matrix_client package

matrix_client.client

class **matrix_client.client.CACHE** [source]

Bases: `int`

ALL= *1*

NONE= *-1*

SOME= *0*

class **matrix_client.client.MatrixClient**(*base_url, token=None, user_id=None, valid_cert_check=True, sync_filter_limit=20, cache_level=1*) [source]

Bases: `object`

The client API for Matrix. For the raw HTTP calls, see MatrixHttpApi.

Parameters:
- **base_url** (*str*) – The url of the HS preceding /_matrix. e.g. (ex: https://localhost:8008)
- **token** (*Optional[str]*) – If you have an access token supply it here.
- **user_id** (*Optional[str]*) – You must supply the user_id (as obtained when initially logging in to obtain the token) if supplying a token; otherwise, ignored.
- **valid_cert_check** (*bool*) – Check the homeservers certificate on connections?

Returns: *MatrixClient*

Raises: *MatrixRequestError*, *ValueError*

Examples

Create a new user and send a message:

```
client = MatrixClient("https://matrix.org")
token = client.register_with_password(username="foobar",
    password="monkey")
room = client.create_room("myroom")
room.send_image(file_like_object)
```

Send a message with an already logged in user:

```
client = MatrixClient("https://matrix.org", token="foobar",
    user_id="@foobar:matrix.org")
client.add_listener(func)  # NB: event stream callback
client.rooms[0].add_listener(func)  # NB: callbacks just for this room.
room = client.join_room("#matrix:matrix.org")
response = room.send_text("Hello!")
response = room.kick("@bob:matrix.org")
```

Incoming event callbacks (scopes):

```
def user_callback(user, incoming_event):
    pass

def room_callback(room, incoming_event):
    pass
```

```
def global_callback(incoming_event):
    pass
```

add_ephemeral_listener(*callback, event_type=None*) [source]

Add an ephemeral listener that will send a callback when the client recieves an ephemeral event.

Parameters:
- **callback** (*func(roomchunk*) – Callback called when an ephemeral event arrives.
- **event_type** (*str*) – The event_type to filter for.

Returns: Unique id of the listener, can be used to identify the listener.

Return type: uuid.UUID

add_invite_listener(*callback*) [source]

Add a listener that will send a callback when the client receives an invite.

Parameters: **callback** (*func(room_id, state*) – Callback called when an invite arrives.

add_leave_listener(*callback*) [source]

Add a listener that will send a callback when the client has left a room.

Parameters:
- **callback** (*func(room_id, room*) – Callback called when the client
- **left a room.** (*has*) –

add_listener(*callback, event_type=None*) [source]

Add a listener that will send a callback when the client recieves an event.

Parameters:
- **callback** (*func(roomchunk*) – Callback called when an event arrives.
- **event_type** (*str*) – The event_type to filter for.

Returns: Unique id of the listener, can be used to identify the listener.

Return type: uuid.UUID

add_presence_listener(*callback*) [source]

Add a presence listener that will send a callback when the client receives a presence update.

Parameters: **callback** (*func(roomchunk*) – Callback called when a presence update arrives.

Returns: Unique id of the listener, can be used to identify the listener.

Return type: uuid.UUID

create_room(*alias=None, is_public=False, invitees=None*) [source]

Create a new room on the homeserver.

Parameters:
- **alias** (*str*) – The canonical_alias of the room.
- **is_public** (*bool*) – The public/private visibility of the room.
- **invitees** (*str[]*) – A set of user ids to invite into the room.

Returns: Room

Raises: `MatrixRequestError`

`get_rooms()` [source]

Return a dict of {room_id: Room objects} that the user has joined.

Returns: Rooms the user has joined.

Return type: Room{}

`get_sync_token()` [source]

`get_user(user_id)` [source]

Return a User by their id.

> **NOTE: This function only returns a user object, it does not verify**
>
> the user with the Home Server.

Parameters: **user_id** (*str*) – The matrix user id of a user.

`join_room(room_id_or_alias)` [source]

Join a room.

Parameters: **room_id_or_alias** (*str*) – Room ID or an alias.

Returns: Room

Raises: `MatrixRequestError`

`listen_for_events(timeout_ms=30000)` [source]

This function just calls _sync()

In a future version of this sdk, this function will be deprecated and _sync method will be renamed sync with the intention of it being called by downstream code.

