

# **Managing Email and Folders with 4D NetKit**

By Trina Nguyen, Technical Services Engineer, 4D Inc.

Technical Note 24-13

## Table of Contents

Table of Contents .....	2
Abstract .....	3
Introduction .....	3
Prerequisites to Using 4D NetKit.....	4
Registering a Google Application to get Client Secrets .....	4
Establishing Connection and Obtaining the Access Token.....	8
Send, Receive, Delete Emails via Google Gmail API .....	10
Creating, Updating, Assigning Email Labels .....	11
Registering an Application with Microsoft Identity Platform.....	11
Establishing Connection and Obtaining the Access Token.....	12
Send, Receive, Delete Emails via Microsoft Graph API .....	13
Creating, Renaming, and Deleting Email Folders .....	14
Additional Resources .....	14
Conclusion .....	15

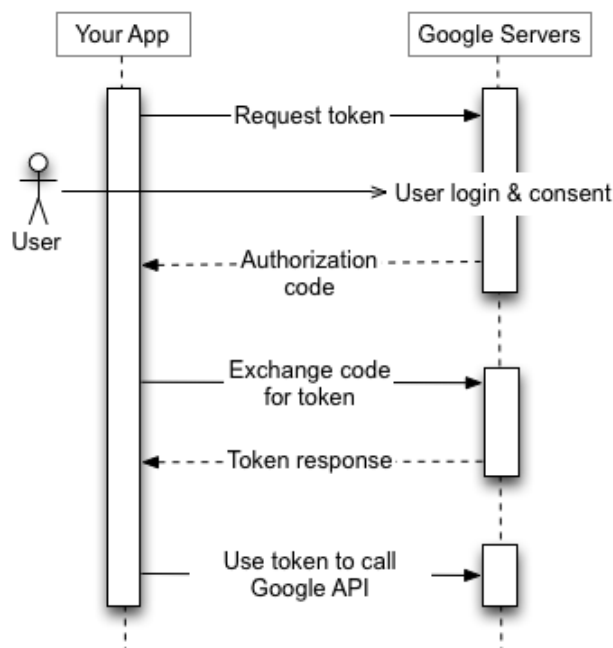
## Abstract

As the Internet moves towards higher security protocols, so do the authorization systems. OAuth 2.0 was introduced as an alternate and more secure login procedure compared to the classic protocol of username and password. OAuth 2.0 utilizes tokens that take the user information from other web apps to authenticate them for your own platform instead of having the user login over and over again. Introduced in 4D 19R3, 4D NetKit manages the OAuth 2.0 connection between 4D applications and 3rd party web services and their API to get these tokens. This technical note will go over the basics of 4D NetKit and how to use it to connect to Google's Gmail and Microsoft's Office365.

## Introduction

4D NetKit revolves around OAuth 2.0, which has become the industry standard protocol for online authorization. 4D previously had support for OAuth 2.0 for IMAP, SMTP, and POP3 transporters, but the introduction of 4D NetKit allows easier management for launching the web browser and managing the authentication process.

OAuth 2.0 work with two types of tokens, access and refresh. The access token works in place of the standard username and password and comes up as a form of an authorization request to the user from the web browser. When the access token expires, then OAuth 2.0 will check to see whether the refresh token flag has been enabled. The refresh token is obtained through REST and allows the connection to stay open after the access token expires, and continually obtains another one as it expires itself. Otherwise, if the flag is not enabled, then the user would have to go through the web browser authentication again to get a new access token.



Once the access token is obtained, it will be sent back to a third-party API to grant the ability to fulfill whatever set of operations it has permission to do. For example, a token issued for the Google Calendar API does not have the same access as a token issued for Google Gmail API.

## Prerequisites to Using 4D NetKit

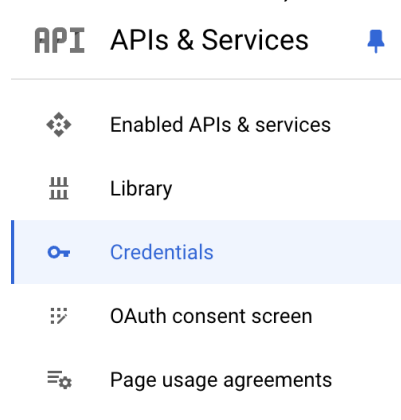
Before anything, the latest version of 4D needs to be installed and a Web Application Expansion license to be able to listen for and receive tokens. The methods gone over in this tech note, as well as the attached sample database, will need a minimum version of 4D 20 R6.

