IBM Cloud Pak for Business Automation Demos and Labs 2025

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

V 1.1s (for CP4BA 25.0.0)

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 at https://graphql.org/. It was designed to overcome problems with RESTful interfaces. Like 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 GraphQL request 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 an environment using GraphQL to access the FileNet Content Engine repositories.

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/docs/en/filenet-p8-platform/5.7.0?topic=development-overview.

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

Since a while, IBM now also publishes the Content Services GraphQL schema. It is available in the Content Platform Engine Samples repository on Github through the URL https://github.com/ibm-ecm/ibm-content-platform-engine-samples/tree/master/CS-GraphQL-Schema. The schema can be used for example with interface generators, which are publicly available for TypeScript, React, Angular, as well as Java Spring Boot applications, for example. Such interface generators aren't required for using GraphQL – GraphQL requires only HTTP for being used – but they can make the life of a developer a bit easier.

1.2 Lab Overview

The GraphQL lab can be performed independently of the lab "Setting up FileNet Content Platform Engine for Automation Projects on Cloud Pak for Business Automation".

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 GraphiQL component, which supports auto-completion and content sensitive help. The target is not to cover the complete GraphQL query language, but to summarize the key concepts to use the GraphQL FileNet Interface efficiently.

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 be deleted again.

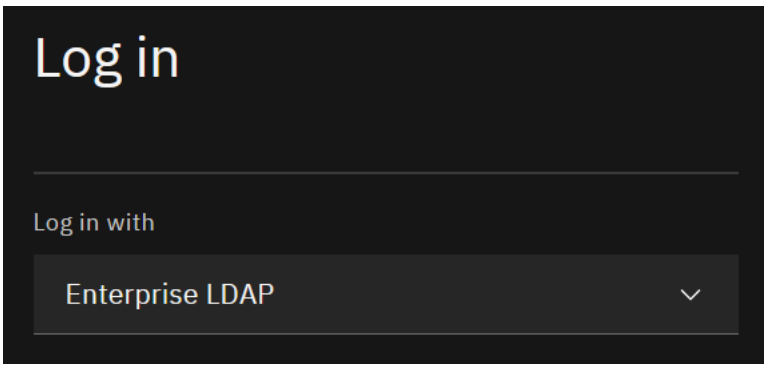
The last **exercise "Parameters"** introduces the possibility to parameterize GraphQL queries, freeing a custom application using the GraphQL interface from making a lot of string manipulations to build up the final queries with the custom information required. This way, the GraphQL queries themselves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

1.3 Lab Setup Instructions

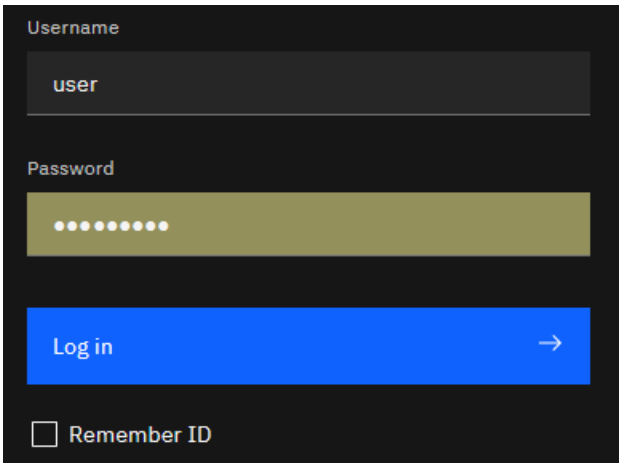
_1. If you are performing this lab as a part of an IBM event, access the document that lists the available systems and URLs along with login instructions. For this lab, you will need to access **GraphiQL**.

_2. Paste the Content Services GraphiQL URL into your web browser.