Parameters: **timeout_ms** (*int*) – How long to poll the Home Server for before retrying.

`listen_forever(timeout_ms=30000, exception_handler=None, bad_sync_timeout=5)` [source]

Keep listening for events forever.

Parameters:
- **timeout_ms** (*int*) – How long to poll the Home Server for before retrying.
- **exception_handler** (*func(exception)*) – Optional exception handler function which can be used to handle exceptions in the caller thread.
- **bad_sync_timeout** (*int*) – Base time to wait after an error before retrying. Will be increased according to exponential backoff.

`login(username, password, limit=10, sync=True, device_id=None)` [source]

Login to the homeserver.

Parameters:
- **username** (*str*) – Account username
- **password** (*str*) – Account password
- **limit** (*int*) – Deprecated. How many messages to return when syncing. This will be replaced by a filter API in a later release.
- **sync** (*bool*) – Optional. Whether to initiate a /sync request after logging in.
- **device_id** (*str*) – Optional. ID of the client device. The server will auto-generate a device_id if this is not specified.

Returns: Access token

Return type: str

Raises: `MatrixRequestError`

`login_with_password`(*username*, *password*, *limit=10*) [source]

Deprecated. Use `login` with `sync=True`.

Login to the homeserver.

Parameters:
- **username** (*str*) – Account username
- **password** (*str*) – Account password
- **limit** (*int*) – Deprecated. How many messages to return when syncing. This will be replaced by a filter API in a later release.

Returns: Access token

Return type: str

Raises: `MatrixRequestError`

`login_with_password_no_sync`(*username*, *password*) [source]

Deprecated. Use `login` with `sync=False`.

Login to the homeserver.

Parameters:
- **username** (*str*) – Account username
- **password** (*str*) – Account password

Returns: Access token

Return type: str

Raises: `MatrixRequestError`

`logout`() [source]

Logout from the homeserver.

`register_as_guest`() [source]

Register a guest account on this HS. Note: HS must have guest registration enabled. :returns: Access Token :rtype: str

Raises: `MatrixRequestError`

`register_with_password`(*username*, *password*) [source]

Register for a new account on this HS.

Parameters:
- **username** (*str*) – Account username
- **password** (*str*) – Account password

Returns: Access Token

Return type: str

Raises: `MatrixRequestError`

`remove_ephemeral_listener`(*uid*) [source]

Remove ephemeral listener with given uid.

Parameters: **uuid.UUID** – Unique id of the listener to remove.

`remove_listener`(*uid*) [source]

Remove listener with given uid.

Parameters: **uuid.UUID** – Unique id of the listener to remove.

`remove_presence_listener`(*uid*) [source]

Remove presence listener with given uid

Parameters: **uuid.UUID** – Unique id of the listener to remove

`remove_room_alias`(*room_alias*) [source]

Remove mapping of an alias

Parameters: **room_alias** (*str*) – The alias to be removed.

Returns: True if the alias is removed, False otherwise.

Return type: bool

`set_sync_token`(*token*) [source]

`set_user_id`(*user_id*) [source]

`should_listen`= *None*

Time to wait before attempting a /sync request after failing.

`start_listener_thread`(*timeout_ms=30000, exception_handler=None*) [source]

Start a listener thread to listen for events in the background.

Parameters:
- **timeout** (*int*) – How long to poll the Home Server for before retrying.
- **exception_handler** (*func(exception*) – Optional exception handler function which can be used to handle exceptions in the caller thread.

`stop_listener_thread`() [source]

Stop listener thread running in the background

`upload`(*content, content_type*) [source]

Upload content to the home server and recieve a MXC url.

Parameters:
- **content** (*bytes*) – The data of the content.
- **content_type** (*str*) – The mimetype of the content.

Raises:
- `MatrixUnexpectedResponse` – If the homeserver gave a strange response
- `MatrixRequestError` – If the upload failed for some reason.

matrix_client.api

class `matrix_client.api.MatrixHttpApi`(*base_url, token=None, identity=None, default_429_wait_ms=5000*) [source]

Bases: `object`

Contains all raw Matrix HTTP Client-Server API calls.

For room and sync handling, consider using MatrixClient.

