Package 'Microsoft365R'

July 21, 2025

Title Interface to the 'Microsoft 365' Suite of Cloud Services

Version 2.4.1

Description An interface to the 'Microsoft 365' (formerly known as 'Office 365') suite of cloud services, building on the framework supplied by the 'AzureGraph' package. Enables access from R to data stored in 'Teams', 'SharePoint Online' and 'OneDrive', including the ability to list drive folder contents, upload and download files, send messages, and retrieve data lists. Also provides a full-featured 'Outlook' email client, with the ability to send emails and manage emails and mail folders.

URL https://github.com/Azure/Microsoft365R

https://github.com/Azure/AzureR

BugReports https://github.com/Azure/Microsoft365R/issues

License MIT + file LICENSE

VignetteBuilder knitr

Depends R (>= 3.3)

Imports AzureAuth, AzureGraph (>= 1.3.1), utils, parallel, tools, curl, httr, jsonlite, R6, vctrs, mime

Suggests openssl, knitr, rmarkdown, testthat, blastula, emayili, readr, readxl

RoxygenNote 7.2.1

NeedsCompilation no

Author Hong Ooi [aut, cre], Roman Zenka [ctb], Robert Ashton [ctb], Philip Zheng [ctb], Microsoft [cph]

Maintainer Hong Ooi <hongooi73@gmail.com>

Repository CRAN

Date/Publication 2025-03-02 04:00:06 UTC

Contents

`add_methods` *Microsoft 365 object accessor methods*

Description

Methods for the `AzureGraph::ms_graph`, `AzureGraph::az_user` and `AzureGraph::az_group` classes.

Usage

```
## R6 method for class 'az_user'
get_chat(chat_id)

## R6 method for class 'ms_graph'
get_drive(drive_id)

## R6 method for class 'az_user'
get_drive(drive_id = NULL)

## R6 method for class 'az_group'
get_drive(drive_name = NULL, drive_id = NULL)
```

```
## R6 method for class 'az_group'
get_plan(plan_title = NULL, plan_id = NULL)

## R6 method for class 'ms_graph'
get_sharepoint_site(site_url = NULL, site_id = NULL)

## R6 method for class 'az_group'
get_sharepoint_site()

## R6 method for class 'ms_graph'
get_team(team_id = NULL)

## R6 method for class 'az_group'
get_team()

## R6 method for class 'az_user'
list_chats(filter = NULL, n = Inf)

## R6 method for class 'az_user'
list_drives(filter = NULL, n = Inf)

## R6 method for class 'az_group'
list_drives(filter = NULL, n = Inf)

## R6 method for class 'az_group'
list_plans(filter = NULL, n = Inf)

## R6 method for class 'az_user'
list_sharepoint_sites(filter = NULL, n = Inf)

## R6 method for class 'az_user'
list_teams(filter = NULL, n = Inf)
```

Arguments

- `drive_name`,`drive_id`: For `get_drive`, the name or ID of the drive or shared document library. Note that only the `az_group` method has the `drive_name` argument, as user drives do not have individual names (and most users will only have one drive anyway). For the `az_user` and `az_group` methods, leaving the argument(s) blank will return the default drive/document library.
- `site_url`,`site_id`: For `ms_graph$get_sharepoint_site()`, the URL and ID of the site. Provide one or the other, but not both.
- `team_name`,`team_id`: For `az_user$get_team()`, the name and ID of the site. Provide one or the other, but not both. For `ms_graph$get_team`, you must provide the team ID.
- `plan_title`,`plan_id`: For `az_group$get_plan()`, the title and ID of the site. Provide one or the other, but not both.
- `filter, n`: See 'List methods' below.

Details

`get_sharepoint_site` retrieves a SharePoint site object. The method for the top-level Graph client class requires that you provide either the site URL or ID. The method for the `az_group` class will retrieve the site associated with that group, if applicable.

`get_drive` retrieves a OneDrive or shared document library, and `list_drives` retrieves all such drives/libraries that the user or group has access to. Whether these are personal or business drives depends on the tenant that was specified in `AzureGraph::get_graph_login()`/`create_graph_login()`: if this was "consumers" or "9188040d-6c67-4c5b-b112-36a304b66dad" (the equivalent GUID), it will be the personal OneDrive. See the examples below.

`get_plan` retrieves a plan (not to be confused with a Todo task list), and `list_plans` retrieves all plans for a group.

`get_team` retrieves a team. The method for the Graph client class requires the team ID. The method for the `az_user` class requires either the team name or ID. The method for the `az_group` class retrieves the team associated with the group, if it exists.

`get_chat` retrieves a one-on-one, group or meeting chat, by ID. `list_chats` retrieves all chats that the user is part of.

Note that Teams, SharePoint and OneDrive for Business require a Microsoft 365 Business license, and are available for organisational tenants only. Similarly, only Microsoft 365 groups can have associated sites/teams/plans/drives, not any other kind of group.

List methods

All `list_*` methods have `filter` and `n` arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are `filter=NULL` and `n=Inf`. If `n=NULL`, the `ms_graph_pager` iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

Value

For `get_sharepoint_site`, an object of class `ms_site`.

For `get_drive`, an object of class `ms_drive`. For `list_drives`, a list of `ms_drive` objects.

For `get_plan`, an object of class `ms_plan`. For `list_plans`, a list of `ms_plan` objects.

For `get_team`, an object of class `ms_team`. For `list_teams`, a list of `ms_team` objects.

For `get_chat`, an object of class `ms_chat`. For `list_chats`, a list of `ms_chat` objects.

See Also

`ms_site`, `ms_drive`, `ms_plan`, `ms_team`, `ms_chat`, `AzureGraph::az_user`, `AzureGraph::az_group`

Examples

```
## Not run:

# 'consumers' tenant -> personal OneDrive for a user
gr <- AzureGraph::get_graph_login("consumers", app="myapp")
me <- gr$get_user()
me$get_drive()

# organisational tenant -> business OneDrive for a user
gr2 <- AzureGraph::get_graph_login("mycompany", app="myapp")
myuser <- gr2$get_user("username@mycompany.onmicrosoft.com")
myuser$get_drive()

# get a site/drive directly from a URL/ID
gr2$get_sharepoint_site("My site")
gr2$get_drive("drive-id")

# site/drive(s) for a group
grp <- gr2$get_group("group-id")
grp$get_sharepoint_site()
grp$list_drives()
grp$get_drive()


## End(Not run)
```

`get_personal_onedrive` *Login clients for Microsoft 365*

Description

Microsoft365R provides functions for logging into each Microsoft 365 service.

Usage

```
get_personal_onedrive(
  app = .microsoft365r_app_id,
  scopes = c("Files.ReadWrite.All", "User.Read"),
  token = NULL,
  ...
)

get_business_onedrive(
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
  app = Sys.getenv("CLIMICROSOFT365_AADAPPID"),
  scopes = c("Files.ReadWrite.All", "User.Read"),
  token = NULL,
  ...
```

```
)

get_sharepoint_site(
  site_name = NULL,
  site_url = NULL,
  site_id = NULL,
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
  app = Sys.getenv("CLIMICROSOFT365_AADAPPID"),
  scopes = c("Group.ReadWrite.All", "Directory.Read.All", "Sites.ReadWrite.All",
    "Sites.Manage.All"),
  token = NULL,
  ...
)

list_sharepoint_sites(
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
  app = Sys.getenv("CLIMICROSOFT365_AADAPPID"),
  scopes = c("Group.ReadWrite.All", "Directory.Read.All", "Sites.ReadWrite.All",
    "Sites.Manage.All"),
  token = NULL,
  ...
)

get_team(
  team_name = NULL,
  team_id = NULL,
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
  app = Sys.getenv("CLIMICROSOFT365_AADAPPID"),
  scopes = c("Group.ReadWrite.All", "Directory.Read.All"),
  token = NULL,
  ...
)

list_teams(
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
  app = Sys.getenv("CLIMICROSOFT365_AADAPPID"),
  scopes = c("Group.ReadWrite.All", "Directory.Read.All"),
  token = NULL,
  ...
)

get_personal_outlook(
  app = .microsoft365r_app_id,
  scopes = c("Mail.Send", "Mail.ReadWrite", "User.Read"),
  token = NULL,
  ...
)
```

```
get_business_outlook(
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
  app = .microsoft365r_app_id,
  shared_mbox_id = NULL,
  shared_mbox_name = NULL,
  shared_mbox_email = NULL,
  scopes = c("User.Read", "Mail.Send", "Mail.ReadWrite"),
  token = NULL,
  ...
)

get_chat(
  chat_id,
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
  app = .microsoft365r_app_id,
  scopes = c("User.Read", "Directory.Read.All", "Chat.ReadWrite"),
  token = NULL,
  ...
)

list_chats(
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
  app = .microsoft365r_app_id,
  scopes = c("User.Read", "Directory.Read.All", "Chat.ReadWrite"),
  token = NULL,
  ...
)
```

Arguments

app	A custom app registration ID to use for authentication. See below.
scopes	The Microsoft Graph scopes (permissions) to obtain. It should never be necessary to change these.
token	An AAD OAuth token object, of class AzureAuth::AzureToken. If supplied, the tenant, app, scopes and ... arguments will be ignored. See "Authenticating with a token" below.
...	Optional arguments that will ultimately be passed to AzureAuth::get_azure_token.
tenant	For get_business_onedrive, get_sharepoint_site and get_team, the name of your Azure Active Directory (AAD) tenant. If not supplied, use the value of the CLIMICROSOFT365_TENANT environment variable, or "common" if that is unset.
site_name, site_url, site_id	For get_sharepoint_site, either the name, web URL or ID of the SharePoint site to retrieve. Supply exactly one of these.
team_name, team_id	For get_team, either the name or ID of the team to retrieve. Supply exactly one of these.

`shared_mbox_id`, `shared_mbox_name`, `shared_mbox_email`
: For `get_business_outlook`, an ID/principal name/email address. Supply exactly one of these to retrieve a shared mailbox. If all are NULL (the default), retrieve your own mailbox.

`chat_id`
: For `get_chat`, the ID of a group, one-on-one or meeting chat in Teams.

