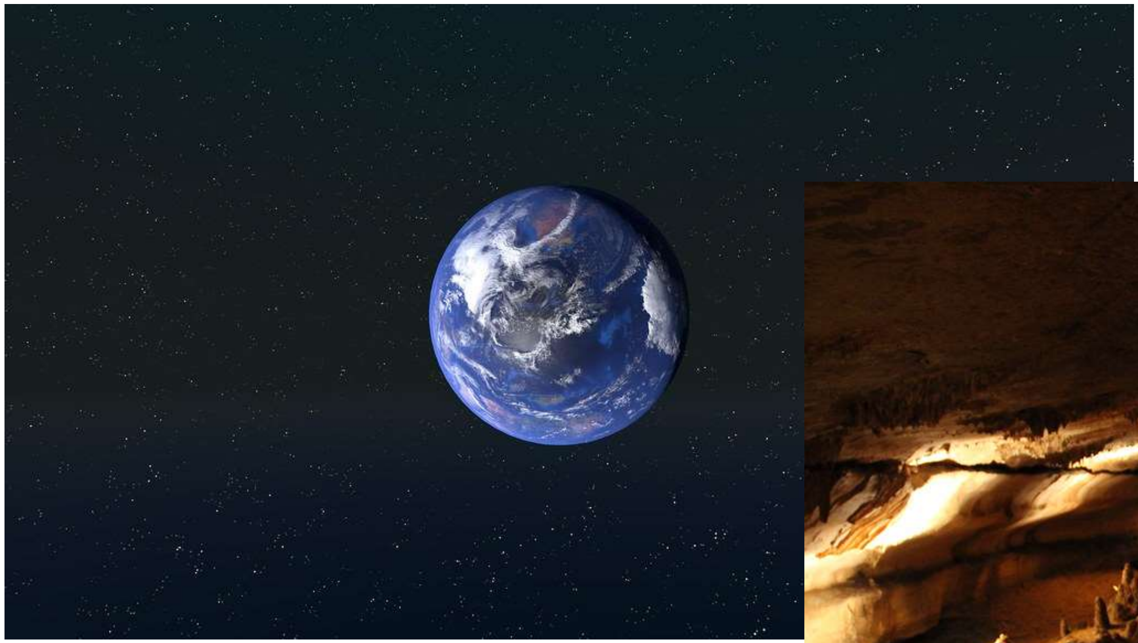
Open Liberty

MicroProfile GraphQL

Everything you ever wanted – and ONLY what you want!

Andy McCright – IBM Web Services Architect
@AndrewMcCright



"Earth" by kristian fagerström is licensed under CC BY-SA 2.0

"Dripstone Trail Inside Cave" by daveynin is licensed under CC BY 2.0

The Entire History of Enterprise Java in under 60 seconds!

J2EE

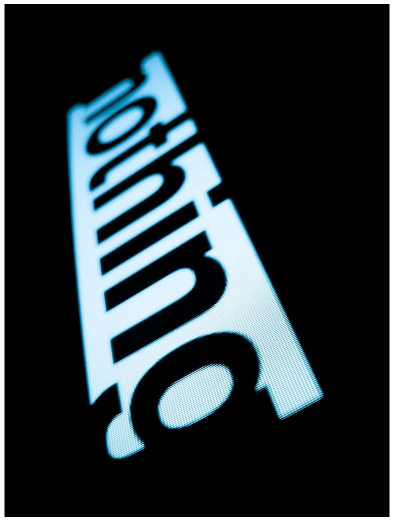
servlets EJBs

Java EE

JSP / JSF SOAP Web services

Dependency Injection RESTful Services

Persistence & Transactions



"nothing" by kozumel is licensed under CC BY-ND 2.0

A New Hope!

"Jar Jar's lightsaber" by Chris Makarsky is licensed under CC BY 2.0

What is MicroProfile?

https://microprofile.io

Open Source!

MicroProfile 3.3

🟪 Rest Client 1.4	🟪 Fault Tolerance 2.1	🟪 Health 2.2	🟪 Metrics 2.3
🟦 Open API 1.1	🟦 Open Tracing 1.3	🟦 JWT 1.1	🟪 Config 1.4
🟩 CDI 2.0	🟩 JAX-RS 2.1	🟩 JSON-B 1.0	🟩 JSON-P 1.1

🟪 Updated specs 🟦 Unchanged specs 🟩 Java EE specs

What is Jakarta EE?

- Jakarta EE 8 == Java EE 8
  – Servlets, EJBs, JSPs, JSF, JAX-WS, JAX-RS, JPA, etc.
- Jakarta EE 9 == Java EE 8, except for package names
  – javax.* → jakarta.*
- The future is wide open!
  – NoSQL, Caching, MVC, incorporating some MP technologies…

Open Source!

What is GraphQL?

- GraphQL is an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data. GraphQL interprets strings from the client, and returns data in an understandable, predictable, pre-defined manner. GraphQL is an alternative, though not necessarily a replacement for REST.

- GraphQL was developed internally by Facebook in 2012 before being publicly released in 2015. Facebook delivered both a specification and a reference implementation in JavaScript.

- On 7 November 2018, Facebook moved the GraphQL project to the newly-established GraphQL foundation, hosted by the non-profit Linux Foundation. This is a significant milestone in terms of industry and community adoption. GraphQL is widely used by many customers.

What is GraphQL?

- GraphQL enables clients to invoke queries (read) and mutations (create/update/delete) on entity types and specify which fields from the entity is returned.
- GraphQL schemas allow developers to specify output types, input types, interfaces, and enumerated types.
- GraphQL "primitives" are known as scalars. The spec defines that all GraphQL implementations handle String, Int, Float, Boolean, and ID. Implementations are allowed to define custom scalars.

Why GraphQL?

- Avoiding over-fetching or under-fetching data. Clients can retrieve several types of data in a single request or can limit the response data based on specific criteria.
- Enabling data models to evolve. The schema can change without requiring changes in existing clients, and vice versa - this can be done without a need for a new version of the application.
- Partial results on errors.
- The schema defines how the data can be accessed and serves as the contract between the client and the server. Development teams on both sides can work without further communication.
- Native schema introspection enables users to discover APIs and to refine the queries on the client-side. This advantage is increased with graphical tools such as GraphiQL and GraphQL Voyager enabling smooth and easy API discovery.

What is MicroProfile GraphQL?

- MP Community with many active participants (more are always welcome!)
  https://github.com/eclipse/microprofile-graphql
  https://gitter.im/eclipse/microprofile-graphql
- Goal is a “code-first” approach to building GraphQL applications.
- MP GraphQL API intends to borrow ideas from well-known APIs like JAX-RS.
- Based around the GraphQL-SPQR project
  https://github.com/leangen/graphql-spqr

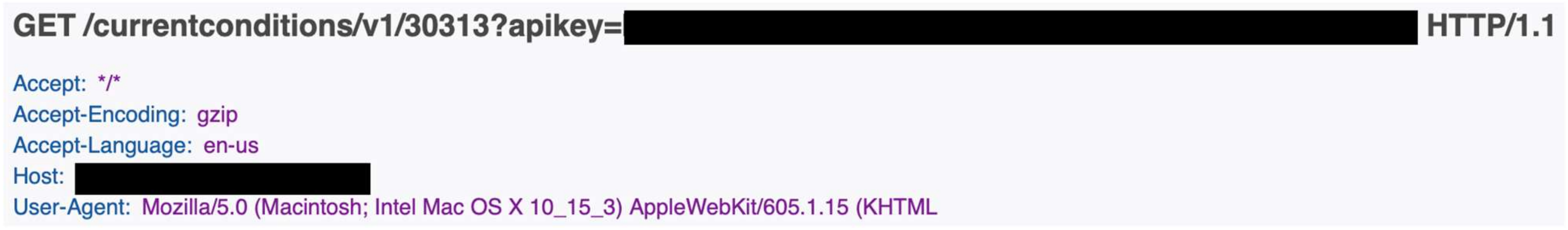
Sample App

Open Liberty

- This app will provide current conditions and forecast for various locations, similar to services offered by accuweather.com, weather.com, etc.
- Unlike “real” weather stations, our data is generated at random.
- At least at first, this sample app will function as the complete service. In the future, we could incorporate a micro-services approach using IoT-based weather stations, etc.

The Scenario: Local conditions

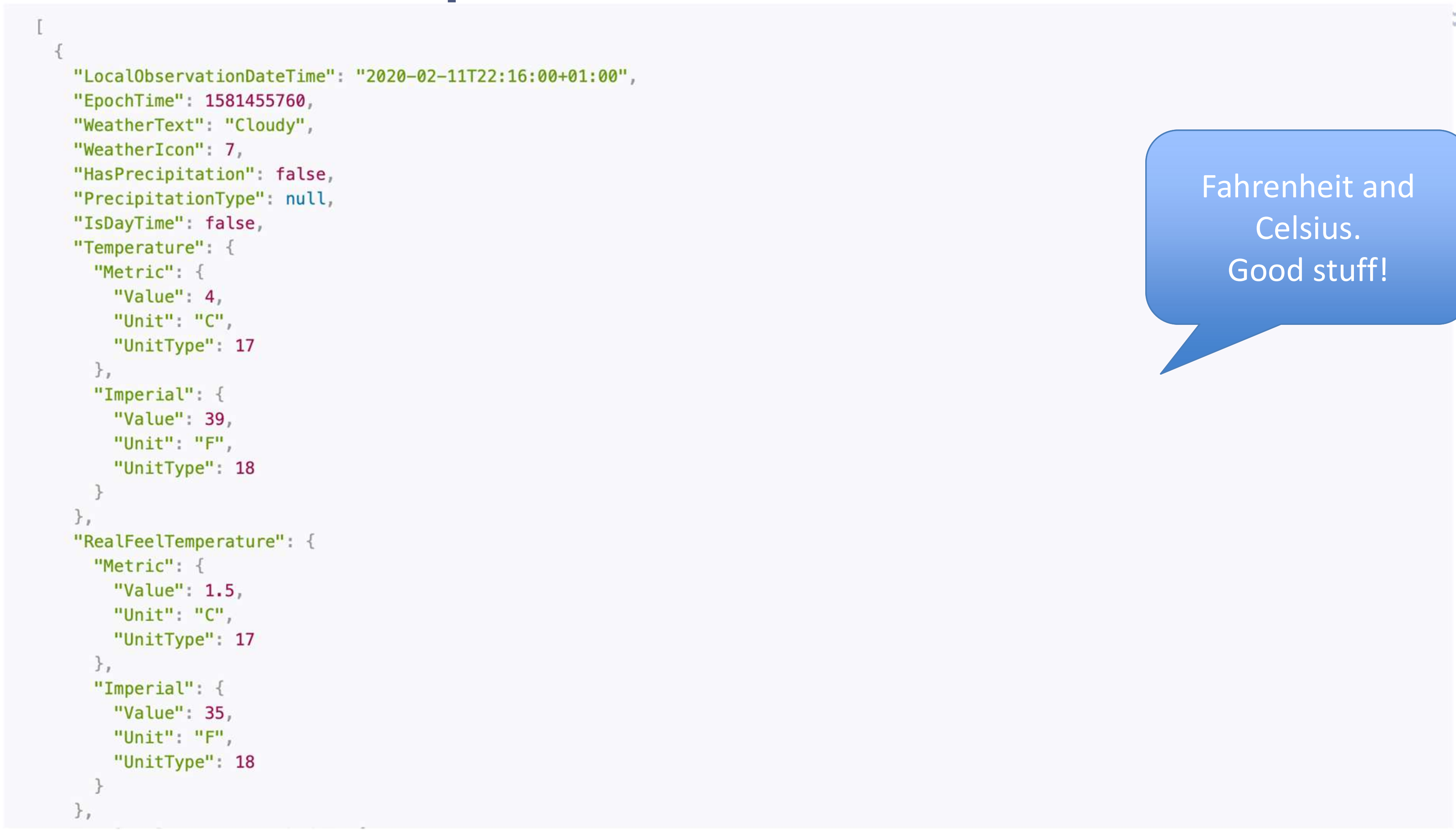
- Our “portal” app would like to include a widget on the webpage that will show the current temperature and tell if it is raining, snowing, sunny, etc.
- There are a few service providers that will provide current weather data via RESTful APIs.
- So we sign up for one and make our RESTful request:



```
GET /currentconditions/v1/30313?apikey=[redacted] HTTP/1.1

Accept:  */*
Accept-Encoding:  gzip
Accept-Language:  en-us
Host:  [redacted]
User-Agent:  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_3) AppleWebKit/605.1.15 (KHTML
```