Parameters:
- **base_url** (*str*) – The home server URL e.g. 'http://localhost:8008'
- **token** (*str*) – Optional. The client's access token.
- **identity** (*str*) – Optional. The mxid to act as (For application services only).
- **default_429_wait_ms** (*int*) – Optional. Time in millseconds to wait before retrying a request when server returns a HTTP 429 response without a 'retry_after_ms' key.

Examples

Create a client and send a message:

```
matrix = MatrixHttpApi("https://matrix.org", token="foobar")
response = matrix.sync()
response = matrix.send_message("!roomid:matrix.org", "Hello!")
```

`add_user_tag`(*user_id, room_id, tag, order=None, body=None*) [source]

`ban_user`(*room_id, user_id, reason=''*) [source]

Perform POST /rooms/$room_id/ban

Parameters:
- **room_id** (*str*) – The room ID
- **user_id** (*str*) – The user ID of the banee(sic)
- **reason** (*str*) – The reason for this ban

`claim_keys`(*key_request, timeout=None*) [source]

Claims one-time keys for use in pre-key messages.

Parameters:
- **key_request** (*dict*) – The keys to be claimed. Format should be <user_id>: { <device_id>: <algorithm> }.
- **timeout** (*int*) – Optional. The time (in milliseconds) to wait when downloading keys from remote servers.

`create_filter`(*user_id, filter_params*) [source]

`create_room`(*alias=None, is_public=False, invitees=None*) [source]

Perform /createRoom.

Parameters:
- **alias** (*str*) – Optional. The room alias name to set for this room.
- **is_public** (*bool*) – Optional. The public/private visibility.
- **invitees** (*list<str>*) – Optional. The list of user IDs to invite.

`delete_device`(*auth_body*, *device_id*) [source]

Deletes the given device, and invalidates any access token associated with it.

NOTE: This endpoint uses the User-Interactive Authentication API.

Parameters:
- **auth_body** (*dict*) – Authentication params.
- **device_id** (*str*) – The device ID of the device to delete.

`delete_devices`(*auth_body*, *devices*) [source]

Bulk deletion of devices.

NOTE: This endpoint uses the User-Interactive Authentication API.

Parameters:
- **auth_body** (*dict*) – Authentication params.
- **devices** (*list*) – List of device ID"s to delete.

`event_stream`(*from_token*, *timeout=30000*) [source]

Deprecated. Use sync instead. Performs /events

Parameters:
- **from_token** (*str*) – The 'from' query parameter.
- **timeout** (*int*) – Optional. The 'timeout' query parameter.

`forget_room`(*room_id*) [source]

Perform POST /rooms/$room_id/forget

Parameters: **room_id** (*str*) – The room ID

`get_avatar_url`(*user_id*) [source]

`get_device`(*device_id*) [source]

Gets information on a single device, by device id.

`get_devices`() [source]

Gets information about all devices for the current user.

`get_display_name`(*user_id*) [source]

`get_download_url`(*mxcurl*) [source]

`get_emote_body`(*text*) [source]

`get_filter`(*user_id*, *filter_id*) [source]

`get_membership`(*room_id*, *user_id*) [source]

Perform GET /rooms/$room_id/state/m.room.member/$user_id

Parameters:
- **room_id** (*str*) – The room ID

- **user_id** (*str*) – The user ID

`get_power_levels`(*room_id*) [source]

Perform GET /rooms/$room_id/state/m.room.power_levels

Parameters: **room_id** (*str*) – The room ID

`get_room_id`(*room_alias*) [source]

Get room id from its alias

Parameters: **room_alias** (*str*) – The room alias name.

Returns: Wanted room's id.

`get_room_members`(*room_id*) [source]

Get the list of members for this room.

Parameters: **room_id** (*str*) – The room to get the member events for.

`get_room_messages`(*room_id, token, direction, limit=10, to=None*) [source]

Perform GET /rooms/{roomId}/messages.

Parameters:
- **room_id** (*str*) – The room's id.
- **token** (*str*) – The token to start returning events from.
- **direction** (*str*) – The direction to return events from. One of: ["b", "f"].
- **limit** (*int*) – The maximum number of events to return.
- **to** (*str*) – The token to stop returning events at.

