nylas DOCS

Create grants with OAuth and an access token

This page describes how to use the OAuth 2.0 features included in Nylas with a user's access token to create a grant.

> **Nylas creates only one grant per email address in each application**. If a user authenticates with your Nylas application using the email address associated with an existing grant, Nylas re-authenticates the grant instead of creating a new one.

If you're using Hosted Authentication and developing a Single Page App (SPA) or mobile app, Nylas recommends you use the OAuth 2.0 with PKCE authentication method. This extra layer of security adds a key to the authentication exchange that you can safely store on a mobile device *instead of* including the API key. This is optional for projects that have a backend, but it's a good security practice to implement it anyway.

Using `/me/` syntax to refer to a grant

Nylas uses the `/me/` syntax that you use in access token-authorized API calls *instead of* specifying a user's grant ID (for example, `GET /v3/grants/me/messages`).

The `/me/` syntax looks up the grant associated with the request's access token, and uses the `grant_id` associated with the token as a locator. You can use this syntax for API requests that access account data only, and only if you use access tokens to authorize requests. You can't use this syntax if you're using API key authorization, because there is no grant associated with an API key.

For more information, see /me/ syntax for API calls.

Before you begin

Before you begin, make sure you set up all the required parts for your authentication system:

1. If you haven't already, **log in to the Nylas Dashboard** and **create an application**.

nylas DOCS

3. **Create a provider auth app** in the provider's console or application. See the detailed instructions for creating a Google or Azure provider auth app.
4. **Create a connector** for the provider you want to authenticate with.
5. **Add your project's callback URIs ("redirect URIs")** in the Nylas Dashboard.

Create grants with OAuth and an access token

> **Note**: Because Nylas uses the schema outlined in RFC 9068 to ensure that it's compatible with all OAuth libraries in all languages, the format for the `/v3/connect/tokeninfo` endpoint is different from the other OAuth endpoints.

Make authorization request

The first step of the OAuth process is to collect the information you need to include when starting an authorization request. You usually start the request using either a button or link that the user clicks, which redirects them to `api.us.nylas.com/v3/connect/auth` and includes their information. The example below requests two scopes using a Google provider auth app.

```
/v3/connect/auth?
  client_id=<NYLAS_CLIENT_ID>
  &redirect_uri=<REDIRECT_URI>
  &response_type=code
  &provider=google
  &access_type=offline
  &state=sQ6vFQN
```

This request also uses the optional `access_type=offline` parameter to specify that Nylas should return a refresh token. The rendered URL that the user is directed to resembles the example below.

```
https://accounts.google.com/o/oauth2/auth/oauthchooseaccount?
  client_id=<GCP_CLIENT_ID>   // The client ID from your Google auth app.
  &prompt=consent
  &redirect_uri=https://api.us.nylas.com/connect/callback
  &response_type=code
  &scope=https://www.googleapis.com/auth/gmail.readonly%20profile
  &access_type=offline
  &state=BA630DED06...        // Stored and checked/compared internally by Nylas for security.
```

nylas DOCS

Node.js SDK | Python SDK | Ruby SDK | Java SDK | Kotlin SDK

```
import 'dotenv/config'
import express from 'express'
import Nylas from 'nylas'

const config = {
  clientId: process.env.NYLAS_CLIENT_ID,
  callbackUri: "http://localhost:3000/oauth/exchange",
  apiKey: process.env.NYLAS_API_KEY,
  apiUri: process.env.NYLAS_API_URI,
}

const nylas = new Nylas({
  apiKey: config.apiKey,
  apiUri: config.apiUri,
})

const app = express()
const port = 3000

// Route to initialize authentication.
app.get('/nylas/auth', (req, res) => {
  const authUrl = nylas.auth.urlForOAuth2({
    clientId: config.clientId,
    provider: 'google',
    redirectUri: config.redirectUri,
    loginHint: 'email_to_connect',
    accessType: 'offline',
  })

  res.redirect(authUrl)
})
```

Provider consent flow

The user goes to the URL Nylas starts a secure authentication session and redirects the user to the provider website.

