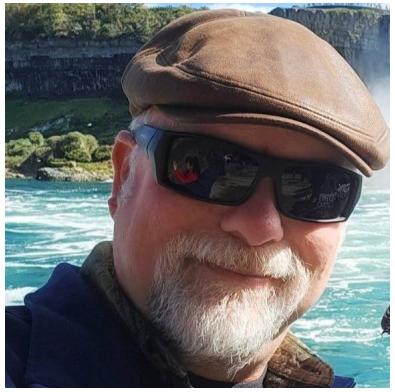

NIAGARA
SUMMIT
2026
SEAMLESS CONNECTIVITY,
POWERFUL INTELLIGENCE
TRIDIUM

Building Tomorrow : HTML5 UX Development and API Integration



James Johnson
Tridium

Charles Johson
Lynxspring

TRIDIUM

This presentation includes content from both Tridium and external speakers. Views expressed by external speakers are their own and do not necessarily reflect the views of Tridium Inc. or its affiliates. Reference to any third-party product, service, customer, or integration does not constitute endorsement by Tridium. Each speaker is responsible for obtaining any required rights or permissions for third-party content included in their portion of this presentation.

Any descriptions of future product direction, intended updates, or new or improved features or functions are for informational purposes only, are subject to change, and do not create any commitment, obligation, or warranty by Tridium. This document contains valuable proprietary and confidential information of Tridium and must not be disclosed to any third party without our written agreement. Content provided by Tridium may not be altered or modified and must remain in the format originally presented. All demos included in this presentation are for informational purposes only.

TRIDIUM

Agenda

- What are Web APIs?
- Niagara APIs
- API Integration Examples
- HTML User Interface Examples

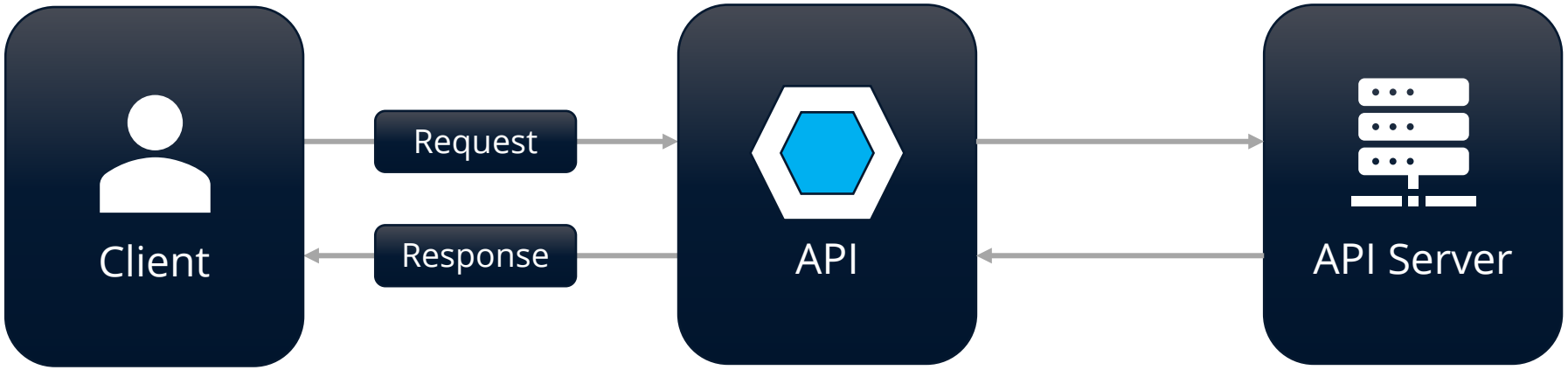
What are Web APIs?

NS26

TRIDIUM

Application Programming Interface (API)

- Well-defined tools that allow software applications to communicate and interact with each other.
- Introduced in the early 2000s and leveraged by companies such as Salesforce, eBay, Amazon, etc.
- Used in social media applications such as Facebook, LinkedIn, etc.
- Since ~2010 used widely in the IoT industry to connect everyday devices such as thermostats, audio visual equipment, cameras, and sensors to the cloud.