`get_room_name`(*room_id*) [source]

Perform GET /rooms/$room_id/state/m.room.name :param room_id: The room ID :type room_id: str

`get_room_state`(*room_id*) [source]

Perform GET /rooms/$room_id/state

Parameters: **room_id** (*str*) – The room ID

`get_room_topic`(*room_id*) [source]

Perform GET /rooms/$room_id/state/m.room.topic :param room_id: The room ID :type room_id: str

`get_text_body`(*text, msgtype='m.text'*) [source]

`get_user_tags`(*user_id, room_id*) [source]

`initial_sync`(*limit=1*) [source]

Warning

Deprecated. Use sync instead.

Perform /initialSync.

Parameters: **limit** (*int*) – The limit= param to provide.

`invite_user`(*room_id*, *user_id*) [source]

Perform POST /rooms/$room_id/invite

Parameters:
- **room_id** (*str*) – The room ID
- **user_id** (*str*) – The user ID of the invitee

`join_room`(*room_id_or_alias*) [source]

Performs /join/$room_id

Parameters: **room_id_or_alias** (*str*) – The room ID or room alias to join.

`key_changes`(*from_token*, *to_token*) [source]

Gets a list of users who have updated their device identity keys.

Parameters:
- **from_token** (*str*) – The desired start point of the list. Should be the next_batch field from a response to an earlier call to /sync.
- **to_token** (*str*) – The desired end point of the list. Should be the next_batch field from a recent call to /sync - typically the most recent such call.

`kick_user`(*room_id*, *user_id*, *reason=''*) [source]

Calls set_membership with membership="leave" for the user_id provided

`leave_room`(*room_id*) [source]

Perform POST /rooms/$room_id/leave

Parameters: **room_id** (*str*) – The room ID

`login`(*login_type*, ***kwargs*) [source]

Perform /login.

Parameters:
- **login_type** (*str*) – The value for the 'type' key.
- ****kwargs** – Additional key/values to add to the JSON submitted.

`logout`() [source]

Perform /logout.

`media_upload`(*content*, *content_type*) [source]

`query_keys`(*user_devices*, *timeout=None*, *token=None*) [source]

Query HS for public keys by user and optionally device.

Parameters:
- **user_devices** (*dict*) – The devices whose keys to download. Should be formatted as <user_id>: [<device_ids>]. No device_ids indicates all devices for the corresponding user.
- **timeout** (*int*) – Optional. The time (in milliseconds) to wait when downloading keys from remote servers.

- **token** (*str*) – Optional. If the client is fetching keys as a result of a device update received in a sync request, this should be the 'since' token of that sync request, or any later sync token.

`redact_event`(*room_id, event_id, reason=None, txn_id=None, timestamp=None*) [source]

Perform PUT /rooms/$room_id/redact/$event_id/$txn_id/

Parameters:
- **room_id** (*str*) – The room ID to redact the message event in.
- **event_id** (*str*) – The event id to redact.
- **reason** (*str*) – Optional. The reason the message was redacted.
- **txn_id** (*int*) – Optional. The transaction ID to use.
- **timestamp** (*int*) – Optional. Set origin_server_ts (For application services only)

`register`(*content=None, kind='user'*) [source]

Performs /register.

Parameters:
- **content** (*dict*) –
  The request payload.

  Should be specified for all non-guest registrations.
  username (string): The local part of the desired Matrix ID.
  If omitted, the homeserver MUST generate a Matrix ID local part.
  bind_email (boolean): If true, the server binds the email used for
  authentication to the Matrix ID with the ID Server.
  Email Registration not currently supported
  password (string): Required. The desired password for the account.
  auth (dict): Authentication Data
  session (string): The value of the session key given by the
  homeserver.
  type (string): Required. The login type that the client is
  attempting to complete. "m.login.dummy" is the only
  non-interactive type.
- **kind** (*str*) – Specify kind="guest" to register as guest.

`remove_room_alias`(*room_alias*) [source]

Remove mapping of an alias

Parameters: **room_alias** (*str*) – The alias to be removed.