- And then we get this…

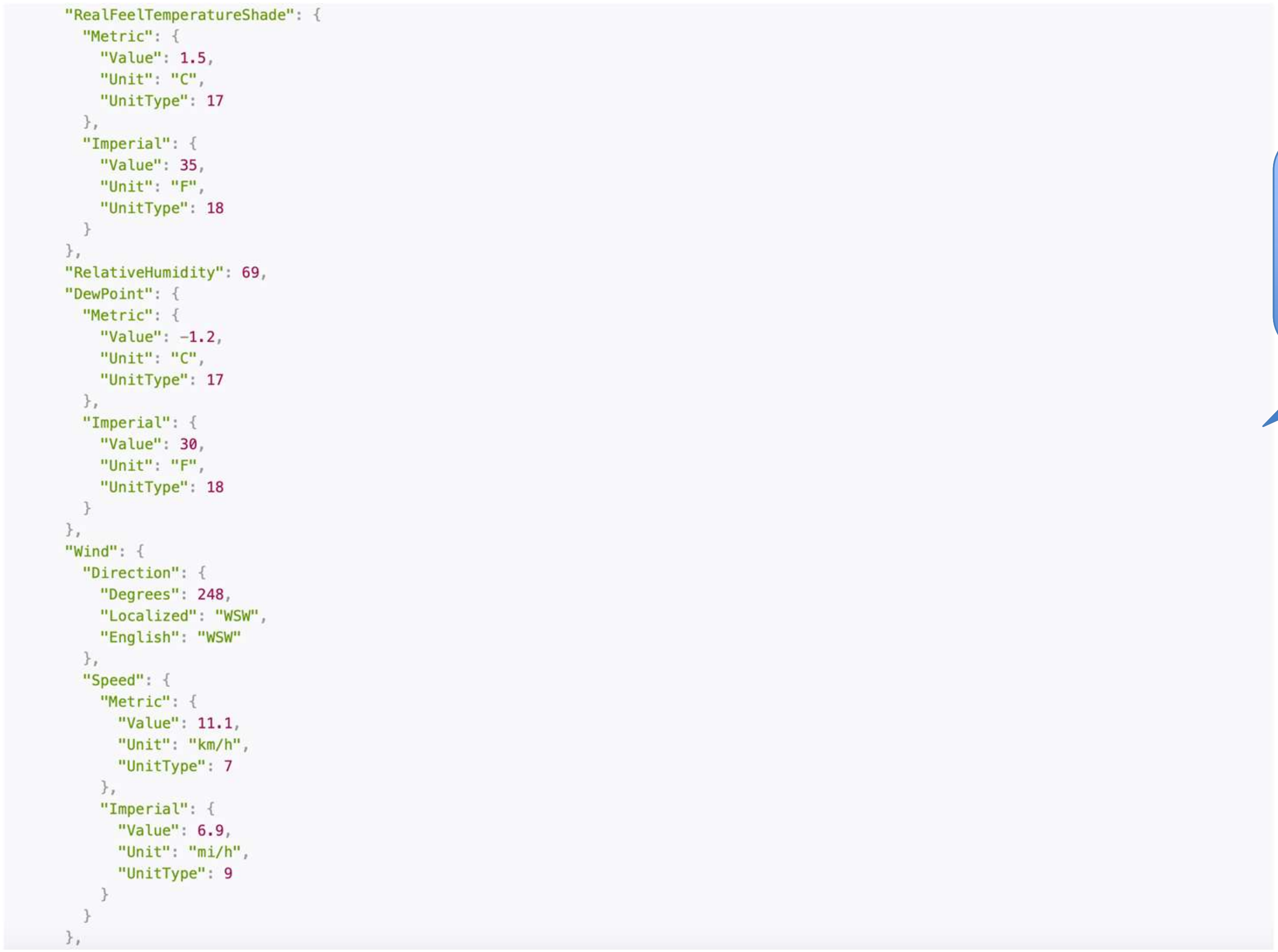
RESTful Response



```
[
  {
    "LocalObservationDateTime": "2020-02-11T22:16:00+01:00",
    "EpochTime": 1581455760,
    "WeatherText": "Cloudy",
    "WeatherIcon": 7,
    "HasPrecipitation": false,
    "PrecipitationType": null,
    "IsDayTime": false,
    "Temperature": {
      "Metric": {
        "Value": 4,
        "Unit": "C",
        "UnitType": 17
      },
      "Imperial": {
        "Value": 39,
        "Unit": "F",
        "UnitType": 18
      }
    },
    "RealFeelTemperature": {
      "Metric": {
        "Value": 1.5,
        "Unit": "C",
        "UnitType": 17
      },
      "Imperial": {
        "Value": 35,
        "Unit": "F",
        "UnitType": 18
      }
    },
```

Fahrenheit and Celsius.
Good stuff!

RESTful Response



```
  "RealFeelTemperatureShade": {
    "Metric": {
      "Value": 1.5,
      "Unit": "C",
      "UnitType": 17
    },
    "Imperial": {
      "Value": 35,
      "Unit": "F",
      "UnitType": 18
    }
  },
  "RelativeHumidity": 69,
  "DewPoint": {
    "Metric": {
      "Value": -1.2,
      "Unit": "C",
      "UnitType": 17
    },
    "Imperial": {
      "Value": 30,
      "Unit": "F",
      "UnitType": 18
    }
  },
  "Wind": {
    "Direction": {
      "Degrees": 248,
      "Localized": "WSW",
      "English": "WSW"
    },
    "Speed": {
      "Metric": {
        "Value": 11.1,
        "Unit": "km/h",
        "UnitType": 7
      },
      "Imperial": {
        "Value": 6.9,
        "Unit": "mi/h",
        "UnitType": 9
      }
    }
  },
```

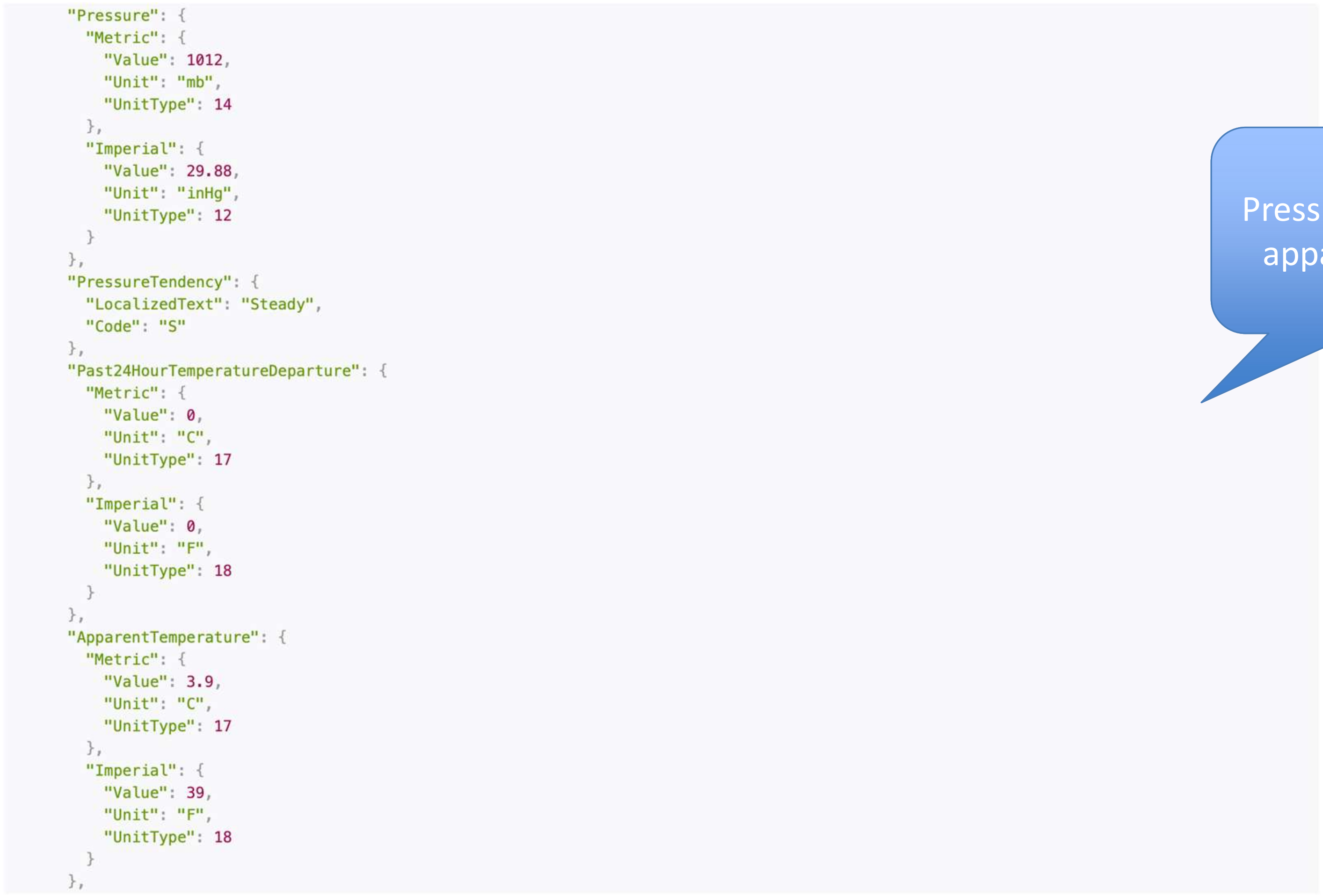
Humidity and wind speed… umm… Okay…

RESTful Response

```
    "WindGust": {
      "Speed": {
        "Metric": {
          "Value": 23.1,
          "Unit": "km/h",
          "UnitType": 7
        },
        "Imperial": {
          "Value": 14.3,
          "Unit": "mi/h",
          "UnitType": 9
        }
      }
    },
    "UVIndex": 0,
    "UVIndexText": "Low",
    "Visibility": {
      "Metric": {
        "Value": 9.7,
        "Unit": "km",
        "UnitType": 6
      },
      "Imperial": {
        "Value": 6,
        "Unit": "mi",
        "UnitType": 2
      }
    },
    "ObstructionsToVisibility": "",
    "CloudCover": 91,
    "Ceiling": {
      "Metric": {
        "Value": 2134,
        "Unit": "m",
        "UnitType": 5
      },
      "Imperial": {
        "Value": 7000,
        "Unit": "ft",
        "UnitType": 0
      }
    },
```