Each provider displays the consent and approval steps differently, and it's only visible to the user. In all cases the user authenticates, then either accepts or declines the scopes your project requested.

nylas DOCS

screen twice. This is a normal part of Nylas' Hosted Auth flow, and ensures that all necessary scopes are approved during the auth process.

Authorization response and grant creation

Next, the auth provider sends the user to the Nylas `redirect_uri` (`https://api.us.nylas.com/v3/connect/callback`), and includes information about the outcome of the session as query parameters in the URL.

Nylas uses the information in the response to find your Nylas application by `client_id` and, if the auth was successful, create an unverified grant record for the user and record their details.

Get the user's code

Nylas uses your project's `callback_uri` (for example, `app.example.com/callback-handler`) to send the user back to your project. The callback URI includes query parameters to indicate to your project if authentication was successful.

The example below shows parameters for a successful authentication. The `code` and `state` query parameters are standard OAuth 2.0 fields, but Nylas provides some optional fields to give you more context about the authentication.

```
https://myapp.com/callback-handler?
  state=... // Passed value of initial state if it was provided.
  &code=... // Use this code value for the next step of authentication.
```

Exchange code for access token

Next, exchange the `code` for an access token. Make a `POST /v3/connect/token` request and include the `code` parameter from the success response.

⚠ **OAuth codes are unique, one-time-use credentials**. This means that if your `POST /v3/connect/token` request fails, you must restart the OAuth flow to generate a new `code`. If you try to pass the original `code` in another token exchange request, Nylas returns an error.

nylas DOCS

```
Host: /v3/connect/token
Content-Type: application/json

{
  "code": "<AUTH_EXCHANGE_CODE>",
  "client_id": "<NYLAS_CLIENT_ID>",
  "client_secret": "<NYLAS_API_KEY>",
  "redirect_uri": "<REDIRECT_URI>",
  "grant_type": "authorization_code"
}
```

The auth provider responds with an access token, a refresh token (because you asked for one by setting `access_type=offline` when you made the authorization request), and some other information. When the user completes the authentication flow successfully, Nylas marks their grant as verified and sends you their `grant_id` and email address.

⚠ **OAuth 2.0 access tokens expire after one hour**. When the access token expires, you can either use the *refresh token* you received during authentication to get a new *access token*, or the user can re-authenticate their grant to access your project.

Your application should store the `grant_id`, the `access_token`, and `refresh_token` (for later re-authentication).

```
{
  "access_token": "<ACCESS_TOKEN>",
  "refresh_token": "<REFRESH_TOKEN>",
  "scope": "https://www.googleapis.com/auth/gmail.readonly profile",
  "token_type": "Bearer",
  "id_token": "<ID_TOKEN>",
  "grant_id": "<NYLAS_GRANT_ID>"
}
```

You can also exchange the user's `code` using the Nylas SDKs.

Node.js SDK | Python SDK | Ruby SDK | Java SDK | Kotlin SDK

```
  console.log(res.status)

  const code = req.query.code

  if (!code) {
    res.status(400).send('No authorization code returned from Nylas')
    return
  }

  try {
    const codeExchangeResponse = nylas.auth.exchangeCodeForToken({
      redirectUri: "REDIRECT_URI",
      clientId: "CLIENT_ID",
      clientSecret: "API_KEY",
      code: "CODE"
  });

    const { grantId } = response

    res.status(200)
  } catch (error) {
    console.error('Error exchanging code for token:', error)

    res.status(500).send('Failed to exchange authorization code for token')
  }
})
```

Authorize API calls with access token and /me/ syntax

When you have a user's access token, you can use it to authorize API requests for that user's data, and that user's data *only*. You can't use an access token to authorize API requests that access or modify data at the application level. Those requests require an API key for authorization.

