IBM Cloud Pak for Business Automation Demos and Labs 2023

Interfacing FileNet Content Platform Engine with GraphQL on Cloud Pak for Business Automation

V 1.3s

Matthias Jung, Ph.D.
matthias.jung@de.ibm.com

NOTICES

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

TRADEMARKS

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a Registered Trade Mark of AXELOS Limited.

ITIL is a Registered Trade Mark of AXELOS Limited.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Table of Contents

1 Introduction

1.1 GraphQL

GraphQL is a specification for a Query Language, which you can read more about it on https://graphql.org/. It was designed to overcome problems with RESTful interfaces. Similar to RESTful interfaces it also uses HTTP. A key advantage of GraphQL is its flexibility to define what information should be contained in the response to a request. Thus, what would be several different requests in traditional interfaces for FileNet Content Platform Engine, can be combined into one to achieve higher efficiency.

Another advantage is, that applications interfacing to GraphQL only require to send and receive HTTP messages and can be written without binding with any specific FileNet Content Platform Engine Client libraries. This results in less version dependencies, and easier upgrades of the environment.

For further reading, the FileNet P8 Platform Documentation contains some sections about developing with the GraphQL API for Content Platform Engine:
https://www.ibm.com/support/knowledgecenter/SSNW2F_5.5.0/com.ibm.p8.graphql.dev.doc/gql_overview.htm.

Furthermore, there is also a white paper on the GraphQL interface, with lots of examples:
https://www.ibm.com/support/pages/sites/default/files/inline-files/$FILE/IBM%20Content%20Services%20GraphQL%20API%20Developer%20Guide_1.pdf.

1.2 Lab Overview

The **exercise "GraphQL Queries"** introduces you to the GraphQL query language in a series of short, easy to understand examples. For this, it uses the Graph iQL component, which supports auto-completion and content sensitive help. The target is not to cover the complete GraphQL query language, but to summarize the required concepts to use the GraphQL FileNet Interface efficiently.

The examples of the first exercise can be performed independently of the lab "Setting up FileNet Content Platform Engine for Automation Projects on Cloud Pak for Business Automation". The following exercises though access documents and folders created on that lab.

The **exercise "Mutations"** introduces the reader to performing changes on the FileNet Content Platform Engine environment through GraphQL. In the examples, a folder will be created, its security settings will be modified, and the folder will also be deleted again.

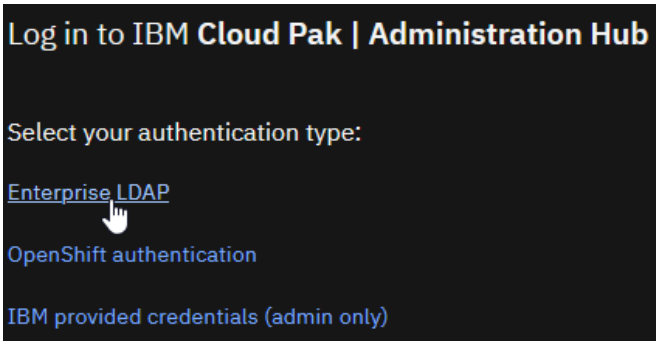
The last **exercise "Parameters"** introduces the possibility to parameterize GraphQL queries, freeing a custom application using the GraphQL interface from making a lot of string manipulations to build up the final queries with the custom information needed in parameters. This way, the GraphQL queries themselves can be assembled in a kind of library, defined in constant strings.

1.3 Lab Setup Instructions

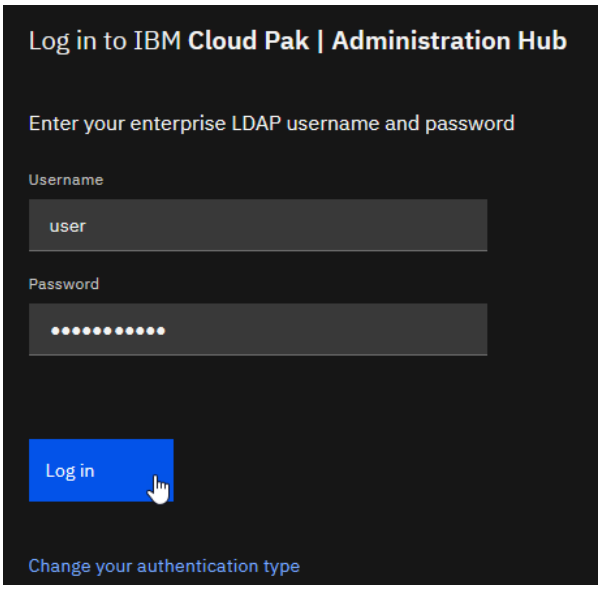
_1. If you are performing this lab as a part of an IBM event, access the document that lists the available systems and URLs along with login instructions. For this lab, you will need to access **Graph IQL**.