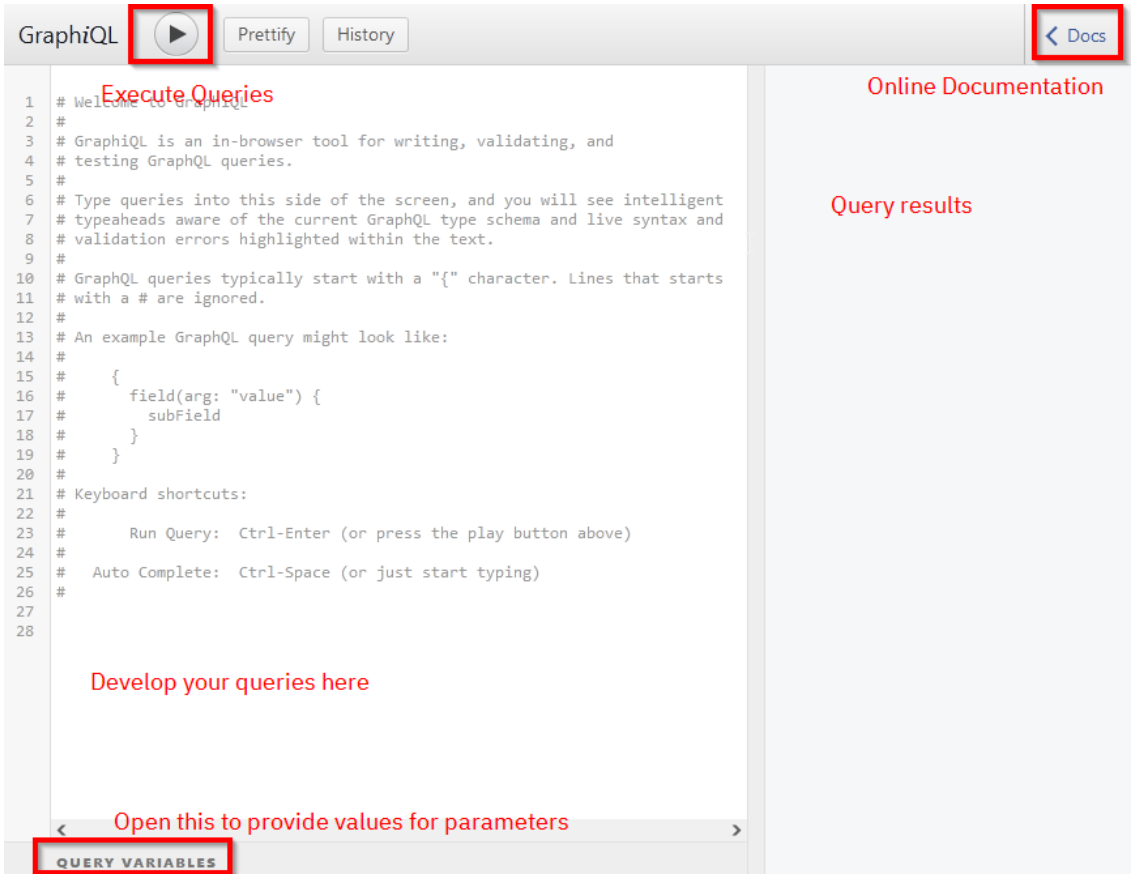
_3. Select Enterprise LDAP login option



_4. Enter the *Username* and *Password* which were supplied to you, click **Log in**.



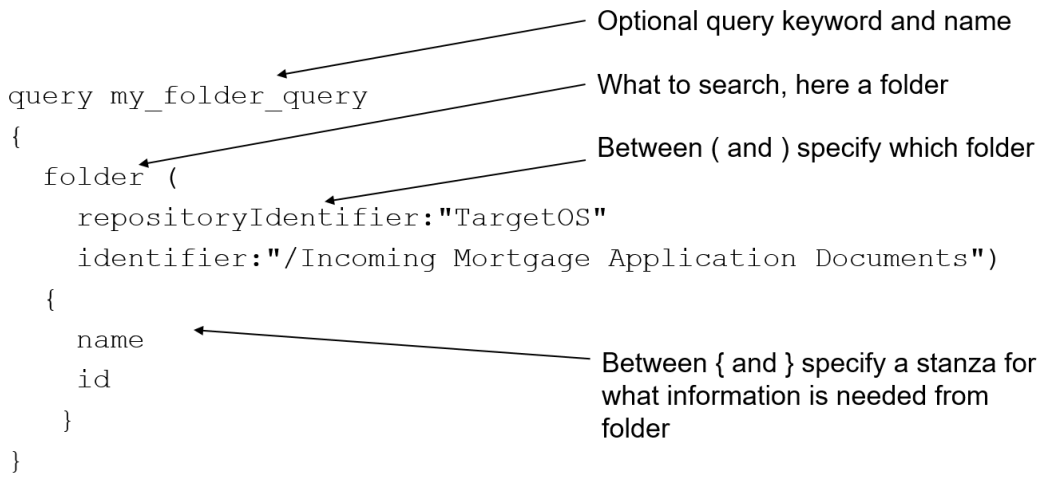
_5. The GraphiQL web page should come up, with which you can explore and develop GraphQL queries and mutations.



