Building Mutations with the SDK

The architecture for a mutation is to pass in an input object and one or more return objects to define what data a successful mutation should return. Because a mutation also makes a query, it is recommended you read GraphQL Queries section first.

The input object is an implementation of the IDataSenderBuilder interface, **InputDataSenderBuilder**. Most mutations expect objects, so take an **InputData SenderBuilder**. This is a Java object containing the **mutationName** and the **fieldsMap**, a Map of field keys and values which contains the content being passed.The fields are defined in an enum in the relevant GraphQLMutation.

In addition, one of more return objects also need to be passed, to determine the content to return from the query. These return objects also implement the IDataSenderBuilder interface and define what should be returned. Unlike queries, the response may be an object or it may be just a single scalar json value. Consequently, there are two different classes that a return object may be, an **ObjectDataSenderBuilder** and a **ScalarDataSenderBuilder**. See specific mutations for further details.

The input and output objects are passed into the mutation object, a class extending **BaseGraphQLMutation** (the mutation equivalent of what exists for queries, the BaseGraphQLQuery). This is passed to a GraphQLRequest, which is added as the request for an endpoint. This process should be familiar from queries. There are also helper methods from the WWClient class to reduce the complexity of layers.

The following code block will demonstrate the full process (*this* is a WWClient object that corresponds to an authenticated WWClient).

updateSpace

```
ArrayList<String> members = new ArrayList<String>();
members.add("ae81bc5d-70fa-4bdb-bab7-29f4df10ddf3y");
InputDataSenderBuilder spaceInput = new InputDataSenderBuilder(Space.UPDATE_SPACE_MUTATION_NAME);
spaceInput.addField(UpdateSpaceFields.ID, spaceId);
spaceInput.addField(UpdateSpaceFields.TITLE, "My new title");
spaceInput.addField(UpdateSpaceFields.MEMBERS, members);
spaceInput.addField(UpdateSpaceFields.MEMBER_OPERATION, UpdateSpaceMemberOperation.ADD);
ObjectDataSenderBuilder returnObject1 = new ObjectDataSenderBuilder(Space.ONE_SPACE_QUERY_OBJECT_NAME);
returnObject1.addField(SpaceFields.TITLE);
ScalarDataSenderBuilder returnObject2 = new ScalarDataSenderBuilder(MEMBER_IDS_CHANGED_FIELD);
SpaceUpdateGraphQLMutation mutationObject = new SpaceUpdateGraphQLMutation(spaceInput, returnObject1,
returnObject2);
ep.setRequest(new GraphQLRequest(mutationObject));
ep.executeRequest();
UpdateSpaceContainer output = ep.getResultContainer().getData().getUpdateSpaceContainer()
```

Lines 1-2 create an ArrayList containing the single member we want to add to access for the Space.

Lines 3 - 7 create the InputDataSenderBuilder, passing the ID of the space we want to update, the title we want to rename the space with, the members we want to update and the "ADD" enum to tell Watson Workspace we want to add rather than remove this user.

Lines 8 - 9 create an ObjectDataSenderBuilder which is a very basic *getSpace* query to return the space's title. Unlike the normal *getSpace* query, we don't need to ask for the ID of which space to return. We're returning data for the space we're updating in the mutation.

Line 10 creates a ScalarDataSenderBuilder to echo the array of members updated.

Line 11 passes the input object and the two return objects into a new instance of a class extending BaseGraphQLMutation. Classes are available for all query types and will create a default method name, e.g. "updateSpace".

Line 12 passes the query object into a new GraphQLRequest, and sets this as the request for the GraphQLEndpoint. Line 20 executes the request.

Line 14 then uses GSON to parse the result container and get an UpdateSpaceContainer which will contain the Space and an array of the member ids changed. Lines 12 - 14 could be reduced into a single line by using a helper method from the WWClient class that takes just the new instance of a class extending BaseGraphQLQuery:

```
UpdateSpaceContainer result = client.updateSpaceWithMutation(new GraphQLRequest(mutationObject));
```

On occasion a mutation may only accept fixed input and only return a fixed output. The **deleteSpace** mutation is one such mutation. As a result, there are no methods for passing input and output objects, just simple methods in WWClient.