

# Visualizing Queries

**A User-Friendly CQL2 Filter Builder for Geospatial Data**

---

Noam Rabi, July 2025



# Hi, I'm Noam

- Senior Front-End Engineer at UP42
- Based in **Berlin**
- My favourite ice cream flavour is **Chocolate**

# Why visualize queries?

We want to make  
exploring geospatial  
datasets easier.

***Web interfaces  
are good for that!***

The screenshot displays the UP42 console search interface. At the top, there is a search bar with a magnifying glass icon and a '18' indicator. Below the search bar, there are two radio buttons: 'My account' (selected) and 'My workspace'. The interface is divided into several sections with filters:

- Creation date:** Includes 'From' and 'To' date pickers.
- Acquisition date:** Includes 'From' and 'To' date pickers.
- Ground sampling distance:** A horizontal slider ranging from 0 m to 100 m, with intermediate markers at 2.5 m, 10 m, and 30 m.
- Maximum cloud coverage:** A horizontal slider ranging from 0% to 100%.
- Data product, provider, and collection:** A text input field with the placeholder 'Search by data product, provider, and collection'.
- Order IDs:** A text input field with the placeholder 'Search by complete order IDs'.
- Asset IDs:** A text input field with the placeholder 'Search by complete asset IDs'.
- Tags:** Two text input fields labeled 'Include' and 'Exclude', both with the placeholder 'Search by complete tags'.
- Source:** Three checkboxes: 'Catalog' (checked), 'Processing', and 'Tasking'.

At the bottom, there is a 'Clear' button on the left and a 'Search' button on the right.

# Why visualize queries?

How to translate the form to a query?

Unfortunately, there was no JavaScript library to help me...

So I made one.

Microsoft | Planetary Computer Explore Data Catalog

Explore datasets Advanced | Clear

Landsat Collection 2 Level-2

Custom filters Select filters

Acquired	08/22/1982 - 06/11/2025	<input checked="" type="checkbox"/> Acquired
Gsd	Greater than 0	<input type="checkbox"/> Description
WRS Row	(Empty)	<input checked="" type="checkbox"/> Cloud Cover
Cloud Cover	Under 50%	<input checked="" type="checkbox"/> Gsd
Platform	Include all	<input type="checkbox"/> Item ID
Off Nadir	Under 45.00°	<input type="checkbox"/> Instrument
Sun Azimuth	Under 180.00°	<input type="checkbox"/> Cloud Cover Land
		<input type="checkbox"/> Collection Cat.
		<input type="checkbox"/> Collection Number
		<input type="checkbox"/> Correction
		<input type="checkbox"/> Scene ID
		<input type="checkbox"/> WRS Path
		<input checked="" type="checkbox"/> WRS Row
		<input type="checkbox"/> Wrs Type
		<input checked="" type="checkbox"/> Platform
		<input type="checkbox"/> Data DOI
		<input checked="" type="checkbox"/> Off Nadir
		<input checked="" type="checkbox"/> Sun Azimuth
		<input type="checkbox"/> Sun Elevation

Natural color

**Landsat Collection 2 Level-2**  
Sorry, no items matched your filter. Try adjusting the query parameters or expand the map area.

[Planetary Computer](#)

**Filters**

**Temporal Extent**

All times in Coordinated Universal Time (UTC).

**Spatial Extent**

☐ Filter by spatial extent

**Additional filters**

☒ Match all filters (and) ☐ Match any filters (or)

Add filter

**Item ID** = foo

**Acquired** < 2025-06-10 09:4

All times in Coordinated Universal Time (UTC).

**Sort**

Default

Not all of the options may be supported.

**Items per page**

Default (12)

Number of items requested per page, max. 1000 items.

[STAC Browser](#)

# CQL2-filters-parser library

- **Unified** way to parse CQL2 Text and JSON encodings
- Runs in **Browsers** and **JavaScript** hosts
- **Extendable** to your needs

## CQL2 text or JSON

## Filter Builder

```
city='Mostar' AND T_EQUALS(talk_date, DATE('2025-07-16'))
```



# CQL2 filters

What is CQL2 and why is it helpful?

