)
  )
}

# Display the updated viewer with highlighted binding sites
if (exists("viewer")) {
  viewer
}

# In this step, specific residues that are
# hypothesized to participate in ligand binding are
#highlighted using a spherical representation.
# The residues are selected manually based on either
# experimental data or computational predictions.
# By highlighting these sites, researchers can
# visually inspect the spatial relationship between
# these residues and other parts of the protein,
# which may provide insights into the
# protein's functional mechanisms.

# This visualization approach offers a powerful
# way to explore and communicate the 3D structure
# of proteins, making it easier to hypothesize about their function and
# interaction with other molecules.
```

`handle_api_errors` *Handle API Errors*

Description

This function checks for errors in the HTTP response and stops execution if the request was not successful.

Usage

```
handle_api_errors(response, url = "")
```

Arguments

response	An HTTP response object.
url	A string representing the requested URL (for more informative error messages).

Value

None. It stops execution if an error is detected.

---

infer_id_type *Infer RCSB Identifier Type*

Description

Infers the likely identifier category for one or more RCSB-related IDs. This helper is additive and does not affect existing workflows unless called explicitly.

Usage

```
infer_id_type(id)
```

Arguments

id	A character vector of identifiers.

Value

A character vector with inferred types: "ENTRY", "ASSEMBLY", "ENTITY", "INSTANCE", "CHEMICAL_COMPONENT", or "UNKNOWN".

---

infer_search_service *Infer the Appropriate Search Service for RCSB PDB Queries*

Description

The ‘infer_search_service‘ function determines the appropriate search service for a given search operator. This function is essential for ensuring that queries are directed to the correct search service, such as basic search, text search, sequence search, etc.

Usage

```
infer_search_service(search_operator)
```

Arguments

search_operator
: A query operator object that specifies the type of search being performed.

Value

A string representing the inferred search service, which is necessary for constructing a valid query.

InOperator *Create an Inclusion Search Operator*

Description

Constructs an 'InOperator' object for search operations where the attribute value must be within a specified set. This operator is useful when the search criteria require the attribute to match one of several possible values. It can handle multiple potential matches and is ideal for scenarios where multiple values are acceptable.

Usage

```
InOperator(attribute, value)
```

Arguments

attribute	The attribute to be evaluated. This should be the field within the RCSB PDB that you want to search against.
value	The set of values to include in the search. This should be a vector of possible values that the attribute can match.

Value

An object of class 'InOperator' that can be used in search queries to retrieve entries where the attribute matches any of the specified values.

Examples

```
# Search for entries where the attribute matches one of several values
operator <- InOperator(attribute = "rcsb_entity_source_organism.taxonomy_lineage.name",
                       value = c("Homo sapiens", "Mus musculus"))
print(operator)
```

join_structure_sequence *Join Structure and Sequence Summaries*

Description

Join Structure and Sequence Summaries

Usage

```
join_structure_sequence(structure, sequences)
```

Arguments

structure	Structure object returned by get_pdb_file().
sequences	FASTA sequence list (typically from get_fasta_from_rcsb_entry()).

Value

Tibble joining available chain-level C-alpha counts and sequence lengths.

list_rcsb_fields *List Known RCSB Fields by Data Type*

Description

Returns the package's built-in field registry used for schema-aware property validation. This registry is intentionally conservative and additive.

Usage

```
list_rcsb_fields(data_type = NULL)
```

Arguments

data_type	Optional data type filter (e.g., "ENTRY"). If NULL, all supported data types are returned.

Value

A data frame with columns data_type, field, and subfield.

parse_fasta_text_to_list *Helper Function: Parse FASTA Text to List Grouped by Header*

Description

This function parses a FASTA-formatted text into a list of sequences, where each sequence is keyed by the header (which includes organism name and chain information).

Usage

```
parse_fasta_text_to_list(fasta_text)
```

Arguments

fasta_text	A string containing FASTA-formatted text.

Value

A list where each element is a FASTA sequence keyed by the header.

parse_rcsb_id *Parse an RCSB Identifier*

Description

Parses an identifier into a structured record when possible.

Usage

```
parse_rcsb_id(id)
```

Arguments

id	A scalar identifier.

Value

A named list with parsed fields and inferred type.

parse_response *Parse API Response*

Description

This function parses the content of an HTTP response based on the specified format. It supports JSON and plain text formats.

Usage