Wind gusts, UV index, cloud cover? Maybe if I'm going to go kiting at the beach…

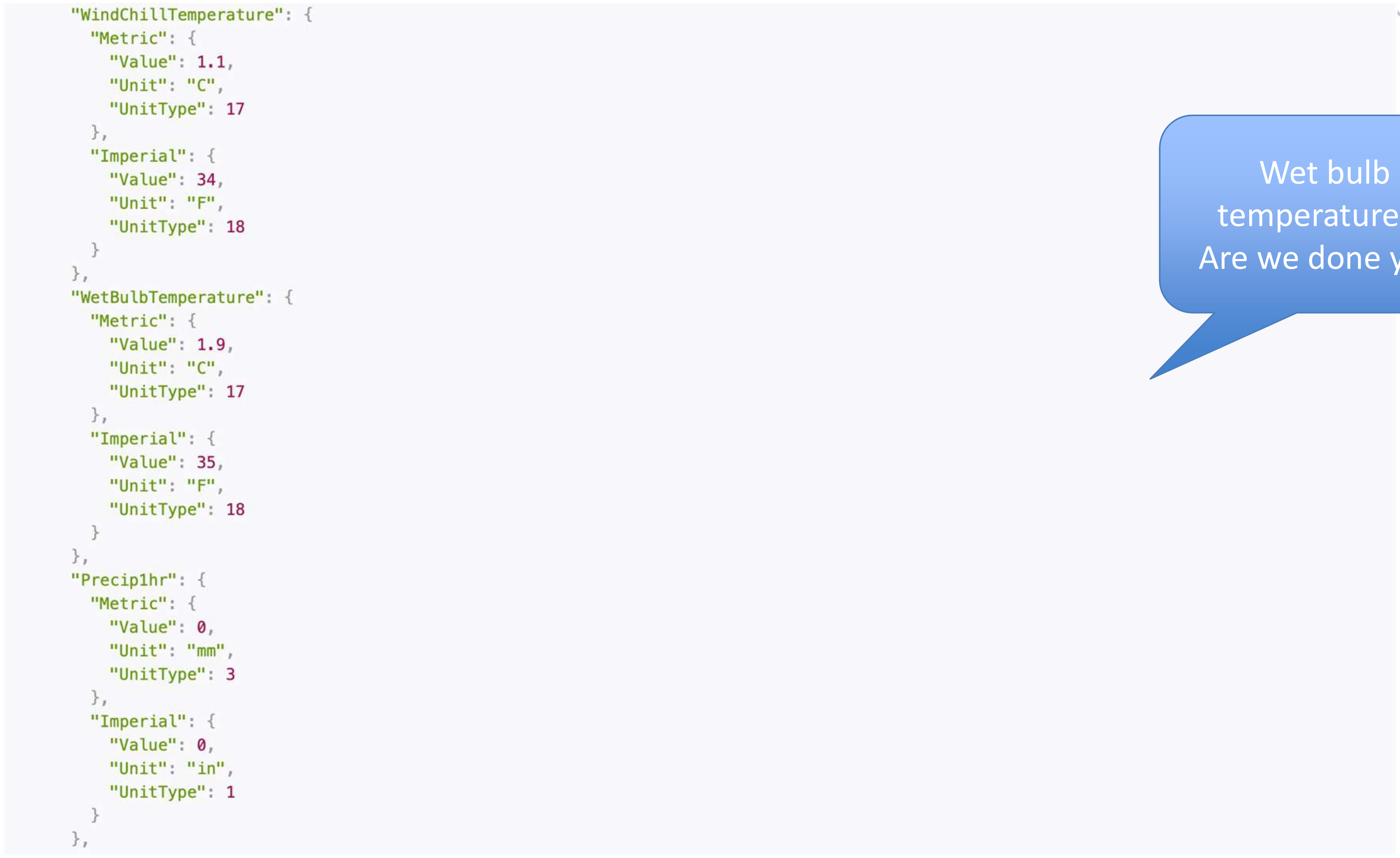
RESTful Response



```
    "Pressure": {
      "Metric": {
        "Value": 1012,
        "Unit": "mb",
        "UnitType": 14
      },
      "Imperial": {
        "Value": 29.88,
        "Unit": "inHg",
        "UnitType": 12
      }
    },
    "PressureTendency": {
      "LocalizedText": "Steady",
      "Code": "S"
    },
    "Past24HourTemperatureDeparture": {
      "Metric": {
        "Value": 0,
        "Unit": "C",
        "UnitType": 17
      },
      "Imperial": {
        "Value": 0,
        "Unit": "F",
        "UnitType": 18
      }
    },
    "ApparentTemperature": {
      "Metric": {
        "Value": 3.9,
        "Unit": "C",
        "UnitType": 17
      },
      "Imperial": {
        "Value": 39,
        "Unit": "F",
        "UnitType": 18
      }
    },
```

Pressure tendency, apparent temp?

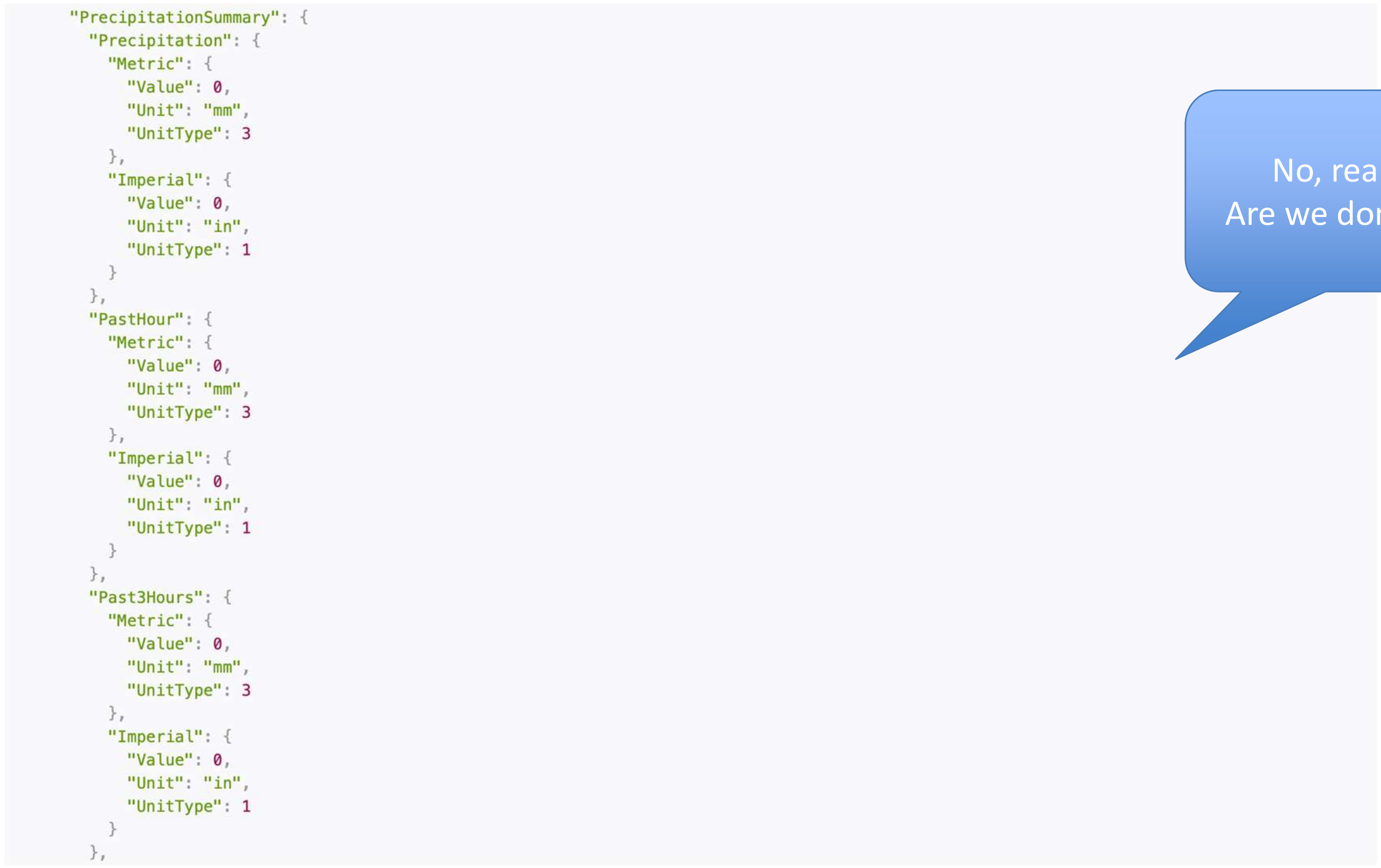
RESTful Response



```
    "WindChillTemperature": {
      "Metric": {
        "Value": 1.1,
        "Unit": "C",
        "UnitType": 17
      },
      "Imperial": {
        "Value": 34,
        "Unit": "F",
        "UnitType": 18
      }
    },
    "WetBulbTemperature": {
      "Metric": {
        "Value": 1.9,
        "Unit": "C",
        "UnitType": 17
      },
      "Imperial": {
        "Value": 35,
        "Unit": "F",
        "UnitType": 18
      }
    },
    "Precip1hr": {
      "Metric": {
        "Value": 0,
        "Unit": "mm",
        "UnitType": 3
      },
      "Imperial": {
        "Value": 0,
        "Unit": "in",
        "UnitType": 1
      }
    },
```

Wet bulb temperature??
Are we done yet?

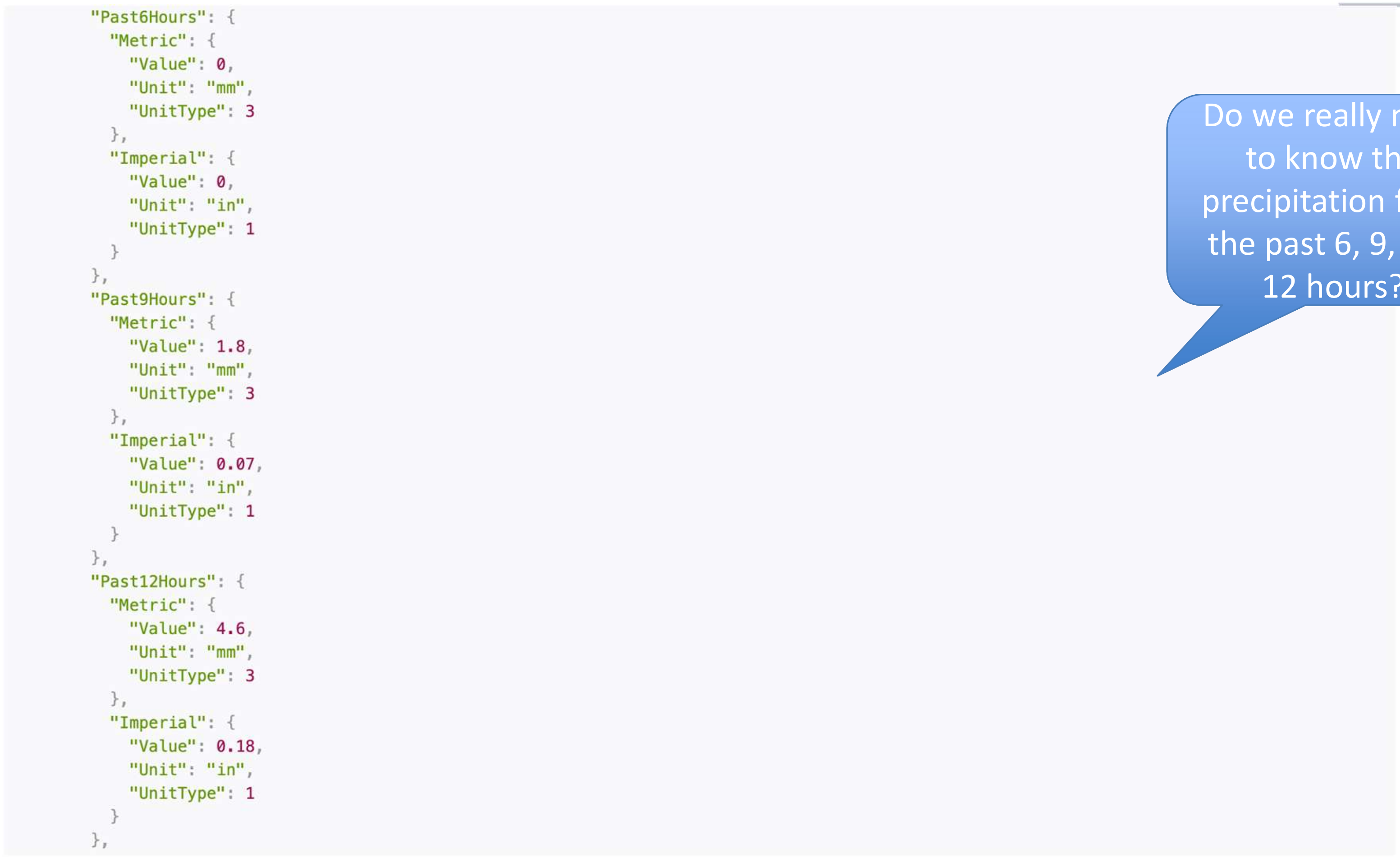
RESTful Response



```
    "PrecipitationSummary": {
      "Precipitation": {
        "Metric": {
          "Value": 0,
          "Unit": "mm",
          "UnitType": 3
        },
        "Imperial": {
          "Value": 0,
          "Unit": "in",
          "UnitType": 1
        }
      },
      "PastHour": {
        "Metric": {
          "Value": 0,
          "Unit": "mm",
          "UnitType": 3
        },
        "Imperial": {
          "Value": 0,
          "Unit": "in",
          "UnitType": 1
        }
      },
      "Past3Hours": {
        "Metric": {
          "Value": 0,
          "Unit": "mm",
          "UnitType": 3
        },
        "Imperial": {
          "Value": 0,
          "Unit": "in",
          "UnitType": 1
        }
      },
```

No, really…
Are we done yet?

