Linx – Google Sheets Integration

Author: R Worthington

Date: Sep 2019

Linx Version: 5.16.6.0

Contents

Introduction

Google Drive and (specific to this sample) Google Sheets, offers a RESTful Webservice API which allows you to create apps that leverage Google Drive cloud storage. You can develop applications that integrate with Google Drive, and create robust functionality in your application using Google Drive API.

The API enables you to:

- Download files from Google Drive and Upload files to Google Drive.
- Search for files and folders stored in Google Drive. Create complex search queries that return any of the file metadata fields in the Files resource.
- Let users share files, folders and drives to collaborate on content.
- Combine with the Google Picker API to search all files in Google Drive, then return the file name, URL, last modified date, and user.
- Create *shortcuts* that are external links to data stored outside of Drive, in a different data store or cloud storage system.
- Create a dedicated Drive folder to store your application's data so that the app cannot access all the user's content stored in Google Drive. See Store application-specific data.
- Integrate with the *Google Drive UI*, which is Google's standard web UI you can use to interact with Drive files. To learn all that you can do with a Drive app that you integrate with the Google Drive UI, see Drive UI integration overview

Setup

To begin our integration, we first need to register our application on the Google developer's portal, this will allow us to generate the necessary authentication keys as well as grant the API specific access within the Google Drive instances.

In order to progress with the Linx- Google integration, you must have downloaded and installed the Linx Application Designer which will allow you to build/use the sample solution. Following this, you can install the Linx Application Server which will allow you to automate your solution.

If you haven't already downloaded the Linx Application Designer, then do so here.

The following steps are required to configure the initial API setup.

1. Register as a developer
2. Register your Application/Project and Enable API scope
3. Retrieve Authorization Code using created Credentials
4. Use authentication to retrieve Access Tokens
5. Connect your Linx solution

1. Register as a developer

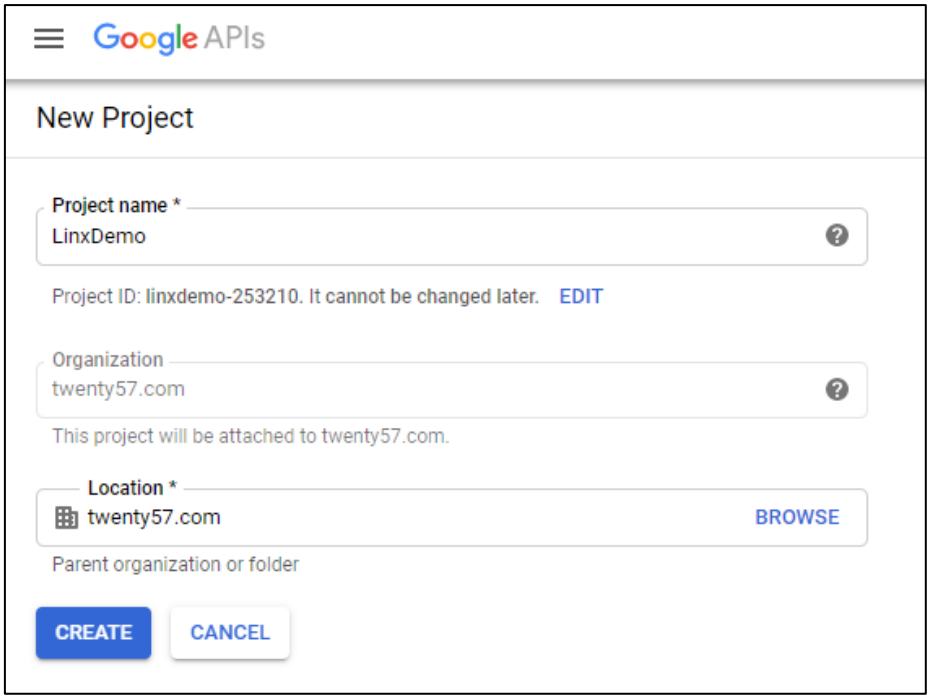
In order for you to interact with the Google Drive API you must first register as a developer, this will give you access to the Google Developer Console which allows you to configure numerous components related to app development and integration.

Navigate to the Google Developer Console, if you are not already registered as a developer then register yourself.

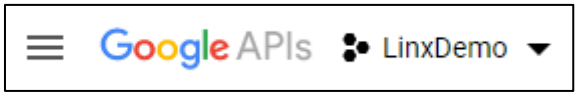
2. Register your Project/Application and Enable API Scope

Once logged in to the Google Developer Console, you should see the API Console Dashboard. On the Menu bar you should see the option to select a project:

For this sample, we are going to create a new Project named 'LinxDemo'

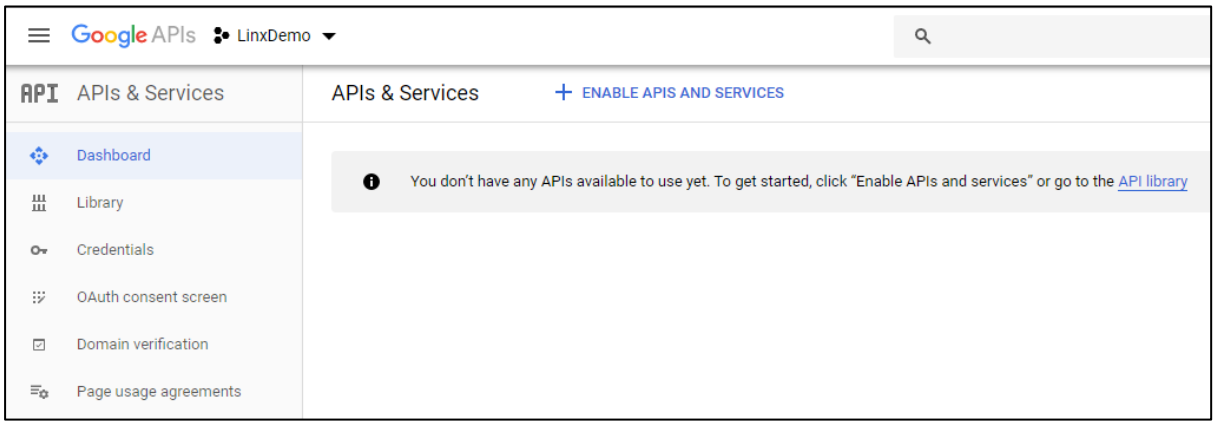


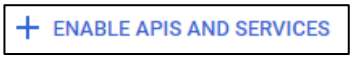
Now we have created our project, make sure it is selected as the current project next to the main menu.



Next, we need to enable the relevant APIs for our Project.

On your dashboard you should see the following screen:



Click on the 

button and you will be redirected to the API Library page.

The API Library contains all the possible Google APIs that you can enable for your application.

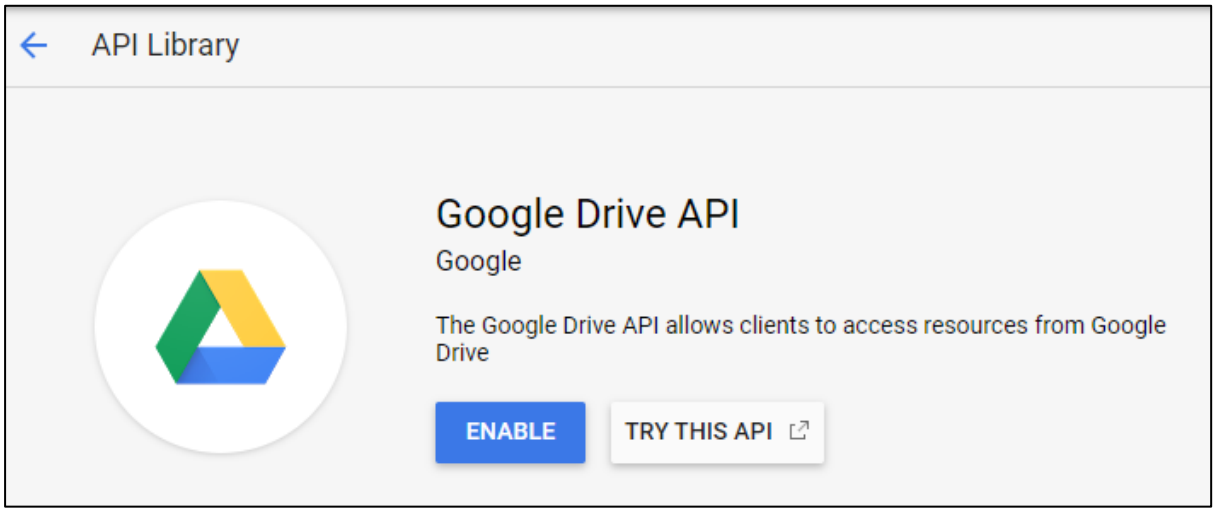
For the scope of this sample we will be using:

- Google Drive API: Allows access to resources from Google Drive

- Google Sheets API: Gives apps full control over the content and appearance of spreadsheet data

- Google+ API: Enables developers to build on top of the Google+ platform (Make sure this is enabled as there appears to be a bug with the API if this isn't enabled).

Search for each of the above and click on the relevant API, then "enable" the relevant APIs:



We now have enabled the relevant scope of the APIs we are going to make user of, in the next section we'll look into generating authorization credentials.

3. Retrieve Authorization Code using created Credentials

In order to create a secure "link" between our solution/application and Google Drive, we need to go through the OAuth 2.0 process which results in Access and Refresh Tokens being generated that we can use to authenticate our solution with Google Drive.

In the following steps we will generate the necessary tokens but for a more information about the OAuth process for Google Drive can be found here.