```
parse_response(response, format = "json")
```

Arguments

response	An HTTP response object.
format	A string indicating the expected response format ("json" or "text").

Value

Parsed content from the response.

perform_search *Perform a Search in the RCSB PDB*

Description

This function allows users to perform highly customizable searches in the RCSB Protein Data Bank (PDB) by specifying detailed search criteria. It interfaces directly with the RCSB PDB's RESTful API, enabling complex queries to retrieve specific data, such as PDB entries, assemblies, polymer entities, non-polymer entities, and more.

Usage

```
perform_search(
  search_operator,
  return_type = "ENTRY",
  request_options = NULL,
  return_with_scores = FALSE,
  return_raw_json_dict = FALSE,
  verbosity = TRUE
)
```

Arguments

`search_operator`

An object that specifies the search criteria. This object can be constructed using various operator functions:

`DefaultOperator` A basic search operator for general search operations.

`ExactMatchOperator` For exact match searches, matching an exact attribute value.

`InOperator` For searching where the attribute value must be within a specified set of values.

`ContainsWordsOperator` For searching attributes that contain certain words.

`ContainsPhraseOperator` For searching attributes that contain a specific phrase.

`ComparisonOperator` For comparison-based searches, such as finding values greater than, less than, or equal to a specified value.

`RangeOperator` For searching within a range of values for a given attribute.

`ExistsOperator` To check the existence of a specific attribute in the database.

`StructureOperator` For structure-based searches, using PDB entry IDs, assembly IDs, and search modes.

`SequenceOperator` For sequence-based searches, using sequences, sequence types, and cutoffs for e-value and identity.

`SeqMotifOperator` For searching sequence motifs, using pattern types like SIMPLE, PROSITE, or REGEX.

`ChemicalOperator` For chemical structure searches, using SMILES or InChI descriptors and various matching criteria.

Please see the Details section.

`return_type` A string specifying the type of data to return. The available options for `return_type` include:

`"ENTRY"` Returns a list of PDB IDs corresponding to the entries that match the search criteria. This is the default option and provides entry-level information.

`"ASSEMBLY"` Returns a list of PDB IDs appended with assembly IDs (formatted as `"PDB_ID-ASSEMBLY_ID"`). Useful for accessing specific biological assemblies.

`"POLYMER_ENTITY"` Returns a list of PDB IDs appended with entity IDs for polymeric molecular entities. Useful for examining specific polymer chains.

`"NON_POLYMER_ENTITY"` Returns a list of PDB IDs appended with entity IDs for non-polymeric entities, such as ligands or small molecules. Useful for detailed chemical analysis.

`"POLYMER_INSTANCE"` Returns a list of PDB IDs appended with asym IDs, representing specific instances of polymeric entities (e.g., protein chains).

`"CHEMICAL_COMPONENT"` Returns a list of chemical component identifiers, useful for detailed chemical analysis.

`request_options`

A list of additional options to further customize the search request. These options can include:

`facets` Faceted queries allow aggregation of search results into categories (buckets) based on the requested field values. Useful for statistical analysis and data aggregation.

`sort_by` Defines the sorting criteria for the search results (e.g., by resolution, release date).

`pagination` Controls how many results to return per page and which page of results to return. Useful for handling large datasets.

`return_all_hits` If set to `TRUE`, the search returns all matching results; otherwise, a limited set is returned.

`return_with_scores`
: Logical; if `TRUE`, the search results will include relevance scores. Useful when prioritizing results based on their relevance to the search criteria. Default is `FALSE`.

`return_raw_json_dict`
: Logical; if `TRUE`, the function returns the raw JSON response from the PDB API. This option is valuable for advanced users who wish to process the raw data themselves or need access to additional details. Default is `FALSE`.

`verbosity`
: Logical; if `TRUE`, detailed messages will be displayed during execution, providing insights into the query being sent and the response received. Verbose mode is useful for debugging or when you need insights into the function's operation. Default is `TRUE`.

Details

The operators allow you to build complex search queries tailored to your specific needs. Detailed documentation for each search operator can be found in the RCSB PDB Search Operators. The searchable attributes include annotations from the mmCIF dictionary, external resources, and those added by RCSB PDB. Both internal additions to the mmCIF dictionary and external resource annotations are prefixed with 'rcsb_'. For a complete list of available attributes for text searches, refer to the Structure Attributes Search and Chemical Attributes Search pages.

Value

The function returns search results based on the specified `return_type`:

`ENTRY` A vector of PDB IDs that match the search criteria.

`ASSEMBLY` A list of PDB IDs with appended assembly IDs, formatted as `"PDB_ID-ASSEMBLY_ID"`.

`POLYMER_ENTITY` A list of PDB IDs with appended entity IDs for polymeric chains.

`NON_POLYMER_ENTITY` A list of PDB IDs with appended entity IDs for non-polymeric components.

`POLYMER_INSTANCE` A list of PDB IDs with appended asym IDs for specific polymer instances.

`CHEMICAL_COMPONENT` A list of chemical component identifiers.

For identifier output, the vector is tagged with class `"rPDBapi_search_ids"`. When `return_with_scores = TRUE`, returned results are tagged with class `"rPDBapi_search_scores"`. Raw JSON responses are tagged with class `"rPDBapi_search_raw_response"`.

Examples