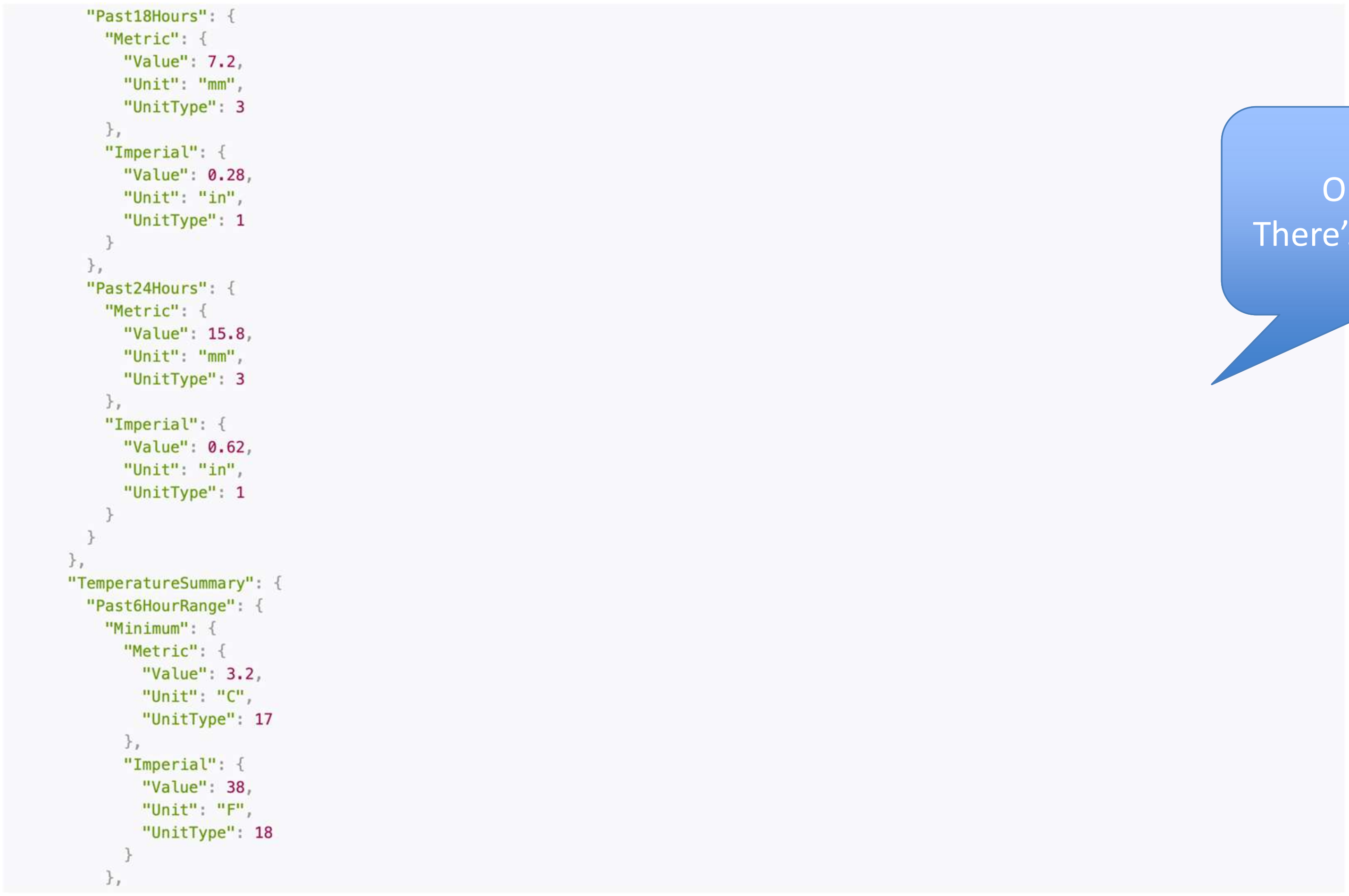
RESTful Response



```
        "Past6Hours": {
          "Metric": {
            "Value": 0,
            "Unit": "mm",
            "UnitType": 3
          },
          "Imperial": {
            "Value": 0,
            "Unit": "in",
            "UnitType": 1
          }
        },
        "Past9Hours": {
          "Metric": {
            "Value": 1.8,
            "Unit": "mm",
            "UnitType": 3
          },
          "Imperial": {
            "Value": 0.07,
            "Unit": "in",
            "UnitType": 1
          }
        },
        "Past12Hours": {
          "Metric": {
            "Value": 4.6,
            "Unit": "mm",
            "UnitType": 3
          },
          "Imperial": {
            "Value": 0.18,
            "Unit": "in",
            "UnitType": 1
          }
        },
```

Do we really need to know the precipitation from the past 6, 9, and 12 hours?

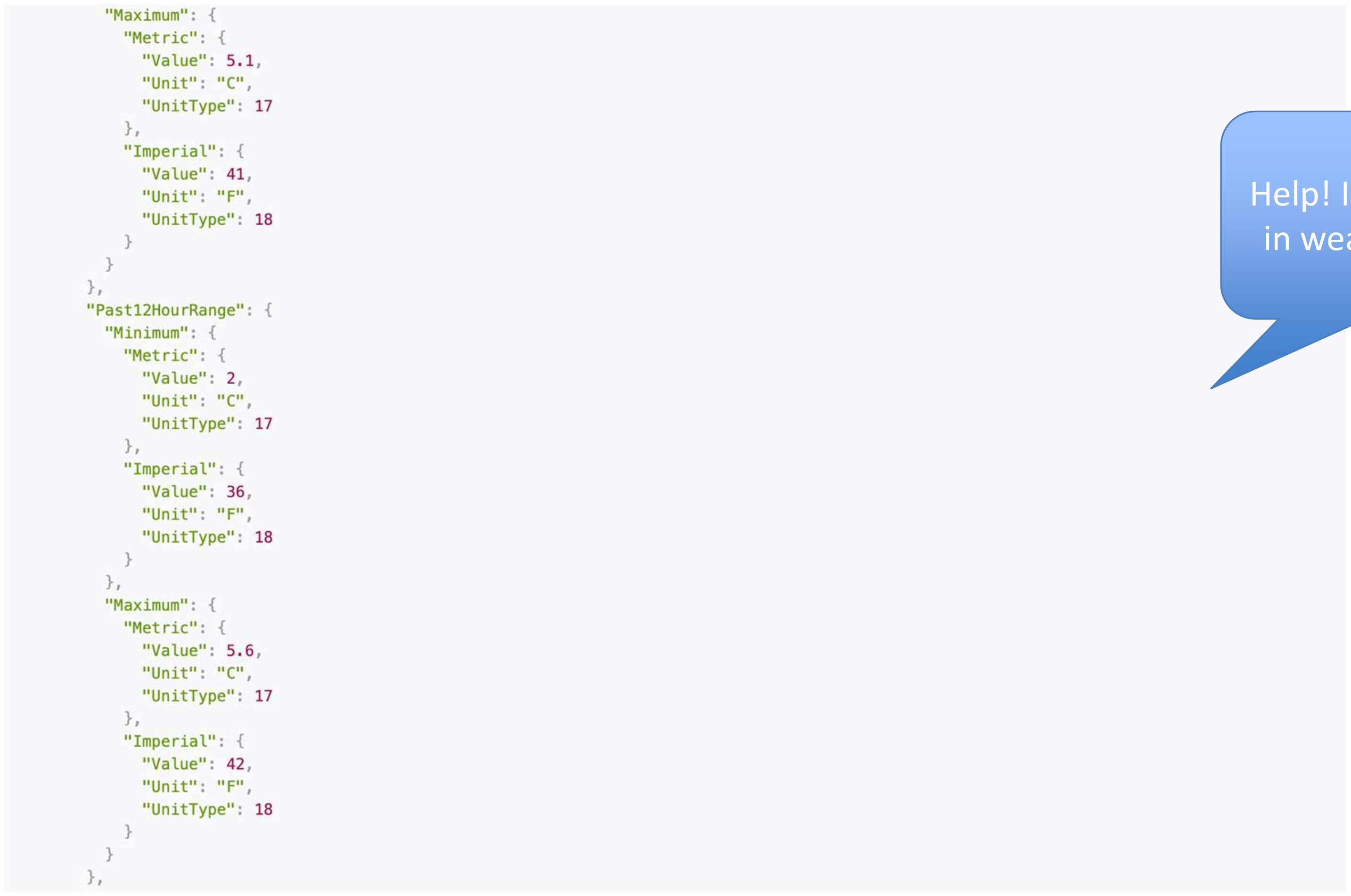
RESTful Response



```
    "Past18Hours": {
      "Metric": {
        "Value": 7.2,
        "Unit": "mm",
        "UnitType": 3
      },
      "Imperial": {
        "Value": 0.28,
        "Unit": "in",
        "UnitType": 1
      }
    },
    "Past24Hours": {
      "Metric": {
        "Value": 15.8,
        "Unit": "mm",
        "UnitType": 3
      },
      "Imperial": {
        "Value": 0.62,
        "Unit": "in",
        "UnitType": 1
      }
    }
  },
  "TemperatureSummary": {
    "Past6HourRange": {
      "Minimum": {
        "Metric": {
          "Value": 3.2,
          "Unit": "C",
          "UnitType": 17
        },
        "Imperial": {
          "Value": 38,
          "Unit": "F",
          "UnitType": 18
        }
      },
```

Oh dear…
There's still more…

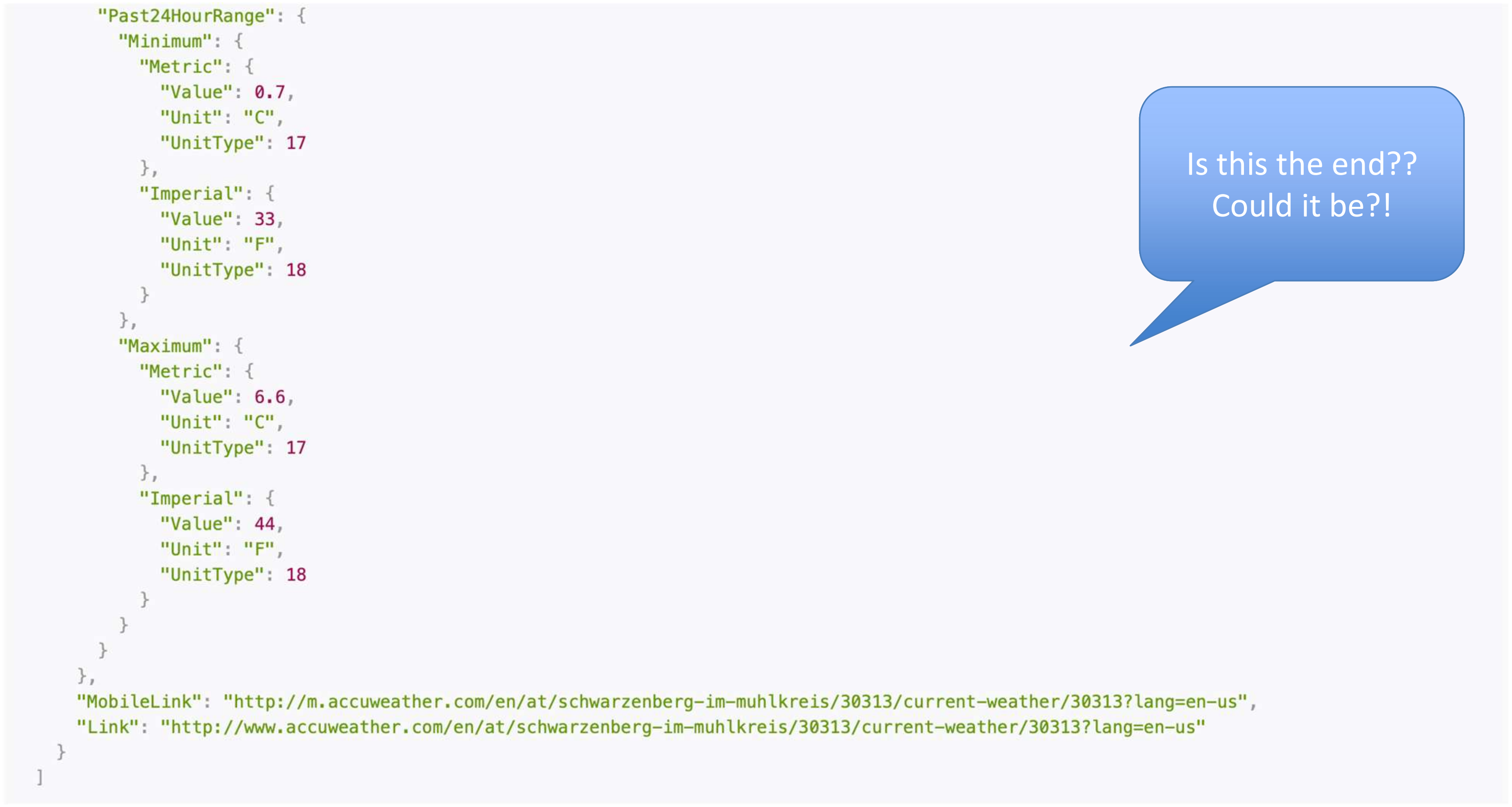
RESTful Response



```
      "Maximum": {
        "Metric": {
          "Value": 5.1,
          "Unit": "C",
          "UnitType": 17
        },
        "Imperial": {
          "Value": 41,
          "Unit": "F",
          "UnitType": 18
        }
      }
    },
    "Past12HourRange": {
      "Minimum": {
        "Metric": {
          "Value": 2,
          "Unit": "C",
          "UnitType": 17
        },
        "Imperial": {
          "Value": 36,
          "Unit": "F",
          "UnitType": 18
        }
      },
      "Maximum": {
        "Metric": {
          "Value": 5.6,
          "Unit": "C",
          "UnitType": 17
        },
        "Imperial": {
          "Value": 42,
          "Unit": "F",
          "UnitType": 18
        }
      }
    },
```

Help! I’m drowning in weather data!!