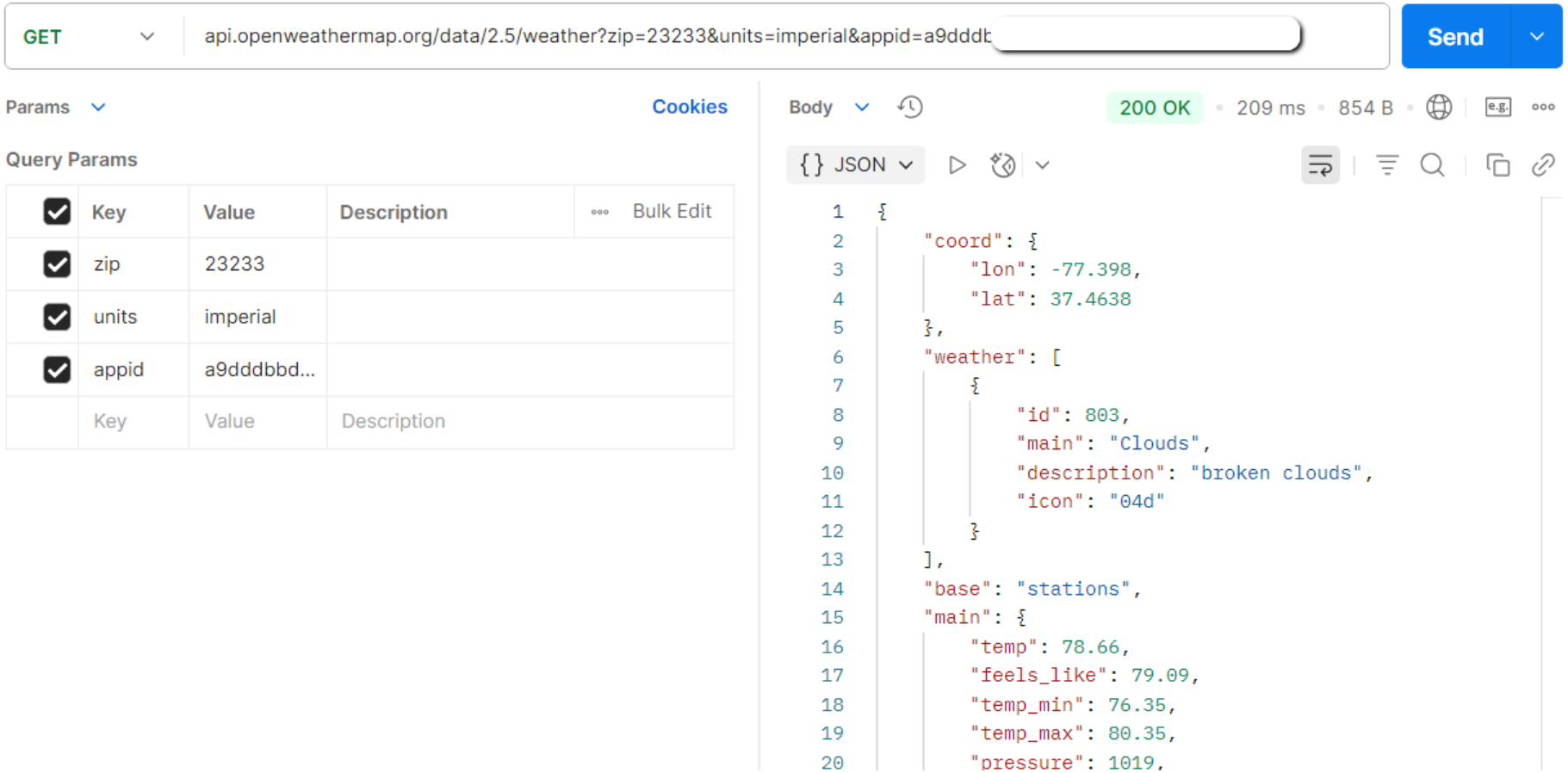


API Types

- **Simple Object Access Protocol (SOAP)** – client and server exchange messages using XML
- **Remote Procedure Calls (RPC)** – client calls a software function, and the server sends the output back to the client.
- **Representational State Transfer (REST)** – the most popular and flexible APIs used today. The client and server exchange data using HTTP.
- **WebSocket** – two-way communication between client and server that uses JSON objects, making it more efficient than REST.

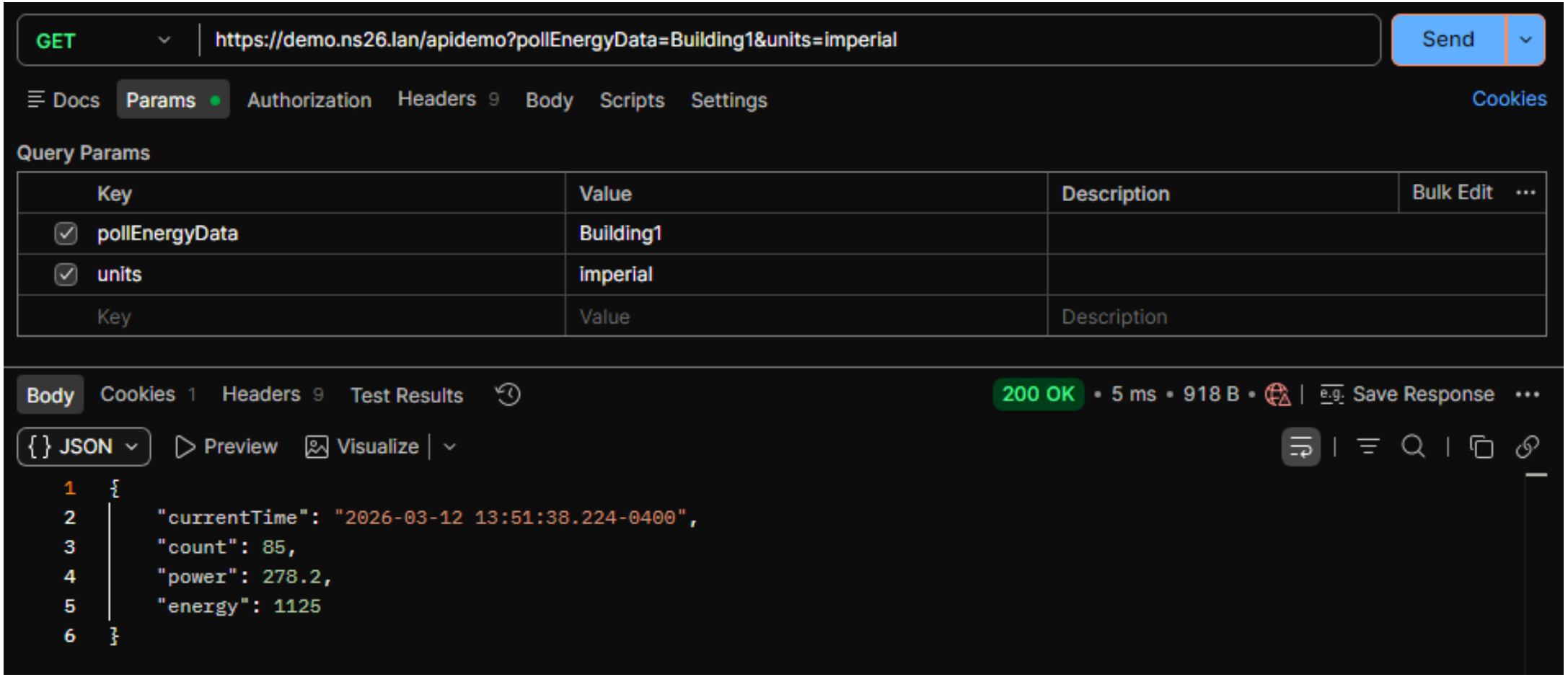
Hypertext Transfer Protocol (HTTP)

- Enables communication between clients and servers over a network.
- Client sends a request, and the server replies with a response.