OAuth Basic Flow:

1. Obtain OAuth 2.0 credentials from the Google API Console.
2. Grant Access and Obtain Authorization Code
3. Obtain an access token from the Google Authorization Server.
4. Connect to the Google API with Access Token

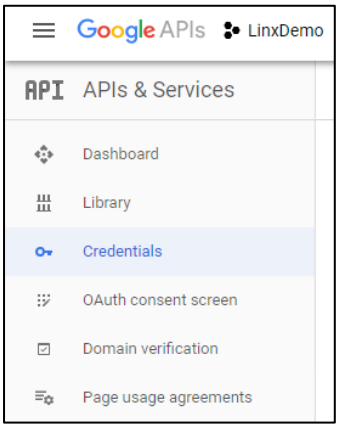
Steps:

1. Obtain OAuth 2.0 credentials from the Google API Console.

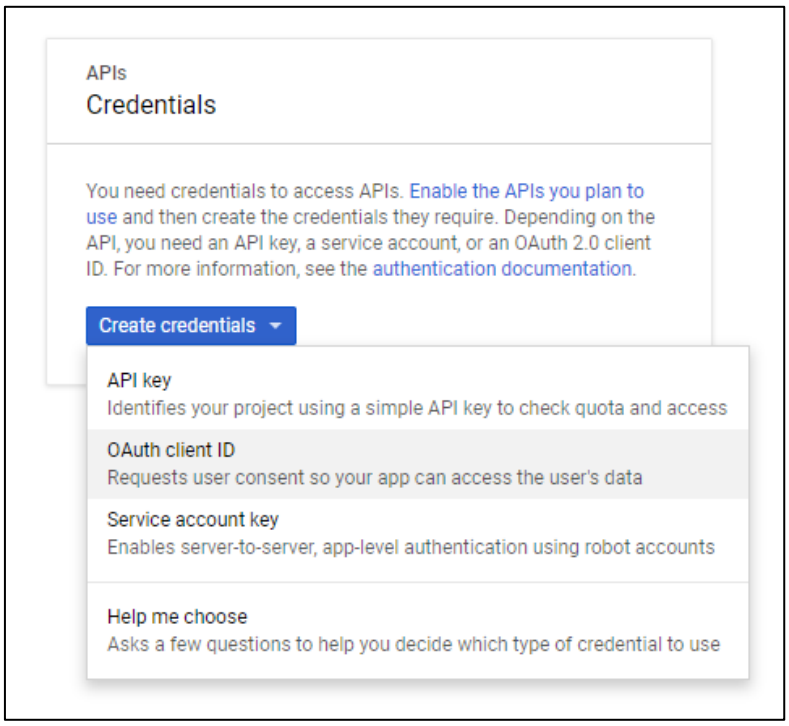
Visit the Google API Console to obtain OAuth 2.0 credentials such as a client ID and client secret that are known to both Google and your application. The set of values varies based on what type of application

you are building. For example, a JavaScript application does not require a secret, but a web server application does.

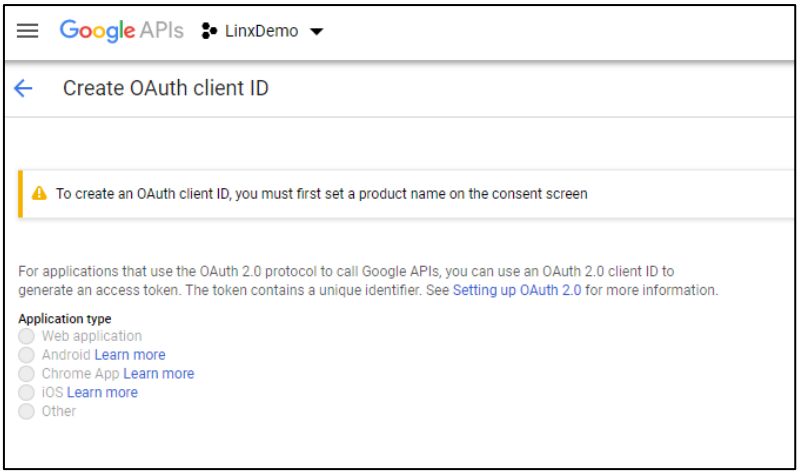
To begin, on the Google API Console dashboard, from the menu on the left panel, select 'Credentials'.



Next, click on 'Create credentials', and from the drop-down options, select 'OAuth client ID' (If you are unable to view options for OAuth credentials then you need to 'Configure consent screen' – described below).



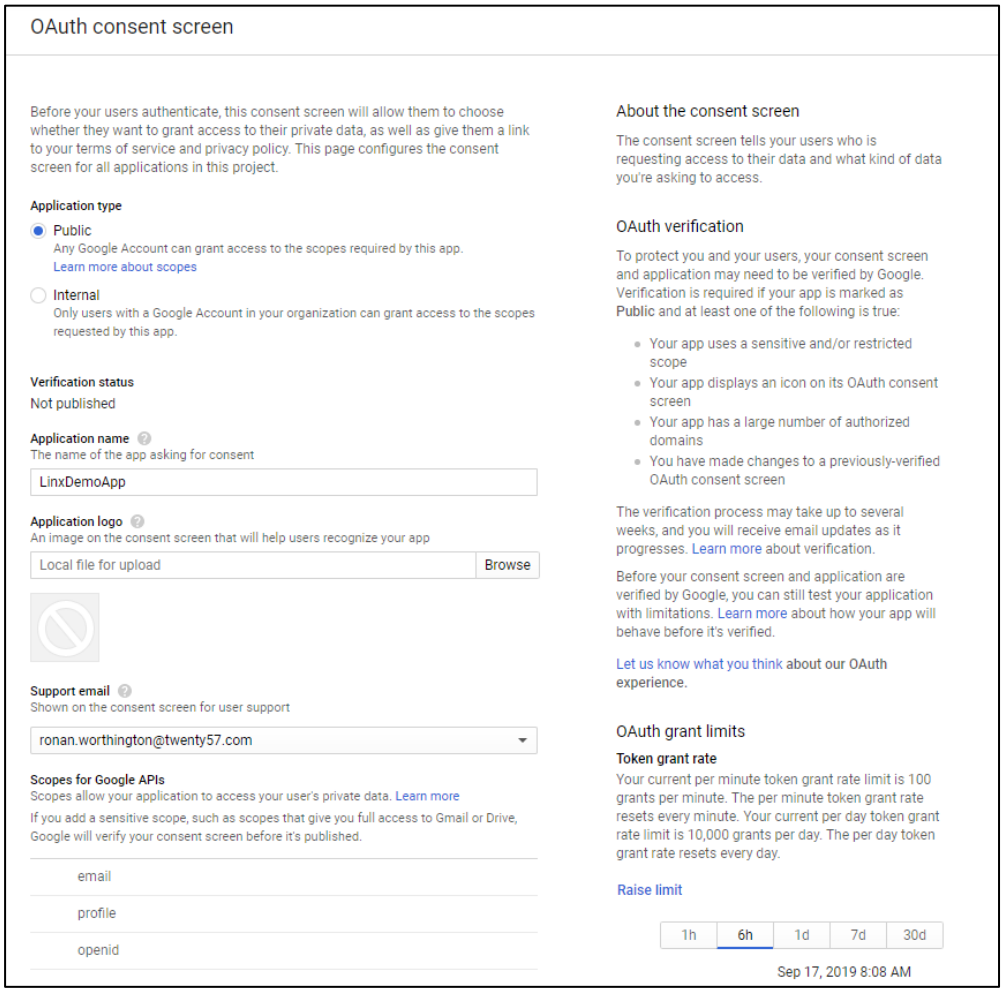
You should see the below screen:

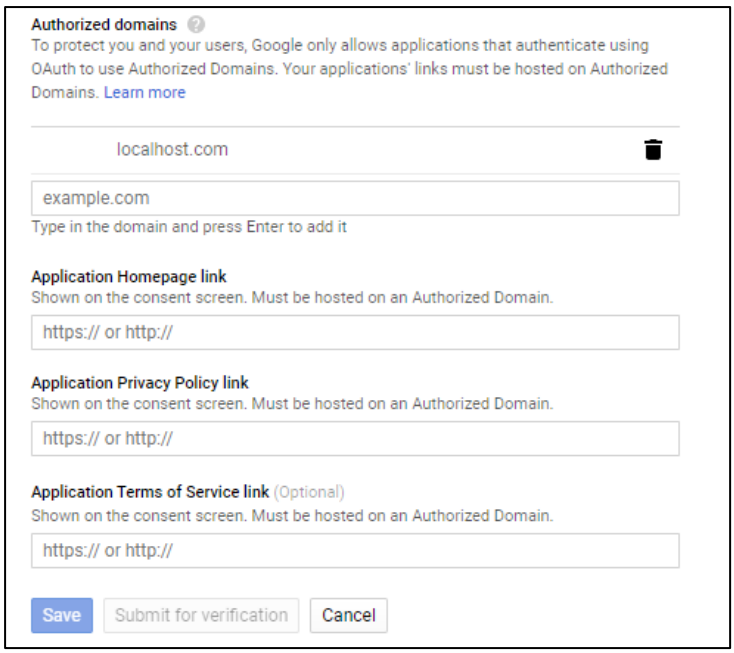


As the message states, we first need to configure the OAuth consent screen, this is the screen that users will see when they initially generate the authorization code on the front-end.

If you click on the Configure consent screen button you will be taken to this page.