RESTful Response



```
      "Past24HourRange": {
        "Minimum": {
          "Metric": {
            "Value": 0.7,
            "Unit": "C",
            "UnitType": 17
          },
          "Imperial": {
            "Value": 33,
            "Unit": "F",
            "UnitType": 18
          }
        },
        "Maximum": {
          "Metric": {
            "Value": 6.6,
            "Unit": "C",
            "UnitType": 17
          },
          "Imperial": {
            "Value": 44,
            "Unit": "F",
            "UnitType": 18
          }
        }
      }
    },
    "MobileLink": "http://m.accuweather.com/en/at/schwarzenberg-im-muhlkreis/30313/current-weather/30313?lang=en-us",
    "Link": "http://www.accuweather.com/en/at/schwarzenberg-im-muhlkreis/30313/current-weather/30313?lang=en-us"
  }
]
```

RESTful Response

- No, just kidding. We’re really done now. ;-)
- This is what we call OVER-FETCHING.
- But what if we want to get the current conditions from all of our sites?

Under-Fetching

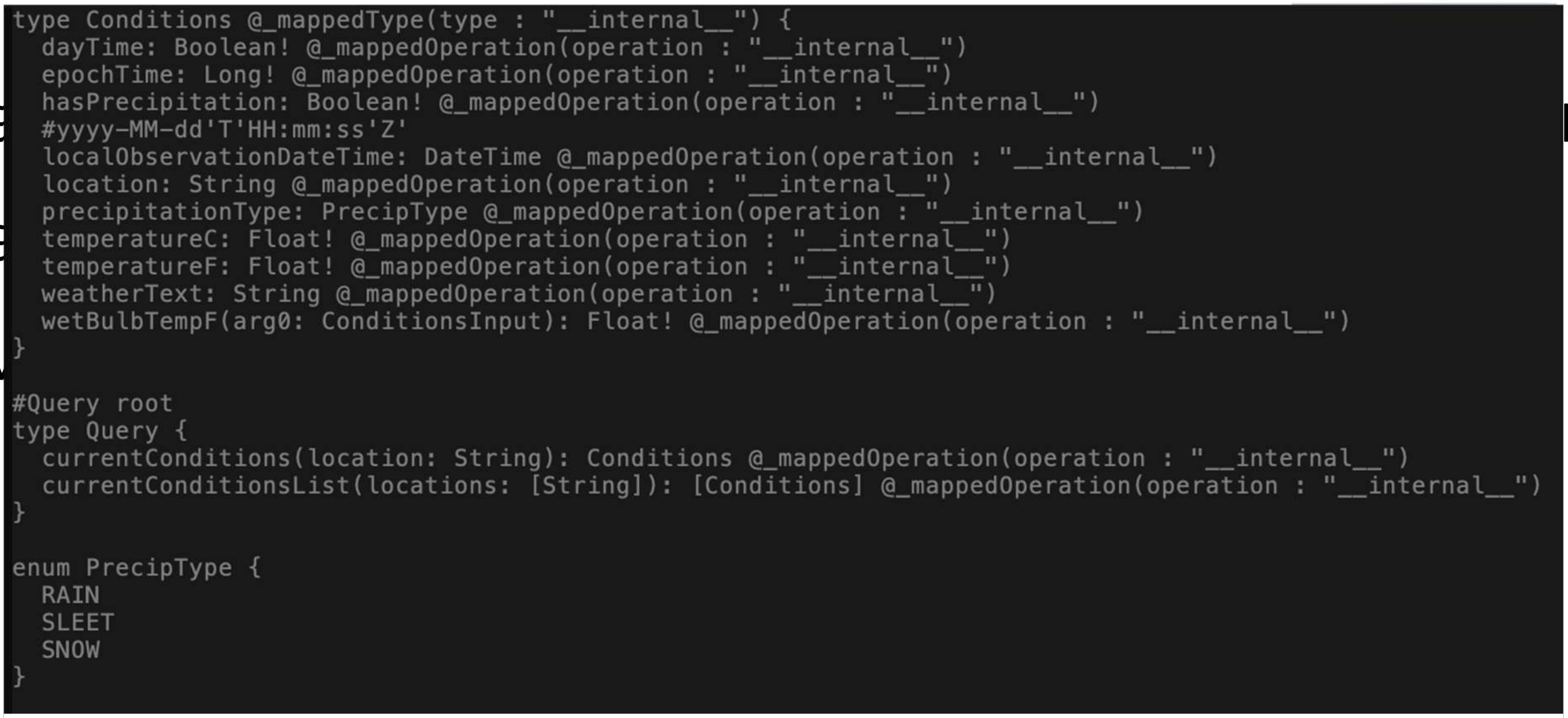
- Under-fetching is when we need to make multiple requests to get the data we want.
- So if we need to get the current conditions for three different locations, we would need to make three different REST requests.
- Now we get to sift through all that data – times three!!

No Thank You! GraphQL to the rescue!

- GraphQL was designed with the idea of reducing over- and under-fetching.
- GraphQL queries specify exactly what the client wants.
- Multiple queries can be submitted on the same request.

No Thank You! GraphQL to the rescue!

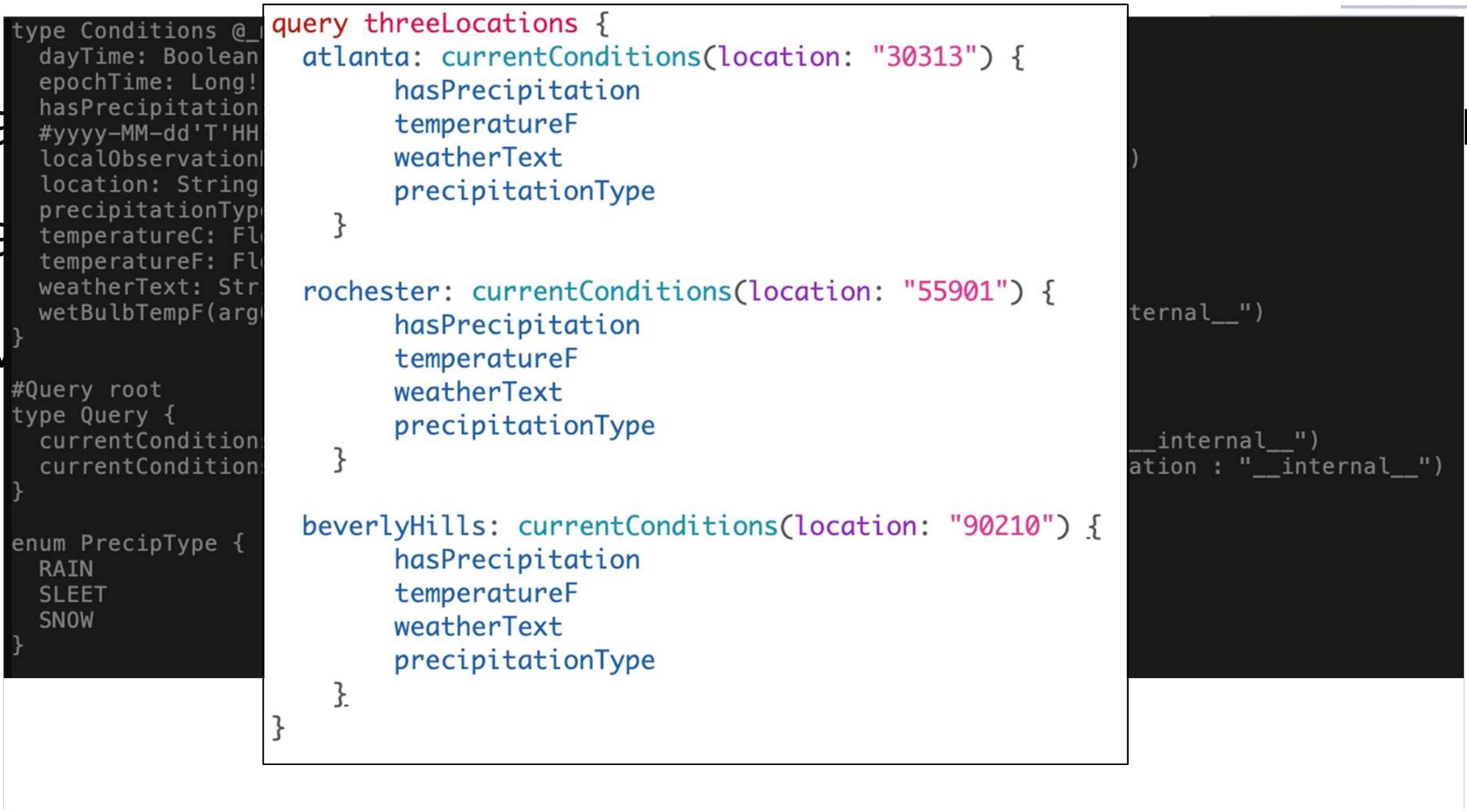
- G hing.
- G
- M



```
type Conditions @_mappedType(type : "__internal__") {
  dayTime: Boolean! @_mappedOperation(operation : "__internal__")
  epochTime: Long! @_mappedOperation(operation : "__internal__")
  hasPrecipitation: Boolean! @_mappedOperation(operation : "__internal__")
  #yyyy-MM-dd'T'HH:mm:ss'Z'
  localObservationDateTime: DateTime @_mappedOperation(operation : "__internal__")
  location: String @_mappedOperation(operation : "__internal__")
  precipitationType: PrecipType @_mappedOperation(operation : "__internal__")
  temperatureC: Float! @_mappedOperation(operation : "__internal__")
  temperatureF: Float! @_mappedOperation(operation : "__internal__")
  weatherText: String @_mappedOperation(operation : "__internal__")
  wetBulbTempF(arg0: ConditionsInput): Float! @_mappedOperation(operation : "__internal__")
}

#Query root
type Query {
  currentConditions(location: String): Conditions @_mappedOperation(operation : "__internal__")
  currentConditionsList(locations: [String]): [Conditions] @_mappedOperation(operation : "__internal__")
}

enum PrecipType {
  RAIN
  SLEET
  SNOW
}
```

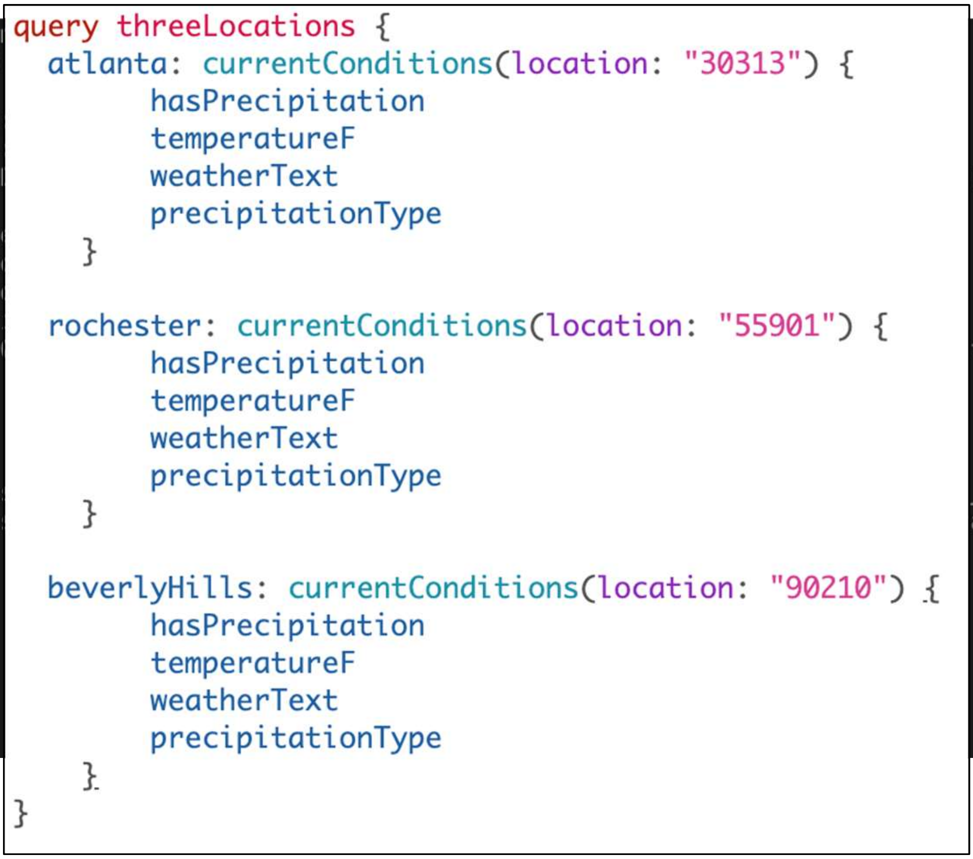
No Thank You! GraphQL to the rescue!