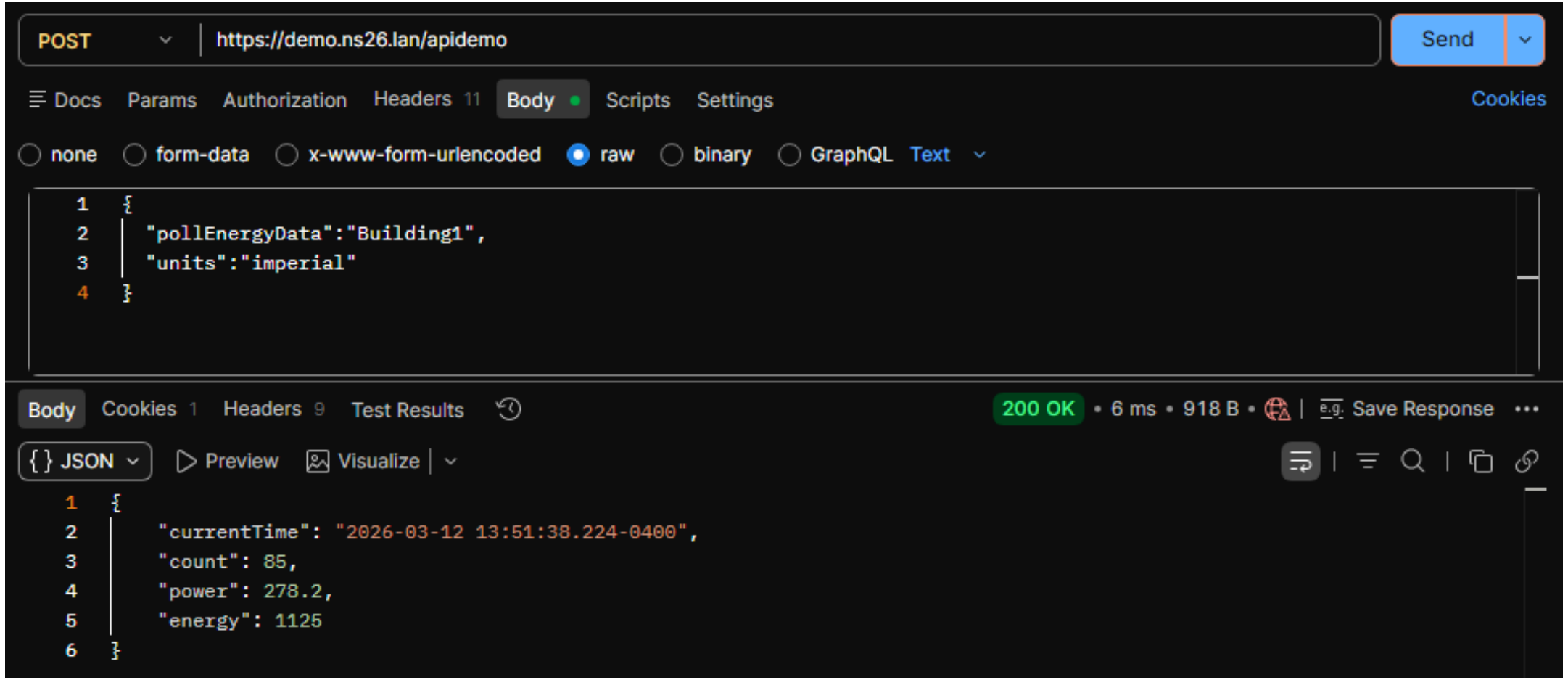
HTTP GET

- GET is used to request data with parameters encoded in the URL as key value pairs.