Raises: `MatrixRequestError`

`remove_user_tag`(*user_id, room_id, tag*) [source]

`send_content`(*room_id, item_url, item_name, msg_type, extra_information=None, timestamp=None*) [source]

`send_emote`(*room_id, text_content, timestamp=None*) [source]

Perform PUT /rooms/$room_id/send/m.room.message with m.emote msgtype

Parameters:
- **room_id** (*str*) – The room ID to send the event in.
- **text_content** (*str*) – The m.emote body to send.
- **timestamp** (*int*) – Set origin_server_ts (For application services only)

`send_location`(*room_id, geo_uri, name, thumb_url=None, thumb_info=None, timestamp=None*) [source]

Send m.location message event

Parameters:
- **room_id** (*str*) – The room ID to send the event in.
- **geo_uri** (*str*) – The geo uri representing the location.
- **name** (*str*) – Description for the location.
- **thumb_url** (*str*) – URL to the thumbnail of the location.
- **thumb_info** (*dict*) – Metadata about the thumbnail, type ImageInfo.
- **timestamp** (*int*) – Set origin_server_ts (For application services only)

`send_message`(*room_id, text_content, msgtype='m.text', timestamp=None*) [source]

Perform PUT /rooms/$room_id/send/m.room.message

Parameters:
- **room_id** (*str*) – The room ID to send the event in.
- **text_content** (*str*) – The m.text body to send.
- **timestamp** (*int*) – Set origin_server_ts (For application services only)

`send_message_event`(*room_id, event_type, content, txn_id=None, timestamp=None*) [source]

Perform PUT /rooms/$room_id/send/$event_type

Parameters:
- **room_id** (*str*) – The room ID to send the message event in.
- **event_type** (*str*) – The event type to send.
- **content** (*dict*) – The JSON content to send.
- **txn_id** (*int*) – Optional. The transaction ID to use.
- **timestamp** (*int*) – Set origin_server_ts (For application services only)

`send_notice`(*room_id, text_content, timestamp=None*) [source]

Perform PUT /rooms/$room_id/send/m.room.message with m.notice msgtype

Parameters:
- **room_id** (*str*) – The room ID to send the event in.
- **text_content** (*str*) – The m.notice body to send.
- **timestamp** (*int*) – Set origin_server_ts (For application services only)

`send_state_event`(*room_id, event_type, content, state_key='', timestamp=None*) [source]

Perform PUT /rooms/$room_id/state/$event_type

Parameters:
- **room_id** (*str*) – The room ID to send the state event in.
- **event_type** (*str*) – The state event type to send.
- **content** (*dict*) – The JSON content to send.
- **state_key** (*str*) – Optional. The state key for the event.
- **timestamp** (*int*) – Set origin_server_ts (For application services only)

`send_to_device`(*event_type, messages, txn_id=None*) [source]

Sends send-to-device events to a set of client devices.

Parameters:
- **event_type** (*str*) – The type of event to send.
- **messages** (*dict*) – The messages to send. Format should be <user_id>: {<device_id>: <event_content>}. The device ID may also be '*', meaning all known devices for the user.
- **txn_id** (*str*) – Optional. The transaction ID for this event, will be generated automatically otherwise.

`set_account_data`(*user_id, type, account_data*) [source]

`set_avatar_url`(*user_id, avatar_url*) [source]

`set_display_name`(*user_id, display_name*) [source]

`set_guest_access`(*room_id, guest_access*) [source]

Set the guest access policy of the room.

Parameters:
- **room_id** (*str*) – The room to set the rules for.
- **guest_access** (*str*) – Wether guests can join. One of: ["can_join", "forbidden"]

`set_join_rule`(*room_id, join_rule*) [source]

Set the rule for users wishing to join the room.

Parameters:
- **room_id** (*str*) – The room to set the rules for.
- **join_rule** (*str*) – The chosen rule. One of: ["public", "knock", "invite", "private"]

`set_membership`(*room_id, user_id, membership, reason='', profile=None, timestamp=None*) [source]

Perform PUT /rooms/$room_id/state/m.room.member/$user_id

Parameters:
- **room_id** (*str*) – The room ID
- **user_id** (*str*) – The user ID
- **membership** (*str*) – New membership value
- **reason** (*str*) – The reason
- **timestamp** (*int*) – Set origin_server_ts (For application services only)

`set_power_levels`(*room_id, content*) [source]

Perform PUT /rooms/$room_id/state/m.room.power_levels

Note that any power levels which are not explicitly specified in the content arg are reset to default values.

Parameters:
- **room_id** (*str*) – The room ID
- **content** (*dict*) – The JSON content to send. See example content below.

