

# Package ‘notionapi’

May 9, 2026

**Title** Client for the 'Notion API'

**Version** 0.2.0

**Description** Enable programmatic interaction with 'Notion' pages, databases, blocks, comments, and users through the 'Notion API' <<https://developers.notion.com/>>. Provides both synchronous and asynchronous client interfaces for building workflows and automations that integrate with 'Notion' workspaces. Supports all 'Notion API' endpoints including content creation, data retrieval, and workspace management.

**License** MIT + file LICENSE

**URL** <https://brenwin1.github.io/notionapi/>,  
<https://github.com/brenwin1/notionapi>

**BugReports** <https://github.com/brenwin1/notionapi/issues>

**Imports** cli, httr2, jsonlite, R6, rlang

**Suggests** curl, promises, testthat (>= 3.0.0), vcr (>= 2.0.0)

**Config/testthat/edition** 3

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** Brenwin Ang [aut, cre, cph]

**Maintainer** Brenwin Ang <brenwinalj@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-04-13 07:30:15 UTC

## Contents

BlocksChildrenEndpoint . . . . .	2
BlocksEndpoint . . . . .	4
CommentsEndpoint . . . . .	6
CustomEmojisEndpoint . . . . .	8

DatabasesEndpoint . . . . .	9
DataSourcesEndpoint . . . . .	11
FileUploadsEndpoint . . . . .	15
notion_token_exists . . . . .	19
no_config . . . . .	19
PagesEndpoint . . . . .	20
PagesPropertiesEndpoint . . . . .	24
print.notion_response . . . . .	26
UsersEndpoint . . . . .	26
ViewsEndpoint . . . . .	28
ViewsQueriesEndpoint . . . . .	31

<b>Index</b>	<b>34</b>
--------------	-----------

---

## BlocksChildrenEndpoint

*R6 Class for Blocks children endpoint*

---

### Description

Handle all block children operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$blocks$children`. Not to be instantiated directly.

### Value

A list containing the parsed API response.

### Methods

#### Public methods:

- [BlocksChildrenEndpoint\\$new\(\)](#)
- [BlocksChildrenEndpoint\\$list\(\)](#)
- [BlocksChildrenEndpoint\\$append\(\)](#)

**Method** `new()`: Initialise block children endpoint. Not to be called directly, e.g., use `notion$pages$children` instead.

*Usage:*

```
BlocksChildrenEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `list()`: Retrieve a block's children

*Usage:*

```
BlocksChildrenEndpoint$list(block_id, start_cursor = NULL, page_size = NULL)
```

*Arguments:*

`block_id` String (required). The ID for a Notion block.  
`start_cursor` Character. For pagination. If provided, returns results starting from this cursor.  
 If NULL, returns the first page of results.  
`page_size` Integer. Number of items to return per page (1-100). Defaults to 100.

*Details:* [Endpoint documentation](#)

**Method** `append()`: Append block children

*Usage:*

```
BlocksChildrenEndpoint$append(block_id, children, position = NULL)
```

*Arguments:*

`block_id` String (required). The ID for a Notion block.  
`children` List of lists (JSON array) (required). Block objects to append as children to the block.  
`position` Named list (JSON object). Controls where new blocks are inserted among parent's children. Defaults to end of parent block's children when omitted.

*Details:* [Endpoint documentation](#)

## Examples

```
notion <- notion_client()

# ----- Append children to a block
notion$blocks$children$append(
  "34033ea0-c1e4-81c4-afa0-d1ec98de4bec",
  list(
    list(
      object = "block",
      heading_2 = list(
        rich_text = list(list(
          text = list(content = "Test Heading")
        ))
      )
    )
  ),
  position = list(type = "start")
)

# ----- Retrieve children of a block
notion$blocks$children$list("34033ea0-c1e4-81c4-afa0-d1ec98de4bec")

# ----- Iterate through paginated results
## Not run:
start_cursor <- NULL
has_more <- FALSE
resps <- list()
i <- 1

while (has_more) {
```

```

resps[[i]] <- notion$blocks$children$list(
  "2926b407e3c44b49a1830609abe6744f",
  start_cursor
)
has_more <- resps[[i]][["has_more"]]
start_cursor <- resps[[i]][["next_cursor"]]
i <- i + 1
}

## End(Not run)

```

---

BlocksEndpoint

*R6 Class for Blocks Endpoint*


---

### Description

Handle all block operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$blocks`. Not to be instantiated directly.

### Value

A list containing the parsed API response.

### Public fields

`children` Block children endpoint

### Methods

#### Public methods:

- [BlocksEndpoint\\$new\(\)](#)
- [BlocksEndpoint\\$retrieve\(\)](#)
- [BlocksEndpoint\\$update\(\)](#)
- [BlocksEndpoint\\$delete\(\)](#)

**Method** `new()`: Initialise block endpoint. Not to be called directly, e.g., use `notion$blocks` instead.

*Usage:*

```
BlocksEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `retrieve()`: Retrieve a block

*Usage:*

```
BlocksEndpoint$retrieve(block_id)
```

*Arguments:*

block\_id Character (required). The ID for a Notion block.

*Details:* [Endpoint documentation](#)

**Method** update(): Update a block

*Usage:*

```
BlocksEndpoint$update(block_id, in_trash = NULL, ...)
```

*Arguments:*

block\_id Character (required). The ID for a Notion block.

in\_trash Boolean. Set to TRUE to trash (delete) a block. Set to FALSE to restore a block.