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 omitted, and the query name is also optional. A sample query can look as follows:



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.

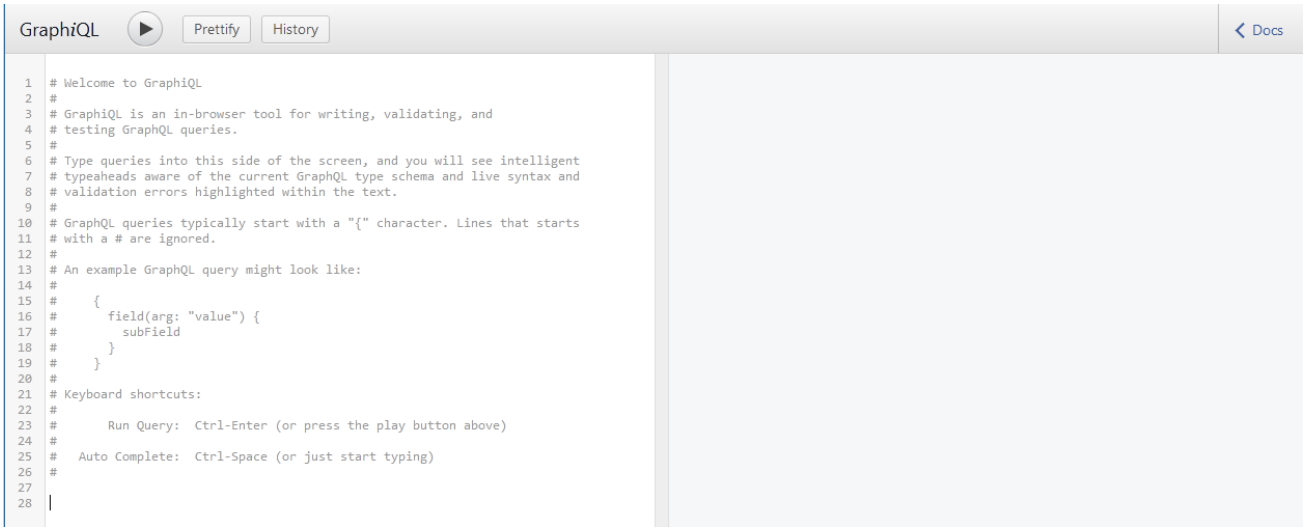
> For security reasons, a production environment would normally not contain the GraphiQL web-application, and that application is by default disabled. For accessing GraphQL there, GraphQL requests can be sent as HTTP requests, the response is sent in JSON format.

When you are working with the GraphiQL interface for some time, it might be that at some point 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

_1. 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.



_2. Try **entering** and **executing** a first sample request to the query editor, you can remove the comments from the first lines, if you like:

```
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 usernames may be different in below screenshot and all further screenshots showing the result of a query).

_3. 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:

_4. Try adding fields to the stanza of your query. **Add** the **properties** field. When you execute the query, notice that GraphiQL has automatically extended your query such as follows:

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

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

_5. 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 (the username shown as value may differ depending on your environment):

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

_6. 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 in addition show information about all the object stores defined in the Content Platform Engine environment.

_7. As an **exercise** create a **GraphQL query**, that **prints the name and value of the "DomainType" property** in addition. You'll find the solution in the appendix.

2.2.2 Folder Queries

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

_2. For 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 further information between parentheses “(“ … “)” for specifying exactly which folder to operate on. Start with leaving them out at first and reviewing 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 “CONTENT” 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. 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 only shows those folders, to which the logged-on user has at least view-access. You will get a result similar to this:


```
{
  "data": {
    "folder": {
      "name": "",
      "id": "{0F1E2D3C-4B5A-6978-8796-A5B4C3D2E1F0}",
      "subFolders": {
        "folders": [
          {
            "id": "{106C6989-0000-CC10-89C2-C7C63132C6C0}",
            "pathName": "/Case Folders"
          },
          {
            "id": "{00237489-0000-C216-8E82-6006C432D0B0}",
            "pathName": "/SOLUTION Client Onboarding"
          },
          {
            "id": "{C378CA31-C354-4528-8A27-C77E4373286E}",
            "pathName": "/Saved Searches"
          },
          {
            "id": "{FD752EA1-76CA-48EB-86C8-CAED1A2A05FA}",
            "pathName": "/usr001 Client Onboarding"
          }
        ]
      }
    }
  }
}
```


_5. Listing the contained documents works very similar. The **field** in the **folder**, which lists the documents contained in the folder is named **containedDocuments** field, you can see it in the online documentation when listing the parameters of the “Folder” type. It is of type **DocumentSet**, and its **documents** field expands to the array of documents contained in the folder.

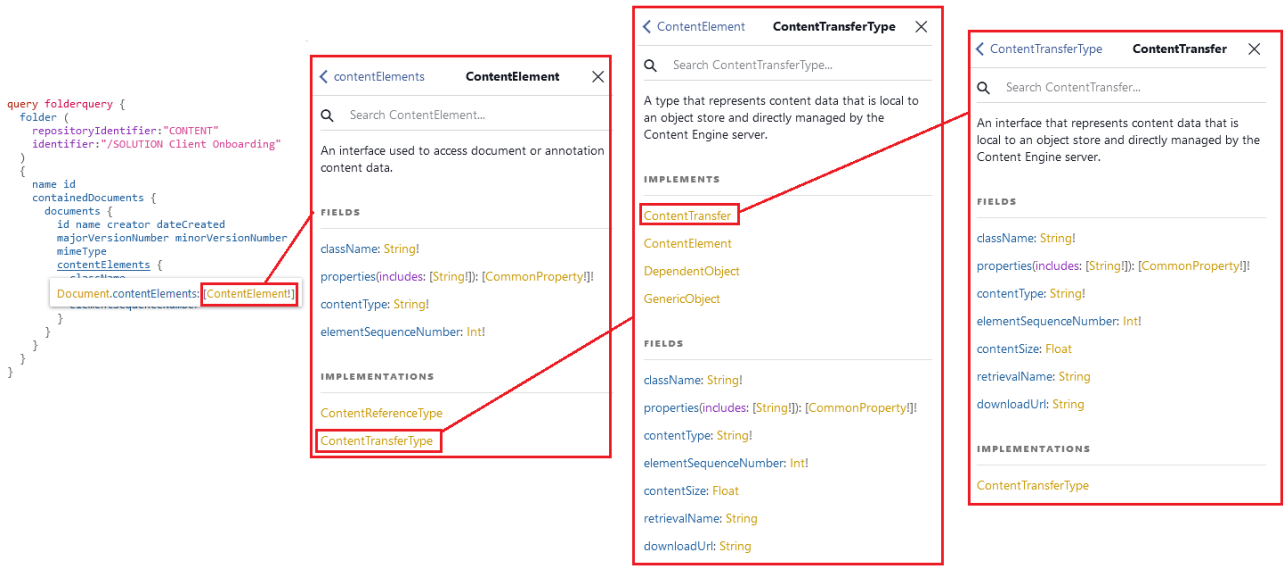
Each of the documents again has potentially many **content elements**, to which **access** is possible via the **contentElements** field.