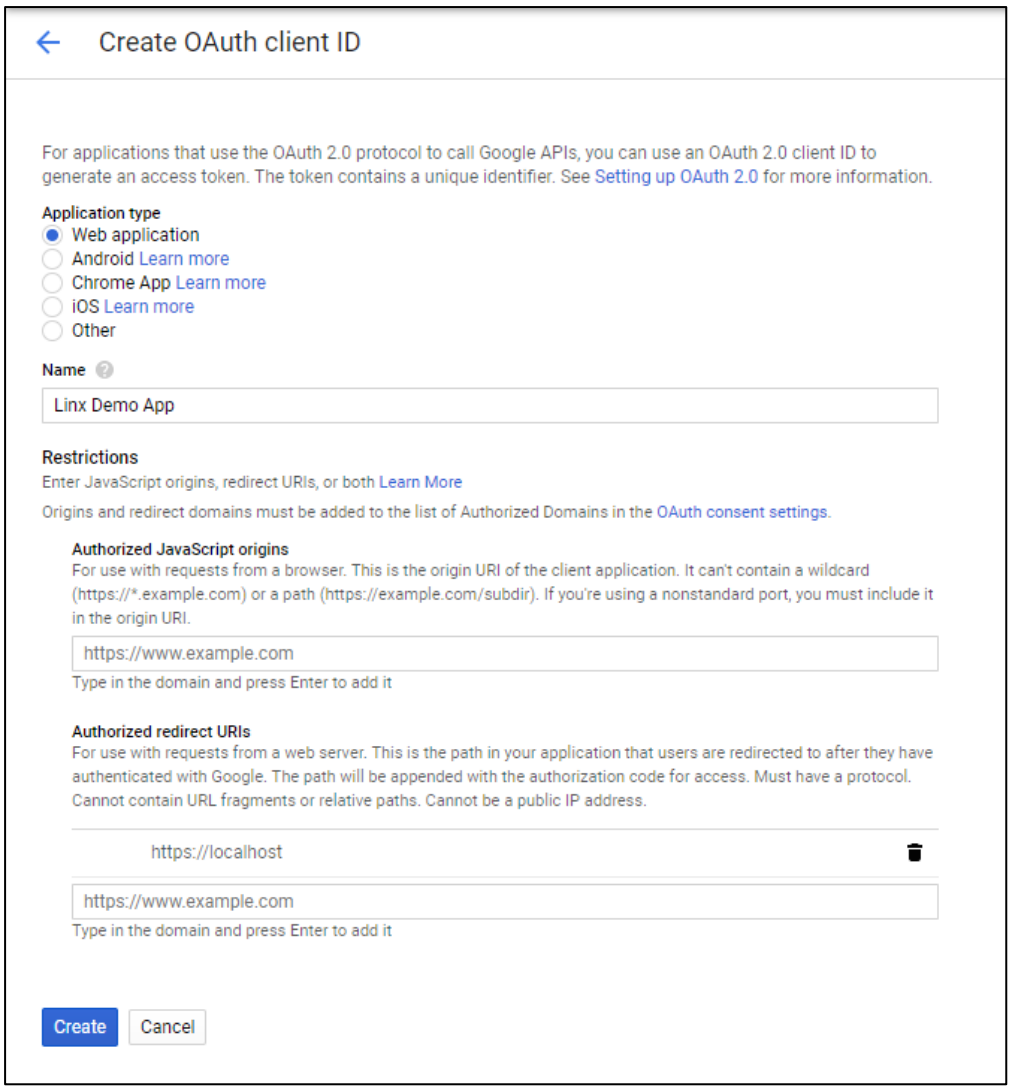
Fill out the basic details of your solution such as the Name and Authorized Domains.



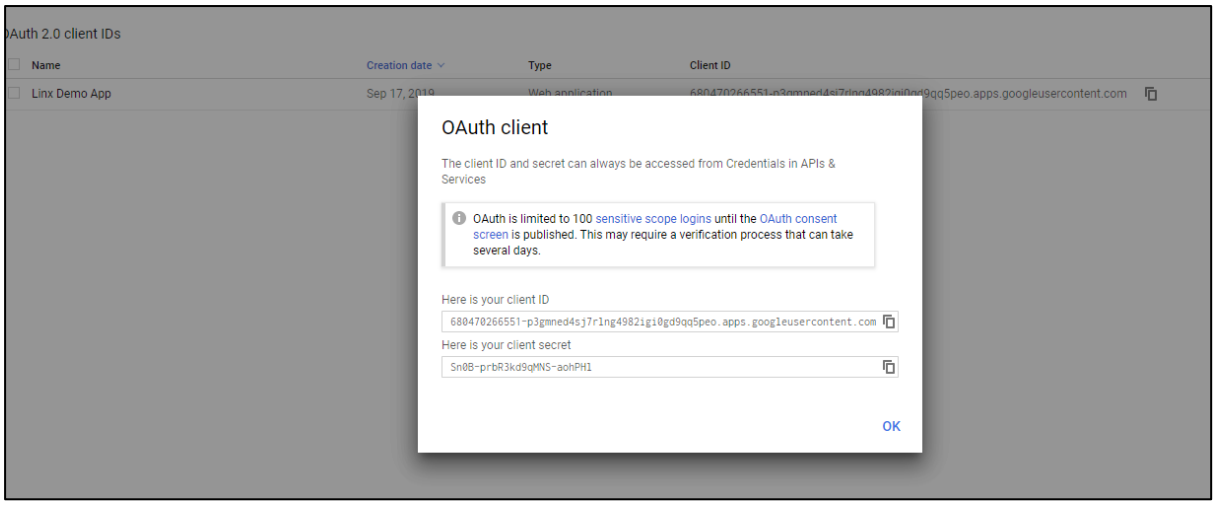


Then click 'Save', you should be redirected to the below page:

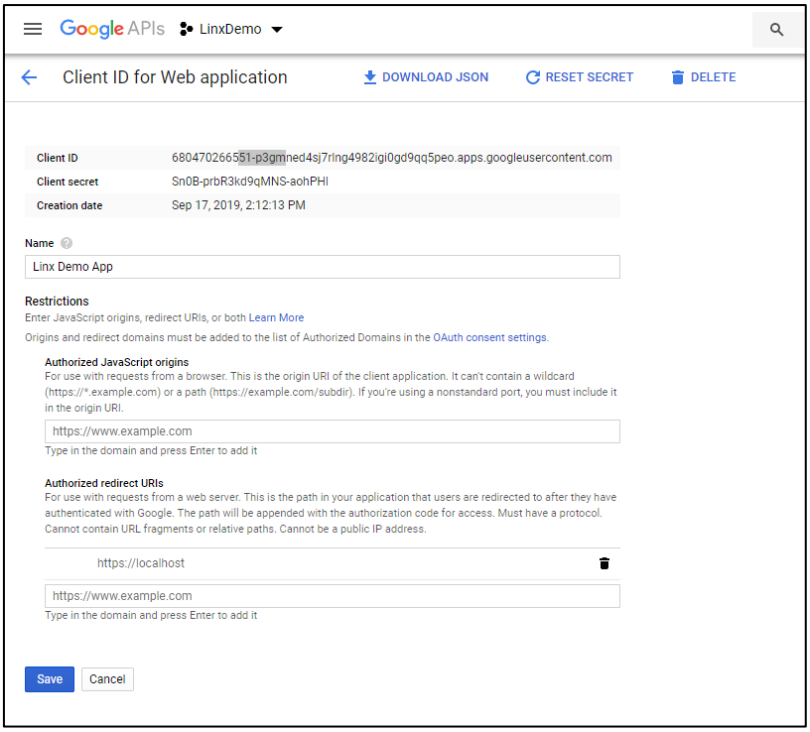
Select the 'Web application' option and complete the fields like below, make note of the redirect URI.



Once created you will see your Client ID and Client Secret displayed:



Copy these values out and store them in a file somewhere just for the moment (you can also download these values as a Json string by clicking on the credentials that you just added)



Json string containing configuration:


```
{
   "web":{
      "client_id":"680470266551-
p3gmned4sj7rlng4982igi0gd9qq5peo.apps.googleusercontent.com",
      "project_id":"linxdemo-253210",
      "auth_uri":"https://accounts.google.com/o/oauth2/auth",
      "token_uri":"https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url":"https://www.googleapis.com/oauth2/v1/certs
",
      "client_secret":"Sn0B-prbR3kd9qMNS-aohPHl",
      "redirect_uris":[
         "https://localhost"
      ]
   }
}
```

Now that we have configured our Google Project, we must now authorize our application from the front-end.

2. Grant Access and Obtain Authorization Code

Before we can generate the necessary access tokens, we first need to manually authorize our application, from here we will obtain an access code which is then used to generate tokens.

Navigate to the following URL:
https://accounts.google.com/o/oauth2/auth?scope=https://www.googleapis.com/auth/drive&response_type=code&access_type=offline&redirect_uri=<redirecturl>&client_id=<client id>

Where <client id> and <redirect url> are replaced by the configuration values obtained in Step 1.