To begin using 4D NetKit, the client ID and client secrets will need to be set up first. The attached sample database will already have a registered application for demo purposes; however, the developer is expected to register their own application for their own programs. This tech note will go over how to set that up for both Google's Gmail API and Microsoft's Office365 API.

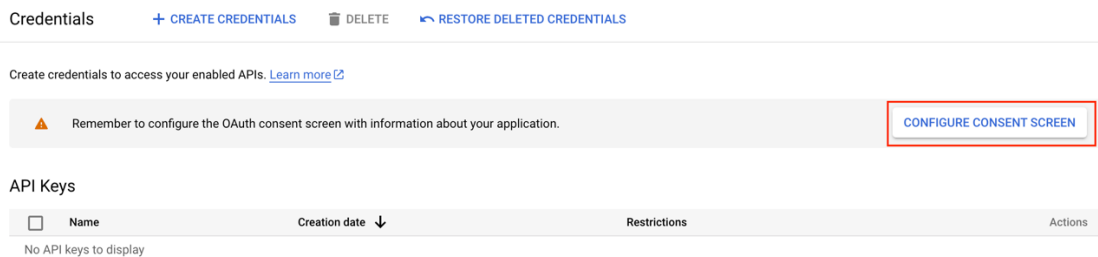
## Registering a Google Application to get Client Secrets

To obtain client secrets from a Google application, follow the steps below:

1. Login to <https://console.developers.google.com/>
2. On the left sidebar under "API's & Services," click on **Credentials**



3. Click on **Create Project** to create the application
4. Name the project and click **Create** to be returned to the Credentials screen
5. Click **Configure Consent Screen**



6. Select **External** for User Type and click **Create** to continue

OAuth consent screen

Choose how you want to configure and register your app, including your target users. You can only associate one app with your project.

User Type

☐ Internal

Only available to users within your organization. You will not need to submit your app for verification. [Learn more about user type](#)

☒ External

Available to any test user with a Google Account. Your app will start in testing mode and will only be available to users you add to the list of test users. Once your app is ready to push to production, you may need to verify your app. [Learn more about user type](#)

CREATE

7. On the **Edit app registration** screen, fill out the required fields. The app domain field could be left blank for now. Click **Save and Continue** to move on
8. On the **Scopes** screen, click **Save and Continue**. Additional scopes can be added later to better suit the application.
9. On the **Test users** screen, click **Add Users** to add the emails of those who will be using this application in Testing mode. Click **Save and continue**

Edit app registration

☒ OAuth consent screen
 —
 ☒ Scopes
 —
 ☒ 3 Test users
 —
 ☐ 4 Summary

### Test users

While publishing status is set to "Testing", only test users are able to access the app. Allowed user cap prior to app verification is 100, and is counted over the entire lifetime of the app. [Learn more](#)