```
query folderquery {
  folder (
    repositoryIdentifier:"CONTENT"
    identifier:"/SOLUTION Client Onboarding"
  )
  {
    name id
    containedDocuments {
      documents {
        id name creator dateCreated
        majorVersionNumber minorVersionNumber
        mimeType
        contentElements {
          className
          contentType
          elementSequenceNumber
        }
      }
    }
  }
}
```

The result is shown as follows (only 1st document shown here):


```
{
  "data": {
    "folder": {
      "name": "SOLUTION Client Onboarding",
      "id": "{00237489-0000-C216-8E82-6006C432D0B0}",
      "containedDocuments": {
        "documents": [
          {
            "id": "{80537489-0000-CC1E-82E5-2F919DE3F248}",
            "name": "Banking Information - Automation Elite Inc",
            "creator": "cp4badmin",
            "dateCreated": "2023-07-20T17:22:59.982Z",
            "majorVersionNumber": 1,
            "minorVersionNumber": 0,
            "mimeType": "application/pdf",
            "contentElements": [
              {
                "className": "ContentTransfer",
                "contentType": "application/pdf",
                "elementSequenceNumber": 0
              }
            ]
          },
          { ↔ },
          { ↔ },
          { ↔ },
          { ↔ }
        ]
      }
    }
  }
}
```


_6. By hovering with the mouse over contentElements, it can be seen that contentElements is an array of objects which all implement the ContentElement interface. That interface is implemented among others by the type "ContentTransferType" which also implements the “ContentTransfer” interface. Follow the online documentation as follows:



_7. By using a special notation, it can be specified, that if a Content Element implements the ContentTransfer interface, further information should be retrieved. Extend the query as follows:

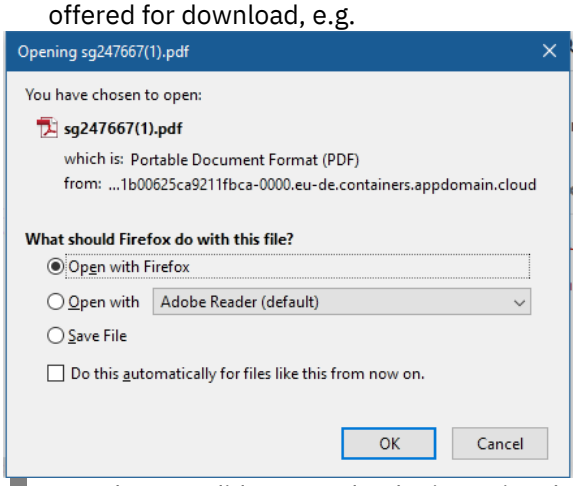
```
query folderquery {
  folder (
    repositoryIdentifier:"CONTENT"
    identifier:"/SOLUTION Client Onboarding"
  )
  {
    name id
    containedDocuments {
      documents {
        id name creator dateCreated
        majorVersionNumber minorVersionNumber
        mimeType
        contentElements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

In the returned information, important further information is contained for content elements implementing the “ContentTransfer” interface. Similarly further information could also be included for other implementations, e.g. ones which implement the “ContentReference” interface. That is not shown here.


```
{
  "data": {
    "folder": {
      "name": "SOLUTION Client Onboarding",
      "id": "{00237489-0000-C216-8E82-6006C432D0B0}",
      "containedDocuments": {
        "documents": [
          {
            "id": "{80537489-0000-CC1E-82E5-2F919DE3F248}",
            "name": "Banking Information - Automation Elite Inc",
            "creator": "cp4badmin",
            "dateCreated": "2023-07-20T17:22:59.982Z",
            "majorVersionNumber": 1,
            "minorVersionNumber": 0,
            "mimeType": "application/pdf",
            "contentElements": [
              {
                "contentType": "application/pdf",
                "elementSequenceNumber": 0,
                "contentSize": 128145,
                "retrievalName": "Banking%20Information%20-%20Automation%20Elite%20Inc.pdf",
                "downloadUrl": "/content?repositoryIdentifier=CONTENT&documentId={80537489-0000-
CC1E-82E5-2F919DE3F248}&elementSequenceNumber=0"
              }
            ]
          },
          { ↔ },
          { ↔ },
          { ↔ },
          { ↔ }
        ]
      }
    }
  }
}
```

_8. For constructing the download URL, open a text editor (e.g. notepad if you are working on a Windows machine) and copy the GraphiQL URL address into it. Search for the first question mark and everything starting from the slash in front of the question mark in the text editor. Then copy one of the downloadUrl values and append it to the address in the text editor, giving the completed download URL. Copy & paste that URL into a new tab of your browser.