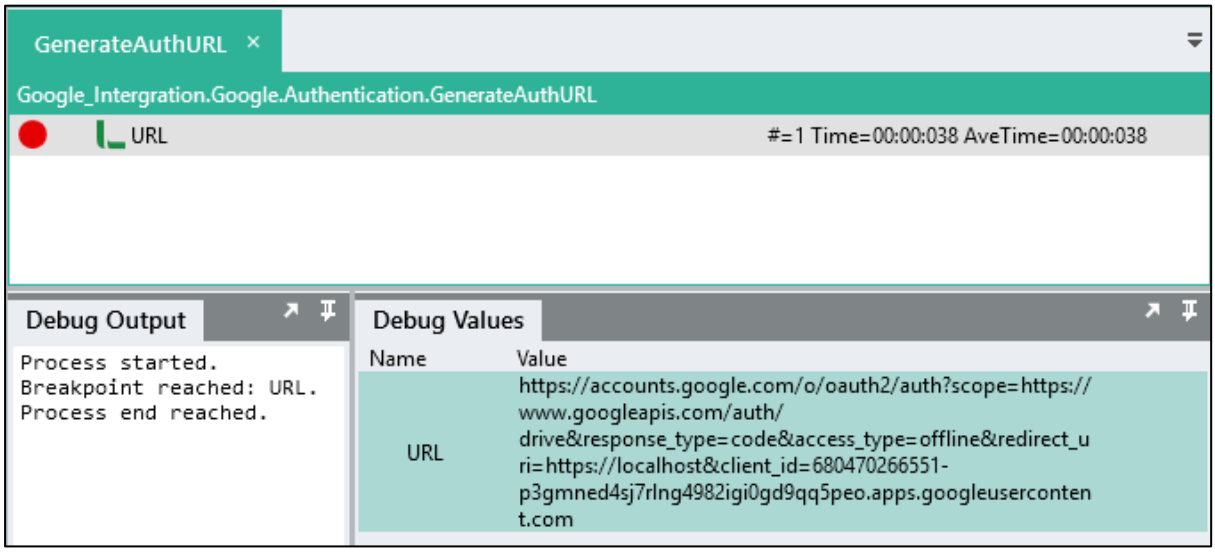
For ease of sake, as you may have to do this a few times, we are going to use Linx to populate the URL for us.

If you open up the sample solution and open the Setting values, replace the following values with the ones obtained in Step 1:

- google_ClientID: Client ID
- google_ClientSecret: Client Secret
- google_RedirectURI: Redirect URI i.e. http://localhost

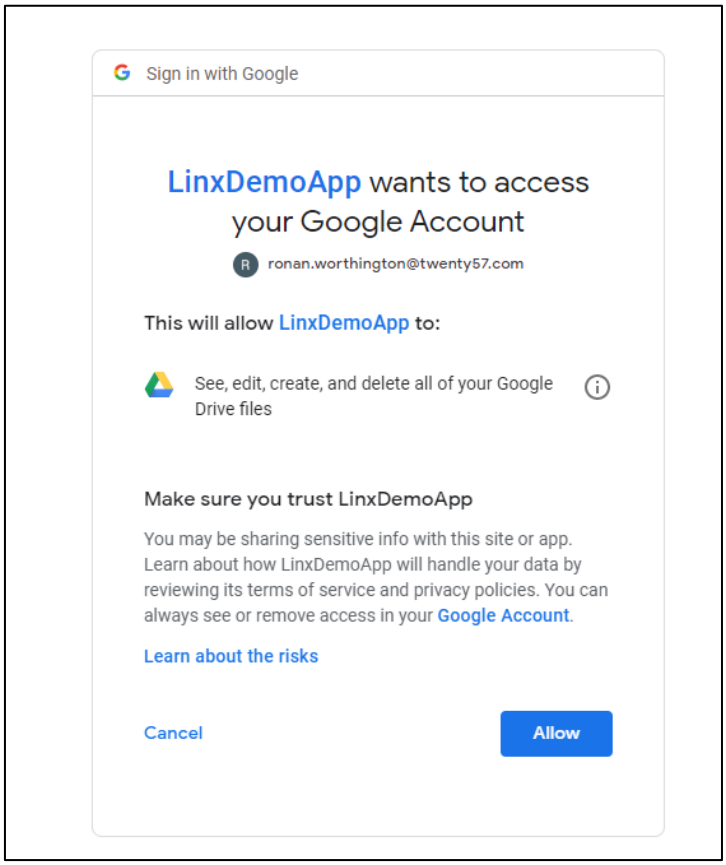
Next, in your Linx sample solution, debug the process [Google] > [Authentication] > GenerateAuthURL.

You will see a URL being constricted with the dynamic parameters:

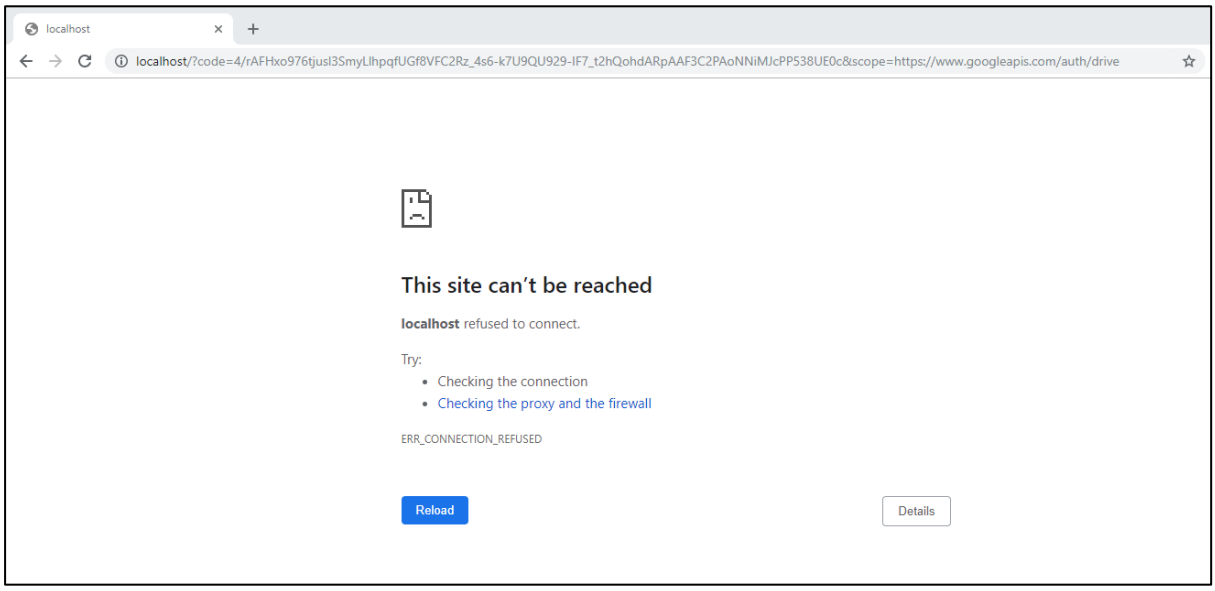


Next, navigate to this URL from a Web browser.

You should then be asked to log in to your Google Account, once logged in you will be presented with the below screen, when you are, click 'Allow':



You will then be redirected to you Redirect URI that we configured in Step 1, in our case, as we are using localhost, it will look like there is an error:



However, if you take a look at the URL, it contains some information, one of those is the Authorization Code in the format of "https://localhost/?code=<your new auth code here>&scope=https://www.googleapis.com/auth/drive.

In this example the Authorization Code will be:
https://localhost/?code=4/rAFHxo976tjusl3SmyLlhpqfUGf8VFC2Rz_4s6-k7U9QU929-lF7_t2hQohdARpAAF3C2PAoNNiMJcPP538UE0c&scope=https://www.googleapis.com/auth/drive

Copy the value after "code=" up until "&scope" and paste it in the Linx sample solution setting value google_AuthCode.

This Authorization code is only valid for one request to get your Access and Refresh tokens (Step 3.3), if your request fails then you will have to generate a new Authorization Code by repeating Step 3.2.

3. Obtain an access token from the Google Authorization Server.

In this next step we are going to make a Webservice request with our authorization values retrieved from previous steps in order to obtain an Access and Refresh token.

Creating Automated Access Token Grant

In order to create a dynamic process that is automated, we need to create 3 processes that will handle the granting and refreshing of access tokens, these will be:

1. GrantTokens: To initially grant the access tokens by using the Auth Code obtained in Step 3.2.
2. RefreshTokens: To handle the renewal of Access Tokens when they are expiring.
3. Authenticate: A handler process that will call the relevant process above.

The above is all included in the sample solution but I will explain how it was built so you can understand for future Webservice authentication as the OAuth process is very generic.

1. Grant Tokens

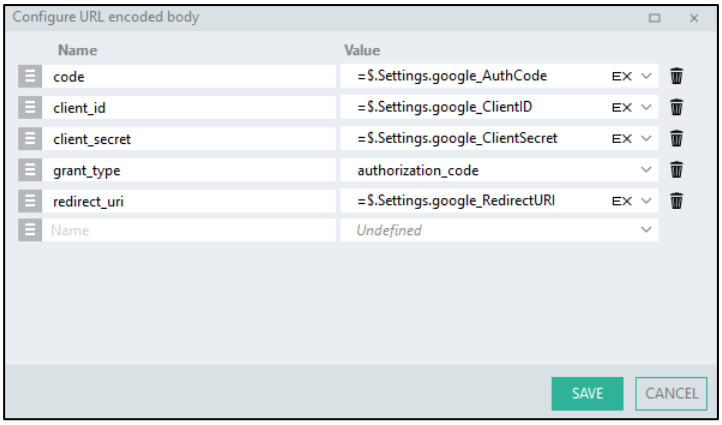
Sample solution process: [Google_Intergration] > [Google] > [Authentication] > [GrantTokens]

This process will make a RESTful WebService request containing our authentication credentials, if successful, it will return the new Access and Refresh tokens.

To begin, create a new process and give it the name of 'GrantTokens'.