- G hing.
- G
- M



```
type Conditions @_
  dayTime: Boolean
  epochTime: Long!
  hasPrecipitation
  #yyyy-MM-dd'T'HH
  localObservation
  location: String
  precipitationTyp
  temperatureC: Fl
  temperatureF: Fl
  weatherText: Str
  wetBulbTempF(arg
}

#Query root
type Query {
  currentCondition
  currentCondition
}

enum PrecipType {
  RAIN
  SLEET
  SNOW
}
```

```
)
ternal__")
__internal__")
ation : "__internal__")
```

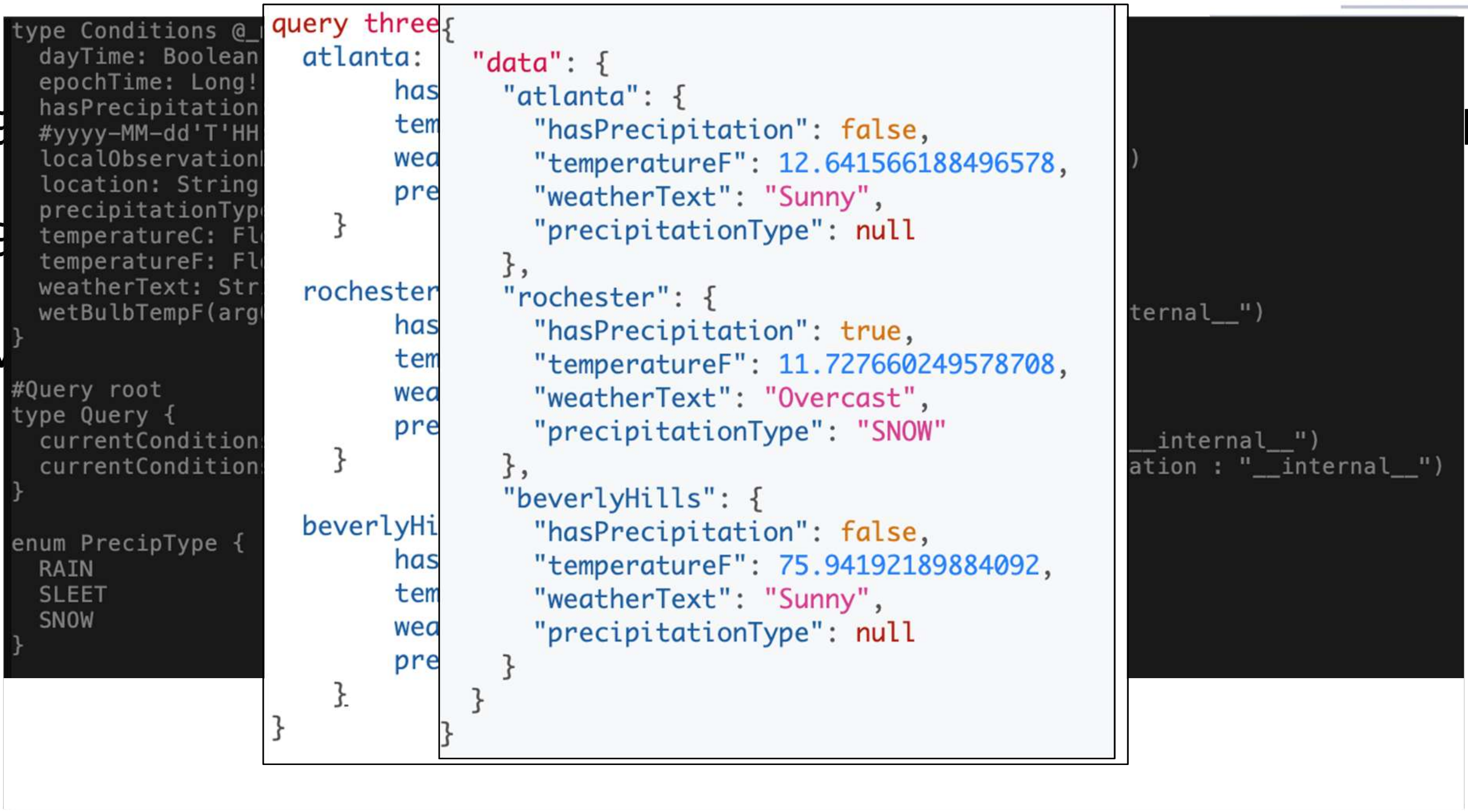


```
query threeLocations {
  atlanta: currentConditions(location: "30313") {
        hasPrecipitation
        temperatureF
        weatherText
        precipitationType
   }

  rochester: currentConditions(location: "55901") {
        hasPrecipitation
        temperatureF
        weatherText
        precipitationType
   }

  beverlyHills: currentConditions(location: "90210") {
        hasPrecipitation
        temperatureF
        weatherText
        precipitationType
   }
}
```

No Thank You! GraphQL to the rescue!

Open Liberty

- G … hing.
- G
- M



```
type Conditions @_
  dayTime: Boolean
  epochTime: Long!
  hasPrecipitation
  #yyyy-MM-dd'T'HH
  localObservation
  location: String
  precipitationTyp
  temperatureC: Fl
  temperatureF: Fl
  weatherText: Str
  wetBulbTempF(arg
}

#Query root
type Query {
  currentCondition
  currentCondition
}

enum PrecipType {
  RAIN
  SLEET
  SNOW
}
```

```
)
ternal__")
__internal__")
ation : "__internal__")
```

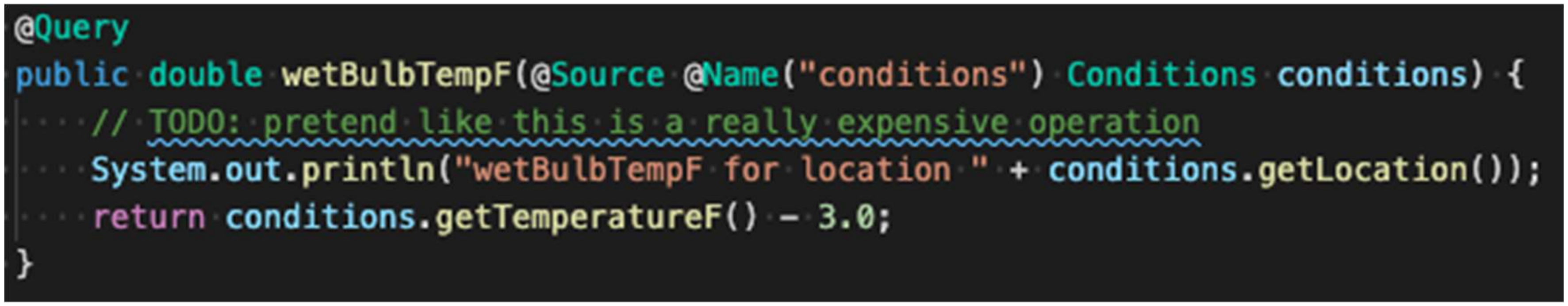
```
query three
  atlanta:
        has
        tem
        wea
        pre
    }

  rochester
        has
        tem
        wea
        pre
    }

  beverlyHi
        has
        tem
        wea
        pre
    }
}
```

```
{
  "data": {
    "atlanta": {
      "hasPrecipitation": false,
      "temperatureF": 12.641566188496578,
      "weatherText": "Sunny",
      "precipitationType": null
    },
    "rochester": {
      "hasPrecipitation": true,
      "temperatureF": 11.727660249578708,
      "weatherText": "Overcast",
      "precipitationType": "SNOW"
    },
    "beverlyHills": {
      "hasPrecipitation": false,
      "temperatureF": 75.94192189884092,
      "weatherText": "Sunny",
      "precipitationType": null
    }
  }
}
```

Let’s see the code!!

Only load what I want to load

- The `@Source` annotation allows apps to avoid expensive data lookups when the client doesn't want that data anyway!



```
@Query
public double wetBulbTempF(@Source @Name("conditions") Conditions conditions) {
    // TODO: pretend like this is a really expensive operation
    System.out.println("wetBulbTempF for location " + conditions.getLocation());
    return conditions.getTemperatureF() - 3.0;
}
```

Let’s see it work!!

What if we run into an exception?

- GraphQL supports partial results.
- For multiple queries, the successful results are returned while error data is returned for unsuccessful queries.
- Developers can throw a `GraphQLException` that contains partial results.

Let’s check it out!!

What’s next?

- Refinements / Clarifications / Fixes
- GraphQL Client APIs
- Subscriptions?
- Custom Scalars?

Summary

- GraphQL fills in some gaps in REST – over-fetching, under-fetching, partial results, etc.
- MicroProfile GraphQL makes it easy to develop and deploy GraphQL applications.
- Open is Awesome!!

Questions?

Open Liberty

Links and Coordinates

- MP GraphQL:
  https://github.com/eclipse/microprofile-graphql
  https://gitter.im/eclipse/microprofile-graphql

- Maven Coordinates:

```
<dependency>
    <groupId>org.eclipse.microprofile.graphql</groupId>
    <artifactId>microprofile-graphql-api</artifactId>
    <version>1.0.3</version>
</dependency>
```

- Sample app:
  https://github.com/OpenLiberty/sample-mp-graphql

- GraphQL-SPQR project
  https://github.com/leangen/graphql-spqr

- Andy McCright
  @AndrewMcCright – j.andrew.mccright@gmail.com – andymc@us.ibm.com