_2. Paste the Content Services GraphQL URL to your web browser.

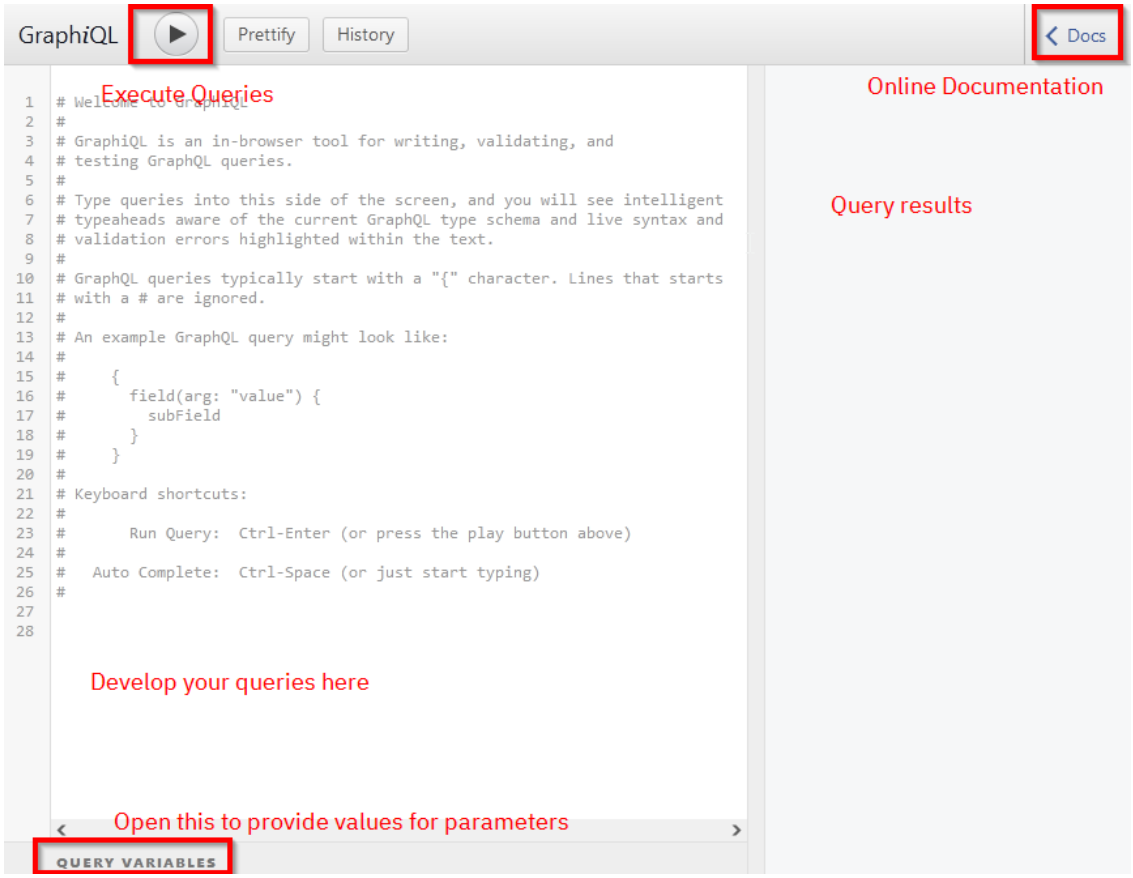
_3. Select Enterprise LDAP login option



_4. Enter the supplied to you *Username* and *Password* and then click **Log in**



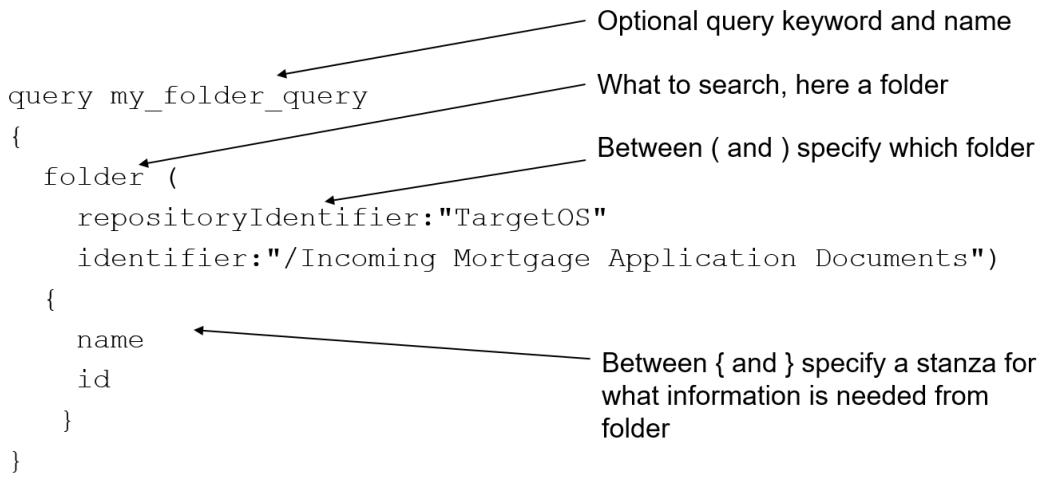
_5. The Graph iQL Webpage should come up, with which you can explore and develop GraphQL queries and mutations.