Next, add a *CallRESTWebService* function to the process and configure the properties as follows:

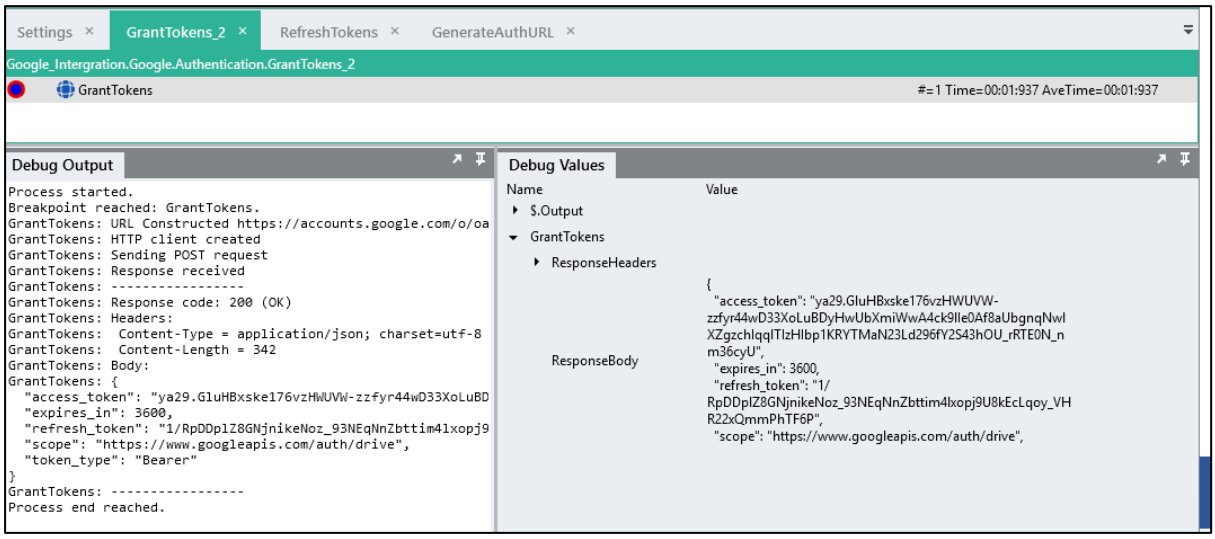
- URL: https://accounts.google.com/o/oauth2/token
- Method: POST
- Body format: URL Encoded Content
- URL Encoded body:



-
- Output type: String

Before running this process, repeat Step 3.2 (Obtain Auth Code) and update the Auth Code setting value.

Once you've updated the code, run the process and take note of the debug output:



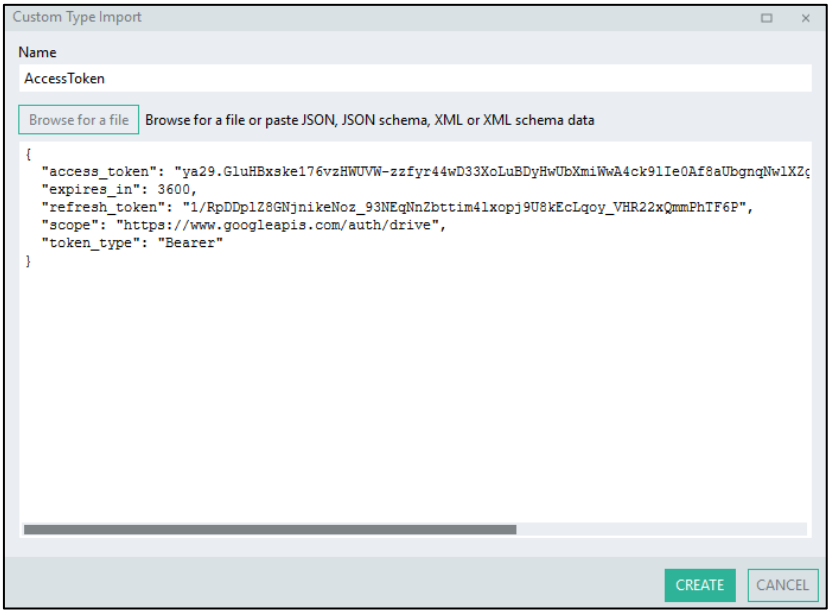
If your request was successful, you should see that the CallRESTWebservice function returns a response "200" along with a Json string containing our access and refresh tokens.

If your response was like the below, then you need to repeat Step 3.2.

```
GrantTokens Trace Log:

URL Constructed https://accounts.google.com/o/oauth2/token

HTTP client created

Sending POST request

Response received

-----------------

Response code: 400 (BadRequest)

Body:

{

 "error": "invalid_grant",

 "error_description": "Bad Request"

}

-----------------
```

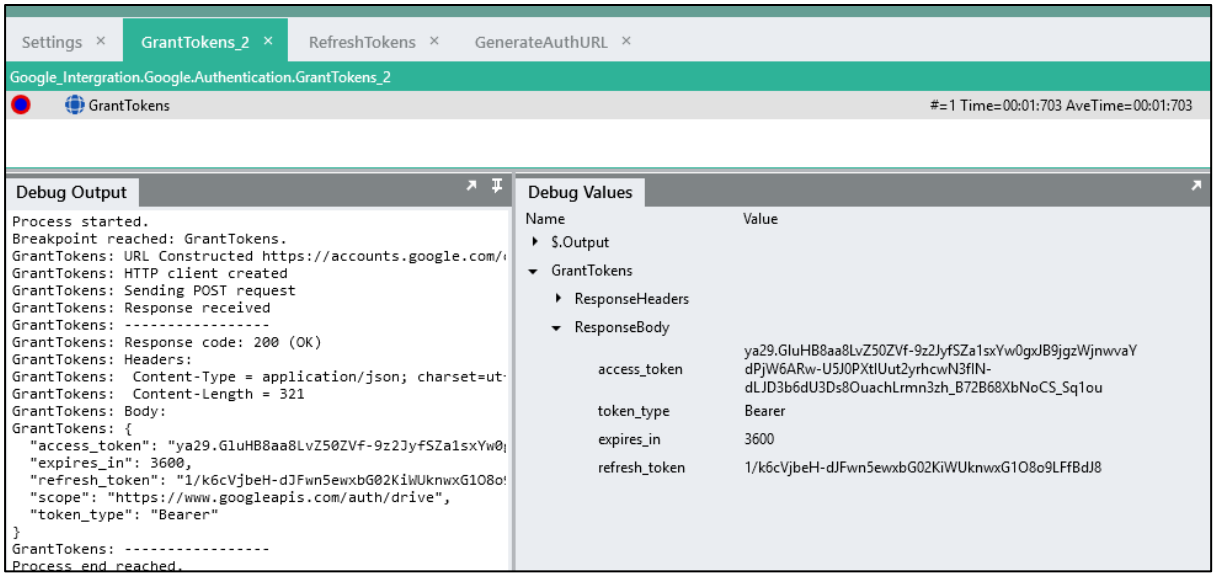
Note: If your request was successful, but there was no 'RefreshToken' present in the response, then go to https://myaccount.google.com/u/0/permissions and remove permissions for the App, then repeat Step 3.2. This is because the RefreshToken is only returned once when you first authorize the App.

Now, for use of use later in our solution, we want to create a CustomType based on this output so we can pass the values into our other sub-processes that will require the credentials. To do this, copy the ResponseBody String output from the CallRESTWebservice function and import it as a custom type names 'AccessToken'

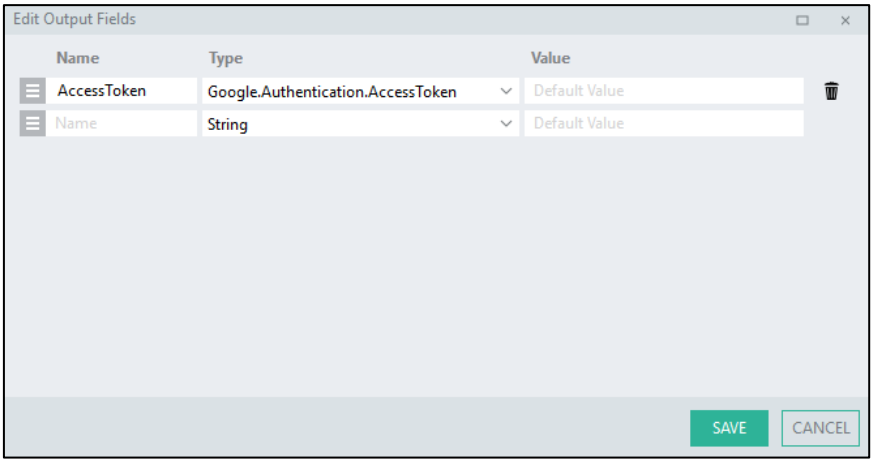


Now that we have our AccessToken *CustomType* structure configured, we can configure the CallRESTWebservice function, to return the output of the request as this CustomType.

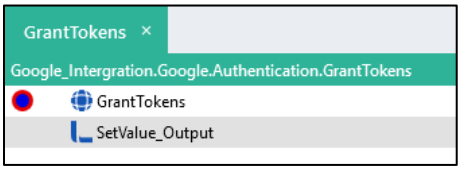
If we repeat the actions from Step 3.2 up until now, and debug our process, you will see the structured output of the Webservice call.



Now to make this process modular, we are going to set this response as the output for the GrantTokens process. To do this, configure a Process output named "AccessToken" and set it of the type CustomType AccessToken.



Then, add a SetValue function to your process that will set the value of the output AccessToken = CallRESTWebservice Response Body:



By doing this, we can call the process from another process and retrieve the Access token in a more user-friendly fashion, it also makes the process more modular so that you can copy it into other solutions without altering much.

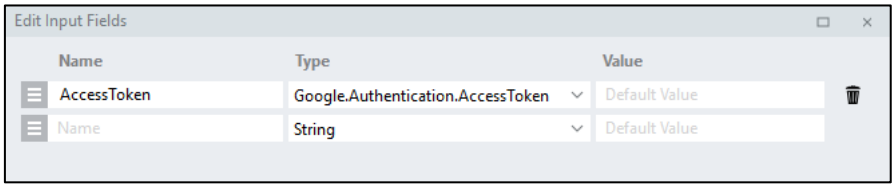
Now that we have completed the process that grants us the access tokens, we are going to build one that refreshes the access tokens for us.