... [<dynamic-dots>](#) Block-specific properties to update. Each argument should be named after a **block type** (e.g., heading\_1, paragraph) with a named list value containing the block configuration.

*Details:* [Endpoint documentation](#)

**Method** delete(): Delete a block

*Usage:*

```
BlocksEndpoint$delete(block_id)
```

*Arguments:*

block\_id Character (required). The ID for a Notion block.

*Details:* [Endpoint documentation](#)

## Examples

```
notion <- notion_client()

# ----- Retrieve a block
notion$blocks$retrieve("34033ea0-c1e4-8123-be95-e77dcc78e47d")

# ----- Update a block
notion$blocks$update(
  "34033ea0-c1e4-8123-be95-e77dcc78e47d",
  heading_2 = list(
    rich_text = list(list(
      text = list(
        content = "Updated Test Heading"
      )
    ))
  )
)

# ----- Delete a block
notion$blocks$delete(
  "34033ea0-c1e4-8123-be95-e77dcc78e47d"
```

)

---

 CommentsEndpoint      *R6 Class for Comments Endpoint*


---

### Description

Handle all comments operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$comments`. Not to be instantiated directly.

### Value

A list containing the parsed API response.

### Methods

#### Public methods:

- [CommentsEndpoint\\$new\(\)](#)
- [CommentsEndpoint\\$create\(\)](#)
- [CommentsEndpoint\\$retrieve\(\)](#)
- [CommentsEndpoint\\$list\(\)](#)

**Method** `new()`: Initialise comments endpoint. Not to be called directly, e.g., use `notion$comments` instead.

*Usage:*

```
CommentsEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `create()`: Create a comment

*Usage:*

```
CommentsEndpoint$create(
  rich_text,
  parent = NULL,
  discussion_id = NULL,
  attachments = NULL,
  display_name = NULL
)
```

*Arguments:*

`rich_text` List of lists (JSON array) (required). **Rich text object(s)** representing the content of the comment.

parent List (JSON object). The parent of the comment. This can be a page or a block. Required if discussion\_id is not provided.

discussion\_id Character. The ID of the discussion to comment on. Required if parent is not provided.

attachments List of lists (JSON array). An array of files to attach to the comment. Maximum of 3 allowed.

display\_name Named list (JSON object). Display name for the comment.

*Details:* [Endpoint documentation](#)

**Method** retrieve(): Retrieve comments for a block

*Usage:*

```
CommentsEndpoint$retrieve(comment_id)
```

*Arguments:*

comment\_id Character (required). The ID of the comment to retrieve.

*Details:* [Endpoint documentation](#)

**Method** list(): List comments

*Usage:*

```
CommentsEndpoint$list(block_id, start_cursor = NULL, page_size = NULL)
```

*Arguments:*

block\_id Character (required). The ID for a Notion block or page.

start\_cursor Character. For pagination. If provided, returns results starting from this cursor. If NULL, returns the first page of results.

page\_size Integer. Number of items to return per page (1-100). Defaults to 100.

## Examples

```
notion <- notion_client()

# ----- Create comment
notion$comments$create(
  parent = list(
    page_id = "34033ea0-c1e4-81c4-afa0-d1ec98de4bec"
  ),
  rich_text = list(
    list(
      text = list(
        content = "Hello world!"
      )
    )
  )
)

# ----- Retrieve comment
notion$comments$retrieve("34033ea0-c1e4-8193-a220-001d29ae83c5")
```

```
# ----- List un-resolved comments from a page or block
notion$comments$list("34033ea0-c1e4-81c4-afa0-d1ec98de4bec")
```

---

CustomEmojisEndpoint *R6 Class for Custom Emojis Endpoint*

---

## Description

Handle all custom emojis operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$custom_emojis`. Not to be instantiated directly.

## Value

A list containing the parsed API response.

## Methods

### Public methods:

- [CustomEmojisEndpoint\\$new\(\)](#)
- [CustomEmojisEndpoint\\$list\(\)](#)

**Method** `new()`: Initialise custom emojis endpoint. Not to be called directly, e.g., use `notion$custom_emojis` instead.

*Usage:*

```
CustomEmojisEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `list()`: List custom emojis

*Usage:*

```
CustomEmojisEndpoint$list(start_cursor = NULL, page_size = NULL, name = NULL)
```

*Arguments:*

`start_cursor` Character. For pagination. If provided, returns results starting from this cursor.

If NULL, returns the first page of results.

`page_size` Integer. Number of items to return per page (1-100). Defaults to 100

`name` Character. Filters custom emojis by exact name match. Useful for resolving a custom emoji name to its ID.

*Details:* [Endpoint documentation](#)

**Examples**

```
notion <- notion_client()

notion$custom_emojis$list()
```

---

DatabasesEndpoint      *R6 Class for Databases Endpoint*

---

**Description**

Handle all databases operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$databases`. Not to be instantiated directly.

**Value**

A list containing the parsed API response.