```
# Example 1: Search for Polymer Entities from Mus musculus and Homo sapiens
search_operator <- InOperator(
  attribute = "rcsb_entity_source_organism.taxonomy_lineage.name",
  value = c("Mus musculus", "Homo sapiens")
)
results <- perform_search(
  search_operator = search_operator,
  return_type = "POLYMER_ENTITY"
)
results

# Example 2: Search for Entries Released After a Specific Date
operator_date <- ComparisonOperator(
  attribute = "rcsb_accession_info.initial_release_date",
  value = "2019-08-20",
  comparison_type = "GREATER"
)
request_options <- list(
  facets = list(
    list(
      name = "Methods",
      aggregation_type = "terms",
      attribute = "exptl.method"
    )
  )
)
results <- perform_search(
  search_operator = operator_date,
  return_type = "ENTRY",
  request_options = request_options
)
results

# Example 3: Search for Symmetric Dimers with DNA-Binding Domain
operator_symbol <- ExactMatchOperator(
  attribute = "rcsb_struct_symmetry.symbol",
  value = "C2"
)
operator_kind <- ExactMatchOperator(
  attribute = "rcsb_struct_symmetry.kind",
  value = "Global Symmetry"
)
operator_full_text <- DefaultOperator(
  value = "\"heat-shock transcription factor\""
)
operator_dna_count <- ComparisonOperator(
  attribute = "rcsb_entry_info.polymer_entity_count_DNA",
  value = 1,
  comparison_type = "GREATER_OR_EQUAL"
)
query_group <- list(
```

```
  type = "group",
  logical_operator = "and",
  nodes = list(
    list(
      type = "terminal",
      service = "text",
      parameters = operator_symbol
    ),
    list(
      type = "terminal",
      service = "text",
      parameters = operator_kind
    ),
    list(
      type = "terminal",
      service = "full_text",
      parameters = operator_full_text
    ),
    list(
      type = "terminal",
      service = "text",
      parameters = operator_dna_count
    )
  )
)
results <- perform_search(
  search_operator = query_group,
  return_type = "ASSEMBLY"
)
results
```

`query_search` *Search Query Function*

Description

This function performs a search query against the RCSB Protein Data Bank using their REST API. It allows for various types of searches based on the provided parameters.

Usage

```
query_search(
  search_term,
  query_type = "full_text",
  return_type = "entry",
  scan_params = NULL,
  num_attempts = 1,
  sleep_time = 0.5
)
```

Arguments

search_term	A string specifying the term to search in the database.
query_type	A string specifying the type of query to perform. Supported values include "full_text", "PubmedIdQuery", "TreeEntityQuery", "ExpTypeQuery", "AdvancedAuthorQuery", "OrganismQuery", "pfam", and "uniprot". Default is "full_text".
return_type	A string specifying the type of search result to return. Possible values are "entry" (default) and "polymer_entity".
scan_params	Additional parameters for the scan, provided as a list. This is 'NULL' by default and typically only used for advanced queries.
num_attempts	An integer specifying the number of attempts to try the query in case of failure.
sleep_time	A numeric value specifying the time in seconds to wait between attempts.

Value

If `return_type = "entry"`, returns a character vector of identifiers with class `"rPDBapi_query_ids"`. Otherwise returns the parsed API payload with class `"rPDBapi_query_response"`.

Examples