Example:

```
api = MatrixHttpApi("http://example.com", token="foobar")
api.set_power_levels("!exampleroom:example.com",
    {
```

```
        "ban": 50, # defaults to 50 if unspecified
        "events": {
            "m.room.name": 100, # must have PL 100 to change room name
            "m.room.power_levels": 100 # must have PL 100 to change PLs
        },
        "events_default": 0, # defaults to 0
        "invite": 50, # defaults to 50
        "kick": 50, # defaults to 50
        "redact": 50, # defaults to 50
        "state_default": 50, # defaults to 50 if m.room.power_levels exists
        "users": {
            "@someguy:example.com": 100 # defaults to 0
        },
        "users_default": 0 # defaults to 0
    }
)
```

set_room_account_data(*user_id, room_id, type, account_data*) [source]

set_room_alias(*room_id, room_alias*) [source]

Set alias to room id

Parameters:
- **room_id** (*str*) – The room id.
- **room_alias** (*str*) – The room wanted alias name.

set_room_name(*room_id, name, timestamp=None*) [source]

Perform PUT /rooms/$room_id/state/m.room.name :param room_id: The room ID :type room_id: str :param name: The new room name :type name: str :param timestamp: Set origin_server_ts (For application services only) :type timestamp: int

set_room_topic(*room_id, topic, timestamp=None*) [source]

Perform PUT /rooms/$room_id/state/m.room.topic :param room_id: The room ID :type room_id: str :param topic: The new room topic :type topic: str :param timestamp: Set origin_server_ts (For application services only) :type timestamp: int

sync(*since=None, timeout_ms=30000, filter=None, full_state=None, set_presence=None*) [source]

Perform a sync request.

Parameters:
- **since** (*str*) – Optional. A token which specifies where to continue a sync from.
- **timeout_ms** (*int*) – Optional. The time in milliseconds to wait.
- **filter** (*int*|*str*) – Either a Filter ID or a JSON string.
- **full_state** (*bool*) – Return the full state for every room the user has joined Defaults to false.
- **set_presence** (*str*) – Should the client be marked as “online” or” offline”

unban_user(*room_id, user_id*) [source]

Perform POST /rooms/$room_id/unban

Parameters:
- **room_id** (*str*) – The room ID
- **user_id** (*str*) – The user ID of the banee(sic)

update_device_info(*device_id, display_name*) [source]

Update the display name of a device.

Parameters:
- **device_id** (*str*) – The device ID of the device to update.
- **display_name** (*str*) – New display name for the device.

`upload_keys`(*device_keys=None, one_time_keys=None*) [source]

Publishes end-to-end encryption keys for the device.

Said device must be the one used when logging in.

Parameters:
- **device_keys** (*dict*) –
  Optional. Identity keys for the device. The required keys are:

  user_id (str): The ID of the user the device belongs to. Must match the user ID used when logging in.
  device_id (str): The ID of the device these keys belong to. Must match the device ID used when logging in.
  algorithms (list<str>): The encryption algorithms supported by this device.
  keys (dict): Public identity keys. Should be formatted as <algorithm:device_id>: <key>.
  signatures (dict): Signatures for the device key object. Should be formatted as <user_id>: {<algorithm:device_id>: <key>}
- **one_time_keys** (*dict*) – Optional. One-time public keys. Should be formatted as <algorithm:key_id>: <key>, the key format being determined by the algorithm.

`validate_certificate`(*valid*) [source]

matrix_client.user

class `matrix_client.user.User`(*api, user_id, displayname=None*) [source]

Bases: `object`

The User class can be used to call user specific functions.

`get_avatar_url`() [source]

`get_display_name`() [source]

Get this users display name.

See also get_friendly_name()

Returns: Display Name

Return type: str

`get_friendly_name`() [source]

`set_avatar_url`(*avatar_url*) [source]

Set this users avatar.

Parameters: **avatar_url** (*str*) – mxc url from previously uploaded

`set_display_name`(*display_name*) [source]

Set this users display name.

Parameters: **display_name** (*str*) – Display Name

matrix_client.room

class `matrix_client.room.Room`(*client*, *room_id*) [source]

Bases: `object`

Call room-specific functions after joining a room from the client.

`add_ephemeral_listener`(*callback*, *event_type=None*) [source]

Add a callback handler for ephemeral events going to this room.