**Methods****Public methods:**

- [DatabasesEndpoint\\$new\(\)](#)
- [DatabasesEndpoint\\$create\(\)](#)
- [DatabasesEndpoint\\$retrieve\(\)](#)
- [DatabasesEndpoint\\$update\(\)](#)

**Method** `new()`: Initialise databases endpoint. Not to be called directly, e.g., use `notion$databases` instead.

*Usage:*

```
DatabasesEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `create()`: Create a database

*Usage:*

```
DatabasesEndpoint$create(
  parent,
  title = NULL,
  description = NULL,
  is_inline = NULL,
  initial_data_source = NULL,
  icon = NULL,
  cover = NULL
)
```

*Arguments:*

**parent** Named list (JSON object) (required). The parent page or workspace where the database will be created.

**title** List of lists (JSON array). The title of the database.

**description** List of lists (JSON array). The description of the database.

**is\_inline** Boolean. Whether the database should be displayed inline in the parent page. Defaults to false.

**initial\_data\_source** Named list (JSON object). Initial data source configuration for the database

**icon** Named list (JSON object). The icon for the database.

**cover** Named list (JSON object). The cover image for the database.

*Details:* [Endpoint documentation](#)

**Method** `retrieve()`: Retrieve a database

*Usage:*

```
DatabasesEndpoint$retrieve(database_id)
```

*Arguments:*

**database\_id** String (required). The ID of a Notion database.

*Details:* [Endpoint documentation](#)

**Method** `update()`: Update a database

*Usage:*

```
DatabasesEndpoint$update(
  database_id,
  parent = NULL,
  title = NULL,
  description = NULL,
  is_inline = NULL,
  icon = NULL,
  cover = NULL,
  in_trash = NULL,
  is_locked = NULL
)
```

*Arguments:*

**database\_id** String (required). The ID of a Notion database.

**parent** Named list (JSON object). The parent page or workspace to move the database to. If not provided, the database will not be moved.

**title** List of lists (JSON array). The updated title of the database.

**description** List of lists (JSON array). The updated description of the database.

**is\_inline** Boolean. Whether the database should be displayed in the parent page.

**icon** Named list (JSON object). The updated icon for the database.

**cover** Named list (JSON object). The updated cover image for the database.

**in\_trash** Boolean. Whether the database should be moved to or from the trash.

**is\_locked** Boolean. Whether the database should be locked from editing.

*Details:* [Endpoint documentation](#)

## Examples

```
notion <- notion_client()

# ----- Create a database
notion$databases$create(
  list(
    type = "page_id",
    page_id = "22f33ea0c1e480b99c77d1ab72aedff9"
  ),
  title = list(list(
    text = list(
      content = "Test Database"
    )
  ))
)

# ----- Retrieve a database
notion$databases$retrieve("ffec20ee-1450-4da8-9904-f4babba0e9c0")

# ----- Update a database
notion$databases$update(
  "ffec20ee-1450-4da8-9904-f4babba0e9c0",
  description = list(list(
    text = list(
      content = "For testing purposes"
    )
  )),
  icon = list(
    icon = list(
      name = "calendar",
      color = "gray"
    )
  )
)
```

---

DataSourcesEndpoint    *R6 Class for DataSources Endpoint*

---

## Description

Handle all data sources operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$databases`. Not to be instantiated directly.

**Value**

A list containing the parsed API response.

**Methods****Public methods:**

- [DataSourcesEndpoint\\$new\(\)](#)
- [DataSourcesEndpoint\\$create\(\)](#)
- [DataSourcesEndpoint\\$retrieve\(\)](#)
- [DataSourcesEndpoint\\$list\\_templates\(\)](#)
- [DataSourcesEndpoint\\$update\(\)](#)
- [DataSourcesEndpoint\\$query\(\)](#)

**Method** `new()`: Initialise data sources endpoint. Not to be called directly, e.g., use `notion$datasources` instead.

*Usage:*

```
DataSourcesEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `create()`: Create a data source

*Usage:*

```
DataSourcesEndpoint$create(parent, properties, title = NULL, icon = NULL)
```

*Arguments:*

`parent` Named list (JSON object) (required). An object specifying the parent of the new data source to be created.

`properties` Named list (JSON object) (required). Property schema of data source.

`title` List of lists (JSON array). Title of data source.

`icon` Named list (JSON object). Page icon.

*Details:* [Endpoint documentation](#)

**Method** `retrieve()`: Retrieve a data source

*Usage:*

```
DataSourcesEndpoint$retrieve(data_source_id)
```

*Arguments:*

`data_source_id` Character (required). ID of a Notion data source.

*Details:* [Endpoint documentation](#)

**Method** `list_templates()`: List page templates available for a data source

*Usage:*

```
DataSourcesEndpoint$list_templates(
  data_source_id,
  name = NULL,
  start_cursor = NULL,
  page_size = NULL
)
```

*Arguments:*

`data_source_id` Character (required). ID of a Notion data source.

`name` Character. Name to filter templates by.

`start_cursor` Character. For pagination. If provided, returns results starting from this cursor. If NULL, returns the first page of results.

`page_size` Integer. Number of items to return per page (1-100). Defaults to 100

**Method** `update()`: Update a data source*Usage:*

```
DataSourcesEndpoint$update(
  data_source_id,
  title = NULL,
  icon = NULL,
  properties = NULL,
  in_trash = NULL,
  parent = NULL
)
```

*Arguments:*

`data_source_id` Character (required). ID of a Notion data source.

`title` List of lists (JSON array). Title of data source.

`icon` Named list (JSON object). Page icon.