2 Exercise: GraphQL Queries

2.1 Introduction

The most important request types for GraphQL are “query” and “mutation”. The query obtains data from FileNet Content Engine, and the mutation will change something, e.g. add a new document to the FileNet Content Engine. Query is the default request type, so the keyword “query” can be left out. A sample query can look as follows:




```
query my_folder_query
{
  folder (
    repositoryIdentifier:"TargetOS"
    identifier:"/Incoming Mortgage Application Documents")
  {
    name
    id
   }
}
```

For developing GraphQL requests, the GraphQL component of Cloud Pak for Business Automation allows to enable the GraphiQL web application. It is used in the next exercises of this lab for exploring the FileNet Content Platform Engine GraphQL implementation.

> For security reasons, a production environment would normally not contain the GraphiQL web-application, and that application is by default disabled. For accessing GraphQL there, GraphQL requests can be sent as HTTP requests, the response is sent in JSON format.

When you are working with the GraphiQL interface for some time, it might be that at some moment the session times out. When this is happening, instead of a query result (or an error) you will get a red error message. To renew the session, it is enough to refresh the browser page.

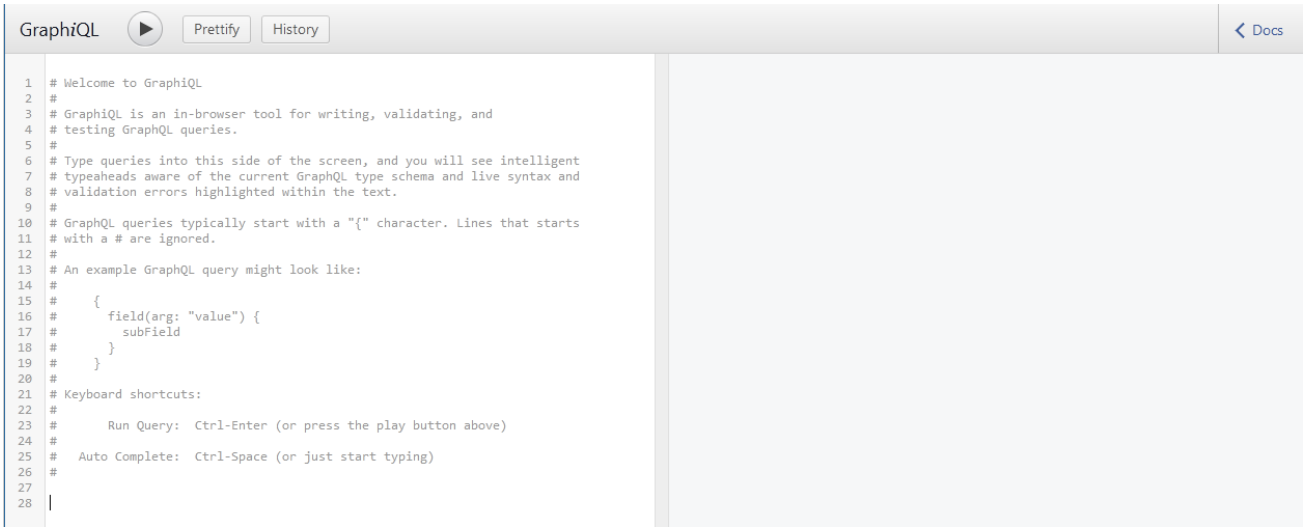
2.2 Exercise Instructions

2.2.1 Domain Queries

The queries of this section can be executed even if the Lab 1 - Content part was not done or not completed.

_1. Open a browser and navigate to GraphiQL, the Development Platform for GraphQL queries. Login using the username and the password which you obtained before.

_2. The left side is for developing a GraphQL request. The editor supports auto-completion using Ctrl + Space. The request can be run by clicking the triangle right from above the editor, or by pressing Ctrl + Enter. Through the “Docs” link in the upper right corner, you can browse through the documentation.