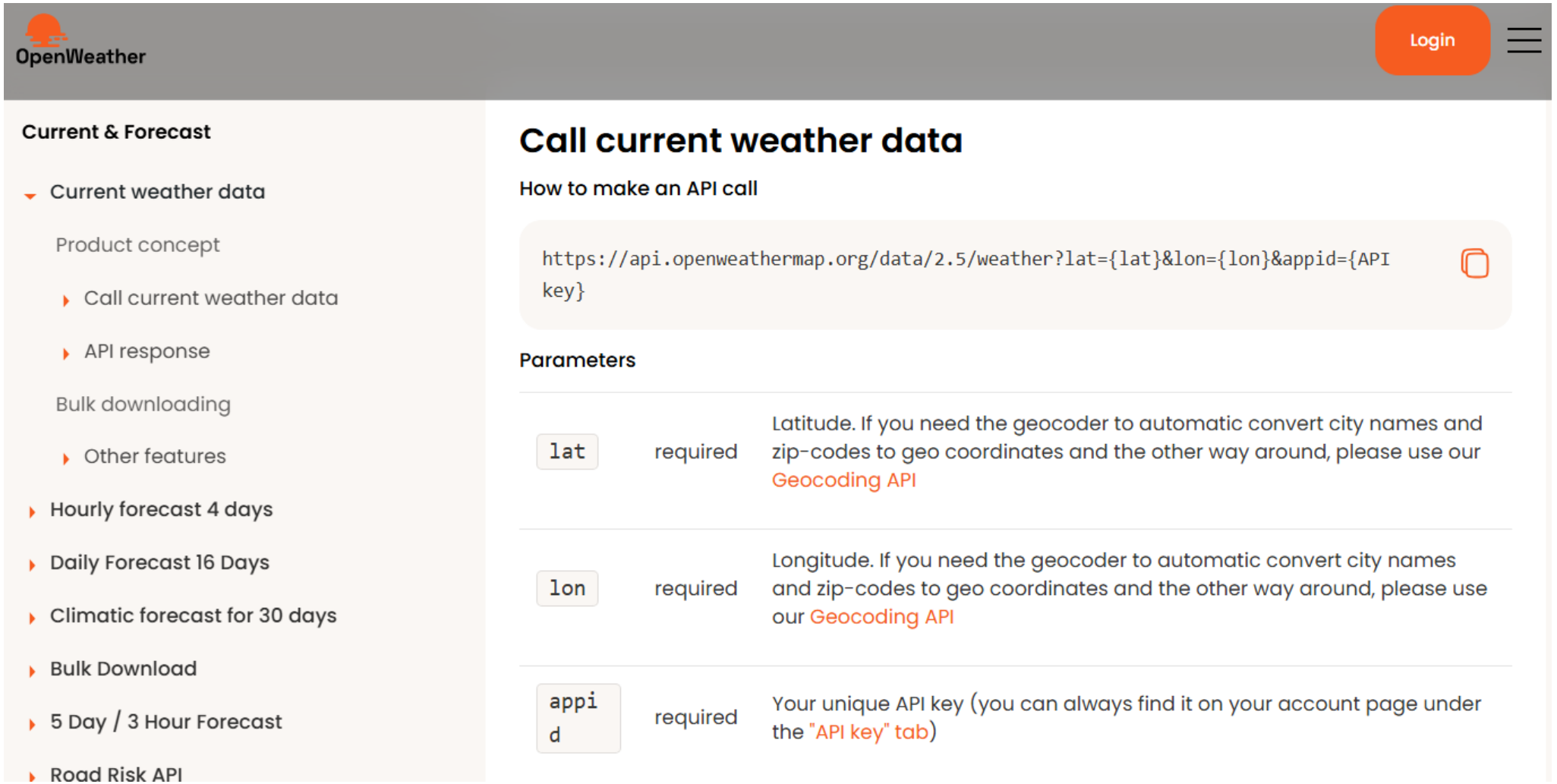
HTTP POST

- POST is used to send data to a server to create or update a resource with data stored in the request body.



API Documentation Example

https://openweathermap.org/current?collection=current_forecast



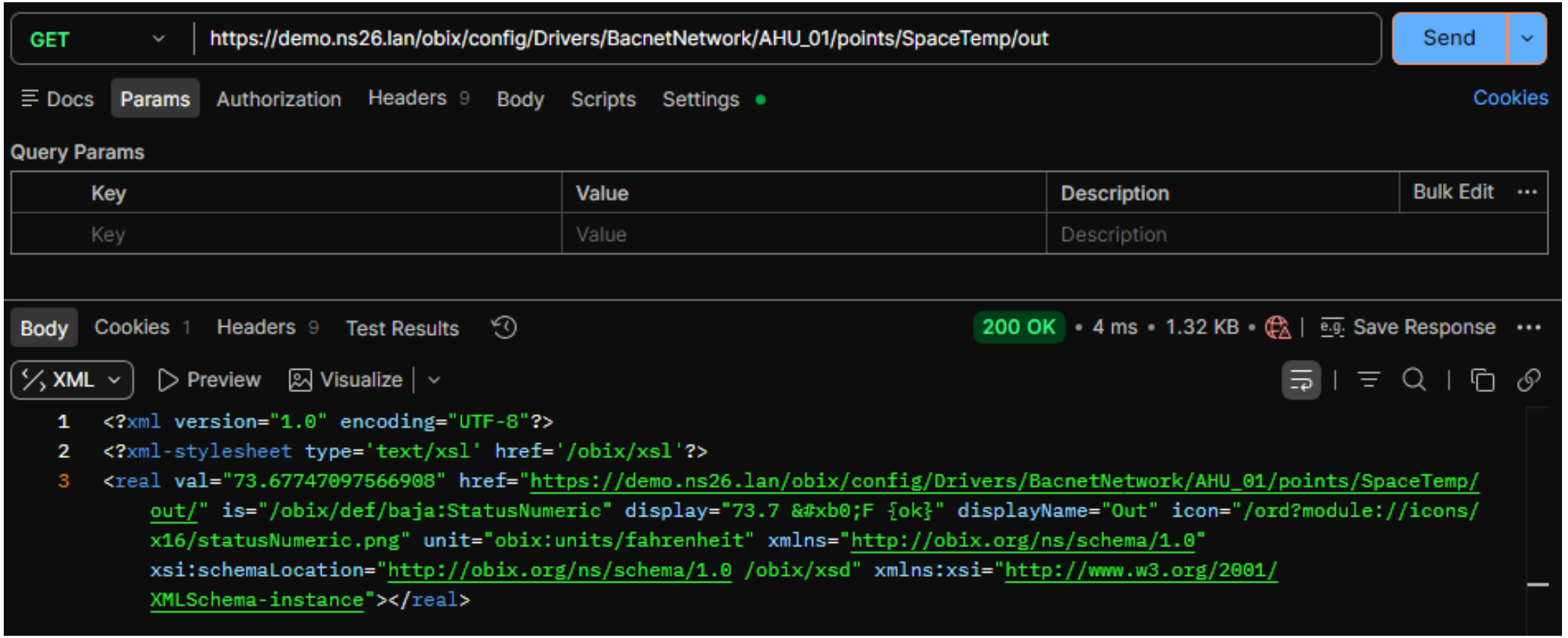
Niagara APIs

Niagara – API Data Access

- oBIX
- Niagara Analytics web API
- Json Toolkit Json Schema
- HTTP Client driver String Servlet
- Custom Servlet
- Niagara Cloud Suite (NCS) using Niagara Data Service (NDS)
  tomorrow @1pm – Cloudy with a Chance of Disruption session