Details

These functions provide easy access to the various collaboration services that are part of Microsoft 365. On first use, they will call your web browser to authenticate with Azure Active Directory, in a similar manner to other web apps. You will get a dialog box asking for permission to access your information. You only have to authenticate once; your credentials will be saved and reloaded in subsequent sessions.

When authenticating, you can pass optional arguments in `...` which will ultimately be received by `AzureAuth::get_azure_token`. In particular, if your machine doesn't have a web browser available to authenticate with (for example if you are in a remote RStudio Server session), pass `auth_type="device_code"` which is intended for such scenarios.

Authenticating to Microsoft 365 Business services:

Authenticating to Microsoft 365 Business services (Teams, SharePoint and business OneDrive/Outlook) has some specific complexities.

The default "common" tenant for `get_team`, `get_business_onedrive` and `get_sharepoint_site` attempts to detect your actual tenant from your saved credentials in your browser. This may not always succeed, for example if you have a personal account that is also a guest account in a tenant. In this case, supply the actual tenant name, either in the `tenant` argument or in the `CLIMICROSOFT365_TENANT` environment variable. The latter allows sharing authentication details with the CLI for Microsoft 365.

The default when authenticating to these services is for Microsoft365R to use its own internal app ID. As an alternative, you (or your admin) can create your own app registration in Azure: for use in a local session, it should have a native redirect URI of `http://localhost:1410`, and the "public client" option should be enabled if you want to use the device code authentication flow. You can supply your app ID either via the `app` argument, or in the environment variable `CLIMICROSOFT365_AADAPPID`.

Authenticating with a token:

In some circumstances, it may be desirable to carry out authentication/authorization as a separate step prior to making requests to the Microsoft 365 REST API. This holds in a Shiny app, for example, since only the UI part can talk to the browser while the server part does the rest of the work. Another scenario is if the refresh token lifetime set by your org is too short, so that the token expires in between R sessions.

In this case, you can authenticate by obtaining a new token with `AzureAuth::get_azure_token`, and passing the token object to the client function. Note that the token is accepted as-is; no checks are performed that it has the correct permissions for the service you're using.

When calling `get_azure_token`, the scopes you should use are those given in the `scopes` argument for each client function, and the API host is `https://graph.microsoft.com/`. The Microsoft365R internal app ID is `d44a05d5-c6a5-4bbb-82d2-443123722380`, while that for the CLI for Microsoft 365 is `31359c7f-bd7e-475c-86db-fdb8c937548e`. However, these app IDs

only work for a local R session; you must create your own app registration if you want to use the package inside a Shiny app.

See the examples below, and also the vignette "Using Microsoft365R in a Shiny app" for a more detailed rundown on combining Microsoft365R and Shiny.

Clearing the cache:

Deleting your cached credentials is a way of rebooting the authentication process, if you are repeatedly encountering errors. To do this, call `AzureAuth::clean_token_directory`, then try logging in again. You may also need to clear your browser's cookies, if you are authenticating interactively.

Value

For `get_personal_onedrive` and `get_business_onedrive`, an R6 object of class `ms_drive`.

For `get_sharepoint_site`, an R6 object of class `ms_site`; for `list_sharepoint_sites`, a list of such objects.

For `get_team`, an R6 object of class `ms_team`; for `list_teams`, a list of such objects.

See Also

`ms_drive`, `ms_site`, `ms_team`, `ms_chat`, Microsoft365R global options

`add_methods` for the associated methods that this package adds to the base AzureGraph classes.

The "Authentication" vignette has more details on the authentication process, including troubleshooting and fixes for common problems. The "Using Microsoft365R in a Shiny app" vignette has further Shiny-specific information, including how to configure the necessary app registration in Azure Active Directory.

CLI for Microsoft 365 – a commandline tool for managing Microsoft 365

Examples

```
## Not run:

get_personal_onedrive()

# authenticating without a browser
get_personal_onedrive(auth_type="device_code")

odb <- get_business_onedrive("mycompany")
odb$list_items()

mysite <- get_sharepoint_site("My site", tenant="mycompany")
mysite <- get_sharepoint_site(site_url="https://mycompany.sharepoint.com/sites/my-site-url")
mysite$get_drive()$list_items()

myteam <- get_team("My team", tenant="mycompany")
myteam$list_channels()
myteam$get_drive()$list_items()

# retrieving chats
```

```
get_chat("chat-id")
list_chats()

# you can also use your own app registration ID:
get_business_onedrive(app="app_id")
get_sharepoint_site("My site", app="app_id")

# using the app ID for the CLI for Microsoft 365: set a global option
options(microsoft365r_use_cli_app_id=TRUE)
get_business_onedrive()
get_sharepoint_site("My site")
get_team("My team")

# authenticating separately to working with the MS365 API
scopes <- c(
    "https://graph.microsoft.com/Files.ReadWrite.All",
    "https://graph.microsoft.com/User.Read",
    "openid", "offline_access"
)
app <- "d44a05d5-c6a5-4bbb-82d2-443123722380" # for local use only
token <- AzureAuth::get_azure_token(scopes, "mycompany", app, version=2)
get_business_onedrive(token=token)


## End(Not run)
```

`microsoft365r_options` *Global options*

Description

Microsoft365R has a number of global options that affect how it interacts with the underlying Graph API.

Usage

```
options(microsoft365r_use_itemid_in_path = TRUE)
options(microsoft365r_use_outlook_immutable_ids = TRUE)
```

Details

The `microsoft365r_use_itemid_in_path` option controls when to use item IDs in requests for OneDrive/SharePoint drive items. The default value of TRUE means to use this always; other possible values are FALSE (the default in previous versions of Microsoft365R) and "remote" (use only when dealing with items shared by another user).

The `microsoft365r_use_outlook_immutable_ids` option controls whether to use immutable object IDs in Outlook. Immutable IDs have the advantage that they don't change when an email is moved or copied between folders, whereas traditional Outlook object IDs can change. The default is to use immutable IDs; set this option to FALSE to revert to traditional Outlook IDs.

ms_channel *Teams channel*

Description

Class representing a Microsoft Teams channel.

Format

An R6 object of class `ms_channel`, inheriting from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for the parent drive.
- `type`: Always "channel" for a channel object
- `team_id`: The ID of the parent team.
- `properties`: The item properties (metadata).

Methods

- `new(...)`: Initialize a new object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete this channel. By default, ask for confirmation first.
- `update(...)`: Update the channel's properties (metadata) in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the channel.
- `sync_fields()`: Synchronise the R object with the channel metadata in Microsoft Graph.
- `send_message(body, content_type, attachments)`: Sends a new message to the channel. See below.
- `list_messages(filter=NULL, n=50)`: Retrieves the messages in the channel. By default, this is limited to the 50 most recent messages; set the `n` argument to change this.
- `get_message(message_id)`: Retrieves a specific message in the channel.
- `delete_message(message_id, confirm=TRUE)`: Deletes a message. Currently the Graph API does not support deleting Teams messages, so this method is disabled.
- `list_files()`: List the files for the channel. See `ms_drive` for the arguments available for this and the file upload/download methods.
- `upload_file()`: Uploads a file to the channel.
- `download_file()`: Downloads a file from the channel.
- `get_folder()`: Retrieves the files folder for the channel, as a `ms_drive_item` object.
- `list_members(filter=NULL, n=Inf)`: Retrieves the members of the channel, as a list of `ms_team_member` objects.
- `get_member(name, email, id)`: Retrieve a specific member of the channel, as a `ms_team_member` object. Supply only one of the member name, email address or ID.

Initialization

Creating new objects of this class should be done via the `get_channel` and `list_channels` methods of the `ms_team` class. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual channel.

Messaging

To send a message to a channel, use the `send_message()` method. This has arguments:

- `body`: The body of the message. This should be a character vector, which will be concatenated into a single string with newline separators. The body can be either plain text or HTML formatted.
- `content_type`: Either "text" (the default) or "html".
- `attachments`: Optional vector of filenames.
- `inline`: Optional vector of image filenames that will be inserted into the body of the message. The images must be PNG or JPEG, and the `content_type` argument must be "html" to include inline content.
- `mentions`: Optional vector of @mentions that will be inserted into the body of the message. This should be either an object of one of the following classes, or a list of the same: `AzureGraph::az_user`, `ms_team`, `ms_channel`, `ms_team_member`. The `content_type` argument must be "html" to include mentions.

Note that message attachments are actually uploaded to the channel's file listing (a directory in the team's primary shared document folder). Support for attachments is somewhat experimental, so if you want to be sure that it works, upload the file separately using the `upload_file()` method.

List methods

All `list_*` methods have `filter` and `n` arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are `filter=NULL` and `n=Inf`. If `n=NULL`, the `ms_graph_pager` iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

`ms_team`, `ms_drive`, `ms_chat_message`

Microsoft Graph overview, Microsoft Teams API reference

Examples

```
## Not run:

myteam <- get_team("my team")
```

```
myteam$list_channels()

chan <- myteam$get_channel()
chan$list_messages()
chan$send_message("hello from R")

# a multi-line message with an attachment
msg_text <- c(
    "message line 1",
    "message line 2",
    "message line 3"
)
chan$send_message(msg_text, attachments="myfile.csv")

# sending an inline image
chan$send_message("", content_type="html", inline="graph.png")

# channel members
chan$list_members()
jane <- chan$get_member("Jane Smith")
bill <- chan$get_member(email="billg@mycompany.com")

# mentioning a team member
chan$send_message("Here is a message", content_type="html", mentions=jane)

# mentioning 2 or more members: use a list
chan$send_message("Here is another message", content_type="html",
    mentions=list(jane, bill))

# mentioning an entire channel or team
chan$send_message("FYI to channel", content_type="html", mentions=chan)
chan$send_message("FYI to everyone", content_type="html", mentions=myteam)

chan$list_files()

chan$upload_file("mydocument.docx")


## End(Not run)
```

`ms_chat` *Teams chat*

Description

Class representing a one-on-one, group or meeting chat in Microsoft Teams.

Format

An R6 object of class `ms_chat`, inheriting from `ms_object`.

Fields

- token: The token used to authenticate with the Graph host.
- tenant: The Azure Active Directory tenant for the parent drive.
- type: Always "chat" for a chat object.
- properties: The item properties (metadata).