`properties` Named list (JSON object). Key-value pairs representing the data source's properties.

`in_trash` Boolean. Whether the database should be moved to or from the trash.

`parent` Named list (JSON object). The parent of the data source, when moving it to a different database.

**Method** `query()`: Query a data source*Usage:*

```
DataSourcesEndpoint$query(
  data_source_id,
  filter_properties = NULL,
  sorts = NULL,
  filter = NULL,
  start_cursor = NULL,
  page_size = NULL,
  in_trash = NULL,
  result_type = NULL
)
```

*Arguments:*

`data_source_id` Character (required). ID of a Notion data source.

`filter_properties` Character vector. Page property value IDs to include in the response schema. If NULL (default), all properties are returned.

`sorts` List of lists (JSON array). **Sort conditions** to apply to the query

`filter` List of lists (JSON array). **Filter conditions** to apply. to the query

`start_cursor` Character. For pagination. If provided, returns results starting from this cursor. If NULL, returns the first page of results.

`page_size` Integer. Number of items to return per page (1-100). Defaults to 100

`in_trash` Boolean. If TRUE, trashed pages are included in the results alongside non-trashed pages. If FALSE or NULL (default), only non-trashed pages are returned.

`result_type` Character. Filter results by type. Available options are "page" and "data\_source". If NULL (default), all result types are returned.

**Examples**

```
notion <- notion_client()

# ----- Create a data source
notion$data_sources$create(
  list(
    database_id = "ffec20ee-1450-4da8-9904-f4babb0e9c0"
  ),
  properties = list(
    Title = list(
      title = no_config()
    )
  ),
  title = list(list(
    text = list(
      content = "Test data source"
    )
  ))
)
```

```
# ----- Update data source
notion$data_sources$update(
  "34033ea0-c1e4-8112-bc3a-000bc940aa45",
  properties = list(
    Status = list(
      status = list(
        options = list(
          list(
            name = "To do",
            color = "red"
          ),
          list(
            name = "Done",
            color = "green"
          )
        )
      )
    )
  )
)
```

```

    )
  )
)

# ----- Retrieve a data source
notion$data_sources$retrieve("34033ea0-c1e4-8112-bc3a-000bc940aa45")

# ----- List data source templates
notion$data_sources$list_templates("34033ea0-c1e4-8112-bc3a-000bc940aa45")

# ----- Query a data source
notion$data_sources$query(
  "34033ea0-c1e4-8112-bc3a-000bc940aa45",
  filter = list(
    property = "Status",
    status = list(
      equals = "To do"
    )
  )
)
)

```

---

FileUploadsEndpoint    *R6 Class for File Uploads Endpoint*

---

### Description

Handle all file uploads operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$file_uploads`. Not to be instantiated directly.

### Value

A list containing the parsed API response.

### Methods

#### Public methods:

- [FileUploadsEndpoint\\$new\(\)](#)
- [FileUploadsEndpoint\\$create\(\)](#)
- [FileUploadsEndpoint\\$send\(\)](#)

- [FileUploadsEndpoint\\$complete\(\)](#)
- [FileUploadsEndpoint\\$retrieve\(\)](#)
- [FileUploadsEndpoint\\$list\(\)](#)

**Method new():** Initialise file uploads endpoint. Not to be called directly, e.g., use `notion$file_uploads` instead.

*Usage:*

```
FileUploadsEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method create():** Create a file upload

*Usage:*

```
FileUploadsEndpoint$create(
  mode = NULL,
  filename = NULL,
  content_type = NULL,
  number_of_parts = NULL,
  external_url = NULL
)
```

*Arguments:*

`mode` Character. How the file is being sent. Use "multi\_part" for files larger than 20MB. Use "external\_url" for files that are temporarily hosted publicly elsewhere. Default is "single\_part".

`filename` Character. Name of the file to be created. Required when mode is "multi\_part". Must include an extension, or have one inferred from the `content_type` parameter.

`content_type` Character. MIME type of the file to be created.

`number_of_parts` Integer. When mode is "multi\_part", the number of parts you are uploading.

`external_url` Character. When mode is "external\_url", provide the HTTPS URL of a publicly accessible file to import into your workspace.

**Method send():** Upload a file

*Usage:*

```
FileUploadsEndpoint$send(file_upload_id, file, part_number = NULL)
```

*Arguments:*

`file_upload_id` Character (required). Identifier for a Notion file upload object.

`file` Named list (JSON object). The raw binary file contents to upload. Must contain named elements:

- `filename`. Character. The name of the file, including its extension (e.g., "report.pdf")
- `data`. Raw. The binary contents of the file, as returned by `e.g., readBin()`
- `type`. Character. Optional. The MIME type of the file (e.g., "application/pdf", "image/png"). If not supplied, the type is inferred from `filename` by `curl::form_file()`. Supported file types are listed [here](#).

`part_number` Character. The current part number when uploading files greater than 20MB in parts. Must be an integer between 1 and 1,000

**Method** `complete()`: Complete a multi-part file upload

*Usage:*

```
FileUploadsEndpoint$complete(file_upload_id)
```

*Arguments:*

`file_upload_id` Character (required). Identifier for a Notion file upload object.

*Details:* [Endpoint documentation](#)

**Method** `retrieve()`: Retrieve a file upload

*Usage:*