+ ADD USERS

Filter Enter property name or value

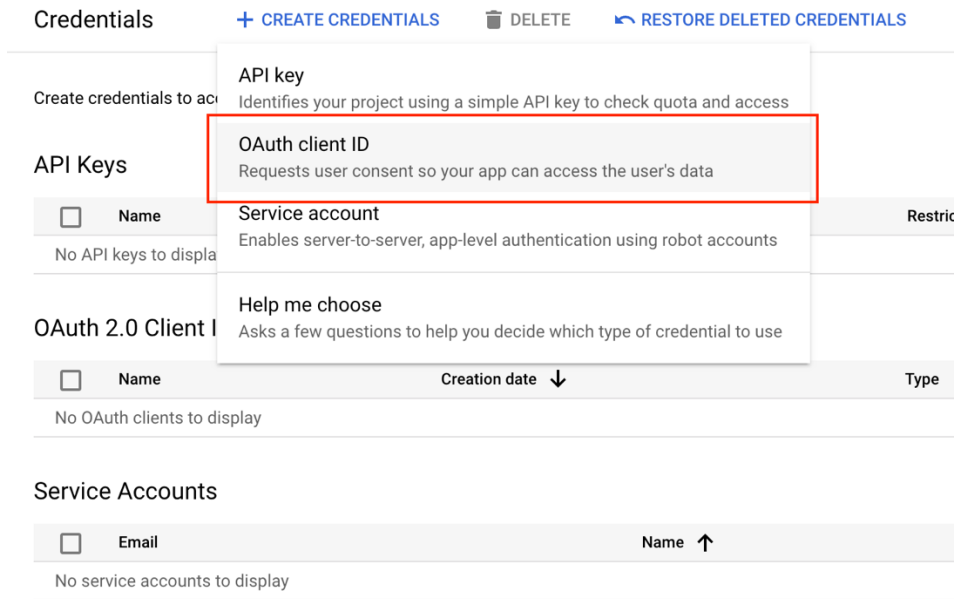
User information

No rows to display

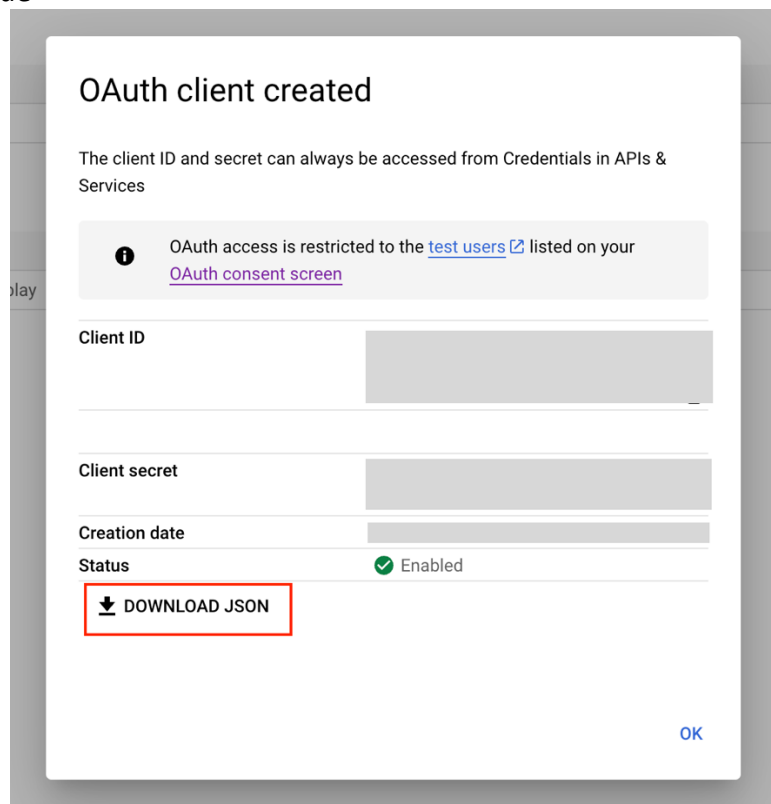
SAVE AND CONTINUE

CANCEL

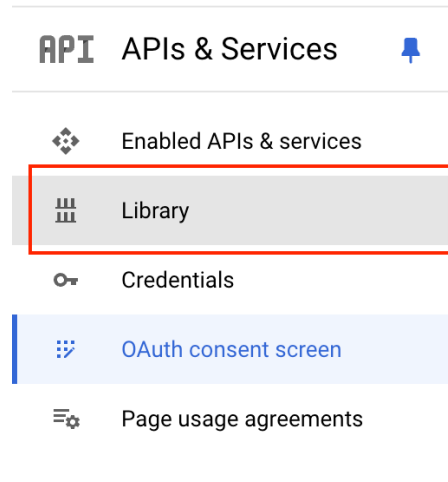
10. On the **Summary** screen, click **Back to dashboard** at the bottom of the page
11. Go back to the **Credentials** screen, click **Create Credentials** and select **OAuth client ID**