Parameters:
- **callback** (*func(room, event)*) – Callback called when an ephemeral event arrives.
- **event_type** (*str*) – The event_type to filter for.

Returns: Unique id of the listener, can be used to identify the listener.

Return type: uuid.UUID

`add_listener`(*callback*, *event_type=None*) [source]

Add a callback handler for events going to this room.

Parameters:
- **callback** (*func(room, event)*) – Callback called when an event arrives.
- **event_type** (*str*) – The event_type to filter for.

Returns: Unique id of the listener, can be used to identify the listener.

Return type: uuid.UUID

`add_room_alias`(*room_alias*) [source]

Add an alias to the room and return True if successful.

`add_state_listener`(*callback*, *event_type=None*) [source]

Add a callback handler for state events going to this room.

Parameters:
- **callback** (*func(roomchunk)*) – Callback called when an event arrives.
- **event_type** (*str*) – The event_type to filter for.

`add_tag`(*tag*, *order=None*, *content=None*) [source]

`backfill_previous_messages`(*reverse=False*, *limit=10*) [source]

Backfill handling of previous messages.

Parameters:
- **reverse** (*bool*) – When false messages will be backfilled in their original order (old to new), otherwise the order will be reversed (new to old).
- **limit** (*int*) – Number of messages to go back.

`ban_user`(*user_id*, *reason*) [source]

Ban a user from this room

Parameters:
- **user_id** (*str*) – The matrix user id of a user.
- **reason** (*str*) – A reason for banning the user.

Returns: The user was banned.

Return type: boolean

`display_name`

Calculates the display name for a room.

`get_events()` [source]

Get the most recent events for this room.

`get_html_content(*html*, *body=None*, *msgtype='m.text'*)` [source]

`get_joined_members()` [source]

Returns list of joined members (User objects).

`get_tags()` [source]

`invite_user(*user_id*)` [source]

Invite a user to this room.

Returns: Whether invitation was sent.

Return type: boolean

`kick_user(*user_id*, *reason=''*)` [source]

Kick a user from this room.

Parameters:
- **user_id** (*str*) – The matrix user id of a user.
- **reason** (*str*) – A reason for kicking the user.

Returns: Whether user was kicked.

Return type: boolean

`leave()` [source]

Leave the room.

Returns: Leaving the room was successful.

Return type: boolean

`modify_required_power_levels(*events=None*, *\*\*kwargs*)` [source]

Modifies room power level requirements.

Parameters:
- **events** (*dict*) – Power levels required for sending specific event types, in the form {“m.room.whatever0”: 60, “m.room.whatever2”: None}. Overrides events_default and state_default for the specified events. A level of None causes the target event to revert to the default level as specified by events_default or state_default.
- ****kwargs** –
  Key/value pairs specifying the power levels required for various actions:
  - events_default(int): Default level for sending message events

- state_default(int): Default level for sending state events
- invite(int): Inviting a user
- redact(int): Redacting an event
- ban(int): Banning a user
- kick(int): Kicking a user

Returns: True if successful, False if not

`modify_user_power_levels`(*users=None, users_default=None*) [source]

Modify the power level for a subset of users

Parameters:
- **users** (*dict*) – Power levels to assign to specific users, in the form {"@name0:host0": 10, "@name1:host1": 100, "@name3:host3", None} A level of None causes the user to revert to the default level as specified by users_default.
- **users_default** (*int*) – Default power level for users in the room

Returns: True if successful, False if not

`prev_batch`

`redact_message`(*event_id, reason=None*) [source]

Redacts the message with specified event_id for the given reason.

See https://matrix.org/docs/spec/r0.0.1/client_server.html#id112

`remove_ephemeral_listener`(*uid*) [source]

Remove ephemeral listener with given uid.

`remove_listener`(*uid*) [source]

Remove listener with given uid.

`remove_tag`(*tag*) [source]

`send_audio`(*url, name, **audioinfo*) [source]

Send a pre-uploaded audio to the room.

See http://matrix.org/docs/spec/client_server/r0.2.0.html#m-audio for audioinfo

Parameters:
- **url** (*str*) – The mxc url of the audio.
- **name** (*str*) – The filename of the audio.
- **()** (*audioinfo*) – Extra information about the audio.

`send_emote`(*text*) [source]

Send an emote (/me style) message to the room.