_9. If you did that correctly, the document should directly be downloaded or 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, its stored in a cookie. 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, and some custom properties.

```
query documentsquery {
  documents (
    repositoryIdentifier:"CONTENT"
    from: "SOLUTION_Client_Document d INNER JOIN ContentSearch c ON d.This =
c.QueriedObject"
    where: "CONTAINS(d.*, 'Automation Elite')"
  )
  {
    documents {
      id
      className
      properties(includes: ["Name", "SWAT_Client_Name"]) {
        id value
      }
      foldersFiledIn {
        folders {
          pathName
        }
      }
    }
  }
}
```

_3. You can 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 networks with a high latency. This below query gets information from the domain, and also on the id of the root folder of the “CONTENT” 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 the same query keyword twice and observe that an error is returned. 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
  }
}
```

2.2.4 Security-related Queries

In Version CPE / GraphQL Version 5.5.11, which is part of Cloud Pak for Business Automation Version 23.0.1, a couple new queries were made available. In this section and the following section, some of these new ones will be explored. The first ones are queries around security.

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

_2. Use auto-completion to determine the name of the new queries related to security. Type in following, then with the cursor directly after "sec" press Ctrl+Space to list matching query names starting with "sec".

```
query whoami {
  sec
}
```

_3. Complete the query as follows as below. The result will give **information** about the **logged-on user**.

```
query whoami {
  secCurrentUser {
    name
    displayName
    memberOfGroups {
      groups {
        name
      }
    }
    shortName
    email
  }
}
```

Depending on the user and the environment you would be getting a result like the following:


```
{
  "data": {
    "secCurrentUser": {
      "name": "cn=usr001,dc=example,dc=com",
      "displayName": "usr001",
      "memberOfGroups": {
        "groups": [
          {
            "name": "cn=generalusers,dc=example,dc=com"
          },
          {
            "name": "cn=cp4bausers,dc=example,dc=com"
          }
        ]
      },
      "shortName": "usr001",
      "email": "usr001@example.com"
    }
  }
}
```


_4. The second example for the security related queries will be **searching for groups starting with the prefix "cp"**. Similar as with folders above, querying for "secGroup" gives information about a specific group, while "secGroups" will be searching for groups. In order to not overload the LDAP Server of the environment, it is important to limit the query, e.g. as below query searches only for group names starting with "ge". Try the following:

```
query Generalusers {
  cp4agroups: secGroups (searchPattern: "ge", searchType: PREFIX_MATCH,
              searchAttribute: SHORT_NAME, sortType: ASCENDING) {
    groups {
      name
    }
  }
}
```

The result is again dependent on the environment (the domain name of the groups might differ):


```
{
  "data": {
    "Generalusers": {
      "groups": [
        {
          "name": "cn=generalusers,dc=example,dc=com"
        }
      ]
    }
  }
}
```


Other new queries and mutations (see below) of the Security group allow to query and update local groups, and managed users.

2.2.5 Administrative Queries

A further area where new queries and mutations were made available is query, update, and delete of Document Classes, and Property Templates.

_1. The first example shows how to **retrieve information about a class definition**. In the query, the symbolic name or object id of the class definition is needed, the information can for example be obtained from querying a document. Notice that through the class definition also information about the property definitions can be obtained.

> Notice that information comes from the Property Definitions which might be different from the Property Templates, which were used to define them.