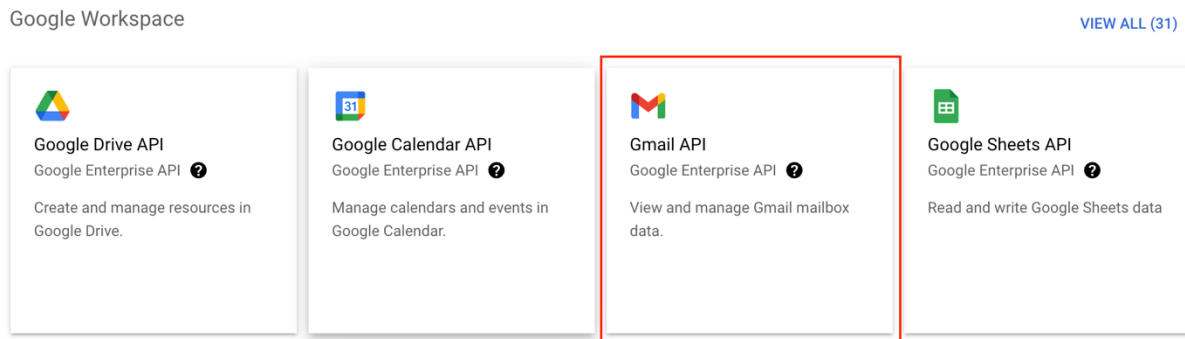
12. Select **Desktop app**, enter in the name of the app, then click **Create**
13. A pop-up window should appear that displays the client ID and client secret for the application. There is the option to click the **DOWNLOAD JSON** link to save the file with the information as a .json file, otherwise it can be accessed anytime from the **Credentials** screen. This information will be used later in this tech note. Click **OK** to continue



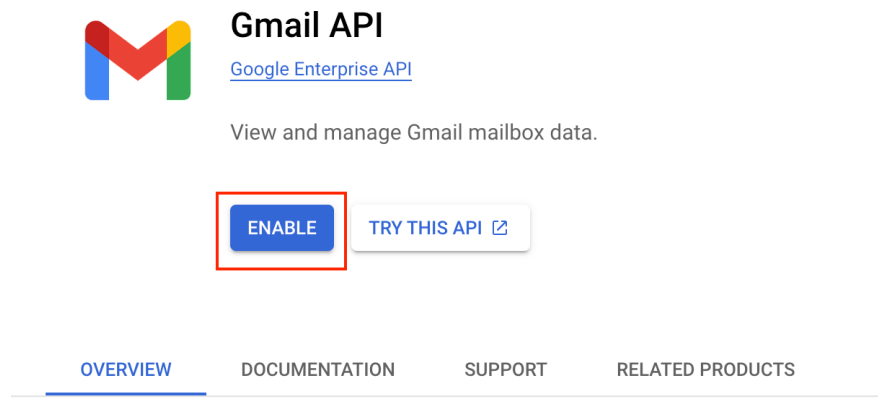
14. Go back to the left sidebar and click on the **Library** tab



15. Scroll down until the **Gmail API** square under **Google Workspace** appears. Click on it



16. Click on the blue **Enable** button to allow this application to call on the Gmail API



From here, the developer can choose to keep their application in “testing” status so that they could test out how their program would work once it goes live. However, if the application will be using any sensitive scopes, such as the permission to send and receive emails from a user’s account, then they must prepare their application to undergo Google’s verification process. More details could be found out through the link below:

[https://support.google.com/cloud/answer/13463073?authuser=3&visit\\_id=638622395066771837-2271496442&rd=1](https://support.google.com/cloud/answer/13463073?authuser=3&visit_id=638622395066771837-2271496442&rd=1)

## Establishing Connection and Obtaining the Access Token

To obtain the access token from Google and establish connection, the developer would need to start by creating the 4D NetKit OAuth2Provider object and filling it in with the client secrets obtained from the previous section.

```
// Creates the object that contains all the information for the OAuth2 connection
#DECLARE : cs.NetKit.OAuth2Provider

var $param : Object

$param:=New object()
$param.name:="Google"
$param.permission:="signIn"
$param.clientId:="YOUR CLIENT ID"
$param.clientSecret:="YOUR CLIENT SECRET"
$param.redirectURI:="http://127.0.0.1:50993/authorize/"
$param.scope:="https://www.googleapis.com/auth/gmail.send"

// Create new OAuth2 object
return cs.NetKit.OAuth2Provider.new($param)
```

The .permission parameter determines whether the application will have Microsoft or Google sign the user in through the “signIn” parameter, or have Microsoft called with its own identity or have Google access without a user. The client ID and secret are the values taken from the secrets JSON file taken from the previous section. The redirect URI is only used in “signIn” mode, and is the location the server sends the user after authentication, usually to confirm a successful authorization. The scope is the collection of API permissions that the developer would want the user to consent to.