_3. Try entering and executing a first sample request to the query editor:

```
query domainquery {
  domain {
    id name
  }
}
```

As there is only one domain on a FileNet Content Manager environment, the section in parentheses (...) does not need to be specified. The following output should be shown (the IDs, time stamps and user names might be different in this and all further screenshots showing results, actually)


```
{
  "data": {
    "domain": {
      "id": "{41CB7F3B-D826-4C7E-BB7B-16C5B29E1F26}",
      "name": "P8DOMAIN"
    }
  }
}
```


_4. Try searching for the “Domain” type in the online documentation, by placing the mouse over the query keyword “domain” and waiting a couple seconds. The box which opens, informs that the domain directive is of type "Domain", all the types shown in yellow color. Click on the yellow “Domain” to list the Domain type and its properties. It should tell you, that following properties can be used in the stanza for the “domain” query:

_5. Try adding fields to the stanza of your query. Add the “properties” field. When you execute the query, notice that Graph iQL has automatically extended your query such as follows:

```
query domainquery {
  domain {
    id name
    properties {
      id
    }
  }
}
```

The output of this request will show up the ids of all the properties supported by GraphQL below the Domain.

_6. But you might not be interested in obtaining all properties. To narrow down to retrieve specific properties, it is possible to specify which properties are needed. Try editing the request in this way:

```
query domainquery {
  domain {
    id name
    properties(includes: ["SystemUserName"]) {
      id value
    }
  }
}
```

When running this query, the output should show up as follows:


```
{
  "data": {
    "domain": {
      "id": "{41CB7F3B-D826-4C7E-BB7B-16C5B29E1F26}",
      "name": "P8DOMAIN",
      "properties": [
        {
          "id": "SystemUserName",
          "value": "cp4admin"
        }
      ]
    }
  }
}
```


_7. Finally, this request might be extended to include data about the Object Stores of the domain too. Extend the query as follows:

```
query domainquery {
  domain {
    id name
    properties(includes: ["SystemUserName"]) {
      id value
    }
    objectStores {
      objectStores {
        id symbolicName
      }
    }
  }
}
```

This should show information about all the object stores defined in the Content Platform Engine environment.

_8. How would a request look like which would print the name and value of the "DomainType" property as well?

2.2.2 Folder Queries

The queries in this, and the following subsection, depend on having completed the **Setting up FileNet Content Manager for Automation Projects on Cloud Pak for Business Automation** lab.

_1. If the browser window with Graph iQL had been closed or if you are (re-)starting here, open a browser and navigate to Graph IQL, the development platform for GraphQL queries. Login using the username and the password which you obtained.

_2. Querying for other data, for example folders or documents, it is required to change “domain” in the query to a different identifier, for example “folder”. A query for folder requires the section between parentheses “(“ … “)” for specifying exactly which folder to operate on. It is suggested to leave them out at first and review the error messages.

```
query folderquery {
  folder
  {
    name
    id
  }
}
```

Above query will result in an empty data field, and the following two error messages, inside a JSON structure:

a) Validation error of type MissingFieldArgument: Missing field argument **identifier** @ 'folder'
b) Validation error of type MissingFieldArgument: Missing field argument **repositoryIdentifier** @ 'folder'

The “repositoryIdentifier” needs to be set to the symbolic name of the Object Store, in this case “CLOS” without a space and “identifier” needs to be set to the document ID or the absolute folder path of the specific folder to query, inside the object store. The root folder would be "/".

_3. Try adding the required parameters.

```
query folderquery {
  folder (
    repositoryIdentifier:"CONTENT"
    identifier:"/"
  )
  {
    name
    id
  }
}
```

_4. Adding the list of subfolders is a bit more complex, as the "subfolders" field is of type FolderSet. FolderSet objects have a single field, which is named "folders" and expands to all folders contained in the FolderSet. For each of these folders, you can query the id and the pathName for example. The pathName lists the complete path as would be suitable for the content of the identifier in a following call.

```
query folderquery {
  folder (
    repositoryIdentifier:"CONTENT"
    identifier:"/"
  )
  {
    name id
    subFolders {
      folders {
        id pathName
      }
    }
  }
}
```

The result of course only shows those folders, to which the logged on user has at least view-access. So we get something like this:


```
{
  "data": {
    "folder": {
      "name": "",
      "id": "{0F1E2D3C-4B5A-6978-8796-A5B4C3D2E1F0}",
      "subFolders": {
        "folders": [
          {
            "id": "{85DDC780-0000-C1AF-9A9F-1A4B685B8AE3}",
            "pathName": "/Case Folders"
          },
          {
            "id": "{85CD6E30-0000-C7FC-A927-8B6566EA7E8A}",
            "pathName": "/ClbRoles"
          },
          {
            "id": "{85CD6E30-0000-C4AB-A72E-C2082E3DB0B9}",
            "pathName": "/ClbTeamspace Templates"
          },
          {
            "id": "{85CD6E20-0000-CA42-90AD-53BB9817C9BA}",
            "pathName": "/ClbTeamspaces"
          },
          {
            "id": "{85CD6F30-0000-C154-A8FA-1E735A22E01B}",
            "pathName": "/CodeModuleEmitter"
          },
          {
            "id": "{85CD70E0-0000-C6F8-B03F-8BE6C61099FC}",
            "pathName": "/Focus Corp"
          },
          {
            "id": "{A686C424-8DC0-46A2-893D-1724DCE69678}",
            "pathName": "/Saved Searches"
          },
          {
            "id": "{85DFA4C0-0000-C1D0-A983-7D916C183DA7}",
            "pathName": "/Workflows"
          },
          {
            "id": "{D786F754-4962-4F8C-89D3-3F83BD3FA27B}",
            "pathName": "/usr1 Client Onboarding"
          }
        ]
      }
    }
  }
}
```