`send_file`(*url, name, **fileinfo*) [source]

Send a pre-uploaded file to the room.

See http://matrix.org/docs/spec/r0.2.0/client_server.html#m-file for fileinfo.

Parameters:
- **url** (*str*) – The mxc url of the file.
- **name** (*str*) – The filename of the image.
- **()** (*fileinfo*) – Extra information about the file

`send_html`(*html, body=None, msgtype='m.text'*) [source]

Send an html formatted message.

Parameters:
- **html** (*str*) – The html formatted message to be sent.
- **body** (*str*) – The unformatted body of the message to be sent.

`send_image`(*url, name, **imageinfo*) [source]

Send a pre-uploaded image to the room.

See http://matrix.org/docs/spec/r0.0.1/client_server.html#m-image for imageinfo

Parameters:
- **url** (*str*) – The mxc url of the image.
- **name** (*str*) – The filename of the image.
- **()** (*imageinfo*) – Extra information about the image.

`send_location`(*geo_uri, name, thumb_url=None, **thumb_info*) [source]

Send a location to the room.

See http://matrix.org/docs/spec/client_server/r0.2.0.html#m-location for thumb_info

Parameters:
- **geo_uri** (*str*) – The geo uri representing the location.
- **name** (*str*) – Description for the location.
- **thumb_url** (*str*) – URL to the thumbnail of the location.
- **()** (*thumb_info*) – Metadata about the thumbnail, type ImageInfo.

`send_notice`(*text*) [source]

Send a notice (from bot) message to the room.

`send_state_event`(*event_type, content, state_key=''*) [source]

Send a state event to the room.

Parameters:
- **event_type** (*str*) – The type of event that you are sending.
- **()** (*content*) – An object with the content of the message.
- **state_key** (*str, optional*) – A unique key to identify the state.

`send_text`(*text*) [source]

Send a plain text message to the room.

`send_video`(*url, name, **videoinfo*) [source]

Send a pre-uploaded video to the room.

See http://matrix.org/docs/spec/client_server/r0.2.0.html#m-video for videoinfo

Parameters:
- **url** (*str*) – The mxc url of the video.
- **name** (*str*) – The filename of the video.
- **()** (*videoinfo*) – Extra information about the video.

`set_account_data`(*type, account_data*) [source]

`set_guest_access`(*allow_guests*) [source]

Set whether guests can join the room and return True if successful.

`set_invite_only`(*invite_only*) [source]

Set how the room can be joined.

Parameters: **invite_only** (*bool*) – If True, users will have to be invited to join the room. If False, anyone who knows the room link can join.

Returns: True if successful, False if not

`set_room_name`(*name*) [source]

Return True if room name successfully changed.

`set_room_topic`(*topic*) [source]

Set room topic.

Returns: True if the topic changed, False if not

Return type: boolean

`set_user_profile`(*displayname=None, avatar_url=None, reason='Changing room profile information'*) [source]

Set user profile within a room.

This sets displayname and avatar_url for the logged in user only in a specific room. It does not change the user's global user profile.

`unban_user`(*user_id*) [source]

Unban a user from this room

Returns: The user was unbanned.

Return type: boolean

`update_aliases`() [source]

Get aliases information from room state.

Returns: True if the aliases changed, False if not

Return type: boolean

`update_room_name`() [source]

Updates self.name and returns True if room name has changed.

`update_room_topic`() [source]

Updates self.topic and returns True if room topic has changed.

matrix_client.checks

`matrix_client.checks.check_room_id(room_id)` [source]

`matrix_client.checks.check_user_id(user_id)` [source]

matrix_client.errors

exception `matrix_client.errors.MatrixError` [source]

Bases: `Exception`

A generic Matrix error. Specific errors will subclass this.

exception `matrix_client.errors.MatrixHttpLibError(original_exception, method, endpoint)` [source]

Bases: `matrix_client.errors.MatrixError`

The library used for http requests raised an exception.

exception `matrix_client.errors.MatrixRequestError(code=0, content='')` [source]

Bases: `matrix_client.errors.MatrixError`

The home server returned an error response.

exception `matrix_client.errors.MatrixUnexpectedResponse(content='')` [source]

Bases: `matrix_client.errors.MatrixError`

The home server gave an unexpected response.