```
FileUploadsEndpoint$retrieve(file_upload_id)
```

*Arguments:*

`file_upload_id` Character (required). Identifier for a Notion file upload object.

*Details:* [Endpoint documentation](#)

**Method** `list()`: List file uploads

*Usage:*

```
FileUploadsEndpoint$list(status = NULL, start_cursor = NULL, page_size = NULL)
```

*Arguments:*

`status` Character. If supplied, the endpoint will return file uploads with the specified status.

Available options are "pending", "uploaded", "expired", "failed".

`start_cursor` Character. For pagination. If provided, returns results starting from this cursor.

If NULL, returns the first page of results.

`page_size` Integer. Number of items to return per page (1-100). Defaults to 100

*Details:* [Endpoint documentation](#)

## Examples

```
notion <- notion_client()

# ----- Direct upload (files <= 20MB)
# Step 1: Create a File Upload object
(resp <- notion$file_uploads$create("single_part"))
file_upload_id <- resp[["id"]]

# Step 2: Upload file contents
#* replace with your file
path <- file.path(tempdir(), "test.pdf")
writeBin(charToRaw("placeholder"), path)
raw <- readBin(path, "raw", file.size(path))

notion$file_uploads$send(
  file_upload_id,
  list(
    filename = basename(path),
    data = raw
  )
)
```

```

)

# Retrieve the file upload
notion$file_uploads$retrieve(file_upload_id)

# ----- Multi-part upload (files > 20MB)
# Step 1: Split raw content into parts
#* replace with your file
raw <- as.raw(rep(65L, 11 * 1024 * 1024))
mid <- ceiling(length(raw) / 2)
parts <- list(raw[seq_len(mid)], raw[seq(mid + 1L, length(raw))])

# Step 2: Create a File Upload object
(resp <- notion$file_uploads$create(
  mode = "multi_part",
  filename = "test-large.txt",
  number_of_parts = 2L
))
file_upload_id <- resp[["id"]]

# Step 3: Send each part
for (i in seq_along(parts)) {
  notion$file_uploads$send(
    file_upload_id,
    file = list(
      filename = "test-large.txt",
      data = parts[[i]]
    ),
    part_number = as.character(i)
  )
}

# Step 4: Complete the upload
notion$file_uploads$complete(
  file_upload_id
)

# Retrieve the file upload
notion$file_uploads$retrieve(
  file_upload_id
)

# ----- Import external files
notion$file_uploads$create(
  "external_url",
  "dummy.pdf",
  content_type = "application/pdf",
  external_url = "https://github.com/brenwin1/notionapi/blob/main/tests/testthat/test.pdf"
)

```

---

notion_token_exists	<i>Check if Notion token is set</i>
---------------------	-------------------------------------

---

**Description**

Checks if the NOTION\_TOKEN environment variable is set.

**Usage**

```
notion_token_exists()
```

**Value**

TRUE if the token exists, FALSE otherwise.

**Examples**

```
notion_token_exists()
```

---

no_config	<i>Create empty property configuration</i>
-----------	--

---

**Description**

Helper function that creates an empty named list for property configurations that require no additional settings.

Many **database properties** like text, checkbox and date do not need configuration settings. This function returns the empty configuration (`{}` in JSON) that these properties expect.

**Usage**

```
no_config()
```

**Value**

An empty named `list` that serialises to `{}` in JSON

## Examples

```
notion <- notion_client()

# ----- Create a data source
notion$data_sources$create(
  list(
    database_id = "5f9759b2-ad71-4b66-880f-d0306614227b"
  ),
  properties = list(
    Title = list(
      title = no_config()
    )
  ),
  title = list(list(
    text = list(
      content = "Test data source"
    )
  ))
)
```

---

PagesEndpoint

*R6 Class for Pages Endpoint*

---

## Description

Handle all pages operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$pages`. Not to be instantiated directly.

## Value

A list containing the parsed API response.

## Public fields

`properties` Pages properties endpoint

## Methods

### Public methods:

- [PagesEndpoint\\$new\(\)](#)
- [PagesEndpoint\\$create\(\)](#)
- [PagesEndpoint\\$retrieve\(\)](#)
- [PagesEndpoint\\$move\(\)](#)
- [PagesEndpoint\\$update\(\)](#)

- [PagesEndpoint\\$retrieve\\_markdown\(\)](#)
- [PagesEndpoint\\$update\\_markdown\(\)](#)

**Method** `new()`: Initialise pages endpoint. Not to be called directly, e.g., use `notion$pages` instead.

*Usage:*

```
PagesEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `create()`: Create a page

*Usage:*

```
PagesEndpoint$create(
  parent,
  properties = NULL,
  icon = NULL,
  cover = NULL,
  content = NULL,
  children = NULL,
  markdown = NULL,
  template = NULL,
  position = NULL
)
```

*Arguments:*

`parent` Named list (JSON object) (required). The parent under which the new page is created.

`properties` Named list (JSON object). Key-value pairs representing the page's properties.

`icon` Named list (JSON object). The page icon.

`cover` Named list (JSON object). The page cover image.

`content` List of lists (JSON array). Block objects to append as children to the page.

`children` List of lists (JSON array). Block objects to append as children to the page.

`markdown` Character. Page content as Notion-flavored Markdown. Mutually exclusive with `content/children`.

`template` Named list (JSON object). A data source template apply to the new page. Cannot be combined with `children`.