2. Refresh Tokens

Similar to the above process, we are now going to make a process that will take in the AccessToken CustomType, it will then make a Webservice request with the token details, upon successful response, it will return an updated AccessToken custom type.

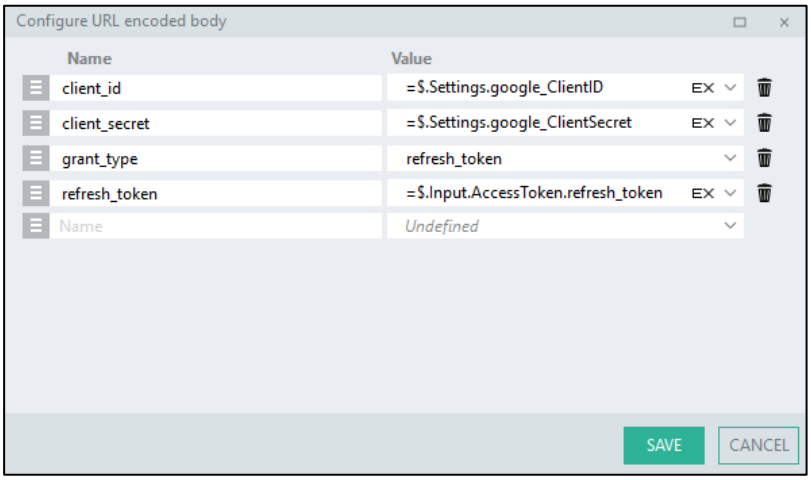
To save you some time, copy the process we just built 'GrantTokens' and rename the copied process to 'RefreshTokens'.

Then, configure the process the process to have an input 'AccessToken' of the CustomType 'AccessToken'



Next, rename the CallRESTWebservice function to 'RefreshTokens' and condifure it as below:

- URL: https://www.googleapis.com/oauth2/v4/token
- Method: POST
- Body format: URL Encoded Content
- URL Encoded body:



- Output: Leave as-is i.e. AccessToken CustomType

We now have a process that will handle the refreshing of access tokens, to test this we need to create a handler process that will call either the GrantTokens or RefreshTokens process depending on a condition.

3. Authentication Handler

We are now going to create an authentication handler process that will determine if the access token exists, if it doesn't then the process will execute the sub-process 'GrantTokens', if it does exist but is about to expire then the sub-process 'RefreshTokens' will be called. The response from either of these processes will then be written to a file/database for later retrieval.

In the provided sample, I have created 2 approaches to retrieving and storing these values, the first one is storing them in a database and the second is storing them in a file, depending on your environment you can choose either one, they only differ on where the data is stored. You will see by comparing the two that they are practically the same.

1. Authentication Handler: File

This process works as follows:

- Check if Access Token file exists on drive
- If exists:
    - Read file contents
    - Check Token expiry
        - If expires soon:
            - Call process RefreshToken
            - Write output to file
- If not exists:
    - Call process GrantToken
    - Write output to file

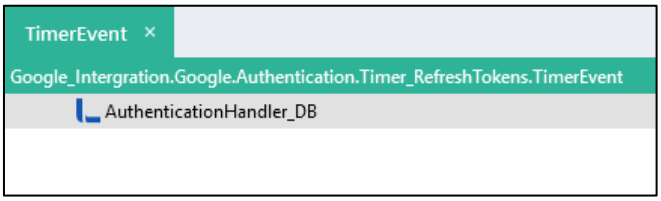
2. Authentication Handler: Database

This process works as follows:

- Retrieve AccessToken object from DB [Sub-process]
- If exists:
    - Check Token expiry
        - If expires soon:
            - Call process RefreshToken
            - Write output to database
- If not exists:
    - Call process GrantToken
    - Write output to database

The above authentication handler process can be added to a Timer service so that the process continually checks if the token is still valid for example every 2 minutes. This will ensure that your access token is always valid, when we will use the access token, we will simply reference a process that will read the database/file and return the current token. This will be discussed in the next section.

For now, add a Timer service that executes every 2 minutes or so, when the event is triggered make a process call to the relevant authentication handler:



Once you have setup the Timer service, deploy your Solution to the Linx Application Server.

Then repeating Step 3.2, update your Auth Code in the Setting values of your solution and start the timer service.

Let the timer service run and if no errors appear, go onto the next section.

If at all this process fails at some point (i.e. details change, timer interval incorrect), repeat step 3.2 up until 3.3.3.

4. Connect to the Google API with Access Token

For this process, we are going to make a simple Webservice request which will retrieve information about our Google Drive instance, this is to demonstrate that the authentication functionality is correctly configured. We are taking the example from the Google Developer Samples.

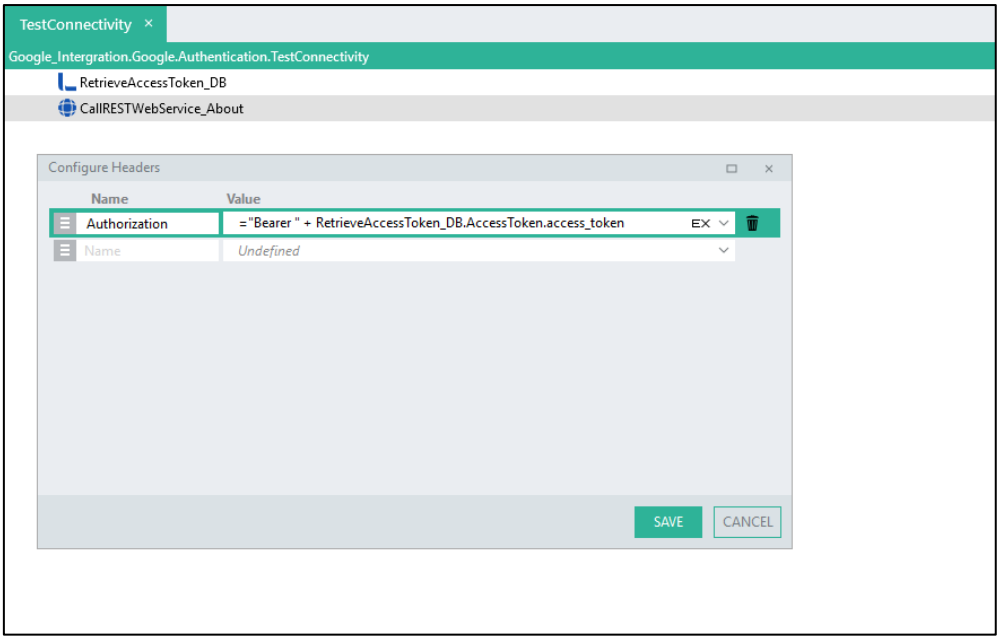
Sample solution process: [Google_Intergration] > [Google] > [Authentication] > [TestConnectivity]

For this process, we are going to make a GET request to this url:
https://www.googleapis.com/drive/v3/about

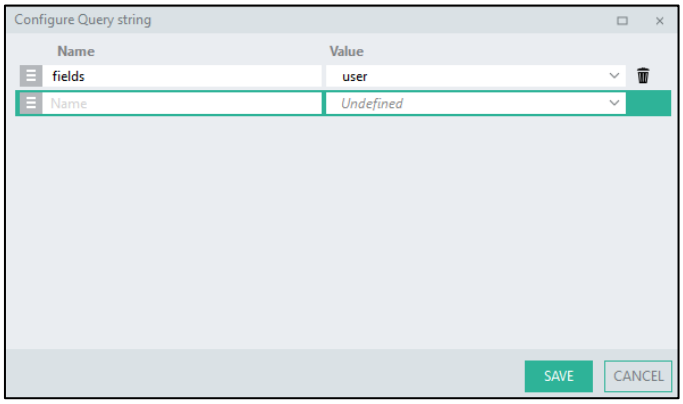
We are going to make use of a CallRESTWebservice function, configure it as follows:

- URL: https://www.googleapis.com/drive/v3/about
- Method: GET
- Query string:
  - fields:user
- Headers:
  - Authorization:"Bearer" + Access Token
- Body format: Text
- Body:
- Output type: String

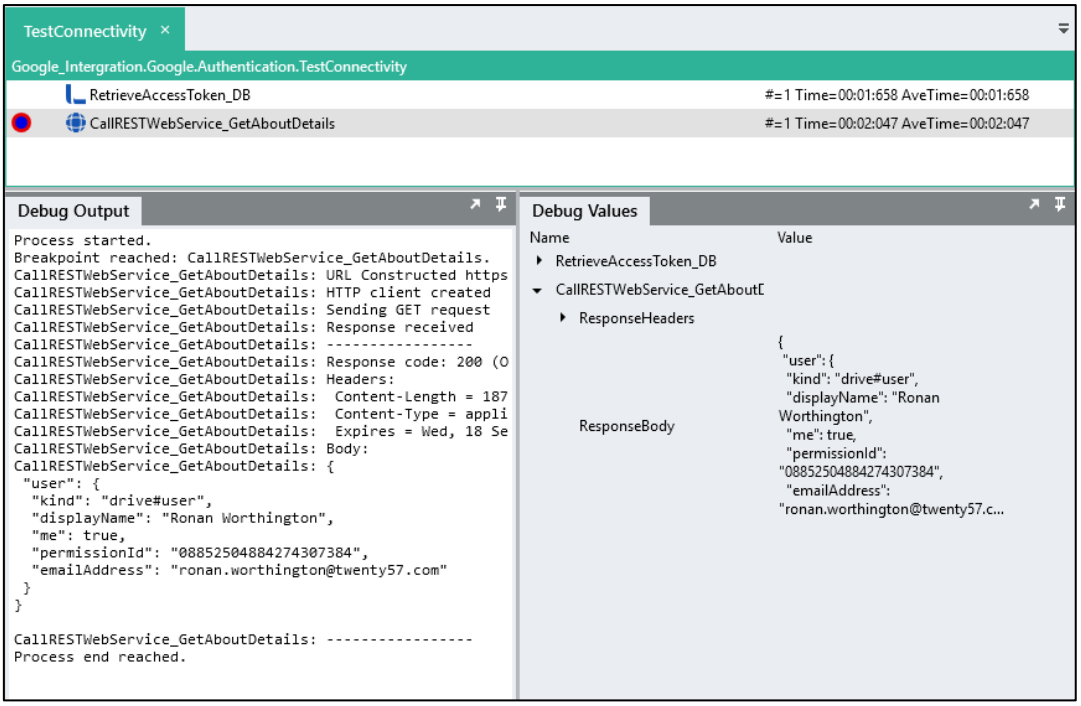
You will notice that we are using a header value, this is often the case with OAuth access, where you pass in the Access Token value as a header value and it authenticates your request. To make the process dynamic, we are going to pass in the Access Token from a sub-process which just returns the AccessToken object (RetrieveAccessToken_DB).