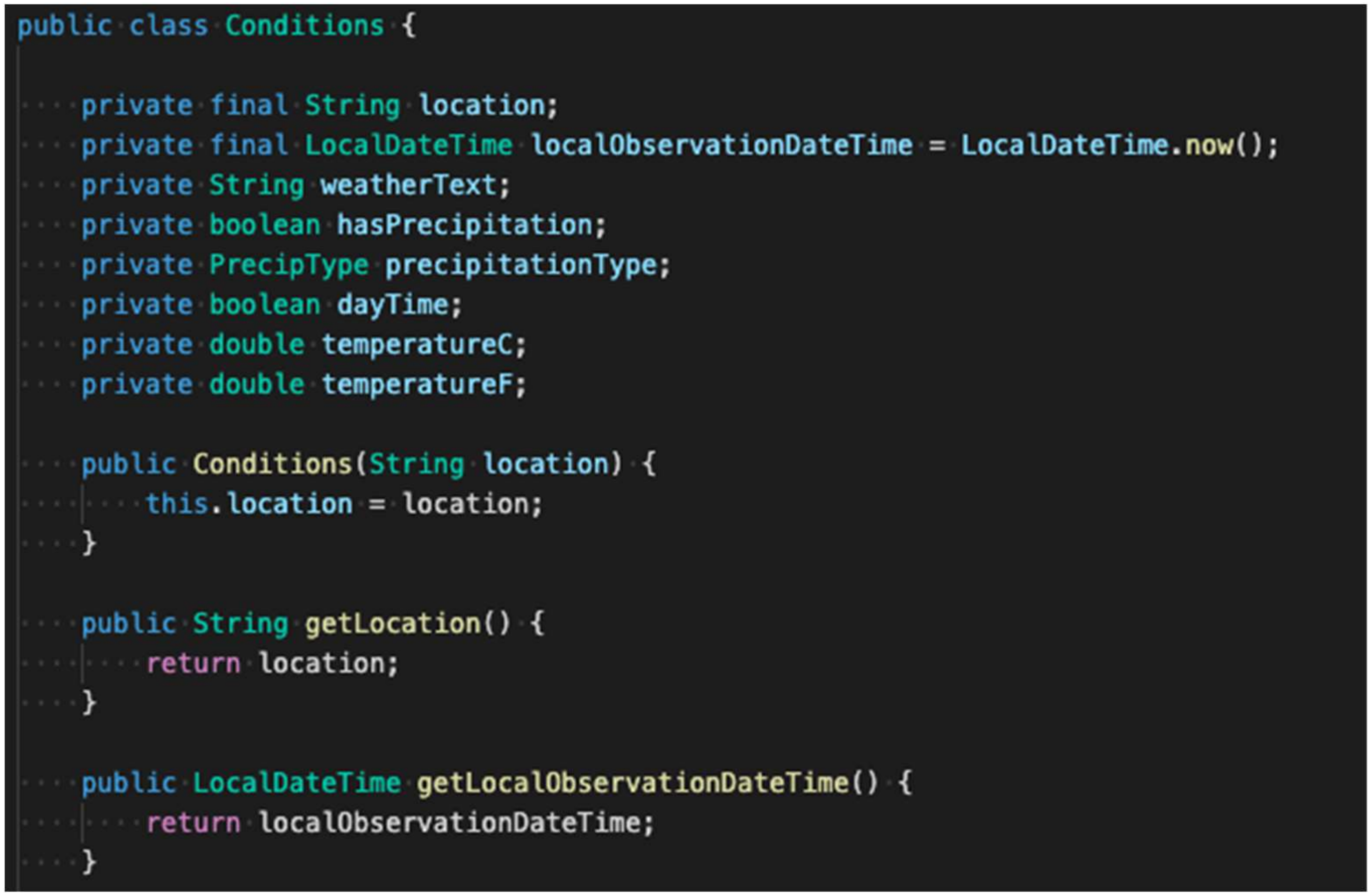
The End

Open Liberty

Backup

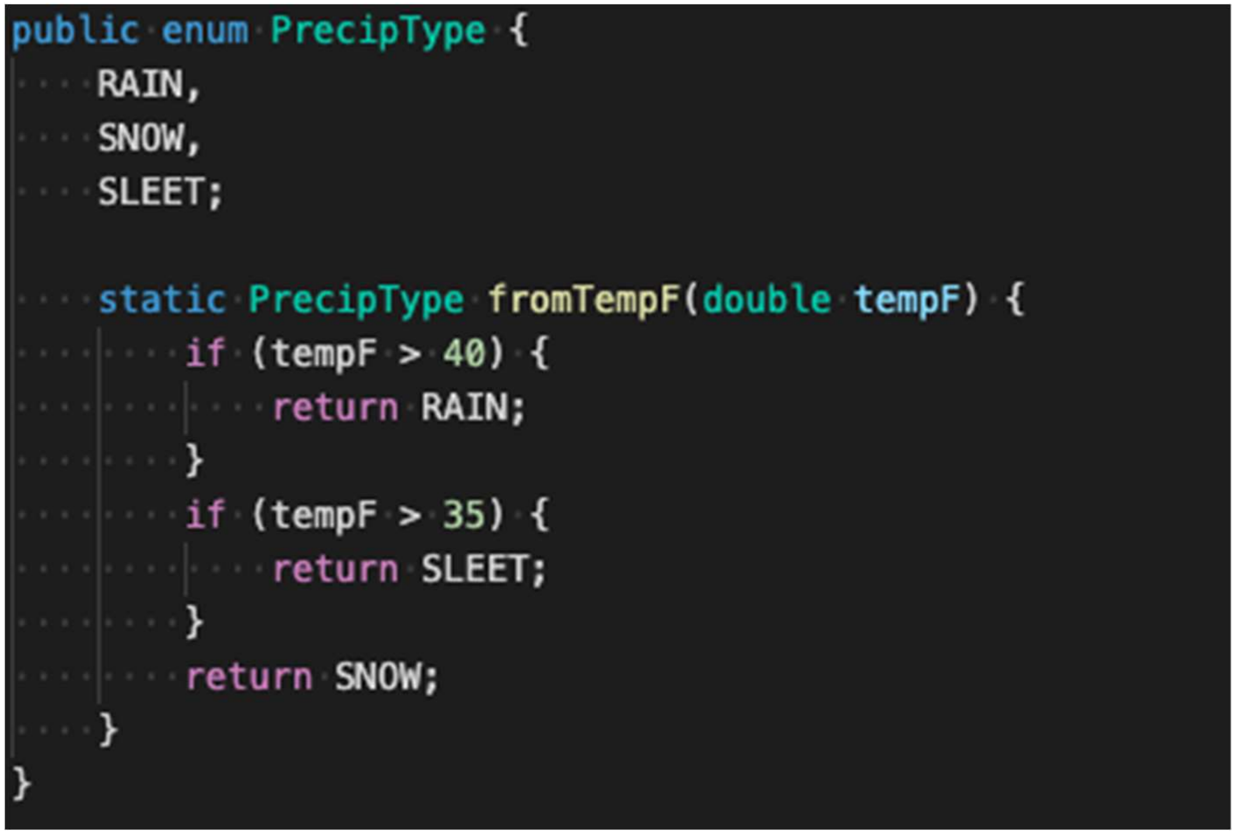
Open Liberty

Conditions.java



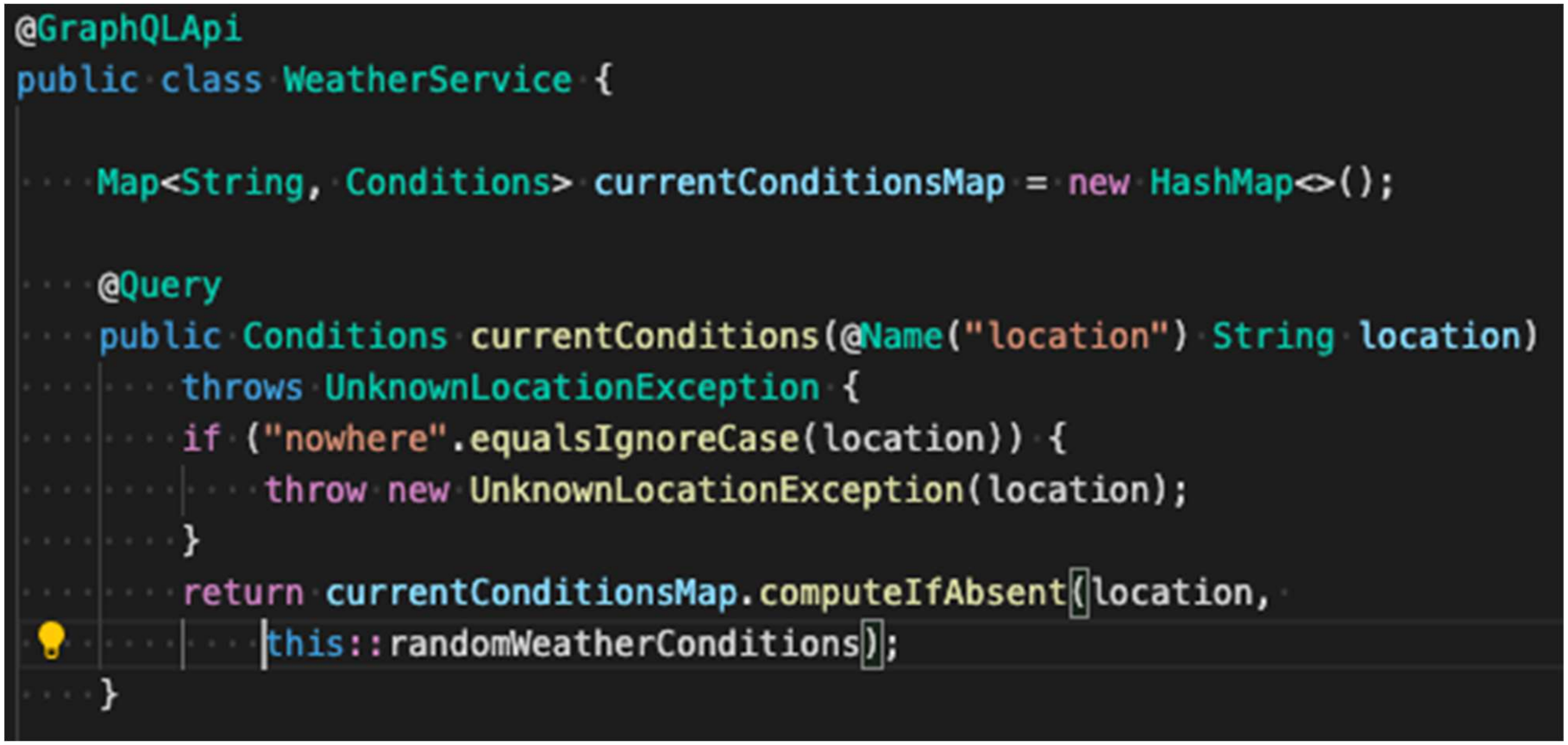
```java
public class Conditions {

    private final String location;
    private final LocalDateTime localObservationDateTime = LocalDateTime.now();
    private String weatherText;
    private boolean hasPrecipitation;
    private PrecipType precipitationType;
    private boolean dayTime;
    private double temperatureC;
    private double temperatureF;

    public Conditions(String location) {
        this.location = location;
    }

    public String getLocation() {
        return location;
    }

    public LocalDateTime getLocalObservationDateTime() {
        return localObservationDateTime;
    }
```

PrecipType.java



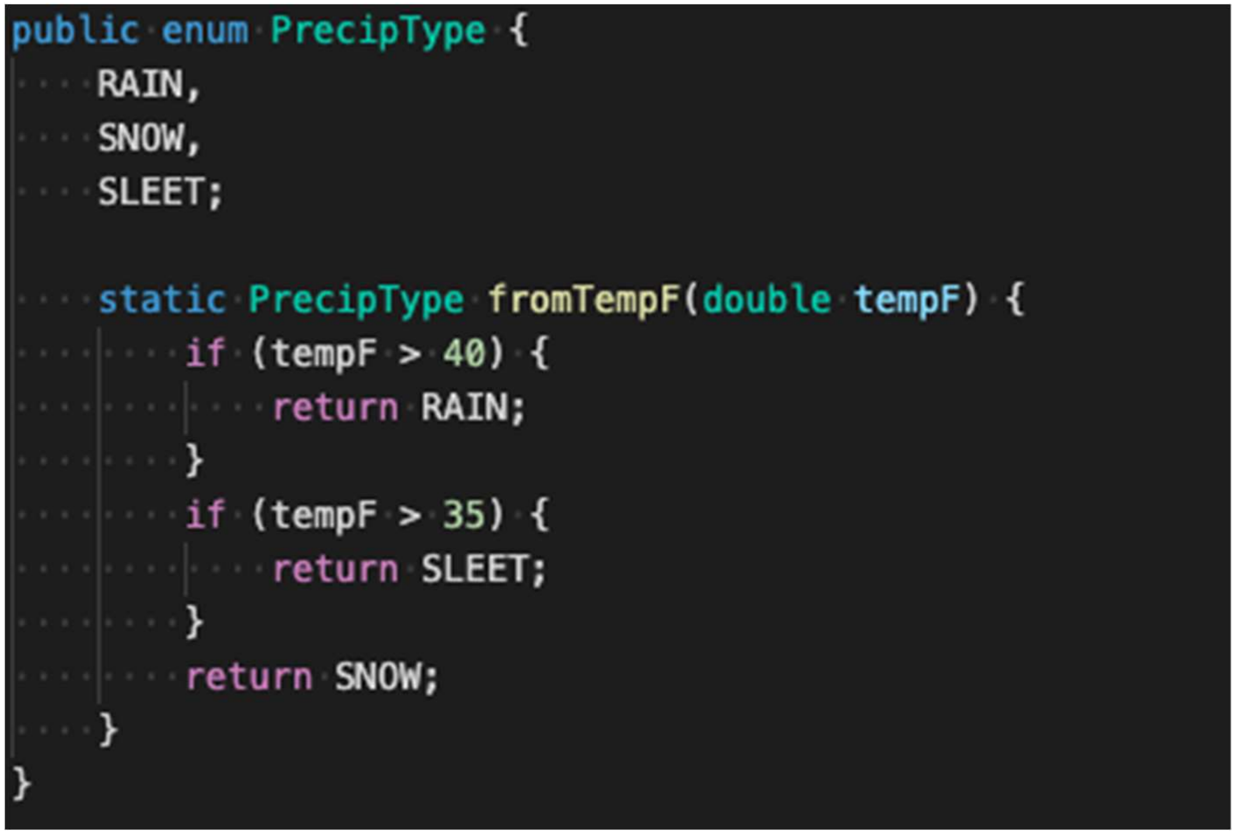
```java
public enum PrecipType {
    RAIN,
    SNOW,
    SLEET;

    static PrecipType fromTempF(double tempF) {
        if (tempF > 40) {
            return RAIN;
        }
        if (tempF > 35) {
            return SLEET;
        }
        return SNOW;
    }
}
```

