Caching

> Providing Object Identifiers allows clients to build rich caches

In an endpoint-based API, clients can use HTTP caching to easily avoid refetching resources, and for identifying when two resources are the same. The URL in these APIs is a **globally unique identifier** that the client can leverage to build a cache. In GraphQL, though, there's no URL-like primitive that provides this globally unique identifier for a given object. It's hence a best practice for the API to expose such an identifier for clients to use.

Globally Unique IDs

One possible pattern for this is reserving a field, like `id`, to be a globally unique identifier. The example schema used throughout these docs uses this approach:

```
{
  starship(id:"3003") {
    id
    name
  }
  droid(id:"2001") {
    id
    name
    friends {
      id
      name
    }
  }
}
```

```
{
  "data": {
    "starship": {
      "id": "3003",
      "name": "Imperial shutt
    },
    "droid": {
      "id": "2001",
      "name": "R2-D2",
      "friends": [
        {
          "id": "1000",
          "name": "Luke Skywa
        },
        {
          "id": "1002",
```

This is a powerful tool to hand to client developers. In the same way that the URLs of a resource-based API provided a globally unique key, the `id` field in this system provides a globally unique key.

If the backend uses something like UUIDs for identifiers, then exposing this globally unique ID may be very straightforward! If the backend doesn't have a globally unique ID for every object already, the GraphQL layer might have to construct this. Oftentimes, that's as simple as appending the name of the type to the ID and using that as the identifier; the server might then make that ID opaque by base64-encoding it.

Optionally, this ID can then be used to work with the Global Object Identification's `node` pattern.

Compatibility with existing APIs

One concern with using the `id` field for this purpose is how a client using the GraphQL API would work with existing APIs. For example, if our existing API accepted a type-specific ID, but our GraphQL API uses globally unique IDs, then using both at once can be tricky.

In these cases, the GraphQL API can expose the previous API's IDs in a separate field. This gives us the best of both worlds:

- GraphQL clients can continue to rely on a consistent mechanism for getting a globally unique ID.
- Clients that need to work with our previous API can also fetch `previousApiId` from the object, and use that.

Alternatives

While globally unique IDs have proven to be a powerful pattern in the past, they are not the only pattern that can be used, nor are they right for every situation. The really critical functionality that the client needs is the ability to derive a globally unique identifier for their caching. While having the server derive that ID simplifies the client, the client can also derive the identifier. Oftentimes, this would be as simple as combining the type of the object (queried with `__typename`) with some type-unique identifier.

Additionally, if replacing an existing API with a GraphQL API, it may be confusing if all of the fields in GraphQL are the same **except** `id`, which changed to be globally unique. This would be another reason why one might choose not to use `id` as the globally unique field.