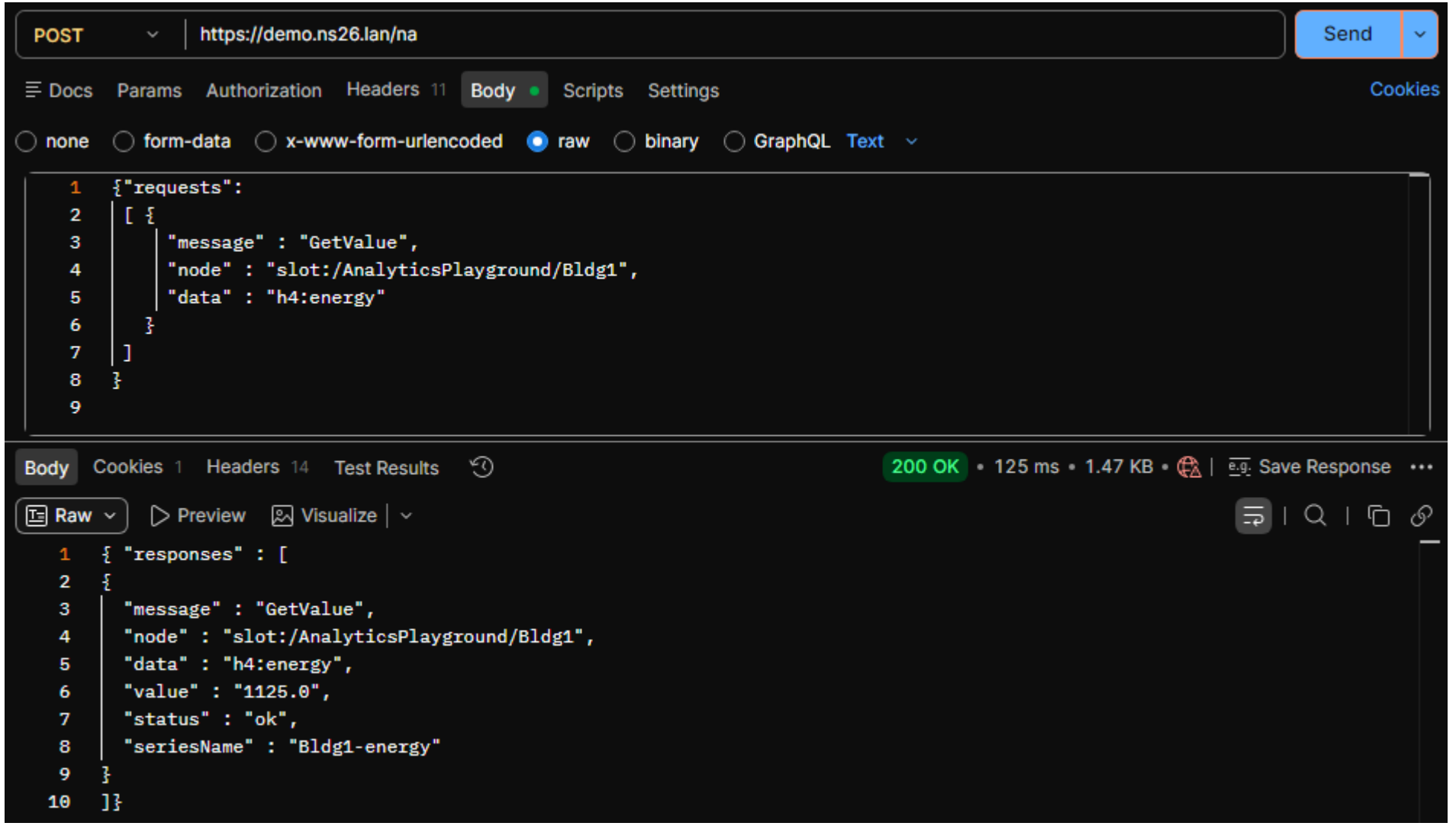
oBix – HTTP GET Example

- Used to read/write live control points, subscribe to alarm events, or query history data.