_5. Listing the contained documents works very similar. The field in the folder to add is the "containedDocuments" field, which is of type DocumentSet. Its "documents" field expands to the array of documents contained in the folder. In the query below, you need to use the identifier according to your username (copy the result from the query above).

```
query folderquery {
  folder (
    repositoryIdentifier:"CONTENT"
    identifier:"/usrxxx Client Onboarding"
  )
  {
    name id
    containedDocuments {
      documents {
        id name creator dateCreated
        majorVersionNumber minorVersionNumber
        mimeType
        contentElements {
          contentType
          elementSequenceNumber
          ... on ContentTransfer {
            contentSize
            retrievalName
            downloadUrl
          }
        }
      }
```

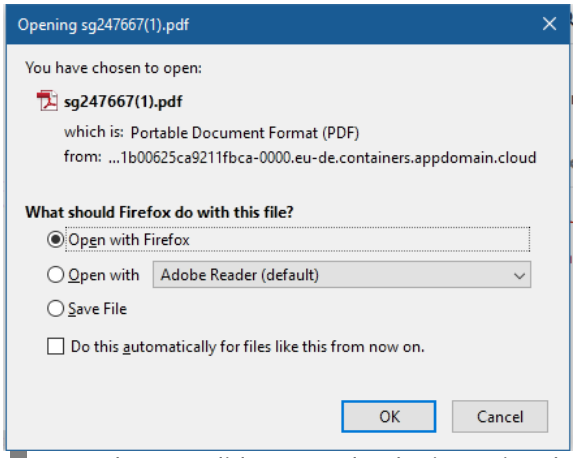
```
      }
    }
}
```

For understanding the magic behind this query, you need to refer to the online documentation. The contentElements field is of type ContentElements, which implements the interface "ContentTransferType", which in turn implements the type "ContentTransfer". So by using the "..." notation it is possible to access fields of one of the other implemented types. The following is returned:


```
{
  "data": {
    "folder": {
      "name": "usr1 Client Onboarding",
      "id": "{D786F754-4962-4F8C-89D3-3F83BD3FA27B}",
      "containedDocuments": {
        "documents": [
          {
            "id": "{85DE77C0-0000-C11E-AD63-F6664991ED1F}",
            "name": "IBM FileNet P8 Platform and Architecture",
            "creator": "user1",
            "dateCreated": "2023-01-23T11:51:10.672Z",
            "majorVersionNumber": 1,
            "minorVersionNumber": 0,
            "mimeType": "application/pdf",
            "contentElements": [
              {
                "contentType": "application/pdf",
                "elementSequenceNumber": 0,
                "contentSize": 5699249,
                "retrievalName": "sg247667.pdf",
                "downloadUrl": "/content?repositoryIdentifier=CONTENT&documentId=
{85DE77C0-0000-C11E-AD63-F6664991ED1F}&elementSequenceNumber=0"
              }
            ]
          },
          {
            "id": "{85DE6770-0000-C1CC-AF27-F8D18235EB89}",
            "name": "IBM FileNet P8 Platform and Architecture",
            "creator": "user1",
            "dateCreated": "2023-01-23T11:33:26.115Z",
            "majorVersionNumber": 1,
            "minorVersionNumber": 0,
            "mimeType": "application/pdf",
            "contentElements": [
              {
                "contentType": "application/pdf",
                "elementSequenceNumber": 0,
                "contentSize": 5699249,
                "retrievalName": "sg247667.pdf",
                "downloadUrl": "/content?repositoryIdentifier=CONTENT&documentId=
{85DE6770-0000-C1CC-AF27-F8D18235EB89}&elementSequenceNumber=0"
              }
            ]
          },
          {
            "id": "{85E495B0-0000-C161-97D2-C089D99DF771}",
            "name": "TEST1 Bank Information",
            "creator": "user1",
            "dateCreated": "2023-01-24T16:21:37.839Z",
            "majorVersionNumber": 1,
            "minorVersionNumber": 0,
            "mimeType": "application/pdf",
            "contentElements": [
              {
                "contentType": "application/pdf",
                "elementSequenceNumber": 0,
                "contentSize": 5699249,
                "retrievalName": "sg247667.pdf",
                "downloadUrl": "/content?repositoryIdentifier=CONTENT&documentId=
{85E495B0-0000-C161-97D2-C089D99DF771}&elementSequenceNumber=0"
              }
            ]
          }
        ]
      }
    }
  }
}
```


_6. For constructing the download URL, open a text editor (e.g. notepad if you are working on a Windows machine) and copy the GraphQL URL address into it. Search for the first question mark and remove the question mark and everything following it from the URL in the text editor. Then copy one of the downloadUrl values, and append it to the addresss in the text editor, giving the completed download URL. Copy & paste that URL into a new tab of your browser.