Methods

- new(...): Initialize a new object. Do not call this directly; see 'Initialization' below.
- update(...): Update the chat's properties (metadata) in Microsoft Graph.
- do_operation(...): Carry out an arbitrary operation on the chat.
- sync_fields(): Synchronise the R object with the chat metadata in Microsoft Graph.
- send_message(body, content_type, attachments): Sends a new message to the chat. See below.
- list_messages(filter=NULL, n=50): Retrieves the messages in the chat. By default, this is limited to the 50 most recent messages; set the n argument to change this.
- get_message(message_id): Retrieves a specific message in the chat.
- delete_message(message_id, confirm=TRUE): Deletes a message. Currently the Graph API does not support deleting Teams messages, so this method is disabled.
- list_members(filter=NULL, n=Inf): Retrieves the members of the chat, as a list of ms_team_member objects.
- get_member(name, email, id): Retrieve a specific member of the chat, as a ms_team_member object. Supply only one of the member name, email address or ID.

Initialization

Creating new objects of this class should be done via the get_chat and list_chats methods of the ms_team class. Calling the new() method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual chat.

Messaging

To send a message to a chat, use the send_message() method. This has arguments:

- body: The body of the message. This should be a character vector, which will be concatenated into a single string with newline separators. The body can be either plain text or HTML formatted.
- content_type: Either "text" (the default) or "html".
- attachments: Optional vector of filenames.
- inline: Optional vector of image filenames that will be inserted into the body of the message. The images must be PNG or JPEG, and the content_type argument must be "html" to include inline content.

- mentions: Optional vector of @mentions that will be inserted into the body of the message. This should be either an object of one of the following classes, or a list of the same: AzureGraph::az_user, ms_team, ms_channel, ms_team_member. The content_type argument must be "html" to include mentions.

Message attachments are uploaded to your OneDrive for Business, in the folder "Microsoft Teams Chat Files". This is the same method as used by the regular Teams app. Unlike the Teams app, no localisation is performed, so the folder is always the same regardless of your language settings. As with channels, support for attachments is still somewhat experimental so please report any bugs found.

List methods

All list_* methods have filter and n arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are filter=NULL and n=Inf. If n=NULL, the ms_graph_pager iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

ms_team, ms_drive, ms_chat, ms_chat_message

Microsoft Graph overview, Microsoft Teams API reference

Examples

```
## Not run:

chat <- get_chat("chat-id")

# a multi-line message with an attachment
msg_text <- c(
    "message line 1",
    "message line 2",
    "message line 3"
)
chat$send_message(msg_text, attachments="myfile.csv")

# sending an inline image
chat$send_message("", content_type="html", inline="graph.png")

# chat members
chat$list_members()
jane <- chat$get_member("Jane Smith")
bill <- chat$get_member(email="billg@mycompany.com")

# mentioning a team member
```

```
chat$send_message("Here is a message", content_type="html", mentions=jane)

# mentioning 2 or more members: use a list
chat$send_message("Here is another message", content_type="html",
    mentions=list(jane, bill))


## End(Not run)
```

`ms_chat_message` *Teams chat message*

Description

Class representing a message in a Teams channel. Currently Microsoft365R only supports channels, not chats between individuals.

Format

An R6 object of class `ms_chat_message`, inheriting from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for the parent drive.
- `type`: Always "Teams message" for a chat message object.
- `properties`: The item properties (metadata).

Methods

- `new(...)`: Initialize a new object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete this message.
- `update(...)`: Update the message's properties (metadata) in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the message.
- `sync_fields()`: Synchronise the R object with the message metadata in Microsoft Graph.
- `send_reply(body, content_type, attachments)`: Sends a reply to the message. See below.
- `list_replies(filter=NULL, n=50)`: List the replies to this message. By default, this is limited to the 50 most recent replies; set the `n` argument to change this.
- `get_reply(message_id)`: Retrieves a specific reply to the message.
- `delete_reply(message_id, confirm=TRUE)`: Deletes a reply to the message. Currently the Graph API does not support deleting Teams messages, so this method is disabled.

Initialization

Creating new objects of this class should be done via the `get_message` and `list_messages` method of the `ms_team` class. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual message.

Replying to a message

To reply to a message, use the `send_reply()` method. This has arguments:

- `body`: The body of the message. This should be a character vector, which will be concatenated into a single string with newline separators. The body can be either plain text or HTML formatted.
- `content_type`: Either "text" (the default) or "html".
- `attachments`: Optional vector of filenames.
- `inline`: Optional vector of image filenames that will be inserted into the body of the message. The images must be PNG or JPEG, and the `content_type` argument must be "html" to include inline content.
- `mentions`: Optional vector of @mentions that will be inserted into the body of the message. This should be either an object of one of the following classes, or a list of the same: `AzureGraph::az_user`, `ms_team`, `ms_channel`, `ms_team_member`. The `content_type` argument must be "html" to include mentions.

Teams channels don't support nested replies, so any methods dealing with replies will fail if the message object is itself a reply.

Note that message attachments are actually uploaded to the channel's file listing (a directory in the team's primary shared document folder). Support for attachments is somewhat experimental, so if you want to be sure that it works, upload the file separately using the channel's `upload_file()` method.

List methods

All `list_*` methods have `filter` and `n` arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are `filter=NULL` and `n=Inf`. If `n=NULL`, the `ms_graph_pager` iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

`ms_team`, `ms_channel`

Microsoft Graph overview, Microsoft Teams API reference

Examples

```
## Not run:

myteam <- get_team("my team")

chan <- myteam$get_channel()
msg <- chan$list_messages()[[1]]
msg$list_replies()
msg$send_reply("Reply from R")


## End(Not run)
```

`ms_drive` *Personal OneDrive or SharePoint document library*

Description

Class representing a personal OneDrive or SharePoint document library.

Format

An R6 object of class `ms_drive`, inheriting from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for this drive.
- `type`: always "drive" for a drive object.
- `properties`: The drive properties.

Methods

- `new(...)`: Initialize a new drive object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete a drive. By default, ask for confirmation first.
- `update(...)`: Update the drive metadata in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the drive.
- `sync_fields()`: Synchronise the R object with the drive metadata in Microsoft Graph.
- `list_items(...)`, `list_files(...)`: List the files and folders under the specified path. See 'File and folder operations' below.
- `download_file(src, srcid, dest, overwrite)`: Download a file.
- `download_folder(src, srcid, dest, overwrite, recursive, parallel)`: Download a folder.
- `upload_file(src, dest, blocksize)`: Upload a file.
- `upload_folder(src, dest, blocksize, recursive, parallel)`: Upload a folder.

- `create_folder(path)`: Create a folder.
- `open_item(path, itemid)`: Open a file or folder.
- `create_share_link(...)`: Create a shareable link for a file or folder.
- `delete_item(path, itemid, confirm, by_item)`: Delete a file or folder. By default, ask for confirmation first. For personal OneDrive, deleting a folder will also automatically delete its contents; for business OneDrive or SharePoint document libraries, you may need to set `by_item=TRUE` to delete the contents first depending on your organisation's policies. Note that this can be slow for large folders.
- `get_item(path, itemid)`: Get an item representing a file or folder.
- `get_item_properties(path, itemid)`: Get the properties (metadata) for a file or folder.
- `set_item_properties(path, itemid, ...)`: Set the properties for a file or folder.
- `copy_item(path, itemid, dest, dest_item_id)`: Copy a file or folder.
- `move_item(path, itemid, dest, dest_item_id)`: Move a file or folder.
- `list_shared_items(...)`, `list_shared_files(...)`: List the drive items shared with you. See 'Shared items' below.
- `load_dataframe(path, itemid, ...)`: Download a delimited file and return its contents as a data frame. See 'Saving and loading data' below.
- `load_rds(path, itemid)`: Download a .rds file and return the saved object.
- `load_rdata(path, itemid)`: Load a .RData or .Rda file into the specified environment.
- `save_dataframe(df, file, ...)` Save a dataframe to a delimited file.
- `save_rds(object, file)`: Save an R object to a .rds file.
- `save_rdata(..., file)`: Save the specified objects to a .RData file.

Initialization

Creating new objects of this class should be done via the `get_drive` methods of the `AzureGraph::ms_graph`, `AzureGraph::az_user` or `ms_site` classes. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual drive.

File and folder operations

This class exposes methods for carrying out common operations on files and folders. They call down to the corresponding methods for the `ms_drive_item` class. In most cases an item can be specified either by path or ID. The former is more user-friendly but subject to change if the file is moved or renamed; the latter is an opaque string but is immutable regardless of file operations.

`get_item(path, itemid)` retrieves a file or folder, as an object of class `ms_drive_item`. Specify either the path or ID, not both.

`open_item` opens the given file or folder in your browser. If the file has an unrecognised type, most browsers will attempt to download it.

`delete_item` deletes a file or folder. By default, it will ask for confirmation first.

`create_share_link(path, itemid, type, expiry, password, scope)` returns a shareable link to the item.

`get_item_properties` is a convenience function that returns the properties of a file or folder as a list.

`set_item_properties` sets the properties of a file or folder. The new properties should be specified as individual named arguments to the method. Any existing properties that aren't listed as arguments will retain their previous values or be recalculated based on changes to other properties, as appropriate. You can also call the `update` method on the corresponding `ms_drive_item` object.

- `copy_item` and `move_item` can take the destination location as either a full pathname (in the `dest` argument), or a name plus a drive item object (in the `dest_folder_item` argument). If the latter is supplied, any path in `dest` is ignored with a warning. Note that copying is an *asynchronous* operation, meaning the method returns before the copy is complete.

For copying and moving, the destination folder must exist beforehand. When copying/moving a large number of files, it's much more efficient to supply the destination folder in the `dest_folder_item` argument rather than as a path.

`list_items(path, info, full_names, pagesize)` lists the items under the specified path.

`list_files` is a synonym for `list_items`.

`download_file` and `upload_file` transfer files between the local machine and the drive. For `download_file`, the default destination folder is the current (working) directory of your R session. For `upload_file`, there is no default destination folder; make sure you specify the destination explicitly.