```
query getClientInformationClassDetails {
  clientInfoClass: admClassDefinition (repositoryIdentifier: "CONTENT",
                                       identifier: "SOLUTION_Client_Information") {
      symbolicName
        propertyDefinitions {
          dataType
          symbolicName
          isSystemOwned
          ... on PropertyDefinitionString {
            maximumLengthString
          }
          ... on PropertyDefinitionDateTime {
            isDateOnly
          }
      }
  }
}
```

The result is not shown completely here, it includes details of all predefined system properties as well as the user defined properties. Notice the technique with which to extract further information about string and datetime properties. Observe in the results, that the maximumLengthString information is present only on properties of type string, for example.

The screenshot below shows partial results of this query:

3 Mutations

3.1 Introduction

Mutations are very similar to queries. They use the keyword "mutation" at the beginning and are otherwise using the same syntax as 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. To change information i.e., create, change, or delete folders or documents with GraphQL, mutations need to be used. The below mutation will create a folder below the root folder, as an example (replace usrxx with your username, to avoid collision with any other user working on this lab at the same time). 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. The permission objects are of type “CmAbstractPermission”, and to access additional properties of the implementation class “AccessPermissionType” we need to use the ... notation to "cast" the permission type.

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

The security shows the 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": "{80842C8A-0000-C61D-ACD0-02D6A6D87165}",
      "pathName": "/usrxxx GraphQL Folder",
      "permissions": [
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "#AUTHENTICATED-USERS",
          "granteeType": "GROUP",
          "accessMask": 131073
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "cn=usr001,dc=example,dc=com",
          "granteeType": "USER",
          "accessMask": 999415
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "cn=cp4badmin,dc=example,dc=com",
          "granteeType": "USER",
          "accessMask": 999415
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "cn=ce_environmentowners,dc=example,dc=com",
          "granteeType": "GROUP",
          "accessMask": 999415
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "cn=p8administrators,dc=example,dc=com",
          "granteeType": "GROUP",
          "accessMask": 999415
        },
        {
          "permissionSource": "SOURCE_DEFAULT",
          "inheritableDepth": "OBJECT_ONLY",
          "granteeName": "cn=generalusers,dc=example,dc=com",
          "granteeType": "GROUP",
          "accessMask": 999415
        }
      ]
    }
  }
}
```


The permissions can also be updated. Since version 5.5.11 of GraphQL, or version 23.0.1 of CP4BA, the permissions can not only be replaced completely but can also be updated. The mutation for updating the security settings for the folder created above will be shown later in its parameterized variant, in section 4.2.

_2. The folder can be deleted again by using the "deleteFolder" operation on the mutation, as follows (replacing usrxx with your login):

```
mutation deletefolder {
  deleteFolder (
    repositoryIdentifier: "CONTENT"
    identifier: "/usrxx 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 need to do 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.

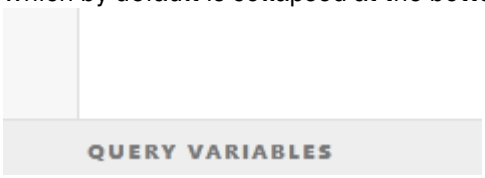
This can be solved by using parameters. This way, the GraphQL queries can be developed and can be defined in a custom application as constant strings. Only the parameters need to be supplied, and they are using JSON syntax.

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/5.7.0?topic=mutations-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. This way it is straightforward to provide the required data. In the data below again substitute the username.

```
{"theRepo": "CONTENT",
"parentFolderId": "/",
"newFolderName": "usrxx GraphQL Folder"}
```

Execute the mutation 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, or to be more precise to replace the complete permissions of the folder by a new set of permissions. 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. For the parameter values, as two different type of LDAP repositories are used for this lab, the names of the users have slight differences. Therefore, please copy & paste the parameters given below to the GraphQL parameters pane, and then update the correct user- and group names from the names given in the reply of the createFolder mutation. Also update the correct folder name by replacing usrxx with your username:

```
{
  "theRepo": "CONTENT",
  "folderId": "/usrxx GraphQL Folder",
  "permissions": [{
      "type": "ACCESS_PERMISSION",
      "inheritableDepth": "OBJECT_ONLY",
      "accessMask": 999415,
      "subAccessPermission": {"accessType": "ALLOW", "granteeName": "cp4admin user" }
    },{
      "type": "ACCESS_PERMISSION",
      "inheritableDepth": "OBJECT_ONLY",
      "accessMask": 999415,
      "subAccessPermission": {"accessType": "ALLOW", "granteeName": "usrxx user" }
    }, {
      "type": "ACCESS_PERMISSION",
      "inheritableDepth": "OBJECT_ONLY",
      "accessMask": 999415,
      "subAccessPermission": {"accessType": "ALLOW", "granteeName": "p8 administrators" }
    }
  ]
}
```

When the mutation is executed without errors, the folder should be visible only for the user, as all access permissions for other users were removed.

With FileNet Content Services GraphQL 5.5.11 provided with Cloud Pak for Business Automation 23.0.1, dependent object arrays can now also be updated, instead of replacing the whole list by a new one.

_7. The parameterized version for the deletion of the GraphQL folder is again left as an exercise to you.

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: "/usrxx 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": "/usrxx GraphQL Folder"
}
```