_7. If you did that correctly, you should get the document offered for download, e.g.



Note that you did not need to login again. The authentication done earlier will be reused on the new browser tab. Optionally, you can try the same but with using a private browser window, then you would need to login again.

2.2.3 Other Queries

_1. The singular form of "folder" will search for information on a specific folder. The plural form "folders" allows to conduct a query for folders in an Object Store. In the below simple form of the query, just a different folder class name is given for the search, but a where clause can also be added.

```
query foldersquery {
  folders (
    repositoryIdentifier:"CONTENT"
    from: "SWAT_JAM_Case_Folder"
  )
  {
    folders {
      id pathName
    }
  }
}
```

_2. With the query on “documents” you can search for documents. Using this kind of query, you can also perform more complex searches, like for example queries using CBR clauses. This specific query will list the folders where the documents are filed, along with the document ids.

```
query documentsquery {
  documents (
    repositoryIdentifier:"CONTENT"
    from: "Document d INNER JOIN ContentSearch c ON d.This = c.QueriedObject"
    where: "CONTAINS(d.*, 'CBR and CSS')"
  )
  {
    documents {
      id foldersFiledIn {
        folders {
          pathName
        }
      }
    }
  }
}
```

_3. You can actually also combine multiple queries into a single GraphQL request, to further reduce the number of roundtrips of your application to the Content Engine Server, e.g. to increase the overall performance, especially on slow networks, or with a high latency. This below query gets information from the domain, and also on the id of the root folder of the “CLOS” repository

```
query multiquery {
  domain {
    name
```

```
    objectStores {
      objectStores {
        displayName
      }
    }
  }

  folder (
    repositoryIdentifier: "CONTENT"
    identifier: "/"
  ) {
    id
  }
}
```

_4. Observe that in the output JSON data, the query keyword is used to store the result data.

_5. Try to use same query keyword twice and observe that an error is logged. The error occurs, since “folder” cannot appear twice in the JSON output under the “data” entry.

```
query multiquery2 {
  folder (
    repositoryIdentifier: "CONTENT"
    identifier: "/"
  ) {
    id
  }
  folder (
    repositoryIdentifier: "CONTENT"
    identifier: "/Case Folders"
  ) {
    id
  }
}
```

_6. To fix it, make use of alias clauses, as follows, also to make the result a lot more self-explanatory.

```
query multiquery {
  rootfolder: folder (
    repositoryIdentifier: "CONTENT"
    identifier: "/"
  ) {
    id
  }
  casefolders: folder (
    repositoryIdentifier: "CONTENT"
    identifier: "/Case Folders"
  ) {
    id
  }
```

```
}
```

3 Mutations

3.1 Introduction

Mutations are very similar to queries. They are using the keyword "mutation" at the beginning and are otherwise using the same syntax like a query.

> In the mutation, the first section between "(" and ")" after the operation name "e.g. “createFolder" does not only contain information to identify where to apply the operation, but also contains values for any new object to create, or values for any changed information, e.g. the new folder name.
>
> The part between "{" and "}" has the same purpose as before, it specifies exactly what information should be provided on the result of the request.

3.2 Exercise Instructions

_1. In order to change information, create, change or delete folders and documents with GraphQL mutations need to be used. The below mutation will create a folder below the usrxxx Client Documents folder, as an example (twice replace usrxxx with your username). Notice how the properties of the new folder are given in the parenthesis part after the createFolder operation identifier for the mutation.
The result shows how to query additionally the permissions of an object, in this case of the created folder. Again, the permission objects implement an interface, in this case the AccessPermissionType, and the code uses the ... notation to "cast" the permission type.

```
mutation createfolder {
  createFolder (
    repositoryIdentifier: "CONTENT"
    classIdentifier: "Folder"
    folderProperties: {
      parent: {
        identifier: "/usrxxx Client Onboarding"
      }
      name: "usrxxx GraphQL Folder"
    })
    {
      id pathName
      permissions {
        permissionSource
        inheritableDepth
        ... on AccessPermissionType {
          granteeName
          granteeType
          accessMask
        }
      }
    }
}
```