`download_folder` and `upload_folder` transfer all the files in a folder. If `recursive` is TRUE, all subfolders will also be transferred recursively. The `parallel` argument can have the following values:

- TRUE: A cluster with 5 workers is created
- A number: A cluster with this many workers is created
- A cluster object, created via the parallel package
- FALSE: The transfer is done serially Transferring files in parallel can result in substantial speedup for a large number of small files.

`create_folder` creates a folder with the specified path. Trying to create an already existing folder is an error.

Saving and loading data

The following methods are provided to simplify the task of loading and saving datasets and R objects. They call down to the corresponding methods for the `ms_drive_item` class. The 'load_*" methods allow specifying the file to be loaded by either a path or item ID.

- `load_dataframe` downloads a delimited file and returns its contents as a data frame. The delimiter can be specified with the `delim` argument; if omitted, this is "," if the file extension is .csv, ";" if the file extension is .csv2, and a tab otherwise. If the readr package is installed, the `readr::read_delim` function is used to parse the file, otherwise `utils::read.delim` is used. You can supply other arguments to the parsing function via the `...` argument.
- `save_dataframe` is the inverse of `load_dataframe`: it uploads the given data frame to a folder item. Specify the delimiter with the `delim` argument. The `readr::write_delim` function is used to serialise the data if that package is installed, and `utils::write.table` otherwise.

- load_rds downloads a .rds file and returns its contents as an R object. It is analogous to the base readRDS function but for OneDrive/SharePoint drive items.
- save_rds uploads a given R object as a .rds file, analogously to saveRDS.
- load_rdata downloads a .RData or .Rda file and loads its contents into the given environment. It is analogous to the base load function but for OneDrive/SharePoint drive items.
- save_rdata uploads the given R objects as a .RData file, analogously to save.

Shared items

The list_shared_items method shows the files and folders that have been shared with you. This is a named list of drive items, that you can use to access the shared files/folders. The arguments are:

- allow_external: Whether to include items that were shared from outside tenants. The default is FALSE.
- filter, n: See 'List methods' below.
- pagesize: The number of results to return for each call to the REST endpoint. You can try reducing this argument below the default of 1000 if you are experiencing timeouts.
- info: Deprecated, will be ignored. In previous versions, controlled the return type of the method.

list_shared_files is a synonym for list_shared_items.

Because of how the Graph API handles access to shared items linked in the root, you cannot directly access subitems of shared folders via the drive get_item method, like this: drv$get_item("shared_folder/path/to/file Instead, get the item into its own object, and use its get_item method: drv$get_item("shared_folder")$get_item("path

List methods

All list_* methods have filter and n arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are filter=NULL and n=Inf. If n=NULL, the ms_graph_pager iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

get_personal_onedrive, get_business_onedrive, ms_site, ms_drive_item

Microsoft Graph overview, OneDrive API reference

Examples

```
## Not run:

# personal OneDrive
mydrv <- get_personal_onedrive()
```

```
# OneDrive for Business
busdrv <- get_business_onedrive("mycompany")

# shared document library for a SharePoint site
site <- get_sharepoint_site("My site")
drv <- site$get_drive()

## file/folder operationss
drv$list_files()
drv$list_files("path/to/folder", full_names=TRUE)

# download a file -- default destination filename is taken from the source
drv$download_file("path/to/folder/data.csv")

# shareable links
drv$create_share_link("myfile")
drv$create_share_link("myfile", type="edit", expiry="24 hours")
drv$create_share_link("myfile", password="Use-strong-passwords!")

# file metadata (name, date created, etc)
drv$get_item_properties("myfile")

# rename a file -- item ID remains the same, while name is changed
obj <- drv$get_item("myfile")
drv$set_item_properties("myfile", name="newname")

# retrieve the renamed object by ID
id <- obj$properties$id
obj2 <- drv$get_item(itemid=id)
obj$properties$id == obj2$properties$id  # TRUE

# saving and loading data
drv$save_dataframe(iris, "path/to/iris.csv")
iris2 <- drv$load_dataframe("path/to/iris.csv")
identical(iris, iris2)  # TRUE

drv$save_rds(iris, "path/to/iris.rds")
iris3 <- drv$load_rds("path/to/iris.rds")
identical(iris, iris3)  # TRUE

# accessing shared files
shared_df <- drv$list_shared_items()
shared_df$remoteItem[[1]]$open()
shared_items <- drv$list_shared_items(info="items")
shared_items[[1]]$open()


## End(Not run)
```

`ms_drive_item` *File or folder in a drive*

Description

Class representing an item (file or folder) in a OneDrive or SharePoint document library.

Format

An R6 object of class `ms_drive_item`, inheriting from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for the parent drive.
- `type`: always "drive item" for a drive item object.
- `properties`: The item properties (metadata).

Methods

- `new(...)`: Initialize a new object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE, by_item=FALSE)`: Delete this item. By default, ask for confirmation first. For personal OneDrive, deleting a folder will also automatically delete its contents; for business OneDrive or SharePoint document libraries, you may need to set `by_item=TRUE` to delete the contents first depending on your organisation's policies. Note that this can be slow for large folders.
- `update(...)`: Update the item's properties (metadata) in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the item.
- `sync_fields()`: Synchronise the R object with the item metadata in Microsoft Graph.
- `open()`: Open the item in your browser.
- `copy(dest, dest_folder_item=NULL)`: Copy the item to the given location.
- `move(dest, dest_folder_item=NULL)`: Move the item to the given location.
- `list_items(...), list_files(...)`: List the files and folders under the specified path.
- `download(dest, overwrite, recursive, parallel)`: Download the file or folder. See below.
- `create_share_link(type, expiry, password, scope)`: Create a shareable link to the file or folder.
- `upload(src, dest, blocksize, , recursive, parallel)`: Upload a file or folder. See below.
- `create_folder(path)`: Create a folder. Only applicable for a folder item.
- `get_item(path)`: Get a child item (file or folder) under this folder.
- `get_parent_folder()`: Get the parent folder for this item, as a drive item object. Returns the root folder for the root. Not supported for remote items.
- `get_path()`: Get the absolute path for this item, as a character string. Not supported for remote items.
- `is_folder()`: Information function, returns TRUE if this item is a folder.

- `load_dataframe(delim=NULL, ...)`: Download an Excel or text file and return its contents as a data frame. See 'Saving and loading data' below.
- `load_rds()`: Download a .rds file and return the saved object.
- `load_rdata(envir)`: Load a .RData or .Rda file into the specified environment.
- `save_dataframe(df, file, delim=",", ...)` Save a dataframe to a delimited file.
- `save_rds(object, file)`: Save an R object to a .rds file.
- `save_rdata(..., file)`: Save the specified objects to a .RData file.

Initialization

Creating new objects of this class should be done via the `get_item` method of the `ms_drive` class. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual item.

File and folder operations

This class exposes methods for carrying out common operations on files and folders. Note that for the methods below, any paths to child items are relative to the folder's own path.

`open` opens this file or folder in your browser. If the file has an unrecognised type, most browsers will attempt to download it.

`list_items(path, info, full_names, filter, n, pagesize)` lists the items under the specified path. It is the analogue of base R's `dir`/`list.files`. Its arguments are

- `path`: The path.
- `info`: The information to return: either "partial", "name" or "all". If "partial", a data frame is returned containing the name, size, ID and whether the item is a file or folder. If "name", a vector of file/folder names is returned. If "all", a data frame is returned containing *all* the properties for each item (this can be large).
- `full_names`: Whether to prefix the folder path to the names of the items.
- `filter, n`: See 'List methods' below.
- `pagesize`: The number of results to return for each call to the REST endpoint. You can try reducing this argument below the default of 1000 if you are experiencing timeouts.

`list_files` is a synonym for `list_items`.

`download` downloads the item to the local machine. If this is a file, it is downloaded; in this case, the `dest` argument can be the path to the destination file, or NULL to return the downloaded content in a raw vector. If the item is a folder, all its files are downloaded, including subfolders if the `recursive` argument is TRUE.

`upload` uploads a file or folder from the local machine into the folder item. The `src` argument can be the path to the source file, a rawConnection or a textConnection object. If `src` is a folder, all its files are uploaded, including subfolders if the `recursive` argument iS TRUE. An `ms_drive_item` object is returned invisibly.

Uploading is done in blocks of 32MB by default; you can change this by setting the `blocksize` argument. For technical reasons, the block size must be a multiple of 320KB.

Uploading and downloading folders can be done in parallel, which can result in substantial speedup when transferring a large number of small files. This is controlled by the `parallel` argument to `upload` and `download`, which can have the following values:

- TRUE: A cluster with 5 workers is created
- A number: A cluster with this many workers is created
- A cluster object, created via the parallel package
- FALSE: The transfer is done serially

`get_item` retrieves the file or folder with the given path, as another object of class `ms_drive_item`.

- `copy` and `move` can take the destination location as either a full pathname (in the `dest` argument), or a name plus a drive item object (in the `dest_folder_item` argument). If the latter is supplied, any path in `dest` is ignored with a warning. Note that copying is an *asynchronous* operation, meaning the method returns before the copy is complete.

For copying and moving, the destination folder must exist beforehand. When copying/moving a large number of files, it's much more efficient to supply the destination folder in the `dest_folder_item` argument rather than as a path.

`create_folder` creates a folder with the specified path. Trying to create an already existing folder is an error. This returns an `ms_drive_item` object, invisibly.

`create_share_link(path, type, expiry, password, scope)` returns a shareable link to the item. Its arguments are

- `path`: The path.
- `type`: Either "view" for a read-only link, "edit" for a read-write link, or "embed" for a link that can be embedded in a web page. The last one is only available for personal OneDrive.
- `expiry`: How long the link is valid for. The default is 7 days; you can set an alternative like "15 minutes", "24 hours", "2 weeks", "3 months", etc. To leave out the expiry date, set this to NULL.
- `password`: An optional password to protect the link.
- `scope`: Optionally the scope of the link, either "anonymous" or "organization". The latter allows only users in your AAD tenant to access the link, and is only available for OneDrive for Business or SharePoint.