# STAC vs CQL2

## STAC

### SpatioTemporal Asset Catalog

a way to organize geospatial data

- STAC Item
- STAC Catalog

[Spec](#)

## CQL2

### Common Query Language

a language to describe filter expressions for spatial and temporal data

[Spec](#)



## Two encodings

CQL2 comes in two encodings,  
**Text** and **JSON**

- Query language
- Only operators and operands
- No flow control, loops, recursion

```
// CQL2 Text encoding  
city = 'Mostar'
```

```
// CQL2 JSON encoding  
{  
  "op": "=",  
  "args": [  
    { "property": "city" },  
    "Mostar"  
  ]  
}
```



## Two encodings

Differences:

- Audience
- Usage

```
// CQL2 Text encoding  
city = 'Mostar'
```

```
// CQL2 JSON encoding  
{  
  "op": "=",  
  "args": [  
    { "property": "city" },  
    "Mostar"  
  ]  
}
```

## Building blocks

- **Literals:** strings, numbers, booleans, etc.
- **Properties:** the variables of CQL2
- **Operators:** logical, comparison, arithmetic
- **Combining:** binary expression, function, etc.
- **Spatial:** bbox, geometries
- **Temporal:** timestamp, date, interval

All of these nodes expressions in a tree data structure.

```
// CQL2 Text encoding  
city = 'Mostar'
```

```
// CQL2 JSON encoding  
{  
  "op": "=",  
  "args": [  
    { "property": "city" },  
    "Mostar"  
  ]  
}
```



# **Parsing Text, Parsing JSON**

# Parsers

Text → tokenizer → parser → Expression tree

JSON → depth first parser → Expression tree

parse() → Expression tree

# Parsers

```
import { parse } from "cql2-filters-parser";

const { encoding, expression } = parse("city='Mostar'");

console.log(encoding); // -> Text
console.log(expression.toText()); // -> "city = 'Mostar'"
console.log(expression.toJSON()); // ->
// {
//   op: '=',
//   args: [ { property: 'city' }, 'Mostar' ]
// }
```



**Visitors**  
Especially for this library :)  
**welcome**

# Visitor design pattern

Separates the operation from the object.  
Allows defining new operations on data structure.



# Visitor design pattern

Each node type has a corresponding visit function:

- Literal → visit**Literal**Expression()
- Operator → visit**Operator**Expression()
- Property → visit**Property**Expression()
- Binary expression → visit**Binary**Expression()

The **visitor object** is something that **implements visit functions**

# HTML Builder Visitor

The **visitor object** is something that **implements visit functions**

```
const HTMLBuilderVisitor = {  
  visitLiteralExpression(expr) { /* ToDo */ },  
  visitOperatorExpression(expr) { /* ToDo */ },  
  visitPropertyExpression(expr) { /* ToDo */ },  
  visitBinaryExpression(expr) { /* ToDo */ },  
};  
  
const { expression } = parse("city='Mostar'");  
const builderForm = expression.accept(HTMLBuilderVisitor);  
document.getElementById("builder").appendChild(builderForm);
```

# HTML Builder Visitor

```
const HTMLBuilderVisitor = {
  visitLiteralExpression(expr) {
    return createInputElement(expr.value, expr.type);
  },
  visitOperatorExpression(expr) {
    return createOperatorSelectElement(expr.text);
  },
  visitPropertyExpression(expr) {
    return createInputElement(expr.name, "text");
  },
  visitBinaryExpression(expr) {
    const left = expr.left.accept(BuilderVisitor);
    const op = expr.operator.accept(BuilderVisitor);
    const right = expr.right.accept(BuilderVisitor);
    return createBinaryPairElement(left, op, right);
  },
};

const builderForm = expression.accept(HTMLBuilderVisitor);
```

# HTML Builder Visitor

```
const HTMLBuilderVisitor = {  
  visitLiteralExpression(expr) {  
    return createLiteralElement(expr.value);  
  },  
  visitOperatorExpression(expr) {  
    return createOperatorElement(expr.operator, expr.left, expr.right);  
  },  
  visitPropertyAccessExpression(expr) {  
    return createPropertyAccessElement(expr.object, expr.property);  
  },  
  visitBinaryExpression(expr) {  
    const left = expr.left.accept(this);  
    const operator = expr.operator;  
    const right = expr.right.accept(this);  
    return createBinaryElement(left, operator, right);  
  },  
};
```

## CQL2 Text

city='Mostar' AND talk\_duration > 20

## Filter Builder

city

=

Mostar

and

talk\_duration

>

20

```
const builderForm = expression.accept(HTMLBuilderVisitor);
```



# Wrapping up

Something funny to add here

## WRAPPING UP

# CQL2-filters-parser library

[GitHub repository](#)

- **Unified** way to parse CQL2 Text and JSON encodings
- Runs in **Browsers** and **JavaScript** hosts
- **Extendable** to your needs



## CQL2 text or JSON

Empty ▾

## Filter Builder

```
city='Mostar' AND T_EQUALS(talk_date, DATE('2025-07-16'))
```

AND ▾

Convert to rule

Property ▾

city

= ▾

String ▾

Mostar

Convert to AND / OR

T\_EQUALS

Property ▾

talk\_date

Date ▾

16.07.2025



CQL2 Playground - <https://noamra.github.io/ogc-cql2-filters>

GitHub repository - <https://github.com/NoamRa/ogc-cql2-filters>

npm i **cql2-filters-parser**