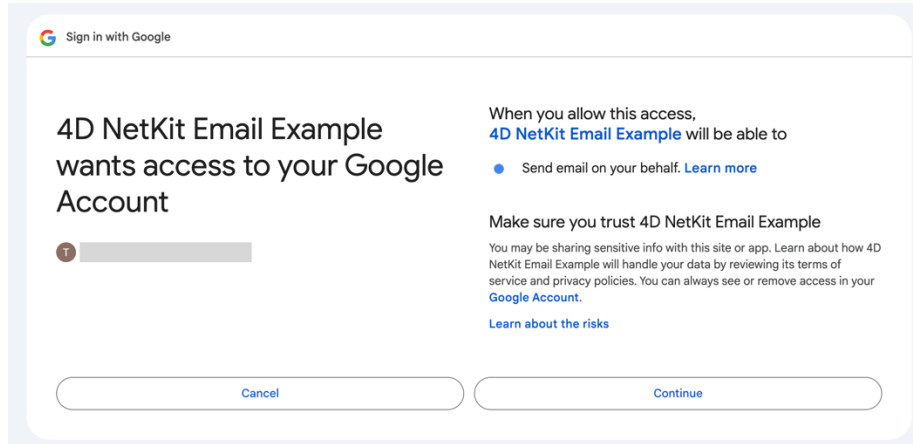
Afterwards, using the method .getToken() will open up the URL through the default browser window so that the user can insert their credentials to get connected. The code below uses the returned OAuth2 object from the *GoogleProvider* method from the code above.

```
var $google : cs.NetKit.Google

Form.oAuth2:=GoogleProvider
Form.oAuth2.getToken()
```

The URL that opens should look like the image below. If the user encounters an Error 403: access\_denied screen, make sure that the email credentials being used matches up with the user added in the **OAuth consent screen** under **Test users**. If the app is in “Testing” status, then it will allow the connection as long as the email being used is part of the **Test users** list.





Please note that if the app is “In Production” status by this stage but not verified, Google will send a notice screen back saying that the app would need to be verified before it will allow the connection to be made.

Since the sample database is a simple test application made for demonstrative purposes, it is not linked to any official web domain, and thus Google may warn the user that the app is not verified. This sample app still adheres to 4D’s privacy policy and no personal info or emails will be shared or altered in any way. To continue with the process, click on **Advanced** and then **Go to 4D Netkit Email Example (unsafe)**.



### Google hasn’t verified this app

The app is requesting access to sensitive info in your Google Account. Until the developer ([tnguyen@4d.com](mailto:tnguyen@4d.com)) verifies this app with Google, you shouldn’t use it.

[Hide Advanced](#)

[BACK TO SAFETY](#)

Continue only if you understand the risks and trust the developer ([tnguyen@4d.com](mailto:tnguyen@4d.com)).

[Go to 4D NetKit Email Example \(unsafe\)](#)

The user will be given the choice to allow or deny the app to have access to emails and email labels. To allow the connection, click **Continue**.

If the connection is successful, the browser will attempt to open the redirect URI that was entered in the OAuth2 object. In this case, it was <http://127.0.0.1:50993/authorize/>. In “signedIn” mode, a web server included in 4D NetKit starts automatically in the port specified (50993) to intercept the provider’s response and display it in the browser. In this example, the default page is shown.

## Send, Receive, Delete Emails via Google Gmail API

The first step in demonstrating how to use 4D NetKit with the Gmail API is through sending emails. To begin, first create a Google object using the OAuth2 object from the previous sections. The mailType parameter refers to the Mail type used to send and receive emails, with the possible types being "MIME" and "JMAP."

```
$google:=cs.NetKit.Google.new(Form.oauth2; New object("mailType"; "JMAP"))
```

The resulting object includes the mail.send() function that allows us to send the emails.

```
// Email creation
$email:=New object
$email.from:="youremail@gmail.com"
$email.to:="destinationmail@mail.com"
$email.subject:="Test Gmail API"
$email.textBody:"this is a Gmail API test"

// Email sending
$status:=$google.mail.send($email)
```

To retrieve mail, the .getMailIds() function is used to grab all the mail IDs in the inbox. Then, to download the mail content of a specific id, the .getMail() function is used. The .getMails() function could also be used to get a collection of emails from \$mailIds. It will return the mail in either the MIME or JMAP formats.

```
#DECLARE($OAuth2Provider : cs.NetKit.OAuth2Provider; $param : Object; $winRef : Integer)

var $mailIds:=[]
var $mails:=[]
var $mailId : Object
var $google : cs.NetKit.Google

$google:=cs.NetKit.Google.new($OAuth2Provider)

// Grabs the email ID
$mailIds:=$google.mail.getMailIds($param).mailIds

// Downloads the collection of emails from the mailID
If ($mailIds.length>0)
    $mails:=$google.mail.getMails($mailIds)
End if
```