The security shows four entries on the permissions list. They are depicted below. You can quickly login to ACCE and verify that the same information is shown there:


```
{
  "data": {
    "createFolder": {
      "id": "{85E4BED0-0000-C132-879B-2AE963D2D7CA}",
      "pathName": "/usr1 Client Onboarding/usr1 GraphQL Folder",
      "permissions": [
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "#AUTHENTICATED-USERS",
          "granteeType": "GROUP",
          "accessMask": 131073
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "uid=cp4admin,dc=example,dc=org",
          "granteeType": "USER",
          "accessMask": 999415
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "uid=user1,dc=example,dc=org",
          "granteeType": "USER",
          "accessMask": 999415
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "cn=ce_environmentowners,dc=example,dc=org",
          "granteeType": "GROUP",
          "accessMask": 131073
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "cn=p8administrators,dc=example,dc=org",
          "granteeType": "GROUP",
          "accessMask": 131073
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "cn=generalusers,dc=example,dc=org",
          "granteeType": "GROUP",
          "accessMask": 999415
        }
      ]
    }
  }
}
```


_2. The permissions can also be updated, by replacing the whole set of permissions with a new set. The below update copies the permissions from the result above, just leaving out the access entry for the group "GeneralUsers" and the group “#AUTHENTICATED-USERS”. This way, the folder is again only visible for your user account. Remember to update your username below on three places, otherwise your folder might not be accessible for you anymore after running the mutation.
The update again displays the new security, so that you can verify success of the modification.

```
mutation updateFolder {
  updateFolder(
    repositoryIdentifier: "CONTENT"
    identifier: "/usrxxx Client Onboarding/usrxxx GraphQL Folder"
    folderProperties: {
      permissions: {
        replace: [
          { type:ACCESS_PERMISSION inheritableDepth:OBJECT_ONLY
            accessMask:999415 subAccessPermission:{
              accessType:ALLOW granteeName:"uid=cp4admin,dc=example,dc=org" }},
          { type:ACCESS_PERMISSION inheritableDepth:OBJECT_ONLY
            accessMask:999415 subAccessPermission:{
              accessType:ALLOW granteeName:"uid=usrxxx,dc=example,dc=org" }},
          { type:ACCESS_PERMISSION inheritableDepth:OBJECT_ONLY
            accessMask:999415 subAccessPermission:{
              accessType:ALLOW  granteeName:"cn=ce_environmentowners,dc=example,dc=org" }},
          { type:ACCESS_PERMISSION inheritableDepth:OBJECT_ONLY
            accessMask:999415 subAccessPermission:{
              accessType:ALLOW  granteeName:"cn=p8administrators,dc=example,dc=org" }}
        ]
  }))
```

```
  {
    id pathName permissions {
      permissionSource inheritableDepth
      ... on AccessPermissionType { granteeName granteeType accessMask
      }
    }
  }
}
```

The output should be very similar to the one above, just without the two groups.

_3. The folder can be deleted again by using the "deleteFolder" operation on the mutation, as follows (twice replacing userxxx with your login):

```
mutation deletefolder {
  deleteFolder (
    repositoryIdentifier: "CONTENT"
    identifier: "/usrxxx Client Onboarding/usrxxx GraphQL Folder"
  ) {
     id
  }
}
```

4 Parameters

4.1 Introduction

Without being able to use parameters, an application wanting to use GraphQL to access FileNet Content Platform Engine would require doing many string manipulations. Imagine performing the queries in the prior chapters for example in a Java program. As values in the queries need to be mixed with GraphQL directives and keywords, the application would need to assemble the final query in a rather complex manner.

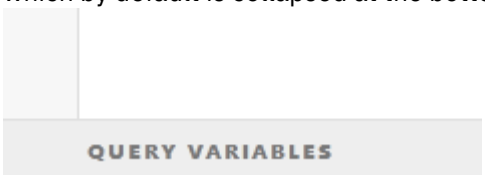
Gladly, in FileNet Content Platform Engine version 5.5.6 support for parameters was introduced. This way, the GraphQL queries can be developed and can be defined in a custom application as constant strings. Only the parameters need to be supplied, and they are using JSON syntax.

The examples in this chapter will create the folder from the last chapter again, but this time the mutations will be parameterized.

In the documentation, you find the description for the query parameters in this section: https://www.ibm.com/docs/en/filenet-p8-platform/5.5.x?topic=mutations-v556-later-graphql-variables.

4.2 Exercise Instructions

_1. In the graphical user interface, the parameter values can be provided in the QUERY VARIABLES pane, which by default is collapsed at the bottom of the screen:



Find the section and click on it to expand it and allow values to be provided.