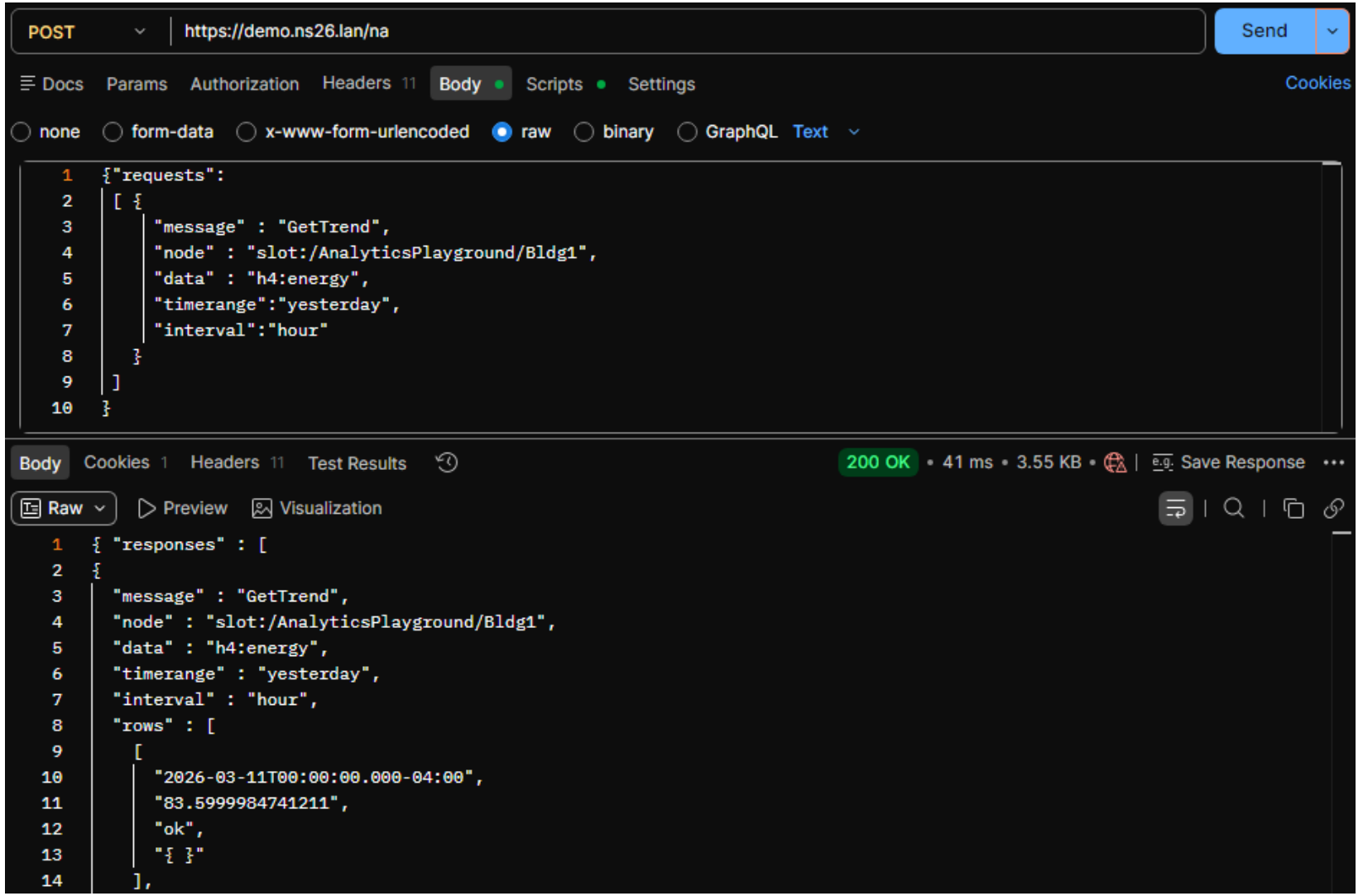
Niagara Analytics – Value Request

- Used to read control point values from the station.



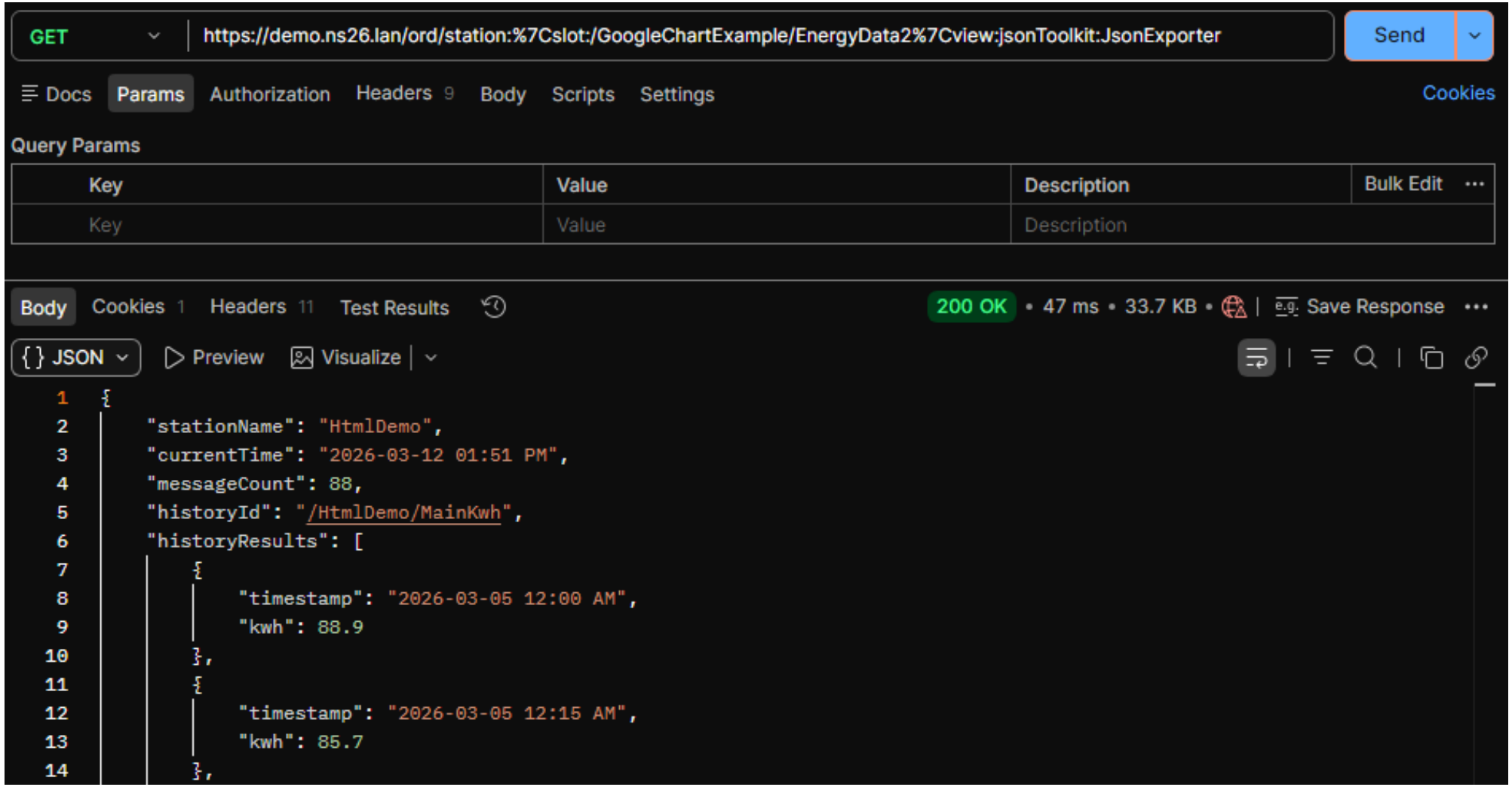
Niagara Analytics – Trend Request

- Used to query history data from the station.