4D NetKit also gives the option to either permanently delete email, or move it to the trash folder with the .delete() function. Below are examples to delete and move the most recent email from an inbox to the trash.

```
// Delete permanently a mail
$status:=$google.mail.delete($mailIds.mailIds.first().id; True)

// Move a mail to the trash
$status:=$google.mail.delete($mailIds.mailIds.first().id; False)
```

## Creating, Updating, Assigning Email Labels

Creating a label for emails is simple. Use the `.createLabel()` function to create a label, such as the “Backup” label below:

```
$status:=$google.mail.createLabel({name: "Backup"})
$labelId:=$status.label.id
```

The label name, total messages, and unread messages can be obtained using the label ID.

```
$info:=$google.mail.getLabel($labelId)
$name:=$info.name
$emailNumber:=$info.messagesTotal
$unread:=$info.messagesUnread
```

Using the `.updateLabel()` function, the name, visibility, and color of labels can be updated if it is a label under the type=“user”. For example, the name of a label is updated to a different name below.

```
$status:=$google.mail.updateLabel($labelId, {name:"Backup January"})
```

To update a label’s assignment to an email, simply use the `.update()` function to add or remove a label to the specific mail ID, or a group of them.

```
// Updates email in $mailIds to have the "Work" and "IMPORTANT" labels
$status:=$google.mail.update($mailIds, {addLabelIds: ["Work", "IMPORTANT"]})

// Updates email in $mailIds to remove the "IMPORTANT" label
$status:=$google.mail.update($mailIds, {removeLabelIds: ["IMPORTANT"]})
```

## Registering an Application with Microsoft Identity Platform

Before looking to use NetKit to connect to Microsoft365, an application in the Microsoft Entra admin center must first be registered.

Please note, there are a couple prerequisites to registering an application with Microsoft. The developer must have an Azure account with an active subscription, the account must be at least a Cloud Application Administrator, and the quick start to setting up a tenant a tenant must be completed.

To do all these steps, please follow the instructions from Microsoft through the link below:  
<https://learn.microsoft.com/en-us/entra/identity-platform/quickstart-register-app?tabs=certificate>

## Establishing Connection and Obtaining the Access Token

Once the application is registered, the connection procedure is extremely similar to the previous example when connecting to Google. First, create the OAuth2 object and fill out the relevant information and client ID taken from the registered application.

```
// Creates the object that contains all the information for the OAuth2 connection
#DECLARE : cs.NetKit.OAuth2Provider

var $param : Object

$param:=New object()
$param.name:="Microsoft"
$param.permission:="signedIn"
$param.clientId:="YOUR CLIENT ID"
$param.redirectURI:="http://127.0.0.1:50993/authorize/"
$param.scope:=" https://graph.microsoft.com/.default"

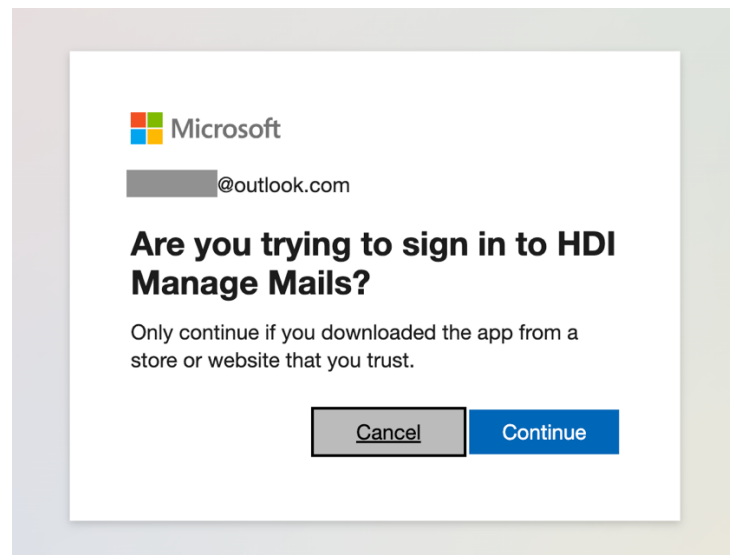
// Create new OAuth2 object
return cs.NetKit.OAuth2Provider.new($param)
```