_2. For the syntax of the parameter type definitions, it is recommended to review how the online documentation is writing the type and use the same notation. For a parameter holding the name of the Object Store to work on, it would for example be denoted as "String!" similarly to following screenshot from the online documentation:

_3. The below example contains a parameterized general-purpose GraphQL mutation for creating a folder, and asking about its security settings in the result data set. Copy it to the GraphiQL entry window (not the parameters section), but don't try to execute it yet.

```
mutation createfolder($theRepo: String!, $parentFolderId: String!, $newFolderName: String!
) {
  createFolder (
    repositoryIdentifier: $theRepo
    classIdentifier: "Folder"
    folderProperties: {
      parent: { identifier: $parentFolderId }
      name: $newFolderName
    })
    {
      id pathName
      permissions {
        permissionSource
        inheritableDepth
        ... on AccessPermissionType {
          granteeName
          granteeType
          accessMask
        }
      }
    }
}
```

_4. In the unfolded QUERY VARIABLES pane, type the opening "{" character to define the json object with the parameter values. A menu appears with the three parameter names. So, it is pretty straightforward to provide the required data.
In the data below again substitute the usernames (twice), otherwise the parent folder ca not be found.

```
{"theRepo": "CONTENT",
"parentFolderId": "/usrxxx Client Onboarding",
"newFolderName": "usrxxx GraphQL Folder"}
```

Execute the query to create the folder again.

_5. The next example shows that also more complex data structures, not only strings can be provided as parameters. In this case a new permission set is passed as a parameter for a request to update the folder security. In the online documentation the permissions to be provided are documented to be having this type:

replace: [BasePermissionInput!]!

Consequently define the mutation as follows:

```
mutation updateFolder(
  $theRepo: String!,
  $folderId: String!,
  $permissions: [BasePermissionInput!]!) {
  updateFolder(
    repositoryIdentifier: $theRepo identifier: $folderId
    folderProperties: {
      permissions: {
        replace: $permissions
  }})
  {
    id pathName permissions {
      permissionSource inheritableDepth
      ... on AccessPermissionType { granteeName granteeType accessMask
      }
    }
  }
}
```

_6. To provide the json parameter values, auto-completion is again of great assistance. Provide the following, and replace usrxxx with your username again three times:

```
{
  "theRepo": "CONTENT",
  "folderId": "/usrxxx Client Onboarding/usrxxx GraphQL Folder",
  "permissions": [{
      "type": "ACCESS_PERMISSION",
      "inheritableDepth": "OBJECT_ONLY",
      "accessMask": 999415,
      "subAccessPermission": {"accessType": "ALLOW", "granteeName":
"uid=cp4admin,dc=example,dc=org" }
    },{
      "type": "ACCESS_PERMISSION",
      "inheritableDepth": "OBJECT_ONLY",
      "accessMask": 999415,
      "subAccessPermission": {"accessType": "ALLOW", "granteeName":
"uid=usrxxx,dc=example,dc=org" }
    }, {
      "type": "ACCESS_PERMISSION",
      "inheritableDepth": "OBJECT_ONLY",
      "accessMask": 999415,
      "subAccessPermission": {"accessType": "ALLOW", "granteeName":
"cn=ce_environmentowners,dc=example,dc=org" }
    }, {
      "type": "ACCESS_PERMISSION",
      "inheritableDepth": "OBJECT_ONLY",
      "accessMask": 999415,
      "subAccessPermission": {"accessType": "ALLOW", "granteeName":
"cn=p8administrators,dc=example,dc=org" }
    }
  ]
}
```

_7. The parameterized version for the deletion of the GraphQL folder is again left open as an exercise to the reader. If you are performing this exercise as part of a SWAT JAM event, you can post the completed query on the slack channel for the event.

Congratulations you have successfully completed the lab “Interfacing FileNet Content Platform Engine with GraphQL on Cloud Pak for Business Automation”!

Appendix A. Solutions to the Questions

_1. A query for listing the name and value for the DomainType property as well, for the Domain Query would just add the second value on the list of property names to extract the value of, e.g.

```
query domainquery {
  domain {
    id name
    properties(includes: ["SystemUserName", "DomainType"]) {
      id value
    }
    objectStores {
      objectStores {
        id symbolicName
      }
    }
  }
}
```

_2. The parameterized form for deleting a folder can be derived from the non-parameterized one given as an example earlier. Here is the one without parameters first, with the values underlines which should be parameterized:

```
mutation deletefolder {
  deleteFolder (
    repositoryIdentifier: "CONTENT"
    identifier: "/usrxxx Client Onboarding/usrxxx GraphQL Folder"
  ) {
     id
  }
}
```

As both are strings, the parameterized version would consequently be:

```
mutation deletefolder($theRepo: String!, $folderID: String!) {
  deleteFolder(
    repositoryIdentifier: $theRepo
    identifier: $folderID) {
    id
  }
}
```

using the following QUERY VARIABLES (replace username twice):

```
{
  "theRepo": "CONTENT",
  "folderID": "/usrxxx Client Onboarding/usrxxx GraphQL Folder"
}
```