Then, we are also specifying in out Query string that we only want "About" information for the field "user":



If you debug your process an enable logging, when the process runs you should see a successful "200" response with your information.



If we wanted to, we could import this response as a CustomType and handle it better but because this is just to perform a "ping" test we are going to leave it as is.

You should now be fully connected to your Google Drive instance, using what you have learnt above you should be fairly comfortable with handling OAuth authentication in general. In the following sections we'll go through some more practical and real-world samples in which we will explore the Google Drive API functionality in more depth.

Sample Processes

If you take a look at the provided sample solution there are a number of samples or recipes/templates that should give you an idea of how to interact with the Google API using Linx. These processes touch on some of the main API functionality as well extended functionality that can be accomplished with Linx.

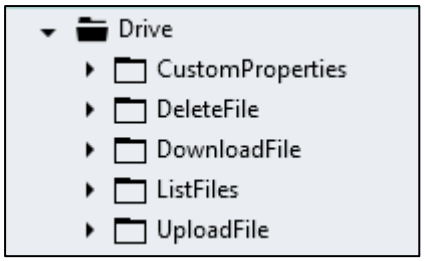
Below is a description of each of the provided sample processes to give you a high-level idea, it is suggested that you tinker and play around with the solution to fully understand the process yourself. Once you've got an idea of the basic structure, you can easily replicate and extend the API functionality.

*Note: These processes cannot be guaranteed to work in production and as such you should do thorough testing before implementing them.

These processes are located in: [Google_Intergration] > [Google] > [Samples]

Google Drive

The below processes demonstrate several areas of API integration which relate to Drive items as a whole, particular Google Sheets integration is explained further on.



Retrieve List of files from Google Drive

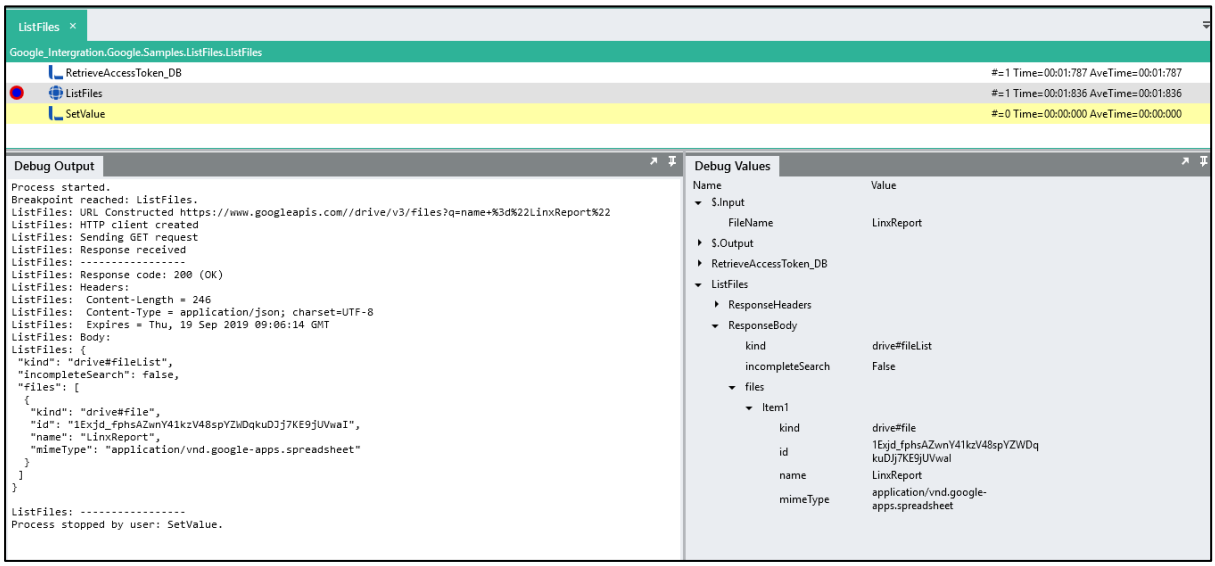
[Google Drive API Documentation]

Sample Process Location: [Google_Intergration] > [Google] > [Samples]>[ListFiles]>[ListFiles]

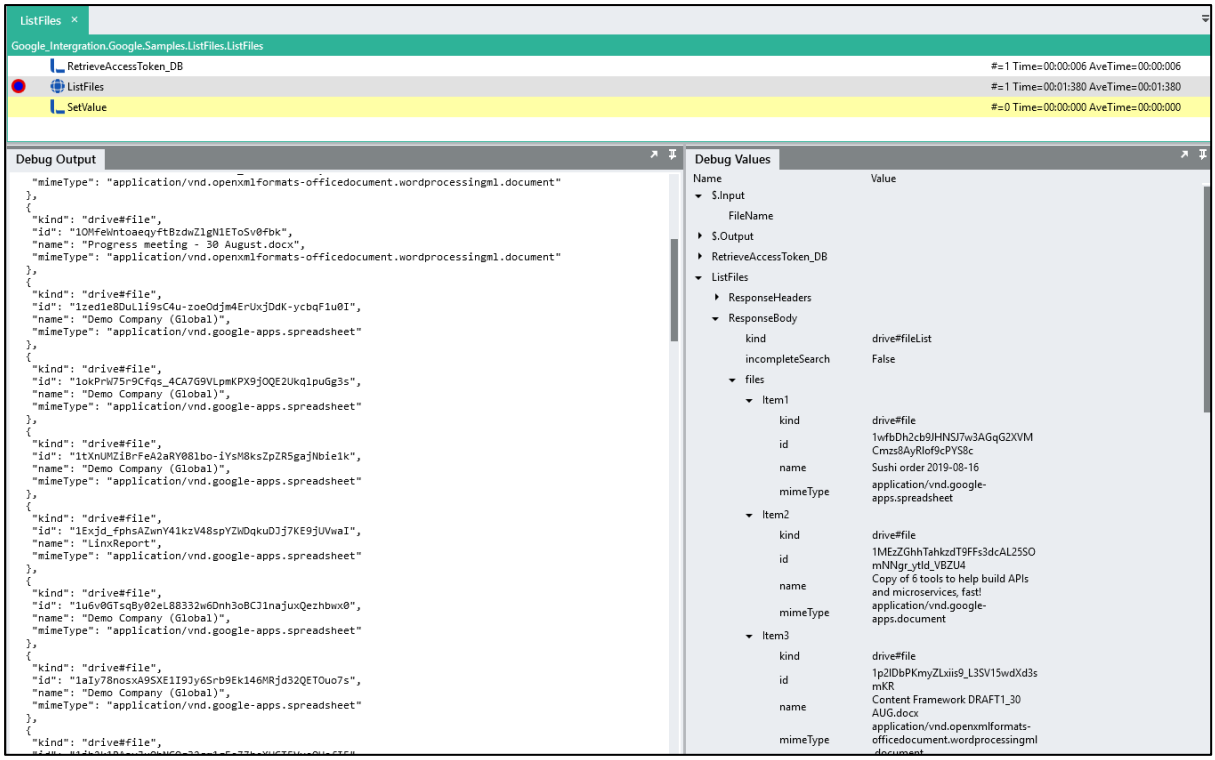
In this process, a webservice GET request is made to the Google Drive API to return a list of all the items on the drive. This will allow you to retrieve the metadata associated with each file as a CustomType object which you can then use in subsequent requests or to store the details somewhere for future use.

For our request, we are indicating what fields we want returned in the 'query string' property to improve performance.

