Content Delivery using Content Fragments with GraphQL

With Adobe Experience Manager (AEM) as a Cloud Service, you can use Content Fragments, together with GraphQL, to deliver content for use in your applications.

The Fundamentals

As a quick introduction, the following fundamentals are introduced.

GraphQL

GraphQL is:

- "*...a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.*".
  See GraphQL.org
- "*...an open spec for a flexible API layer. Put GraphQL over your existing backends to build products faster than ever before....*".
  See Explore GraphQL. "*Explore GraphQL is maintained by the Apollo team. Our goal is to give developers and technical leaders around the world all of the tools they need to understand and adopt GraphQL.*".

GraphQL allows you to perform (complex) queries on your Content Fragments; with each query being according to a specific model type. The content returned can then be used by your applications.

GraphQL Terminology

GraphQL uses the following:

- **Queries**
- **Schemas and Types**
- **Fields**

See the (GraphQL.org) Introduction to GraphQL for comprehensive details, including the Best Practices.

GraphQL Query Types

With GraphQL you can perform queries for either:

- **Single entry**
- **List of entries**

Content Fragments

Content fragments:

- Contain structured content.
- They are based on a Content Fragment Model, which predefines the structure for the resulting fragment.
- Can be nested, using either of the following data types:
  - **Content Reference**
    - This provides a simple reference to another fragment.
  - **Fragment Reference**
    - This references another fragment - dependent on a specific model.

Content Fragment Models

These Content Fragment Models:

- Provide the data types and fields required for GraphQL to ensure that your application only requests what is possible, and receives what is expected.

Fragment References

The **Fragment Reference**:

- Is of particular interest in conjunction with GraphQL.
- Is a specific data type that can be used when defining a Content Fragment Model.
- References another fragment, dependent on a specific Content Fragment Model.
- Allows you to retrieve structured data.
  - When defined as a **multifeed**, multiple sub-fragments can be referenced (retrieved) by the prime fragment.

See:

- A sample Content Fragment structure
- And some sample GraphQL queries, based on the composite content fragments.

A Sample Content Fragment Structure for use with GraphQL

For a simple example we need:

- One, or more, Sample Content Fragment Models
- Sample Content Fragments based on the above models

Sample Content Fragment Models

If we consider the following Content Models, and their interrelationships (references ->):

- Company
  -> Person
    -> Award
- City

Company

The fields:

Field Name	Data Type	Reference
Company Name	Single line text	
CEO	Fragment Reference	Person
Employees	Fragment Reference	Person

Person

The fields:

Field Name	Data Type	Reference
Name	Single line text	
First name	Single line text	
Awards	Fragment Reference	Award

Award

The fields:

Field Name	Data Type	Reference
Shortcut/ID	Single line text	
Title	Single line text	

City

The fields:

Field Name	Data Type	Reference
Name	Single line text	
Country	Single line text	
Population	Number	
Categories	Tags	

Sample Content Fragments

GraphQL - Sample Queries

Sample Query - All Available Schemas

Query

```
{
  __schema {
    types {
      name
    }
  }
}
```

Result

Sample Query - All Persons that have a name of "Jobs" or "Smith"

Query

```
query {
  persons(filter: {
    logOp: AND
    expression: {
      name: {
        logOp: OR
        expressions: [
          {
            operator: EQUALS
            value: "Jobs"
          },
          {
            operator: EQUALS
            value: "Smith"
          }
        ]
      }
    }
  }) {
    name
    firstName
  }
}
```

Results

```
{
  "data": {
    "persons": [
      {
        "name": "Smith",
        "firstName": "Adam"
      },
      {
        "name": "Smith",
        "firstName": "Joe"
      },
      {
        "name": "Jobs",
        "firstName": "Steve"
      }
    ]
  }
}
```