WeatherService.java (part 1)



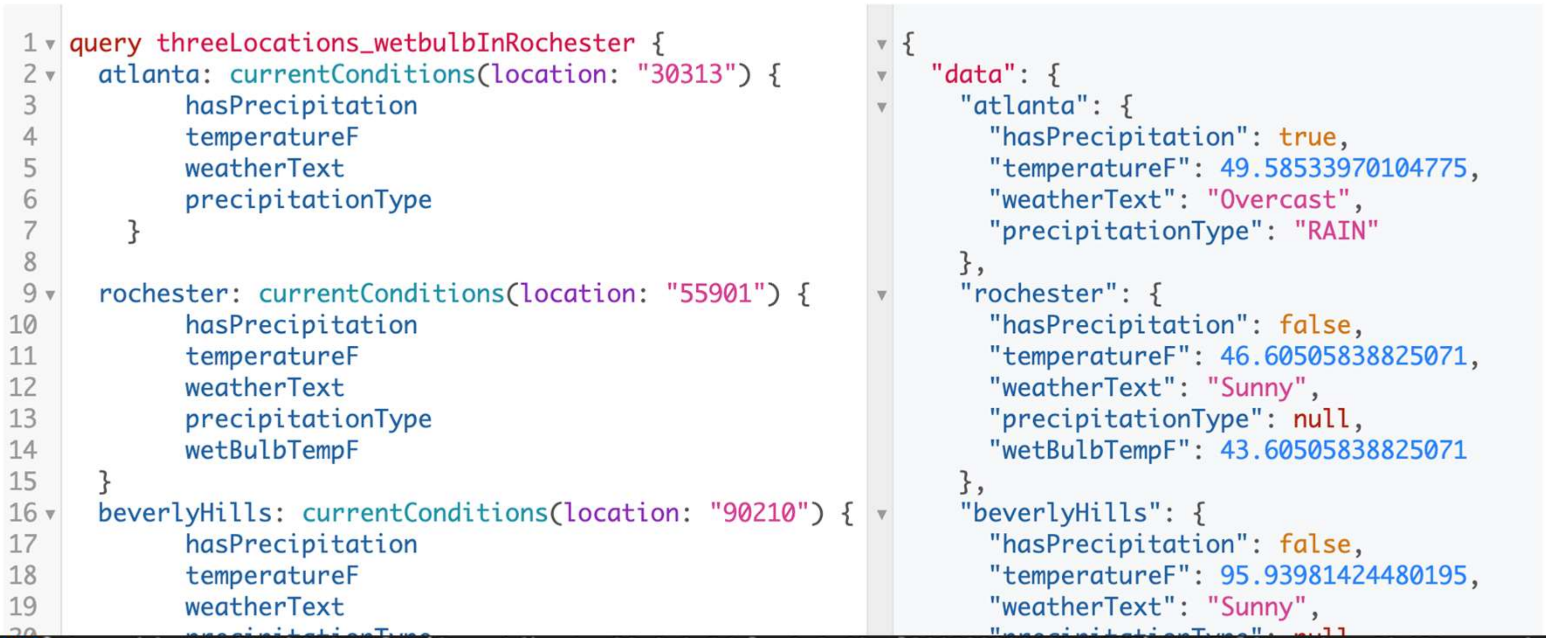
```
@GraphQLApi
public class WeatherService {

    Map<String, Conditions> currentConditionsMap = new HashMap<>();

    @Query
    public Conditions currentConditions(@Name("location") String location)
        throws UnknownLocationException {
        if ("nowhere".equalsIgnoreCase(location)) {
            throw new UnknownLocationException(location);
        }
        return currentConditionsMap.computeIfAbsent(location,
            this::randomWeatherConditions);
    }
```

PrecipType.java



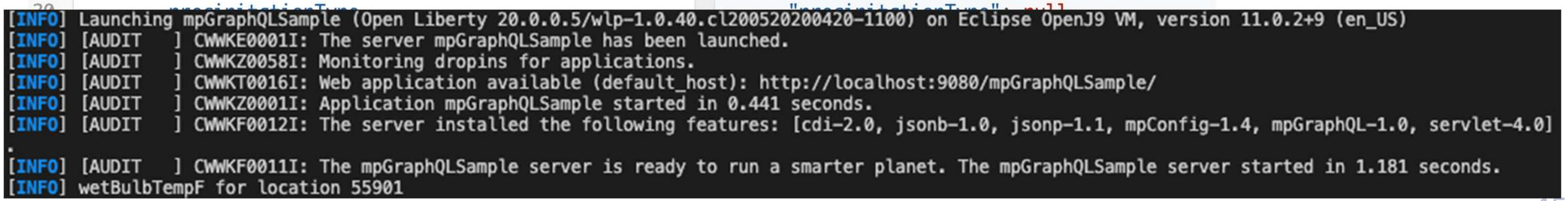
```java
public enum PrecipType {
    RAIN,
    SNOW,
    SLEET;

    static PrecipType fromTempF(double tempF) {
        if (tempF > 40) {
            return RAIN;
        }
        if (tempF > 35) {
            return SLEET;
        }
        return SNOW;
    }
}
```

Lookup 3 Cities – WetBulb in Rochester



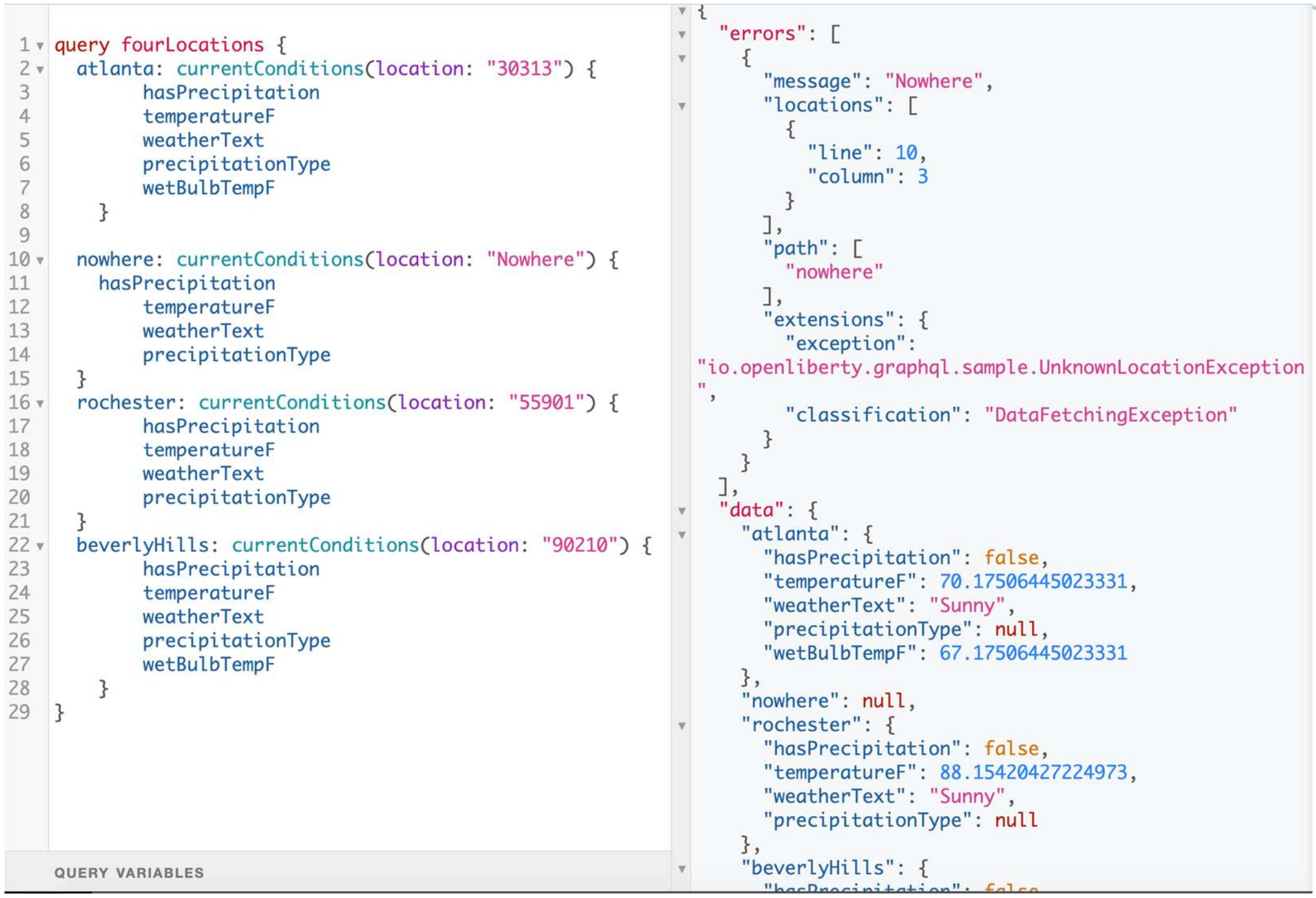
```
query threeLocations_wetbulbInRochester {
  atlanta: currentConditions(location: "30313") {
        hasPrecipitation
        temperatureF
        weatherText
        precipitationType
    }

  rochester: currentConditions(location: "55901") {
        hasPrecipitation
        temperatureF
        weatherText
        precipitationType
        wetBulbTempF
  }
  beverlyHills: currentConditions(location: "90210") {
        hasPrecipitation
        temperatureF
        weatherText
```

```
{
  "data": {
    "atlanta": {
      "hasPrecipitation": true,
      "temperatureF": 49.58533970104775,
      "weatherText": "Overcast",
      "precipitationType": "RAIN"
    },
    "rochester": {
      "hasPrecipitation": false,
      "temperatureF": 46.60505838825071,
      "weatherText": "Sunny",
      "precipitationType": null,
      "wetBulbTempF": 43.60505838825071
    },
    "beverlyHills": {
      "hasPrecipitation": false,
      "temperatureF": 95.93981424480195,
      "weatherText": "Sunny",
```



```
[INFO] Launching mpGraphQLSample (Open Liberty 20.0.0.5/wlp-1.0.40.cl200520200420-1100) on Eclipse OpenJ9 VM, version 11.0.2+9 (en_US)
[INFO] [AUDIT   ] CWWKE0001I: The server mpGraphQLSample has been launched.
[INFO] [AUDIT   ] CWWKZ0058I: Monitoring dropins for applications.
[INFO] [AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/mpGraphQLSample/
[INFO] [AUDIT   ] CWWKZ0001I: Application mpGraphQLSample started in 0.441 seconds.
[INFO] [AUDIT   ] CWWKF0012I: The server installed the following features: [cdi-2.0, jsonb-1.0, jsonp-1.1, mpConfig-1.4, mpGraphQL-1.0, servlet-4.0]
.
[INFO] [AUDIT   ] CWWKF0011I: The mpGraphQLSample server is ready to run a smarter planet. The mpGraphQLSample server started in 1.181 seconds.
[INFO] wetBulbTempF for location 55901
```

Error Handling Example



```
query fourLocations {
  atlanta: currentConditions(location: "30313") {
        hasPrecipitation
        temperatureF
        weatherText
        precipitationType
        wetBulbTempF
    }

  nowhere: currentConditions(location: "Nowhere") {
    hasPrecipitation
        temperatureF
        weatherText
        precipitationType
  }
  rochester: currentConditions(location: "55901") {
        hasPrecipitation
        temperatureF
        weatherText
        precipitationType
  }
  beverlyHills: currentConditions(location: "90210") {
        hasPrecipitation
        temperatureF
        weatherText
        precipitationType
        wetBulbTempF
    }
}
```

QUERY VARIABLES

```
{
  "errors": [
    {
      "message": "Nowhere",
      "locations": [
        {
          "line": 10,
          "column": 3
        }
      ],
      "path": [
        "nowhere"
      ],
      "extensions": {
        "exception":
"io.openliberty.graphql.sample.UnknownLocationException
",
        "classification": "DataFetchingException"
      }
    }
  ],
  "data": {
    "atlanta": {
      "hasPrecipitation": false,
      "temperatureF": 70.17506445023331,
      "weatherText": "Sunny",
      "precipitationType": null,
      "wetBulbTempF": 67.17506445023331
    },
    "nowhere": null,
    "rochester": {
      "hasPrecipitation": false,
      "temperatureF": 88.15420427224973,
      "weatherText": "Sunny",
      "precipitationType": null
    },
    "beverlyHills": {
```