```
# Get a list of PDBs for a specific search term

# Search Functions by Specific Terms
pdbs <- query_search("ribosome")
head(pdbs)

# Search by PubMed ID Number
pdbs_by_pubmedid <- query_search(search_term = 27499440, query_type = "PubmedIdQuery")
head(pdbs_by_pubmedid)

# Search by source organism using NCBI TaxId
pdbs_by_ncbi_taxid <- query_search(search_term = "6239", query_type = "TreeEntityQuery")
head(pdbs_by_ncbi_taxid)

# Search by Experimental Method
pdbs = query_search(search_term = 'SOLID-STATE NMR', query_type='ExpTypeQuery')
head(pdbs)

pdbs = query_search(search_term = '4HHB', query_type="structure")
head(pdbs)

## Advanced Searches

# Search by Author
pdbs = query_search(search_term = 'Rzechorzek, N.J.', query_type='AdvancedAuthorQuery')
head(pdbs)

# Search by Organism
pdbs = query_search(search_term = "Escherichia coli", query_type="OrganismQuery")
head(pdbs)
```

```
# Search by Uniprot ID (Escherichia coli beta-lactamase)
pdbs = query_search(search_term = "P0A877", query_type="uniprot")
head(pdbs)

# Search by PFAM number (protein kinase domain)
pdbs = query_search(search_term = "PF00069", query_type="pfam")
head(pdbs)
```

`QueryGroup` *Create a Grouped Query Object for RCSB PDB Searches*

Description

The 'QueryGroup' function constructs a grouped query object that allows users to perform complex searches in the RCSB Protein Data Bank (PDB). This function is particularly useful when multiple query objects need to be combined using logical operators like 'AND' or 'OR'. The resulting grouped query can be used in advanced search operations to filter or combine results based on multiple criteria.

Usage

```
QueryGroup(queries, logical_operator)
```

Arguments

`queries`	A list of query objects to be grouped together. Each query object can be either a simple query or another grouped query.
`logical_operator`	A string specifying the logical operator to combine the queries. Common values are 'AND' and 'OR', but other logical operators may also be supported.

Value

A list representing the grouped query object, which can be passed to search functions for execution.

QueryNode *Create a Query Node for RCSB PDB Searches*

Description

The 'QueryNode' function constructs a query node, which can be either a terminal node (for a simple query) or a grouped node (for complex queries). This function is crucial for structuring queries that will be sent to the RCSB PDB search system.

Usage

```
QueryNode(search_operator, logical_operator = NULL)
```

Arguments

`search_operator`
: A search operator or group object. This defines the criteria for the search.

`logical_operator`
: A string specifying the logical operator to combine multiple queries. Default is 'NULL'. This is used only if the search_operator is a group.

Value

A list representing the query node, ready to be included in a larger query structure.

Examples

```
node <- QueryNode(search_operator = DefaultOperator("some_value"))
```

RangeOperator *Create a Range Search Operator*

Description

Constructs a 'RangeOperator' object for search operations that specify a range for attribute values. This operator is particularly useful for filtering results based on numeric or date ranges, such as finding entries with resolution between specific values or dates within a certain range.

Usage

```
RangeOperator(
  attribute,
  from_value,
  to_value,
  include_lower = TRUE,
  include_upper = TRUE,
  negation = FALSE
)
```

Arguments

attribute	The attribute to be evaluated within a range. This should be the numeric or date field within the RCSB PDB that you want to search against.
from_value	The starting value of the range. This is the lower bound of the range.
to_value	The ending value of the range. This is the upper bound of the range.
include_lower	Boolean indicating whether to include the lower bound in the range. Default is TRUE.
include_upper	Boolean indicating whether to include the upper bound in the range. Default is TRUE.
negation	Boolean indicating whether to negate the range condition. Default is FALSE.

Value

An object of class 'RangeOperator' that can be used in search queries to retrieve entries where the attribute falls within the specified range.

Examples

```
# Search for entries within a specific range of resolution
operator <- RangeOperator(attribute = "rcsb_entry_info.resolution_combined",
                          from_value = 1.5, to_value = 2.5)
print(operator)
```

RequestOptions *Define Request Options for RCSB PDB Search Queries*

Description

The 'RequestOptions' function sets various options for search requests to the RCSB PDB, such as pagination and sorting preferences. These options help control the volume of search results returned and the order in which they are presented.

Usage