To authorize an API request, pass the token in the request header using HTTP Bearer authentication, then substitute the word `me` in the API calls where you would usually specify a `grant_id`.

When Nylas receives a request using the `/me/` syntax, it checks the authorization header token, finds the `grant_id` associated with that token, and uses that ID to locate data for the user.

The examples below illustrate an API request using an access token and the `/me/` syntax, and the equivalent call using the `grant_id`.

nylas DOCS

```
curl --request GET \
  --url 'https://api.us.nylas.com/v3/grants/me/calendars' \
  --header 'Authorization: Bearer <ACCESS_TOKEN>' \
  --header 'Content-Type: application/json'
```

(Optional) Refresh an expired access token

If you set `access_type=offline` in the initial authorization request, Nylas returns a `refresh_token` along with the `access_token` during the token exchange. When the initial `access_token` expires, you can use the `refresh_token` to request a new one.

> **Refresh tokens don't expire unless revoked**. If your application is client-side-only, you shouldn't request offline access or need this step.

Make a `POST /v3/connect/token` request that specifies `"grant_type": "refresh_token"` and includes the refresh token. The auth provider returns a fresh access token.

Request | Response (JSON) | Node.js SDK | Python SDK | Ruby SDK | Java SDK | Kotlin SDK

```
POST /token HTTP/1.1
Host: /v3/connect/token
Content-Type: application/json

{
  "client_id": "<NYLAS_CLIENT_ID>",
  "client_secret": "<NYLAS_API_KEY>",
  "grant_type": "refresh_token",
  "refresh_token": "<REFRESH_TOKEN>"
}
```

Create grants with OAuth 2.0 and PKCE

The OAuth PKCE (Proof Key for Code Exchange) flow improves security for client-side-only applications, such as browser-based or mobile apps that don't have a back-end server. Even if your application *does* have a back-end server, Nylas recommends you use PKCE for extra security.

nylas DOCS

the `code` exchange flow *without* using your Nylas application's API key. If you're using PKCE, you can set the `platform` parameter to `android`, `desktop`, `ios`, or `js` when you create a callback URI to make the `client_secret` field optional.

The following sections describe how to create grants using OAuth 2.0 and PKCE.

Construct a code challenge

Before you make an authentication request using PKCE, you need to create a `code_challenge`. You'll use this when you make authorization requests.

The following example uses `nylas` as a code verification string and sets the encoding method to `S256` for extra security.

> **If you don't set an encoding method, Nylas assumes you're using a plain text code verification string.** Nylas strongly recommends you use SHA-256 encoding to create a more secure `code_challenge`.

1. Hash the verification string using an SHA-256 encoding tool (`SHA256("nylas")` → `e96bf6686a3c3510e9e927db7069cb1cba9b99b022f49483a6ce3270809e68a2`).
2. Convert the hashed string to Base64 encoding and remove any padding (`e96bf6686a3c3510e9e927db7069cb1cba9b99b022f49483a6ce3270809e68a2` → `ZTk2YmY2Njg2YTNjMzUxMGU5ZTkyN2RiNzA2OWNiMWNiYTliOTliMDIyZjQ5NDgzYTZjZTMyNzA4MDllNjhhMg`).
3. Save the resulting encoded string to use as the `code_challenge` in your authorization request.

Make authorization request with code challenge

Next, make a `GET /v3/connect/auth` request to create a URL that redirects your user to the auth flow.

Request | Node.js SDK | Python SDK | Ruby SDK | Java SDK | Kotlin SDK

```
/connect/auth?
  client_id=<NYLAS_CLIENT_ID>
  &redirect_uri=https://myapp.com/callback-handler
  &response_type=code
  &provider=google
  &scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fgmail.readonly%20profile
  &state=sQ6vFQN

&code_challenge=ZTk2YmY2Njg2YTNjMzUxMGU5ZTkyN2RiNzA2OWNiMWNiYTliOTliMDIyZjQ5NDgzYTZjZTMyNzA4MDllNjhhM
  &code_challenge_method=S256
```

nylas DOCS

The rest of the OAuth flow should proceed as usual: the user is redirected to the auth provider, where they authenticate and accept or reject the requested scopes. The provider then sends them back to Nylas with an authorization `code`, and Nylas creates an unverified grant. Nylas returns the user to your project with the `code`.