This method returns a URL to access the item, for `type="view"` or `"type=edit"`. For `type="embed"`, it returns a list with components `webUrl` containing the URL, and `webHtml` containing a HTML fragment to embed the link in an IFRAME. The default is a viewable link, expiring in 7 days.

Saving and loading data

The following methods are provided to simplify the task of loading and saving datasets and R objects.

- `load_dataframe` downloads an Excel (.xlsx or .xls extension) or text file and returns its contents as a data frame. Loading Excel files uses the `readxl::read_excel` function, and so requires that the readxl package is installed. For a text file, you can specify the delimiter with the `delim` argument; if omitted, this is "," if the file extension is .csv, ";" if the file extension is .csv2, and a tab otherwise. If the readr package is installed, the `readr::read_delim` function is used to parse text files, otherwise `utils::read.delim` is used. You can supply other arguments to these functions via the `...` argument.

- save_dataframe is the inverse of load_dataframe: it uploads the given data frame to a folder item. Specify the delimiter with the delim argument. The readr::write_delim function is used to serialise the data if that package is installed, and utils::write.table otherwise. Note that unlike loading, saving to an Excel file is not supported.
- load_rds downloads a .rds file and returns its contents as an R object. It is analogous to the base readRDS function but for OneDrive/SharePoint drive items.
- save_rds uploads a given R object as a .rds file, analogously to saveRDS.
- load_rdata downloads a .RData or .Rda file and loads its contents into the given environment. It is analogous to the base load function but for OneDrive/SharePoint drive items.
- save_rdata uploads the given R objects as a .RData file, analogously to save.

List methods

All list_* methods have filter and n arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are filter=NULL and n=Inf. If n=NULL, the ms_graph_pager iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

AzureGraph::ms_graph, ms_site, ms_drive

Microsoft Graph overview, OneDrive API reference

Examples

```
## Not run:

# personal OneDrive
mydrv <- get_personal_onedrive()

docs <- mydrv$get_item("Documents")
docs$list_files()
docs$list_items()

# this is the file 'Documents/myfile.docx'
myfile <- docs$get_item("myfile.docx")
myfile$properties

# rename a file
myfile$update(name="newname.docx")

# copy a file (destination folder must exist)
myfile$copy("/Documents/folder2/myfile_copied.docx")
```

```
# alternate way of copying: supply the destination folder
destfolder <- docs$get_item("folder2")
myfile$copy("myfile_copied.docx", dest_folder_item=destfolder)

# move a file (destination folder must exist)
myfile$move("Documents/folder2/myfile_moved.docx")

# open the file in the browser
myfile$open()

# download the file to the working directory
myfile$download()

# shareable links
myfile$create_share_link()
myfile$create_share_link(type="edit", expiry="24 hours")
myfile$create_share_link(password="Use-strong-passwords!")

# delete the file (will ask for confirmation first)
myfile$delete()

# saving and loading data
myfolder <- mydrv$get_item("myfolder")
myfolder$save_dataframe(iris, "iris.csv")
iris2 <- myfolder$get_item("iris.csv")$load_dataframe()
identical(iris, iris2)  # TRUE

myfolder$save_rds(iris, "iris.rds")
iris3 <- myfolder$get_item("iris.rds")$load_rds()
identical(iris, iris3)  # TRUE


## End(Not run)
```

`ms_list` *Sharepoint list*

Description

Class representing a list in a SharePoint site.

Format

An R6 object of class `ms_list`, inheriting from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for the parent drive.
- `type`: always "list" for a SharePoint list object.
- `properties`: The item properties (metadata).

Methods

- `new(...)`: Initialize a new object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete this list. By default, ask for confirmation first.
- `update(...)`: Update the list's properties in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the list.
- `sync_fields()`: Synchronise the R object with the list metadata in Microsoft Graph.
- `list_items(filter, select, all_metadata, as_data_frame, pagesize)`: Queries the list and returns items as a data frame. See 'List querying' below.
- `get_column_info()`: Return a data frame containing metadata on the columns (fields) in the list.
- `get_item(id)`: Get an individual list item.
- `create_item(...)`: Create a new list item, using the named arguments as fields.
- `update_item(id, ...)`: Update the *data* fields in the given item, using the named arguments. To update the item's metadata, use `get_item()` to retrieve the item object, then call its `update()` method.
- `delete_item(confirm=TRUE)`: Delete a list item. By default, ask for confirmation first.
- `bulk_import(data)`: Imports a data frame into the list.

Initialization

Creating new objects of this class should be done via the `get_list` method of the `ms_site` class. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual item.

List querying

`list_items` supports the following arguments to customise results returned by the query.

- `filter`: A string giving an OData expression to filter the rows to return. Note that column names used in the expression must be prefixed with `fields/` to distinguish them from item metadata.
- `n`: The maximum number of (filtered) results to return. If this is NULL, the `ms_graph_pager` iterator object is returned instead to allow manual iteration over the results.
- `select`: A string containing comma-separated column names to include in the returned data frame. If not supplied, includes all columns.
- `all_metadata`: If TRUE, the returned data frame will contain extended metadata as separate columns, while the data fields will be in a nested data frame named `fields`. This is always set to FALSE if `n=NULL` or `as_data_frame=FALSE`.
- `as_data_frame`: If FALSE, return the result as a list of individual `ms_list_item` objects, rather than a data frame.
- `pagesize`: The number of results to return for each call to the REST endpoint. You can try reducing this argument below the default of 5000 if you are experiencing timeouts.

Note that the Graph API currently doesn't support retrieving item attachments.

See Also

get_sharepoint_site, ms_site, ms_list_item

Microsoft Graph overview, SharePoint sites API reference

Examples

```
## Not run:

site <- get_sharepoint_site("My site")
lst <- site$get_list("mylist")

lst$get_column_info()

lst$list_items()
lst$list_items(filter="startswith(fields/firstname, 'John')", select="firstname,lastname")

lst$create_item(firstname="Mary", lastname="Smith")
lst$get_item("item-id")
lst$update_item("item_id", firstname="Eliza")
lst$delete_item("item_id")

df <- data.frame(
    firstname=c("Satya", "Mark", "Tim", "Jeff", "Sundar"),
    lastname=c("Nadella", "Zuckerberg", "Cook", "Bezos", "Pichai")
)
lst$bulk_import(df)


## End(Not run)
```

---

ms_list_item *SharePoint list item*

Description

Class representing an item in a SharePoint list.

Format

An R6 object of class `ms_list_item`, inheriting from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for the parent drive.
- `type`: always "drive item" for a drive item object.
- `properties`: The item properties (data and metadata). This is a list; the item data can be found in the `fields` component.

Methods

- `new(...)`: Initialize a new object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete this item. By default, ask for confirmation first.
- `update(...)`: Update the item's properties (metadata) in Microsoft Graph. To update the list *data*, update the `fields` property. See the examples below.
- `do_operation(...)`: Carry out an arbitrary operation on the item.
- `sync_fields()`: Synchronise the R object with the item data and metadata in Microsoft Graph.

Initialization

Creating new objects of this class should be done via the `get_item` method of the `ms_list` class. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual item.

See Also

`AzureGraph::ms_graph`, `ms_site`, `ms_list`

Microsoft Graph overview, SharePoint sites API reference

Examples

```
## Not run:

site <- get_sharepoint_site("My site")
lst <- site$get_list("mylist")

lst_items <- lst$list_items(as_data_frame=FALSE)

item <- lst_items[[1]]

item$update(fields=list(firstname="Mary"))

# item data (plus some metadata mixed in)
item$properties$fields

item$delete()


## End(Not run)
```

ms_outlook *Outlook mail client*

Description

Class representing a user's Outlook email account.

Format

An R6 object of class ms_outlook, inheriting from ms_outlook_object, which in turn inherits from ms_object.

Fields

- token: The token used to authenticate with the Graph host.
- tenant: The Azure Active Directory tenant for the email account.
- type: always "Outlook account" for an Outlook email account.
- properties: The item properties (metadata).

Methods

- new(...): Initialize a new object. Do not call this directly; see 'Initialization' below.
- update(...): Update the account's properties (metadata) in Microsoft Graph.
- do_operation(...): Carry out an arbitrary operation on the account.
- sync_fields(): Synchronise the R object with the account metadata in Microsoft Graph.
- create_email(...): Creates a new email in the Drafts folder, optionally sending it as well. See 'Creating and sending emails'.
- list_inbox_emails(...): List the emails in the Inbox folder. See 'Listing emails'.
- get_inbox(),get_drafts(),get_sent_items(),get_deleted_items(): Gets the special folder of that name. These folders are created by Outlook and exist in every email account.
- list_folders(filter=NULL, n=Inf): List all folders in this account.
- get_folder(folder_name, folder_id): Get a folder, either by the name or ID.
- create_folder(folder_name): Create a new folder.
- delete_folder(folder_name, folder_id, confirm=TRUE): Delete a folder. By default, ask for confirmation first. Note that special folders cannot be deleted.

Initialization

Creating new objects of this class should be done via the get_personal_outlook() or get_business_outlook() functions, or the get_outlook method of the AzureGraph::az_user class. Calling the new() method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve the account information.

Creating and sending emails

To create a new email, call the `create_email()` method. The default behaviour is to create a new draft email in the Drafts folder, which can then be edited further to add attachments, recipients etc; or the email can be sent immediately.

The `create_email()` method has the following signature:

```
create_email(body = "", content_type = c("text", "html"), subject = "",
             to = NULL, cc = NULL, bcc = NULL, reply_to = NULL, send_now = FALSE)
```

- `body`: The body of the message. This should be a string or vector of strings, which will be pasted together with newlines as separators. You can also supply a message object as created by the blastula or emayili packages—see the examples below.
- `content_type`: The format of the body, either "text" (the default) or HTML.
- `subject`: The subject of the message.
- `to,cc,bcc,reply_to`: These should be lists of email addresses, in standard "user@host" format. You can also supply objects of class `AzureGraph::az_user` representing user accounts in Azure Active Directory.
- `send_now`: Whether the email should be sent immediately, or saved as a draft. You can send a draft email later with its `send()` method.

This returns an object of class `ms_outlook_email`, which has methods for making further edits, attaching files, replying, forwarding, and (re-)sending.