```
RequestOptions(
  result_start_index = NULL,
  num_results = NULL,
  sort_by = "score",
  desc = TRUE
)
```

Arguments

result_start_index	An integer specifying the starting index for result pagination. If 'NULL', pagination is not applied.
num_results	An integer specifying the number of results to return. If 'NULL', the default number of results is returned.
sort_by	A string indicating the attribute to sort the results by. The default value is 'score', which ranks results based on relevance.
desc	A boolean indicating whether the sorting should be in descending order. Default is 'TRUE'.

Value

A list of request options that can be included in a search query to control the results.

Examples

```
options <- RequestOptions(result_start_index = 0, num_results = 100, sort_by = "score", desc = TRUE)
```

---

return_data_as_dataframe

Convert RCSB PDB Response Data into a Dataframe

Description

The 'return_data_as_dataframe' function transforms the response data obtained from a query to the RCSB Protein Data Bank (PDB) into a structured dataframe. This function handles various scenarios, including responses with duplicate names, null or empty responses, and nested data structures. It ensures that the resulting dataframe is consistently formatted and ready for downstream analysis.

Usage

```
return_data_as_dataframe(response, data_type, ids)
```

Arguments

`response`	A list containing the response data from a PDB query. This list is expected to be structured according to the RCSB PDB GraphQL or REST API specifications.
`data_type`	A string indicating the type of data contained in the response (e.g., "ENTRY", "POLYMER_ENTITY"). This parameter is primarily used for contextual information and does not directly influence the function's operations.
`ids`	A vector of identifiers corresponding to the response data. These IDs are used to label the resulting dataframe, ensuring that each row corresponds to a specific query identifier.

Details

The 'return_data_as_dataframe' function is designed to provide a flexible and robust mechanism for converting PDB query responses into dataframes. It addresses several common challenges in handling API responses, such as:

Null or Empty Responses If the response is null or contains no data, the function immediately returns 'NULL', avoiding unnecessary processing.

Duplicate Names Handling The function detects and manages scenarios where the response contains duplicated names. It simplifies such lists by keeping only the first occurrence of each duplicated element, ensuring that the final dataframe has unique column names.

Recursive Flattening The function flattens nested lists within the response, ensuring that all relevant data is captured in a single-level dataframe structure. This is particularly useful for complex responses that contain deeply nested data elements.

Consistent Column Naming After processing the data, the function ensures that column names are consistent and do not retain unnecessary prefixes. This makes the resulting dataframe easier to interpret and work with in subsequent analyses.

Value

A dataframe constructed from the response data, where each row corresponds to an identifier from the 'ids' vector and each column represents a data field from the response. If the response is null or empty, the function returns 'NULL'.

Note

The function is equipped to handle responses with varying degrees of complexity. It is recommended to provide valid 'ids' corresponding to the query to ensure that the dataframe rows are correctly labeled.

ScoredResult *Create a Scored Result Object for PDB Searches*

Description

The 'ScoredResult' function constructs a scored result object, typically used in search results to associate an entity ID with a numerical score. This is useful in ranking search results or displaying relevance scores alongside the results.

Usage

```
ScoredResult(entity_id, score)
```

Arguments

entity_id	A string representing the entity ID. This could be a PDB ID or any identifier relevant to the search.
score	A numeric value representing the score associated with the entity. The score often indicates the relevance or quality of the match.

Value

A list representing the scored result, which can be included in the search results or used for further processing.

Examples

```
result <- ScoredResult(entity_id = "1XYZ", score = 95.6)
```

search_graphql *Perform a GraphQL Query to RCSB PDB*

Description

The 'search_graphql' function sends a GraphQL query to the RCSB Protein Data Bank (PDB) using the provided JSON query format. This function handles the HTTP request, sends the query, and processes the response, including error handling to ensure that the query executes successfully.

Usage

```
search_graphql(graphql_json_query, graphql_url = GRAPHQL_URL)
```

Arguments

graphql_json_query	A list containing the GraphQL query formatted as JSON. This list should include the 'query' key with a value that represents the GraphQL query string. The query string can specify various elements to retrieve, such as entry IDs, experimental methods, cell dimensions, etc.
graphql_url	A string representing the base URL perform GraphQL query. By default, this is set to the global constant GRAPHQL_URL, but users can specify a different URL if needed.

Value

A parsed list containing the content of the response from the RCSB PDB, formatted as an R object. If the request fails, the function stops with an error message.

---

search_rcsb_fields *Search Known RCSB Fields by Pattern*

Description

Searches the built-in field registry by field or subfield name.

Usage