Use the `code` in a `POST /v3/connect/token` request to get an access token. Because you're using PKCE, you must set the `grant_type` to `authorization_code` and include your `code_verifier` string.

For readability, the example below sets `code_verifier` to the original plain text `code_verifier` value.

```
POST /token HTTP/1.1
Host: /v3/connect/token
Content-Type: application/json

{
  "client_id": "<NYLAS_CLIENT_ID>",
  "redirect_uri": "<REDIRECT_URI>",
  "grant_type": "authorization_code",
  "code": "<AUTH_EXCHANGE_CODE>",
  "code_verifier": "nylas"
}
```

⚠ **You must include your Nylas application's API key to refresh your access token**. If you are working on a client-side JavaScript or mobile application, you can use `grant_type=authorization_code` instead of `grant_type=refresh_token` so you don't need to store the API key.

You can also exchange the `code` for an access token using the Nylas SDKs, as in the following examples.

Node.js SDK | Python SDK | Ruby SDK | Java SDK | Kotlin SDK

```
  console.log(res.status)

  const code = req.query.code

  if (!code) {
    res.status(400).send('No authorization code returned from Nylas')
    return
  }

  try {
    const codeExchangeResponse = nylas.auth.exchangeCodeForToken({
      redirectUri: "REDIRECT_URI",
      clientId: "CLIENT_ID",
      clientSecret: "API_KEY",
      code: "CODE"
  });

    const { grantId } = response

    res.status(200)
  } catch (error) {
    console.error('Error exchanging code for token:', error)

    res.status(500).send('Failed to exchange authorization code for token')
  }
})
```

Handling authentication errors

If authentication fails, Nylas returns the standard OAuth 2.0 error fields in the response: `error`, `error_description`, and `error_uri`.

```
https://myapp.com/callback-handler?
  state=...               // Passed value of initial state if it was provided
  &error=...              // Error type/constant
  &error_description=...  // Error description
  &error_uri=...          // Error or event code
```

If an unexpected error occurs during the callback URI creation step at the end of the flow, the response includes an `error_code` field instead of an `error_uri`.

```
&error=internal_error                                          // Error type/constant
&error_description=Internal+error%2C+contact+administrator     // Error description
&error_code=500                                                // Code of internal error
```

Using `access_type` to request refresh tokens

You can use the `access_type` parameter in your token exchange request to indicate whether you want Nylas to return a refresh token for the grant.

If your project is a mobile or client-side-only app, Nylas recommends you use `access_type=online`. This prevents the OAuth process from creating a refresh token, so you must prompt the user to re-authenticate when their access token expires.

If your project *isn't* mobile or client-side-only, you can use `access_type=offline` to get a refresh token when a user authenticates. You can use the refresh token to get a new access token for the user without prompting them to re-authenticate.

⚠ **For security reasons, Nylas strongly recommends against using `access_type=offline` for mobile and client-side-only applications**. In these cases it's best for your users to re-authenticate when their access tokens expire.

Using the `state` parameter to pass information about the user

Nylas Hosted OAuth includes the standard `state` parameter. This is an optional parameter in Nylas, and if you include it in an authorization request, Nylas returns the value unmodified back to the application. You can use this as a verification check, and to track information about the user that you need when creating a grant or logging them in.

Learn more about the `state` parameter in the OAuth 2.0 specification or in the official OAuth 2.0 documentation.

nylas DOCS

Forums

Trust Center

Send Feedback

Cookies