You can also supply message objects as created by the blastula and emayili packages in the `body` argument. Note that blastula objects include attachments (if any), and emayili objects include attachments, recipients, and subject line; the corresponding arguments to `create_email()` will not be used in this case.

Listing emails

To list the emails in the Inbox, call the `list_emails()` method. This returns a list of objects of class `ms_outlook_email`, and has the following signature:

```
list_emails(by = "received desc", n = 100, pagesize = 10)
```

- `by`: The sorting order of the message list. The possible fields are "received" (received date, the default), "from" and "subject". To sort in descending order, add a " desc". You can specify multiple sorting fields, with later fields used to break ties in earlier ones. The last sorting field is always "received desc" unless it appears earlier.
- `filter, n`: See below.
- `pagesize`: The number of emails per page. You can change this to a larger number to increase throughput, at the risk of running into timeouts.

List methods generally

All `list_*` methods have `filter` and `n` arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are `filter=NULL` and `n=100`

for listing emails, and `n=Inf` for listing folders. If `n=NULL`, the `ms_graph_pager` iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

`ms_outlook_folder`, `ms_outlook_email`

Microsoft Graph overview, Outlook API reference

Examples

```
## Not run:

outl <- get_personal_outlook()

##
## listing emails and folders
##

# the default: 100 most recent messages in the inbox
outl$list_emails()

# sorted by subject, then by most recent received date
outl$list_emails(by="subject")

# retrieve a specific email:
# note the Outlook ID is NOT the same as the Internet message-id
email_id <- outl$list_emails()[[1]]$properties$id
outl$get_email(email_id)

# all folders in this account (including nested folders)
outl$list_folders()

# draft (unsent) emails
dr <- outl$get_drafts()
dr$list_emails()

# sent emails
sent <- outl$get_sent_items()
sent$list_emails()

##
## creating/sending emails
##

# a simple text email with just a body (can't be sent)
outl$create_email("Hello from R")
```

```
# HTML-formatted email with all necessary fields, sent immediately
outl$create_email("<emph>Emphatic hello</emph> from R",
    content_type="html",
    to="user@example.com",
    subject="example email",
    send_now=TRUE)

# you can also create a blank email object and call its methods to add content
outl$create_email()$
    set_body("<emph>Emphatic hello</emph> from R", content_type="html")$
    set_recipients(to="user@example.com")$
    set_subject("example email")$
    add_attachment("mydocument.docx")$
    send()

# using blastula to create a HTML email with Markdown
bl_msg <- blastula::compose_email(md(
"
## Hello!

This is an email message that was generated by the blastula package.

We can use **Markdown** formatting with the `md()` function.

Cheers,

The blastula team
"),
    footer=md("Sent via Microsoft365R"))
outl$create_email(bl_msg, subject="example blastula email")


# using emayili to create an email with attachments
ey_email <- emayili::envelope(
    text="Hello from emayili",
    to="user@example.com",
    subject="example emayili email") %>%
    emayili::attachment("mydocument.docx") %>%
    emayili::attachment("mydata.xlsx")
outl$create_email(ey_email)


## End(Not run)
```

`ms_outlook_attachment` *Outlook mail attachment*

Description

Class representing an attachment in Outlook.

Format

An R6 object of class `ms_outlook_attachment`, inheriting from `ms_outlook_object`, which in turn inherits from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for the email account.
- `type`: always "attachment" for an attachment.
- `properties`: The attachment properties (metadata).

Methods

- `new(...)`: Initialize a new object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete this attachment. By default, ask for confirmation first.
- `update(...)`: Update the attachment's properties (metadata) in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the attachment.
- `sync_fields()`: Synchronise the R object with the attachment metadata in Microsoft Graph. This method does *not* transfer the attachment content for a file attachment.
- `download(dest, overwrite)`: For a file attachment, downloads the content to a file. The default destination filename is the name of the attachment.

Initialization

Creating new objects of this class should be done via the the `get_attachment()`, `list_attachments()` or `create_attachment()` methods `ms_outlook_email` class. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual attachment.

In general, you should not need to interact directly with this class, as the `ms_outlook_email` class exposes convenience methods for working with attachments. The only exception is to download an attachment in a reliable way (not involving the attachment name); see the example below.

See Also

`ms_outlook`, `ms_outlook_email`

Microsoft Graph overview, Outlook API reference

Examples

```
## Not run:

outl <- get_personal_outlook()

em <- outl$get_inbox$get_email("email_id")

# download the first attachment in an email
atts <- em$list_attachments()
```

```
atts[[1]]$download()


## End(Not run)
```

`ms_outlook_email` *Outlook mail message*

Description

Class representing an Outlook mail message. The one class represents both sent and unsent (draft) emails.

Format

An R6 object of class `ms_outlook_email`, inheriting from `ms_outlook_object`, which in turn inherits from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for the email account.
- `type`: always "email" for an Outlook mail message.
- `properties`: The item properties (metadata).

Methods

- `new(...)`: Initialize a new object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete this email. By default, ask for confirmation first.
- `update(...)`: Update the email's properties (metadata) in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the email.
- `sync_fields()`: Synchronise the R object with the email metadata in Microsoft Graph.
- `set_body(body=NULL, content_type=NULL)`: Update the email body. See 'Editing an email' below.
- `set_subject(subject)`: Update the email subject line.
- `set_recipients(to=NULL, cc=NULL, bcc=NULL)`: Set the recipients for the email, overwriting any existing recipients.
- `add_recipients(to=NULL, cc=NULL, bcc=NULL)`: Adds recipients for the email, leaving existing ones unchanged.
- `set_reply_to(reply_to=NULL)`: Sets the reply-to field for the email.
- `add_attachment(object, ...)`: Adds an attachment to the email. See 'Attachments' below.
- `add_image(object)`: Adds an inline image to the email.

- `get_attachment(attachment_name=NULL, attachment_id=NULL)`: Gets an attachment, either by name or ID. Note that attachments don't need to have unique names; if multiple attachments share the same name, the method throws an error.
- `list_attachments(filter=NULL, n=Inf)`: Lists the current attachments for the email.
- `remove_attachment(attachment_name=NULL, attachment_id=NULL, confirm=TRUE)`: Removes an attachment from the email. By default, ask for confirmation first.
- `download_attachment(attachment_name=NULL, attachment_id=NULL, ...)`: Downloads an attachment. This is only supported for file attachments (not URLs).
- `send()`: Sends an email. See 'Sending, replying and forwarding'.
- `create_reply(comment="", send_now=FALSE)`: Replies to the sender of an email.
- `create_reply_all(comment="", send_now=FALSE)`: Replies to the sender and all recipients of an email.
- `create_forward(comment="", to=NULL, cc=NULL, bcc=NULL, send_now=FALSE)`: Forwards the email to other recipients.
- `copy(dest),move(dest)`: Copies or moves the email to the destination folder.
- `get_message_headers`: Retrieves the Internet message headers for an email, as a named character vector.

Initialization

Creating new objects of this class should be done via the the appropriate methods for the `ms_outlook` or `ms_outlook_folder` classes. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual folder.

Editing an email

This class exposes several methods for updating the properties of an email. They should work both for unsent (draft) emails and sent ones, although they make most sense in the context of editing drafts.

`set_body(body, content_type)` updates the message body of the email. This has 2 arguments: `body` which is the body text itself, and `content_type` which should be either "text" or "html". For both arguments, you can set the value to NULL to leave the current property unchanged. The `body` argument can also be a message object from either the blastula or emayili packages, much like when creating a new email.

`set_subject(subject)` sets the subject line of the email.

`set_recipients(to, cc, bcc)` sets or clears the recipients of the email. The `to`, `cc` and `bcc` arguments should be lists of either email addresses as character strings, or objects of class `az_user` representing a user account in Azure Active Directory. The default behaviour is to overwrite any existing recipients; to avoid this, pass `NA` as the value for the relevant argument. Alternatively, you can use the `add_recipients()` method.

`add_recipients(to, cc, bcc)` is like `set_recipients()` but leaves existing recipients unchanged.

`set_reply_to(reply_to)` sets or clears the reply-to address for the email. Leave the `reply_to` argument at its default NULL value to clear this property.

Attachments

This class exposes the following methods for working with attachments.

`add_attachment(object, type, expiry, password, scope)` adds an attachment to the email. The arguments are as follows:

- `object`: A character string containing a filename or URL, or an object of class `ms_drive_item` representing a file in OneDrive or SharePoint. In the latter case, a shareable link to the drive item will be attached to the email, with the link details given by the other arguments.
- `type, expiry, password, scope`: The specifics for the shareable link to attach to the email, if `object` is a drive item. See the `create_share_link()` method of the `ms_drive_item` class; the default is to create a read-only link valid for 7 days.

`add_image(object)` adds an image as an *inline* attachment, ie, as part of the message body. The `object` argument should be a filename, and the message content type will be set to "html" if it is not already. Currently Microsoft365R does minimal formatting of the image; consider using a package like blastula for more control over the layout of inline images.

`list_attachments()` lists the attachments for the email, including inline images. This will be a list of objects of class `ms_outlook_attachment` containing the metadata for the attachments.

`get_attachment(attachment_name, attachment_id)`: Retrieves the metadata for an attachment, as an object of class `ms_outlook_attachment`. Note that multiple attachments can share the same name; in this case, you must specify the ID of the attachment.

`download_attachment(attachment_name, attachment_id, dest, overwrite)`: Downloads a file attachment. The default destination filename is the name of the attachment.

`remove_attachment(attachment_name, attachment_id)` removes (deletes) an attachment.

Sending, replying and forwarding

Microsoft365R's default behaviour when creating, replying or forwarding emails is to create a draft message object, to allow for further edits. The draft is saved in the Drafts folder by default, and can be sent later by calling its `send()` method.

The methods for replying and forwarding are `create_reply()`, `create_reply_all()` and `create_forward()`. The first argument to these is the reply text, which will appear above the current message text in the body of the reply. For `create_forward()`, the other arguments are `to`, `cc` and `bcc` to specify the recipients of the forwarded email.

Other methods

The `copy()` and `move()` methods copy and move an email to a different folder. The destination should be an object of class `ms_outlook_folder`.