Afterwards, use the `.getToken()` command to open up the Microsoft Office window to get the access token.

```
var $office365 : cs.NetKit.Office365

Form.oAuth2:=MicrosoftProvider
Form.oAuth2.getToken()
```

The resulting browser window should open up a page that looks similar to below:



## Send, Receive, Delete Emails via Microsoft Graph API

To begin, create a Microsoft365 object from the OAuth2 information above. Similar to the Google object, Office365's mailType could be MIME and JMAP. However, the default option is the additional Mail type, "Microsoft".

```
$office365:=cs.NetKit.Office365.new($oAuth2; New object("mailType"; "Microsoft"))
```

Afterwards, an email object needs to be created to send an email. The following code example are done with the "message resource type" format from Microsoft Graph.

```
var $office365 : cs.NetKit.Office365

$email:=New object
$email.from:=New object
$email.from.emailAddress:"yourEmail@mail.com"

$addressTo:=New object
$addressTo.emailAddress:=New object
$addressTo.emailAddress.email:"sendToAddress@mail.com"

$email.subject:"Hello world"

$email.body:=New object
$email.body.content:"Test content mail"
$email.body.contentType:"text"

$status:=$office365.mail.send($email)
```

This code is a barebones example of an email object, but it can be customized to have the recipient's name, requests for delivery and read receipts, and be sent with importance flags.

To retrieve emails, the mail folder collection must be obtained and then the specific email folder within the collection must be chosen.

```
$folderList:=$office365.mail.getFolderList()

// Collection with all the mail folder directly under the root folder
$folders:=$folderList.folders
```

Afterwards, the .getMails() function is used to retrieve all the emails from that folder. The folder can be identified by either its numerical id number, or its name if the selected folder is a folder created by default for users. Below is a code example showing how to get the emails in the inbox folder:

```
$param:=New object
$param.folderId:"inbox"

$mails:=$office365.mail.getMails($param)
```

Deleting emails utilizes the `.delete()` function. Below is an example to all the email in the `$mails` collection.

```
For each ($mail; $mails)
    $office365.mail.delete($mail.id)
End for each
```

**Note:** *Unlike Google, you may not be able to permanently delete items from Microsoft's recoverable items deletion folder. For more information, please visit Microsoft's documentation website from this link: <https://learn.microsoft.com/en-us/graph/api/message-delete?view=graph-rest-1.0&tabs=http>*

## Creating, Renaming, and Deleting Email Folders

Creating a folder is simple, as it only requires a single line with the function `.createFolder()`. After the folder is created, email can be immediately moved into the newly created folder with the `.move()` command. The code below demonstrates this ability with the creation of the "Backup" folder:

```
// Creates a new folder in the root
$status:=$office365.mail.createFolder("Backup")
If ($status.success)
    $folderId:=$status.id

    // Moves your email in the new folder
    $status:=$office365.mail.move($mailId; $folderId)
End if
```

The newly created folder can be changed with the `.renameFolder()` function.

```
$status:=$office365.mail.renameFolder($folderId; "Renamed Backup")
$folderId:=$status.id
```

Finally, an entire folder is removed with the `.deleteFolder()` command.

```
$status:=$office365.mail.deleteFolder($folderId)
```

**Note:** *This command will delete all of the emails inside the designated folder, so make sure to check thoroughly before running the command*

## Additional Resources

The documentation for 4D NetKit and all its classes and commands could be found through the 4D GitHub repository. More examples on using 4D NetKit with Google and Microsoft, as well as the standard OAuth2Provider class, could be found in the 4D blog posts and sample databases created by Fabrice Mainguené. Both of these resources can be found in the links below:

<https://github.com/4d/4D-NetKit>  
<https://blog.4d.com/tag/4d-netkit/>

## Conclusion

This tech note went over how to get started with setting up applications for both Google's Gmail API and Microsoft's Office365 API and using 4D NetKit to establish a connection to them to call its most used functions. As internet protocols grow and change over time, it becomes more important for a 4D developer to have access to a streamlined and simple way to authenticate and connect to already built and established 3<sup>rd</sup> party APIs.