`position` Named list (JSON object). Controls where new blocks are inserted among parent's children. Defaults to end of parent block's children when omitted.

*Details:* [Endpoint documentation](#)

**Method** `retrieve()`: Retrieve page properties

*Usage:*

```
PagesEndpoint$retrieve(page_id, filter_properties = NULL)
```

*Arguments:*

`page_id` Character (required). The ID for a Notion page.

`filter_properties` Character vector. Page property value IDs to include in the response schema. If NULL (default), all properties are returned.

*Details:* [Endpoint documentation](#)

**Method** `move()`: Move a page

*Usage:*

```
PagesEndpoint$move(page_id, parent)
```

*Arguments:*

`page_id` Character (required). The ID of the page to move.

`parent` Named list (JSON object) (required). The new parent location for the page.

*Details:* [Endpoint documentation](#)

**Method** `update()`: Update attributes of a Notion page

*Usage:*

```
PagesEndpoint$update(
  page_id,
  properties = NULL,
  icon = NULL,
  cover = NULL,
  is_locked = NULL,
  template = NULL,
  erase_content = NULL,
  in_trash = NULL,
  is_archived = NULL
)
```

*Arguments:*

`page_id` Character (required). The ID for a Notion page.

`properties` Named list (JSON object). Key-value pairs representing the page's properties.

`icon` Named list (JSON object). The page icon.

`cover` Named list (JSON object). The page cover image.

`is_locked` Boolean. Whether the page should be locked from editing.

`template` Named list (JSON object). A template to apply to the page.

`erase_content` Boolean. Whether to erase all existing content from the page. Irreversible via the API.

`in_trash` Boolean. Set to TRUE to trash (delete) the page. Set to FALSE to restore the page.

`is_archived` Boolean. Deprecated alias for `in_trash`. Use `in_trash` for new integrations.

*Details:* [Endpoint documentation](#)

**Method** `retrieve_markdown()`: Retrieve a page as markdown

*Usage:*

```
PagesEndpoint$retrieve_markdown(page_id, include_transcript = NULL)
```

*Arguments:*

`page_id` Character (required). The ID of the page (or block) to to retrieve as markdown.

include\_transcript Boolean. Whether to include meeting note transcripts. Defaults to false.

*Details:* [Endpoint documentation](#)

**Method** update\_markdown(): Update a page's content as markdown

*Usage:*

```
PagesEndpoint$update_markdown(
  page_id,
  type,
  update_content = NULL,
  replace_content = NULL,
  insert_content = NULL,
  replace_content_range = NULL
)
```

*Arguments:*

page\_id Character (required). The ID of the page to update.

type Character (required). The update command type. One of "update\_content", "replace\_content", "insert\_content" and "replace\_content\_range". The first two are recommended.

update\_content Named list (JSON object). Update specific content using search-and-replace operations.

replace\_content Named list (JSON object). Replace the entire page content with new markdown.

insert\_content Named list (JSON object). Insert new content into the page.

replace\_content\_range Named list (JSON object). Replace a range of content in the page.

*Details:* [Endpoint documentation](#)

## Examples

```
notion <- notion_client()

# ----- Create a page
notion$pages$create(
  list(page_id = "22f33ea0c1e480b99c77d1ab72aedff9"),
  list(
    title = list(list(
      text = list(
        content = "Test Page for notionapi"
      )
    ))
  )
)

# ----- Retrieve a page
notion$pages$retrieve("34033ea0-c1e4-81c4-afa0-d1ec98de4bec")

# ----- Move page into a database
notion$pages$move(
```

```

    "34033ea0-c1e4-81c4-afa0-d1ec98de4bec",
    list(
      data_source_id = "34033ea0-c1e4-8112-bc3a-000bc940aa45"
    )
  )

# ----- Update a page
notion$pages$update(
  "34033ea0-c1e4-81c4-afa0-d1ec98de4bec",
  icon = list(
    icon = list(
      name = "pizza",
      color = "blue"
    )
  )
)

# ----- Use `NA` to send JSON `null` - below removes the page's icon
notion$pages$update(
  "34033ea0-c1e4-81c4-afa0-d1ec98de4bec",
  icon = NA
)

# ----- Retrieve a page as markdown
notion$pages$retrieve_markdown("34033ea0-c1e4-81c4-afa0-d1ec98de4bec")

# ----- Update/replace a page content
notion$pages$update_markdown(
  "34033ea0-c1e4-81c4-afa0-d1ec98de4bec",
  "replace_content",
  replace_content = list(
    new_str = '## Updated Test Heading\nUsed markdown{color="blue"}'
  )
)

# ----- Trash a page
notion$pages$update(
  "34033ea0-c1e4-81c4-afa0-d1ec98de4bec",
  in_trash = TRUE
)

```

**Description**

Handle all pages properties operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$pages$properties`. Not to be instantiated directly.

**Value**

A list containing the parsed API response.

**Methods****Public methods:**

- [PagesPropertiesEndpoint\\$new\(\)](#)
- [PagesPropertiesEndpoint\\$retrieve\(\)](#)

**Method** `new()`: Initialise pages properties endpoint. Not to be called directly, e.g., use `notion$pages$properties` instead.