The `get_message_headers()` method retrieves the Internet message headers for the email, as a named character vector.

List methods

All `list_*` methods have `filter` and `n` arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are `filter=NULL` and `n=Inf`.

If n=NULL, the ms_graph_pager iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

ms_outlook, ms_outlook_folder, ms_outlook_attachment

Microsoft Graph overview, Outlook API reference

Examples

```
## Not run:

outl <- get_personal_outlook()

##
## creating a new email
##

# a blank text email
em <- outl$create_email()

# add a body
em$set_body("Hello from R", content_type="html")

# add recipients
em$set_recipients(to="user@example.com")

# add subject line
em$set_subject("example email")

# add an attachment
em$add_attachment("mydocument.docx")

# add a shareable link to a file in OneDrive
mysheet <- get_personal_onedrive()$get_item("documents/mysheet.xlsx")
em$add_attachment(mysheet)

# add an inline image
em$add_image("myggplot.jpg")

# oops, wrong recipient, it should be someone else
# this removes user@example.com from the to: field
em$set_recipients(to="user2@example.com")

# and we should also cc a third user
em$add_recipients(cc="user3@example.com")
```

```
# send it
em$send()

# you can also compose an email as a pipeline
outl$create_email()$
    set_body("Hello from R")$
    set_recipients(to="user2@example.com", cc="user3@example.com")$
    set_subject("example email")$
    add_attachment("mydocument.docx")$
    send()

# using blastula to create a HTML email with Markdown
bl_msg <- blastula::compose_email(md(
"
## Hello!

This is an email message that was generated by the blastula package.

We can use **Markdown** formatting with the `md()` function.

Cheers,

The blastula team
"),
    footer=md("Sent via Microsoft365R"))
outl$create_email()
    set_body(bl_msg)$
    set_subject("example blastula email")


##
## replying and forwarding
##

# get the most recent email in the Inbox
em <- outl$list_emails()[[1]]

# reply to the message sender, cc'ing Carol
em$create_reply("I agree")$
    add_recipients(cc="carol@example.com")$
    send()

# reply to everyone, setting the reply-to address
em$create_reply_all("Please do not reply")$
    set_reply_to("do_not_reply@example.com")$
    send()

# forward to Dave
em$create_forward("FYI", to="dave@example.com")$
    send()


##
```

```
## attachments
##

# download an attachment by name (assumes there is only one 'myfile.docx')
em$download_attachment("myfile.docx")

# a more reliable way: get the list of attachments, and download via the object
atts <- em$list_attachments()
atts[[1]]$download()

# add and remove an attachment
em$add_attachment("anotherfile.pptx")
em$remove_attachment("anotherfile.pptx")


##
## moving and copying
##

# copy an email to a nested folder: /folder1/folder2
dest <- outl$get_folder("folder1")$get_folder("folder2")
em$copy(dest)

# move it instead
em$move(dest)


## End(Not run)
```

`ms_outlook_folder` *Outlook mail folder*

Description

Class representing a folder in Outlook.

Format

An R6 object of class `ms_outlook_folder`, inheriting from `ms_outlook_object`, which in turn inherits from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for the email account.
- `type`: always "mail folder" for an Outlook folder object.
- `user_id`: the user ID of the Outlook account.
- `properties`: The item properties (metadata).

Methods

- `new(...)`: Initialize a new object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete this folder. By default, ask for confirmation first. Note that special folders cannot be deleted.
- `update(...)`: Update the item's properties (metadata) in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the item.
- `sync_fields()`: Synchronise the R object with the item metadata in Microsoft Graph.
- `list_emails(...)`: List the emails in this folder.
- `get_email(message_id)`: Get the email with the specified ID.
- `create_email(...)`: Creates a new draft email in this folder, optionally sending it as well. See 'Creating and sending emails'.
- `delete_email(message_id, confim=TRUE)`: Deletes the specified email. By default, ask for confirmation first.
- `list_folders(filter=NULL, n=Inf)`: List subfolders of this folder.
- `get_folder(folder_name, folder_id)`: Get a subfolder, either by the name or ID.
- `create_folder(folder_name)`: Create a new subfolder of this folder.
- `delete_folder(folder_name, folder_id, confirm=TRUE)`: Delete a subfolder. By default, ask for confirmation first.
- `copy(dest),move(dest)`: Copies or moves this folder to another folder. All the contents of the folder will also be copied/moved. The destination should be an object of class `ms_outlook_folder`.

Initialization

Creating new objects of this class should be done via the `get_folder`, `list_folders` or `create_folder` methods of this class or the `ms_outlook` class. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual folder.

Creating and sending emails

Outlook allows creating new draft emails in any folder, not just the Drafts folder (although that is the default location for the Outlook app, and the `ms_outlook` client class). To create a new email, call the `create_email()` method, which has the following signature:

```
create_email(body = "", content_type = c("text", "html"), subject = "",
             to = NULL, cc = NULL, bcc = NULL, reply_to = NULL, send_now = FALSE)
```

- `body`: The body of the message. This should be a string or vector of strings, which will be pasted together with newlines as separators. You can also supply a message object as created by the blastula or emayili packages—see the examples below.
- `content_type`: The format of the body, either "text" (the default) or HTML.
- `subject`: The subject of the message.
- `to,cc,bcc,reply_to`: These should be lists of email addresses, in standard "user@host" format. You can also supply objects of class `AzureGraph::az_user` representing user accounts in Azure Active Directory.

- send_now: Whether the email should be sent immediately, or saved as a draft. You can send a draft email later with its send() method.

This returns an object of class ms_outlook_email, which has methods for making further edits and attaching files.

You can also supply message objects as created by the blastula and emayili packages in the body argument. Note that blastula objects include attachments (if any), and emayili objects include attachments, recipients, and subject line; the corresponding arguments to create_email() will not be used in this case.

To reply to or forward an email, first retrieve it using get_email() or list_emails(), and then call its create_reply(), create_reply_all() or create_forward() methods.

Listing emails

To list the emails in a folder, call the list_emails() method. This returns a list of objects of class ms_outlook_email, and has the following signature:

```
list_emails(by = "received desc", search = NULL, filter = NULL, n = 100, pagesize = 10)
```

- by: The sorting order of the message list. The possible fields are "received" (received date, the default), "from" and "subject". To sort in descending order, add a " desc". You can specify multiple sorting fields, with later fields used to break ties in earlier ones. The last sorting field is always "received desc" unless it appears earlier.
- search: An optional string to search for. Only emails that contain the search string will be returned. See the description of this parameter for more information.
- filter, n: See below.
- pagesize: The number of emails per page. You can change this to a larger number to increase throughput, at the risk of running into timeouts.

Currently, searching and filtering the message list is subject to some limitations. You can only specify one of search and filter; searching and filtering at the same time will not work. Ordering the results is only allowed if neither a search term nor a filtering expression is present. If searching or filtering is done, the result is always sorted by date.

List methods generally

All list_* methods have filter and n arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are filter=NULL and n=100 for listing emails, and n=Inf for listing folders. If n=NULL, the ms_graph_pager iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

ms_outlook, ms_outlook_email

Microsoft Graph overview, Outlook API reference

Examples

```
## Not run:

outl <- get_personal_outlook()

folder <- outl$get_folder("My folder")

##
## listing emails
##

# the default: 100 most recent messages
folder$list_emails()

# sorted by subject, then by most recent received date
folder$list_emails(by="subject")

# sorted by from name in descending order, then by most recent received date
folder$list_emails(by="from desc")

# searching the list
folder$list_emails(search="important information")

# retrieve a specific email:
# note the Outlook ID is NOT the same as the Internet message-id
email_id <- folder$list_emails()[[1]]$properties$id
folder$get_email(email_id)

##
## creating/sending emails
##

# a simple text email with just a body:
# you can add other properties by calling the returned object's methods
folder$create_email("Hello from R")

# HTML-formatted email with all necessary fields, sent immediately
folder$create_email("<emph>Emphatic hello</emph> from R",
    content_type="html",
    to="user@example.com",
    subject="example email",
    send_now=TRUE)

# using blastula to create a HTML email with Markdown
bl_msg <- blastula::compose_email(md(
"
## Hello!
```

```
This is an email message that was generated by the blastula package.

We can use **Markdown** formatting with the `md()` function.

Cheers,

The blastula team
"),
    footer=md("Sent via Microsoft365R"))
folder$create_email(bl_msg, subject="example blastula email")


# using emayili to create an email with attachments
ey_email <- emayili::envelope(
    text="Hello from emayili",
    to="user@example.com",
    subject="example emayili email") %>%
    emayili::attachment("mydocument.docx") %>%
    emayili::attachment("mydata.xlsx")
folder$create_email(ey_email)


## End(Not run)
```

`ms_plan` *Microsoft Planner Plan*

Description

Class representing one plan withing a Microsoft Planner.

Format

An R6 object of class `ms_plan`, inheriting from `ms_object`.

Details

The plans belong to a group.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for this drive.
- `type`: always "plan" for plan object.
- `properties`: The plan properties.

Methods

- `new(...)`: Initialize a new plan object. Do not call this directly; see 'Initialization' below.
- `update(...)`: Update the plan metadata in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the plan
- `sync_fields()`: Synchronise the R object with the plan metadata in Microsoft Graph.
- `list_tasks(filter=NULL, n=Inf)`: List the tasks under the specified plan.
- `get_task(task_title, task_id)`: Get a task, either by title or ID.
- `list_buckets(filter=NULL, n=Inf)`: List the buckets under the specified plan.
- `get_bucket(bucket_name, bucket_id)`: Get a bucket, either by name or ID.
- `get_details()`: Get the plan details.

Initialization

Creating new objects of this class should be done via the `list_plans` methods of the `AzureGraph::az_group` class. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual plan.

Planner operations

This class exposes methods for carrying out common operations on a plan. Currently only read operations are supported.

Call `list_tasks()` to list the tasks under the plan, and `get_task()` to retrieve a specific task. Similarly, call `list_buckets()` to list the buckets, and `get_bucket()` to retrieve a specific bucket.

Call `get_details()` to get a list containing the details for the plan.