```
search_rcsb_fields(pattern, data_type = NULL, ignore_case = TRUE)
```

Arguments

pattern	Pattern to search for.
data_type	Optional data type filter.
ignore_case	Logical, passed to grepl().

Value

A filtered data frame from list_rcsb_fields.

send_api_request *Send API Request to a Specified URL*

Description

This function sends an HTTP request (GET or POST) to the specified URL. It supports optional request bodies for POST requests, customizable encoding, and content type for API interactions. The function is designed to be a general-purpose API handler for use in querying external APIs.

Usage

```
send_api_request(
  url,
  method = "GET",
  body = NULL,
  encode = "json",
  content_type = "application/json",
  verbosity = TRUE
)
```

Arguments

url	A string representing the target URL for the API request.
method	A string specifying the HTTP method to use. The default is "GET", but "POST" can also be used.
body	Optional: The body of the request, typically required for POST requests. Default is NULL.
encode	A string representing the encoding type of the body for POST requests. Default is "json".
content_type	A string specifying the content type for POST requests. Default is "application/json".
verbosity	Logical flag indicating whether to print status messages during the function execution. Default is TRUE.

Details

The send_api_request function is a flexible tool for handling API interactions. It supports both GET and POST methods and provides optional parameters for encoding and content type, making it suitable for a wide range of API requests.

If a network error occurs during the request, the function will throw an error with a detailed message about the failure.

Value

A response object from the httr package representing the server's response to the API request.

SeqMotifOperator *Create a Sequence Motif Operator for RCSB PDB Searches*

Description

The 'SeqMotifOperator' function constructs an operator for searching sequence motifs within the RCSB Protein Data Bank (PDB). This operator is used to specify a search pattern, the type of biological sequence, and the pattern-matching method to be applied in the search.

Usage

```
SeqMotifOperator(pattern, sequence_type, pattern_type)
```

Arguments

pattern	A string representing the motif pattern to search for. This can be a simple string or a more complex pattern, depending on the 'pattern_type'.
sequence_type	A string indicating the type of sequence being searched. Accepted values are 'DNA', 'RNA', or 'PROTEIN'.
pattern_type	A string indicating the pattern matching method to use. Options include 'SIMPLE' for basic patterns, 'PROSITE' for PROSITE-style patterns, and 'REGEX' for regular expressions.

Value

An object of class 'SeqMotifOperator' that encapsulates the specified search criteria. This object can be used as part of a search query within the RCSB PDB system.

Examples

```
# Example of creating a sequence motif operator to search for a DNA motif using a regular expression
seq_motif_operator <- SeqMotifOperator(
  pattern = "A[TU]G",
  sequence_type = "DNA",
  pattern_type = "REGEX"
)
print(seq_motif_operator)
```

SequenceOperator *Create a Sequence Operator for Sequence-Based Searches*

Description

The 'SequenceOperator' function constructs an operator for performing sequence-based searches within the RCSB Protein Data Bank (PDB). This operator allows users to specify a nucleotide or protein sequence, define the type of sequence, and set thresholds for e-value and identity in the search process.

Usage

```
SequenceOperator(
  sequence,
  sequence_type = NULL,
  evalue_cutoff = 100,
  identity_cutoff = 0.95
)
```

Arguments

sequence	A string representing the nucleotide or protein sequence to search for. The sequence should be provided in standard IUPAC format.
sequence_type	Optional: A string indicating the type of sequence. Accepted values are 'DNA', 'RNA', or 'PROTEIN'. If not provided, the sequence type is automatically determined based on the characters present in the sequence using the 'autoresolve_sequence_type' function.
evalue_cutoff	A numeric value for the e-value cutoff in the search. This defines the threshold for statistical significance of the search results. Default is 100.
identity_cutoff	A numeric value for the identity cutoff in the search. This sets the minimum percentage of identity required for a match to be considered. Default is 0.95.

Value

An object of class 'SequenceOperator' that encapsulates the search criteria for sequence-based queries within the RCSB PDB.

Examples