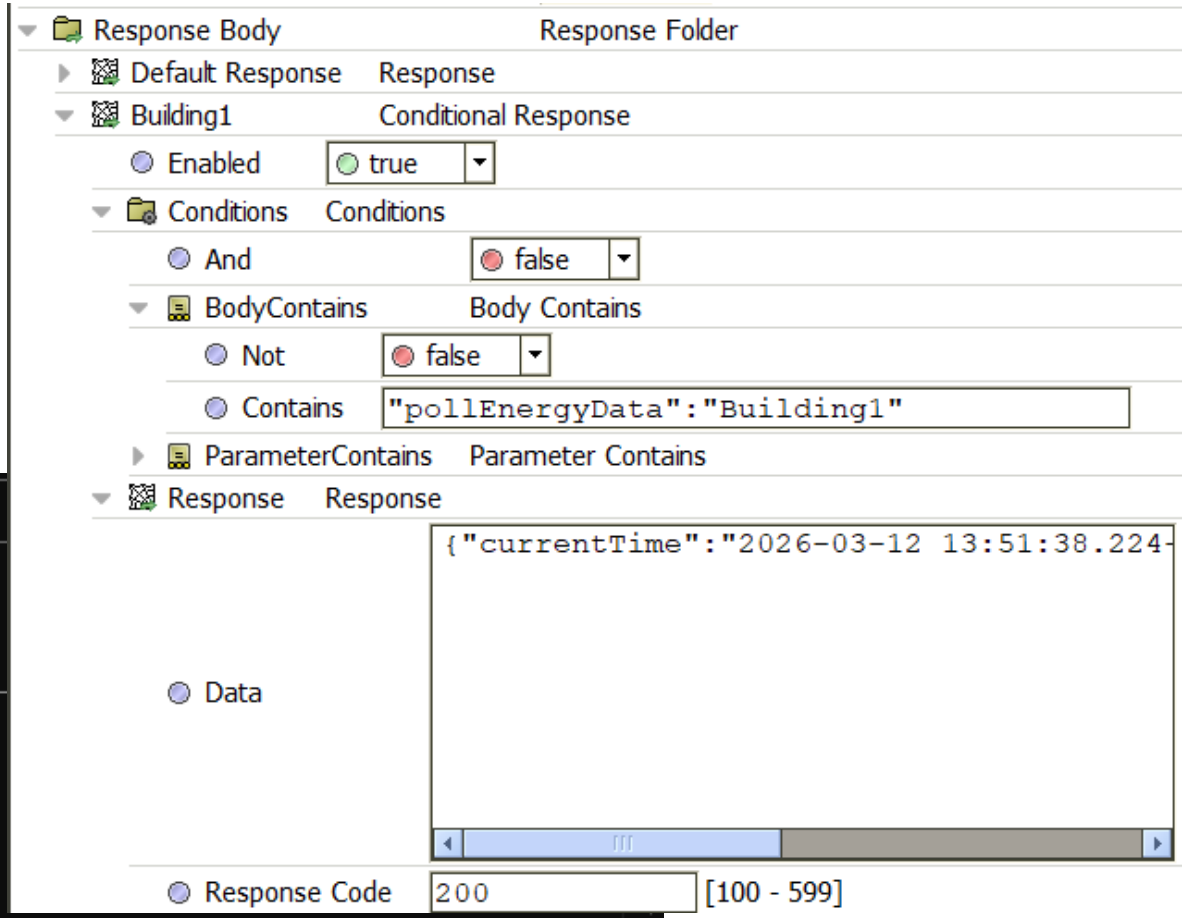


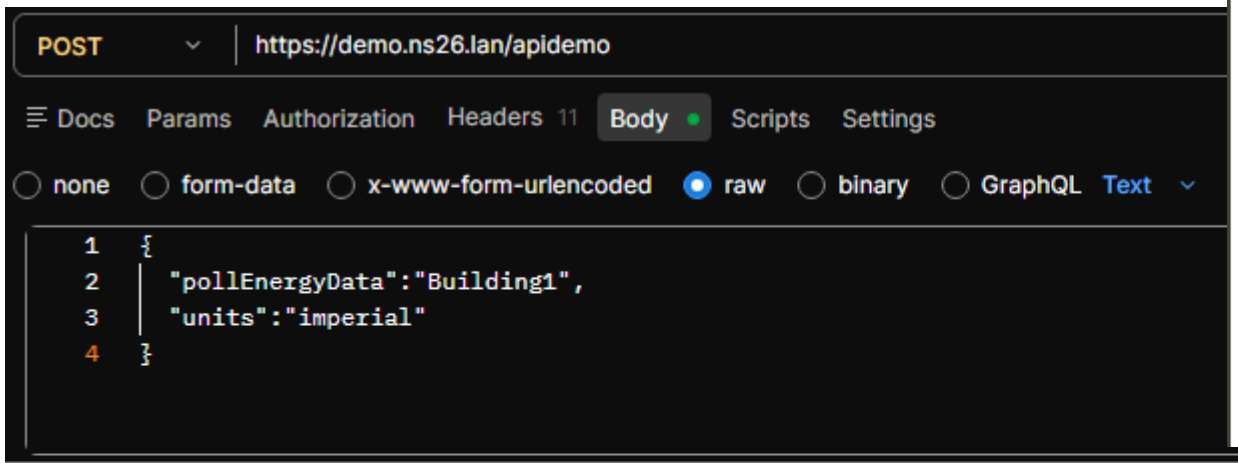
JSON Toolkit – Schema Exporter

- API response returns the schema output as JSON.