*Usage:*

```
PagesPropertiesEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `retrieve()`: Retrieve a page property item

*Usage:*

```
PagesPropertiesEndpoint$retrieve(
  page_id,
  property_id,
  start_cursor = NULL,
  page_size = NULL
)
```

*Arguments:*

`page_id` Character (required). The ID for a Notion page.

`property_id` Character (required). The ID of the property to retrieve.

`start_cursor` Character. For pagination. If provided, returns results starting from this cursor.

If NULL, returns the first page of results.

`page_size` Integer. Number of items to return per page (1-100). Defaults to 100.

*Details:* [Endpoint documentation](#)

**Examples**

```
notion <- notion_client()

# ----- Retrieve a page property
notion$pages$properties$retrieve(
  "34033ea0-c1e4-81c4-afa0-d1ec98de4bec",
  "%60w%5CX"
```

)

---

```
print.notion_response Print the result of a Notion API call
```

---

### Description

Print the result of a Notion API call

### Usage

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

### Arguments

x	The result object.
...	Ignored.

### Value

The JSON result.

---

UsersEndpoint	<i>R6 Class for Users Endpoint</i>
---------------	------------------------------------

---

### Description

Handle all users operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$users`. Not to be instantiated directly.

### Value

A list containing the parsed API response.

## Methods

### Public methods:

- [UsersEndpoint\\$new\(\)](#)
- [UsersEndpoint\\$list\(\)](#)
- [UsersEndpoint\\$retrieve\(\)](#)
- [UsersEndpoint\\$me\(\)](#)

**Method** `new()`: Initialise users endpoint. Not to be called directly, e.g., use `notion$users` instead.

*Usage:*

```
UsersEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `list()`: List all users

*Usage:*

```
UsersEndpoint$list(start_cursor = NULL, page_size = NULL)
```

*Arguments:*

`start_cursor` Character. For pagination. If provided, returns results starting from this cursor.

If NULL, returns the first page of results.

`page_size` Integer. Number of items to return per page (1-100). Defaults to 100.

*Details:* [Endpoint documentation](#)

**Method** `retrieve()`: Retrieve a user

*Usage:*

```
UsersEndpoint$retrieve(user_id)
```

*Arguments:*

`user_id` Character (required). The ID of the user to retrieve.

*Details:* [Endpoint documentation](#)

**Method** `me()`: Retrieve the bot User associated with the API token

*Usage:*

```
UsersEndpoint$me()
```

*Details:* [Endpoint documentation](#)

## Examples

```
notion <- notion_client()
```

```
# ----- List all users
```

```
notion$users$list()
```

```
# ----- Retrieve a user
```

```

notion$users$retrieve("fda12729-108d-4eb5-bbfb-a8f0886794d1")

# ----- Retrieve the bot User associated with the API token
notion$users$me()

```

---

ViewsEndpoint

*R6 Class for Views Endpoint*


---

### Description

Handle all views operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$views`. Not to be instantiated directly.

### Value

A list containing the parsed API response.

### Public fields

`queries` Views Queries Endpoint

### Methods

#### Public methods:

- [ViewsEndpoint\\$new\(\)](#)
- [ViewsEndpoint\\$create\(\)](#)
- [ViewsEndpoint\\$retrieve\(\)](#)
- [ViewsEndpoint\\$update\(\)](#)
- [ViewsEndpoint\\$delete\(\)](#)
- [ViewsEndpoint\\$list\(\)](#)

**Method** `new()`: Initialise views endpoint. Not to be called directly, e.g., use `notion$views` instead.

*Usage:*

```
ViewsEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `create()`: Create a view

*Usage:*

```
ViewsEndpoint$create(
  data_source_id,
  name,
  type,
  database_id = NULL,
  view_id = NULL,
  filter = NULL,
  sorts = NULL,
  quick_filters = NULL,
  create_database = NULL,
  configuration = NULL,
  position = NULL,
  placement = NULL
)
```

*Arguments:*

`data_source_id` Character (required). The ID of the data source this view is scoped to.

`name` Character (required). The name of the view.

`type` Character (required). The type of view to create.

`database_id` Character. The ID of the database to create a view in. Mutually exclusive with `view_id` and `create_database`

`view_id` Character. The ID of a dashboard view to add this view to as a widget. Mutually exclusive with `database_id` and `create_database`.

`filter` Named list (JSON object). Filter to apply to the view.

`sorts` List of lists (JSON array). Sorts to apply to the view.

`quick_filters` Named list (JSON object). Key-value pairs of quick filters to pin in the view's filter bar.

`create_database` Named list (JSON object). Create a new linked database block and add the view to it. Mutually exclusive with `database_id` and `view_id`

`configuration` Named list (JSON object). View presentation configuration.

`position` Named list (JSON object). Where to place the new view in the database's view tab bar. Only applicable when `database_id` is provided. Defaults to "end" (append).

`placement` Named list (JSON object). Where to place the new widget in a dashboard view. Only applicable when `view_id` is provided. Defaults to creating a new row at the end.

*Details:* [Endpoint documentation](#)

**Method** `retrieve()`: Retrieve a view

*Usage:*

```
ViewsEndpoint$retrieve(view_id)
```

*Arguments:*

`view_id` ID of a Notion view.

*Details:* [Endpoint documentation](#)

**Method** `update()`: Update a view

*Usage:*