List methods

All `list_*` methods have `filter` and `n` arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are `filter=NULL` and `n=Inf`. If `n=NULL`, the `ms_graph_pager` iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

`ms_plan_task`, `ms_plan_bucket`

Microsoft Graph overview, Plans overview

`ms_plan_bucket` *Microsoft Planner Plan Bucket*

Description

Class representing a bucket within a plan of a Microsoft Planner.

Format

An R6 object of class `ms_plan_bucket`, inheriting from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for this bucket
- `type`: always "plan_bucket" for plan bucket object.
- `properties`: The plan bucket properties.

Methods

- `new(...)`: Initialize a new plan bucket object. Do not call this directly; see 'Initialization' below.
- `update(...)`: Update the plan bucket metadata in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the plan bucket
- `sync_fields()`: Synchronise the R object with the plan bucket metadata in Microsoft Graph.
- `list_tasks(filter=NULL, n=Inf)`: List the tasks for this bucket.

Initialization

Creating new objects of this class should be done via the `list_buckets` method of the `ms_plan` class. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual plan bucket.

List methods

All `list_*` methods have `filter` and `n` arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are `filter=NULL` and `n=Inf`. If `n=NULL`, the `ms_graph_pager` iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

ms_plan, ms_plan_task

Microsoft Graph overview, Plans overview

---

ms_plan_task *Microsoft Planner Plan Task*

Description

Class representing a task within a plan of a Microsoft Planner.

Format

An R6 object of class ms_plan_task, inheriting from ms_object.

Fields

- token: The token used to authenticate with the Graph host.
- tenant: The Azure Active Directory tenant for this task.
- type: always "plan_task" for plan task object.
- properties: The task properties.

Methods

- new(...): Initialize a new plan task object. Do not call this directly; see 'Initialization' below.
- update(...): Update the plan task metadata in Microsoft Graph.
- do_operation(...): Carry out an arbitrary operation on the plan task
- sync_fields(): Synchronise the R object with the plan task metadata in Microsoft Graph.

Initialization

Creating new objects of this class should be done via the list_tasks methods of the ms_plan class. Calling the new() method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual plan task.

List methods

All list_* methods have filter and n arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are filter=NULL and n=Inf. If n=NULL, the ms_graph_pager iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

ms_plan, ms_plan_bucket

Microsoft Graph overview, Plans overview

---

ms_site *Office 365 SharePoint site*

Description

Class representing a SharePoint site.

Format

An R6 object of class ms_site, inheriting from ms_object.

Fields

- token: The token used to authenticate with the Graph host.
- tenant: The Azure Active Directory tenant for this site.
- type: always "site" for a site object.
- properties: The site properties.

Methods

- new(...): Initialize a new site object. Do not call this directly; see 'Initialization' below.
- delete(confirm=TRUE): Delete a site. By default, ask for confirmation first.
- update(...): Update the site metadata in Microsoft Graph.
- do_operation(...): Carry out an arbitrary operation on the site.
- sync_fields(): Synchronise the R object with the site metadata in Microsoft Graph.
- list_drives(filter=NULL, n=Inf): List the drives (shared document libraries) associated with this site.
- get_drive(drive_name, drive_id): Retrieve a shared document library for this site. If the name and ID are not specified, this returns the default document library.
- list_subsites(filter=NULL, n=Inf): List the subsites of this site.
- get_lists(filter=NULL, n=Inf): Returns the lists that are part of this site.
- get_list(list_name, list_id): Returns a specific list, either by name or ID.
- get_group(): Retrieve the Microsoft 365 group associated with the site, if it exists. A site that backs a private Teams channel will not have a group associated with it.

Initialization

Creating new objects of this class should be done via the get_sharepoint_site method of the AzureGraph::ms_graph or AzureGraph::az_group classes. Calling the new() method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual site.

List methods

All `list_*` methods have `filter` and `n` arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are `filter=NULL` and `n=Inf`. If `n=NULL`, the `ms_graph_pager` iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

`AzureGraph::ms_graph`, `ms_drive`, `AzureGraph::az_user`

Microsoft Graph overview, SharePoint sites API reference

Examples

```
## Not run:

site <- get_sharepoint_site("My site")
site$list_drives()
site$get_drive()


## End(Not run)
```

`ms_team` *Microsoft Teams team*

Description

Class representing a team in Microsoft Teams.

Format

An R6 object of class `ms_team`, inheriting from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for this team.
- `type`: Always "team" for a team object.
- `properties`: The team properties.

Methods

- `new(...)`: Initialize a new team object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete a team. By default, ask for confirmation first.
- `update(...)`: Update the team metadata in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the team.
- `sync_fields()`: Synchronise the R object with the team metadata in Microsoft Graph.
- `list_channels(filter=NULL, n=Inf)`: List the channels for this team.
- `get_channel(channel_name, channel_id)`: Retrieve a channel. If the name and ID are not specified, returns the primary channel.
- `create_channel(channel_name, description, membership)`: Create a new channel. Optionally, you can specify a short text description of the channel, and the type of membership: either standard, shared or private (invitation-only). Note that creating a shared channel is an *asynchronous* operation; the call returns before the creation is finished. You can retrieve the channel with the `get_channel` method after waiting for a short while.
- `delete_channel(channel_name, channel_id, confirm=TRUE)`: Delete a channel; by default, ask for confirmation first. You cannot delete the primary channel of a team. Note that Teams keeps track of all channels ever created, even if you delete them (you can see the deleted channels by going to the "Manage team" pane for a team, then the "Channels" tab, and expanding the "Deleted" entry); therefore, try not to create and delete channels unnecessarily.
- `list_drives(filter=NULL, n=Inf)`: List the drives (shared document libraries) associated with this team.
- `get_drive(drive_name, drive_id)`: Retrieve a shared document library for this team. If the name and ID are not specified, this returns the default document library.
- `get_sharepoint_site()`: Get the SharePoint site associated with the team.
- `get_group()`: Retrieve the Microsoft 365 group associated with the team.
- `list_members(filter=NULL, n=Inf)`: Retrieves the members of the team, as a list of `ms_team_member` objects.
- `get_member(name, email, id)`: Retrieve a specific member of the channel, as a `ms_team_member` object. Supply only one of the member name, email address or ID.

Initialization

Creating new objects of this class should be done via the `get_team` and `list_teams` methods of the `AzureGraph::ms_graph`, `AzureGraph::az_user` or `AzureGraph::az_group` classes. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual team.

List methods

All `list_*` methods have `filter` and `n` arguments to limit the number of results. The former should be an OData expression as a string to filter the result set on. The latter should be a number setting the maximum number of (filtered) results to return. The default values are `filter=NULL` and `n=Inf`. If `n=NULL`, the `ms_graph_pager` iterator object is returned instead to allow manual iteration over the results.

Support in the underlying Graph API for OData queries is patchy. Not all endpoints that return lists of objects support filtering, and if they do, they may not allow all of the defined operators. If your filtering expression results in an error, you can carry out the operation without filtering and then filter the results on the client side.

See Also

AzureGraph::ms_graph, AzureGraph::az_group, ms_channel, ms_site, ms_drive

Microsoft Graph overview, Microsoft Teams API reference

Examples

```
## Not run:

myteam <- get_team("my team")
myteam$list_channels()
myteam$get_channel()
myteam$get_drive()

myteam$create_channel("Test channel", description="A channel for testing")
myteam$delete_channel("Test channel")

# team members
myteam$list_members()
myteam$get_member("Jane Smith")
myteam$get_member(email="billg@mycompany.com")


## End(Not run)
```

---

ms_team_member *Teams/channel member*

Description

Class representing a member of a team or channel (which will normally be a user in Azure Active Directory).

Format

An R6 object of class `ms_team_member`, inheriting from `ms_object`.

Fields

- `token`: The token used to authenticate with the Graph host.
- `tenant`: The Azure Active Directory tenant for the parent object.
- `type`: One of "team member", "channel member" or "chat member" depending on the parent object.
- `properties`: The item properties (metadata).

Methods

- `new(...)`: Initialize a new object. Do not call this directly; see 'Initialization' below.
- `delete(confirm=TRUE)`: Delete this member.
- `update(...)`: Update the member's properties (metadata) in Microsoft Graph.
- `do_operation(...)`: Carry out an arbitrary operation on the member.
- `sync_fields()`: Synchronise the R object with the member metadata in Microsoft Graph.
- `get_aaduser()`: Get the AAD information for the member; returns an object of class `AzureGraph::az_user`.

Initialization

Creating new objects of this class should be done via the `get_member` and `list_members` methods of the `ms_team`and `ms_channel` classes. Calling the `new()` method for this class only constructs the R object; it does not call the Microsoft Graph API to retrieve or create the actual member.

See Also

`ms_team`, `ms_channel`

Microsoft Graph overview, Microsoft Teams API reference

`personal_onedrive` *Deprecated client functions*

Description

Deprecated client functions

Usage

```
personal_onedrive(
  app = .microsoft365r_app_id,
  scopes = c("Files.ReadWrite.All", "User.Read"),
  ...
)

business_onedrive(
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
  app = Sys.getenv("CLIMICROSOFT365_AADAPPID"),
  scopes = ".default",
  ...
)

sharepoint_site(
  site_url = NULL,
  site_id = NULL,
  tenant = Sys.getenv("CLIMICROSOFT365_TENANT", "common"),
```

```
  app = Sys.getenv("CLIMICROSOFT365_AADAPPID"),
  scopes = ".default",
  ...
)
```

Arguments

app	A custom app registration ID to use for authentication. For personal_onedrive, the default is to use Microsoft365R's internal app ID. For business_onedrive and sharepoint_site, see below.
scopes	The Microsoft Graph scopes (permissions) to obtain.
...	Optional arguments to be passed to AzureGraph::create_graph_login.
tenant	For business_onedrive and sharepoint_site, the name of your Azure Active Directory (AAD) tenant. If not supplied, use the value of the CLIMICROSOFT365_TENANT environment variable, or "common" if that is unset.
site_url, site_id	For sharepoint_site, the web URL and ID of the SharePoint site to retrieve. Supply one or the other, but not both.

Details

These functions have been replaced by get_personal_onedrive, get_business_onedrive and get_sharepoint_site. They will be removed in a later version of the package.

Index