```
# Example of creating a sequence operator for a protein sequence with specific cutoffs
seq_operator <- SequenceOperator(
  sequence = "MVLSPADKTNVKAAW",
  sequence_type = "PROTEIN",
  evalue_cutoff = 10,
  identity_cutoff = 0.90
)
```

```
print(seq_operator)

# Example of creating a sequence operator with automatic sequence type detection
seq_operator_auto <- SequenceOperator(
  sequence = "ATGCGTACGTAGC",
  evalue_cutoff = 50,
  identity_cutoff = 0.85
)
print(seq_operator_auto)
```

`StructureOperator` *Create a Structure Operator for Structure-Based Searches*

Description

The 'StructureOperator' function constructs an operator object for conducting structure-based searches within the RCSB Protein Data Bank (PDB). This operator allows users to specify a PDB entry ID, an assembly ID, and the mode of search to be used, facilitating precise structural queries.

Usage

```
StructureOperator(
  pdb_entry_id,
  assembly_id = 1,
  search_mode = "STRICT_SHAPE_MATCH"
)
```

Arguments

`pdb_entry_id`	A string representing the PDB entry ID to search for. The PDB entry ID is a unique identifier for each structure in the PDB.
`assembly_id`	An integer representing the assembly ID within the PDB entry. The assembly ID identifies the specific biological assembly or model within the PDB entry. By default, this is set to 1, which typically corresponds to the first biological assembly or model.
`search_mode`	A string indicating the search mode to be applied during the structure-based search. Accepted values include 'STRICT_SHAPE_MATCH', 'RELAXED_SHAPE_MATCH', etc. The default is 'STRICT_SHAPE_MATCH', which ensures a precise comparison based on the structural shape of the molecules.

Value

An object of class 'StructureOperator' that encapsulates the criteria for performing structure-based searches in the RCSB PDB.

Examples

```
# Example of creating a structure operator for a specific PDB entry and assembly
struct_operator <- StructureOperator(
  pdb_entry_id = "1XYZ",
  assembly_id = 1,
  search_mode = "STRICT_SHAPE_MATCH"
)
print(struct_operator)

# Example of creating a structure operator with a relaxed search mode
struct_operator_relaxed <- StructureOperator(
  pdb_entry_id = "1ABC",
  assembly_id = 2,
  search_mode = "RELAXED_SHAPE_MATCH"
)
print(struct_operator_relaxed)
```

---

summarize_assemblies *Summarize Assembly-Level Data*

Description

Summarize Assembly-Level Data

Usage

```
summarize_assemblies(x)
```

Arguments

x	Assembly-like table or object.

Value

One-row tibble with assembly summary fields.

---

summarize_entries *Summarize Entry-Level Data*

Description

Summarize Entry-Level Data

Usage

```
summarize_entries(x)
```

Arguments

x	Entry-like table or object.

Value

One-row tibble with high-level summary fields.

---

validate_properties *Validate Property Specification Against Known Field Registry*

Description

Validates a property list for a given data type.

Usage

```
validate_properties(properties, data_type, strict = TRUE)
```

Arguments

properties	Property list as used in `data_fetcher()` and `generate_json_query()`.
data_type	RCSB data type.
strict	Logical. If TRUE, unknown fields/subfields raise errors. If FALSE, validation details are returned without error.

Value

Invisibly TRUE when valid in strict mode. In non-strict mode, a list with unknown fields/subfields.

---

walk_nested_dict *Recursively Walk Through a Nested Dictionary*

Description

This function performs a recursive search through a nested dictionary-like structure in R, looking for a specific term and collecting its values. It's useful for extracting specific pieces of data from complex, deeply nested results.

Usage

```
walk_nested_dict(my_result, term, outputs = list(), depth = 0, maxdepth = 25)
```

Arguments

`my_result`	The nested dictionary-like structure to search through.
`term`	The term to search for within the nested dictionary.
`outputs`	An initially empty list to store the results of the search, default is an empty list.
`depth`	The current depth of the recursion, default is 0.
`maxdepth`	The maximum depth to recurse, default is 25. If exceeded, the function issues a warning and returns NULL.

Value

A list of values associated with the term found in the nested dictionary. Returns NULL if the term is not found or if maximum recursion depth is exceeded.

Index