```
ViewsEndpoint$update(
  view_id,
  name = NULL,
  filter = NULL,
  sorts = NULL,
  quick_filters = NULL,
  configuration = NULL
)
```

*Arguments:*

`view_id` ID of a Notion view.

`name` Character. New name for the view.

`filter` Named list (JSON object). Filter to apply to the view.

`sorts` List of lists (JSON array). Property sorts to apply to the view.

`quick_filters` Named list (JSON object). Key-value pairs of quick filters to add/update.

`configuration` Named list (JSON object). View presentation configuration.

*Details:* [Endpoint documentation](#)

**Method** `delete()`: Delete a view

*Usage:*

```
ViewsEndpoint$delete(view_id)
```

*Arguments:*

`view_id` ID of a Notion view.

*Details:* [Endpoint documentation](#)

**Method** `list()`: List all views in a database

*Usage:*

```
ViewsEndpoint$list(
  database_id = NULL,
  data_source_id = NULL,
  start_cursor = NULL,
  page_size = NULL
)
```

*Arguments:*

`database_id` Character. ID of a Notion database to list views for. At least one of `database_id` or `data_source_id` is required.

`data_source_id` Character. ID of a data source to list all views for, including linked views across the workspace. At least one of `database_id` or `data_source_id` is required.

`start_cursor` Character. For pagination. If provided, returns results starting from this cursor. If NULL, returns the first page of results.

`page_size` Integer. Number of items to return per page (1-100). Defaults to 100

*Details:* [Endpoint documentation](#)

## Examples

```
notion <- notion_client()

# ----- Create a view
notion$views$create(
  "34033ea0-c1e4-8112-bc3a-000bc940aa45",
  "Test view",
  "table",
  "ffec20ee-1450-4da8-9904-f4babba0e9c0"
)

# ----- Retrieve a view
notion$views$retrieve("34033ea0-c1e4-8192-ac14-000cdad096ce")

# ----- List views
notion$views$list(data_source_id = "34033ea0-c1e4-8112-bc3a-000bc940aa45")

# ----- Update a view
notion$views$update("34033ea0-c1e4-8192-ac14-000cdad096ce", "Updated view name")

# ----- Delete a view
notion$views$delete("34033ea0-c1e4-8192-ac14-000cdad096ce")
```

---

ViewsQueriesEndpoint *R6 Class for Views Queries Endpoint*

---

## Description

Handle all views queries operations in the Notion API

**Note:** Access this endpoint through the client instance, e.g., `notion$views$queries`. Not to be instantiated directly.

## Value

A list containing the parsed API response.

## Methods

### Public methods:

- [ViewsQueriesEndpoint\\$new\(\)](#)
- [ViewsQueriesEndpoint\\$create\(\)](#)
- [ViewsQueriesEndpoint\\$results\(\)](#)

- [ViewsQueriesEndpoint\\$delete\(\)](#)

**Method** `new()`: Initialise pages properties endpoint. Not to be called directly, e.g., use `notion$views$queries` instead.

*Usage:*

```
ViewsQueriesEndpoint$new(client)
```

*Arguments:*

`client` Notion Client instance

**Method** `create()`: Create a view query

*Usage:*

```
ViewsQueriesEndpoint$create(view_id, page_size = NULL)
```

*Arguments:*

`view_id` Character (required). The ID of the view.

`page_size` Integer. Number of items to return per page (1-100). Defaults to 100.

*Details:* [Endpoint documentation](#)

**Method** `results()`: Get view query results

*Usage:*

```
ViewsQueriesEndpoint$results(  
  view_id,  
  query_id,  
  start_cursor = NULL,  
  page_size = NULL  
)
```

*Arguments:*

`view_id` Character (required). The ID of the view.

`query_id` Character (required). The ID of the query.

`start_cursor` Character. For pagination. If provided, returns results starting from this cursor.  
If NULL, returns the first page of results.

`page_size` Integer. Number of items to return per page (1-100). Defaults to 100

*Details:* [Endpoint documentation](#)

**Method** `delete()`: Delete a view query

*Usage:*

```
ViewsQueriesEndpoint$delete(view_id, query_id)
```

*Arguments:*

`view_id` Character (required). The ID of the view.

`query_id` Character (required). The ID of the query.

*Details:* [Endpoint documentation](#)

**Examples**

```
notion <- notion_client()

# ----- Create a view query
notion$views$queries$create("34033ea0-c1e4-8192-ac14-000cdad096ce")

# ----- Get view query results
notion$views$queries$results(
  "34033ea0-c1e4-8192-ac14-000cdad096ce",
  "9af03bd1-ed79-4842-a57c-0bc04fb61be2"
)

# ----- Delete a view query
notion$views$queries$delete(
  "34033ea0-c1e4-8192-ac14-000cdad096ce",
  "9af03bd1-ed79-4842-a57c-0bc04fb61be2"
)
```

# Index

BlocksChildrenEndpoint, 2  
BlocksEndpoint, 4

CommentsEndpoint, 6  
CustomEmojisEndpoint, 8

DatabasesEndpoint, 9  
DataSourcesEndpoint, 11

FileUploadsEndpoint, 15

no\_config, 19  
notion\_token\_exists, 19

PagesEndpoint, 20  
PagesPropertiesEndpoint, 24  
print.notion\_response, 26

UsersEndpoint, 26

ViewsEndpoint, 28  
ViewsQueriesEndpoint, 31