Run process result with file name input parameter ("LinxReport"):



Results for all drive items:


ListFiles
Google_Intergration.Google.Samples.ListFiles.ListFiles
RetrieveAccessToken_DB
#=1 Time=00:00:006 AveTime=00:00:006
ListFiles
#=1 Time=00:01:380 AveTime=00:01:380
SetValue
#=0 Time=00:00:000 AveTime=00:00:000
Debug Output
"mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document"
},
{
"kind": "drive#file",
"id": "1OMfeWntoaeqyftBzdwZlgN1EToSv0fbk",
"name": "Progress meeting - 30 August.docx",
"mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document"
},
{
"kind": "drive#file",
"id": "1zed1e8DuLli9sC4u-zoeOdjm4ErUxjDdK-ycbqF1u0I",
"name": "Demo Company (Global)",
"mimeType": "application/vnd.google-apps.spreadsheet"
},
{
"kind": "drive#file",
"id": "1okPrW75r9Cfqs_4CA7G9VLpmKPX9jOQE2Ukq1puGg3s",
"name": "Demo Company (Global)",
"mimeType": "application/vnd.google-apps.spreadsheet"
},
{
"kind": "drive#file",
"id": "1tXnUMZiBrFeA2aRY081bo-iYsM8ksZpZR5gajNbie1k",
"name": "Demo Company (Global)",
"mimeType": "application/vnd.google-apps.spreadsheet"
},
{
"kind": "drive#file",
"id": "1Exjd_fphsAZwnY41kzV48spYZWDqkuDJj7KE9jUVwaI",
"name": "LinxReport",
"mimeType": "application/vnd.google-apps.spreadsheet"
},
{
"kind": "drive#file",
"id": "1u6v0GTsqBy02eL88332w6Dnh3oBCJ1najuxQezhbwx0",
"name": "Demo Company (Global)",
"mimeType": "application/vnd.google-apps.spreadsheet"
},
{
"kind": "drive#file",
"id": "1aIy78nosxA9SXE1I9Jy6Srb9Ek146MRjd32QETOuo7s",
"name": "Demo Company (Global)",
"mimeType": "application/vnd.google-apps.spreadsheet"
},
{
"kind": "drive#file",
Debug Values
Name
Value
$.Input
FileName
$.Output
RetrieveAccessToken_DB
ListFiles
ResponseHeaders
ResponseBody
kind
drive#fileList
incompleteSearch
False
files
Item1
kind
drive#file
id
1wfbDh2cb9JHNSJ7w3AGqG2XVMCmzs8AyRlof9cPYS8c
name
Sushi order 2019-08-16
mimeType
application/vnd.google-apps.spreadsheet
Item2
kind
drive#file
id
1MEzZGhhTahkzdT9FFs3dcAL25SOmNNgr_ytld_VBZU4
name
Copy of 6 tools to help build APIs and microservices, fast!
mimeType
application/vnd.google-apps.document
Item3
kind
drive#file
id
1p2IDbPKmyZLxiis9_L3SV15wdXd3smKR
name
Content Framework DRAFT1_30 AUG.docx
mimeType
application/vnd.openxmlformats-officedocument.wordprocessingml

Upload files from local drive to Google Drive
Google Drive API Documentation

Sample Processes Location: [Google_Intergration] > [Google] > [Samples]>[Upload]

In this process, a file list is performed on a local file directory, when a file is picked up by Linx, it is then searched for on Google Drive. If the file does not exist then the file is uploaded to Google Drive making use of a resumable upload. If the file does already exist, a further check is done to determine if the file on the local drive has been modified after the file on Google drive. If it is, then the file is updated with the new file contents and metadata using a resumable upload. The resumable upload involves submitting metadata in one request as a Json body, receiving a URL back, and then uploading the file data to the returned URL in a subsequent request with the request body containing a filestream of the file. Once the file has been uploaded to Google drive, the file is then moved from the local directory into a "processed" directory.

Download files from Google Drive to local drive
Google API Documentation:

- Google Docs Export
- Non-Google Doc Download
- File Management

Sample Process Location: [Google_Intergration] > [Google] > [Samples]>[Download]>[DowloadAllFiles]

In this process, files on Google Drive are downloaded onto your local drive. In order to download/export specific files you need to specify the item ID to download it, to get the ID we can use our process created earlier which returns a list of files.

The process is as follows.

First, a list of all the files on your Google Drive instance are retrieved.

Then, for each file returned the sub-process DownloadFileHandler is run.

In DownloadFileHandler, first a check is done on the MIME type of the file, this is to determine what type of item it is i.e. Sheet, Doc, Folder etc.

If the file is a generic Google doc, then the sub-process 'DownloadFile_ExportGoogleFile' is then executed.

DownloadFile_ExportGoogleFile: The sub-process 'ConvertMIMEType' is called which returns the MS Office equivalent of the GDrive MIME type. Then using this MIMEType, a webservice request is made to 'export' the google doc. A conversion is then done on the filename to remove any special characters for the Windows drive. Finally, using a BinaryFileWrite function, the filestream returned from the webservice is written to a file.

If the item is of type Google Folder, a file list is performed on the Google Folder to list its contents. For each item in the folder, the process DownloadFileHandler is called again which will repeat the above process to download the files within the folder.

If the drive item is not a generic google doc i.e. Excel Document, MS Word Doc, then another branch of logic is triggered. First, a check is done on the file size of the item, if the size exceeds the max download size in bytes then a multipart download is performed by calling the sub-process 'DownloadFile_MultipartDownload'. This works by retrieving batches of the file until all the data has been retrieved, as the data is retrieved it is a built up in a byte list. Finally, once all the data has been stored in the byte list, it is written to a file using the BinaryFileWrite function.

If the file size is below the max download threshold, then the sub-process 'DownloadFile_NonGoogleFile ' is called and inside it one request is made that retrieves all the data of the file, then using a the BinaryFileWrite function the file is written to the local drive.

Delete files from Google Drive
Google Drive API Documentation

Sample Process Location: [Google_Intergration] > [Google] > [Samples]>[Download]>[DeleteFile]>[DeleteFile]

In this process, a DELETE webservice request is made with the FileID of the selected file, the file will then be removed from your Google Drive instance.

Add Custom Properties for a File
Google Drive API Documentation

Sample Process Location:

Location: [Samples] > [Drive] > [CustomProperties] > [AddCustomProperties]

Custom properties which can be thought of as drive item metadata is very useful in adding additional custom "tags" to drive items, these could be individual identifiers or groups of files which you can then search for all at once using the process. Custom properties are added in a key:value pair format.

Examples use cases of custom properties:

- Linking custom IDs to drive items in order to track files across systems eg. "CustomID":"363637""
- Linking files by group, eg: "Department":"Treasury"

These can then be used to retrieve groups or specific drive items in a fairy quick and straight forward way.

The process works as follows,

A list of key:value custom property pairs and a FileID are passed in as an input.

A loop is then performed for each of the key, value pairs in the list, for each item, a JSON string is built up that will contain all the custom properties to be submitted.

The JSON string is then "closed".

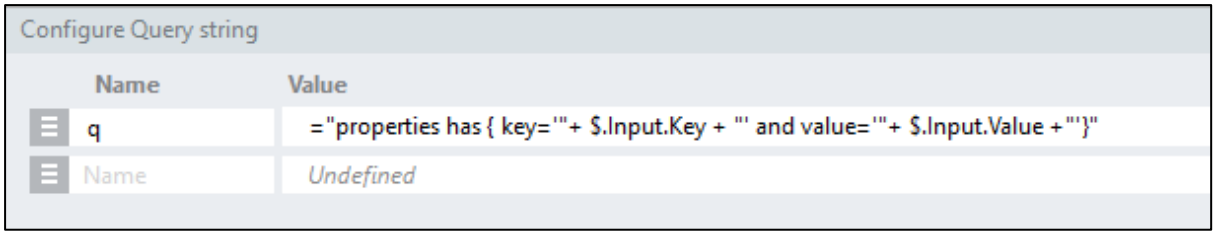
A webservice PATCH request is then made to the endpoint containing the FileID along with the JSON body that was built up and contains all the custom properties for that file.

Upon successful request, the custom properties are linked to the file.

A "real world" example process "Example_LinkCustomIDForFiles" has been created to demonstrate a use case. In this process, all the files from the drive are returned, then a random string is generated (unique ID) and submitted as a custom property for each of the files. Then the DriveID and the UniqueID generated are written to a file to keep track of.

Search for Files by Custom Property
Location: [Samples] > [Drive] > [CustomProperties] > [SearchByCustomProperty]

Now that we can link custom properties to files, we can then use these properties to search for drive items. In this process, a key:value pair are taking in as inputs, then a file list request is made with the key:value pair as query parameters:



Google Sheets

Create a Google Sheet

Google Sheets API Documentation

Sample Process Location: [Google_Intergration] > [Google] > [Samples]>[Sheets]>[Write]>[WriteSheet]

In this process, we are going to submit some metadata to the Google Sheets API from a CustomType which will create a blank sheet on the Drive. We can choose to submit other data like the actual cell values but we will do that with an update in a subsequent process.

The process is as follows:

First, we return the access token by calling the sub-process 'RetrieveAccessToken_DB'.

Then, we initiate a local instance of the 'Spreadsheet' CustomType, with a few of its meta-data properties configured.

Update data values of a Google Sheet

Google Sheets API Documentation

Sample Process Location: [Google_Intergration] > [Google] > [Samples]>[Sheets]>[Create]>[CreateSheet]

In this process, we are going to submit values to the Google Sheets API which will then "write out" the sheet values online. Unlike the previous processes we are not simply uploading a file from the local drive to google drive, in this case we are instructing the Sheets API to create a sheet as well as write out values in certain columns/cells.

As of now, all the processes we have covered relate to the high-level Drive API, that's why we can upload/download different types of files. For this process, its specific to the sheets and that is why the Google Sheets API must be enabled.

To "write" values to a sheet, you need to submit the values that you want in the format you expect them to appear, we do this by submitting a JSON object containing the rows and cells we want to write out.

Create Metadata for a Sheet

Google Sheets API Documentation

This process allows a user to "attach" certain metadata values to a whole spreadsheet (this can be extended down to lower levels such as cells, but this process demonstrates the concept). These values can then be used to identify files or group multiple files with "tags".

In this sample, a spreadsheet with an ID has the metadata tag "testKey":"testValue" associated with it,

Useful Resources:

Google API Console: https://console.developers.google.com

Google Sheet API Samples: https://developers.google.com/sheets/api/samples/

Google Drive API Samples: https://developers.google.com/drive/api/v3/reference/

Google Account Permissions: https://myaccount.google.com/u/0/permissions