HTTP String Servlet



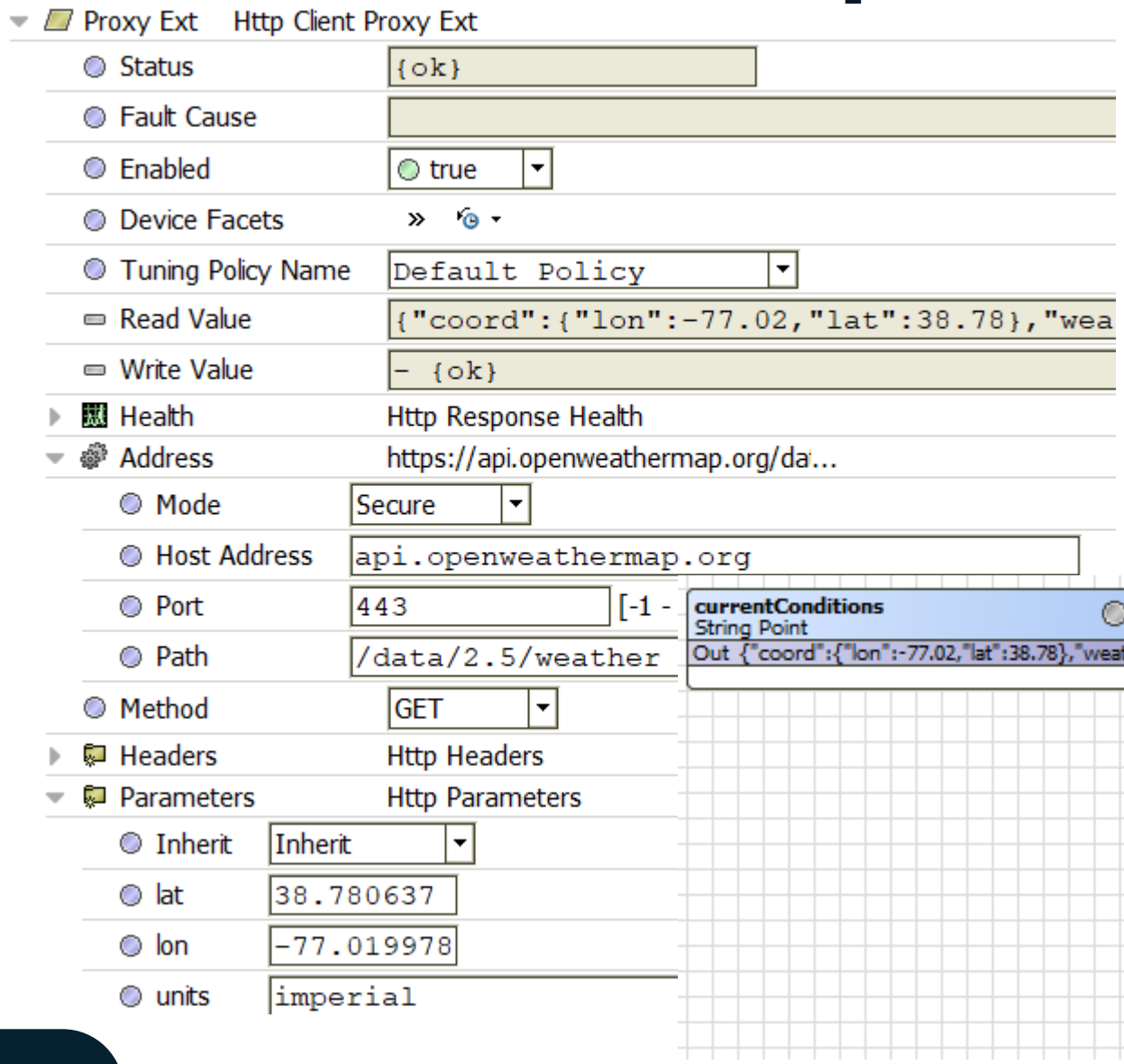


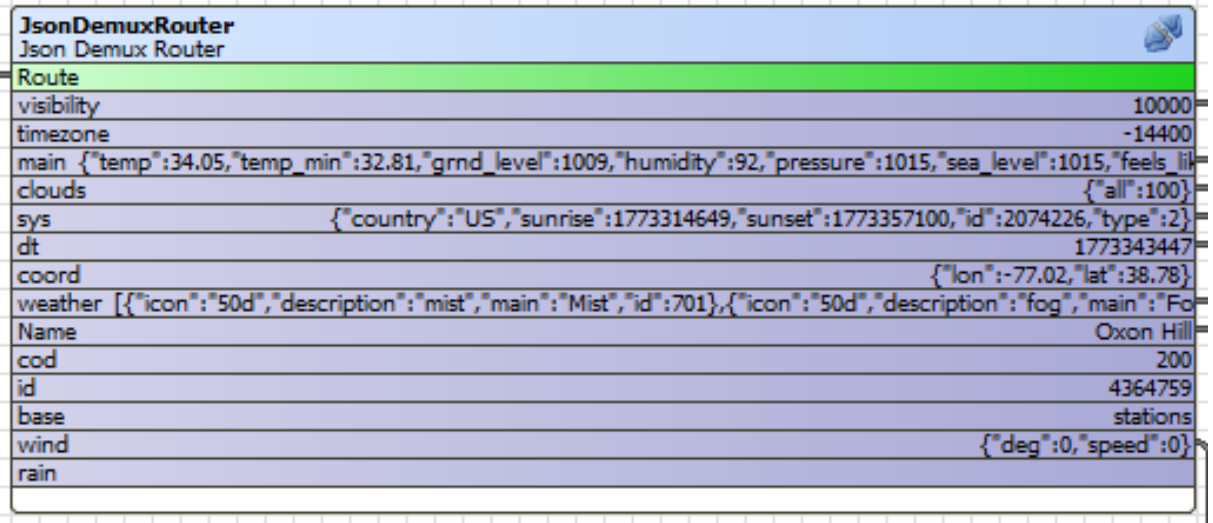
API Integration Examples

NS26

TRIDIUM

HTTP Client – openweathermap.org





NS26

TRIDIUM

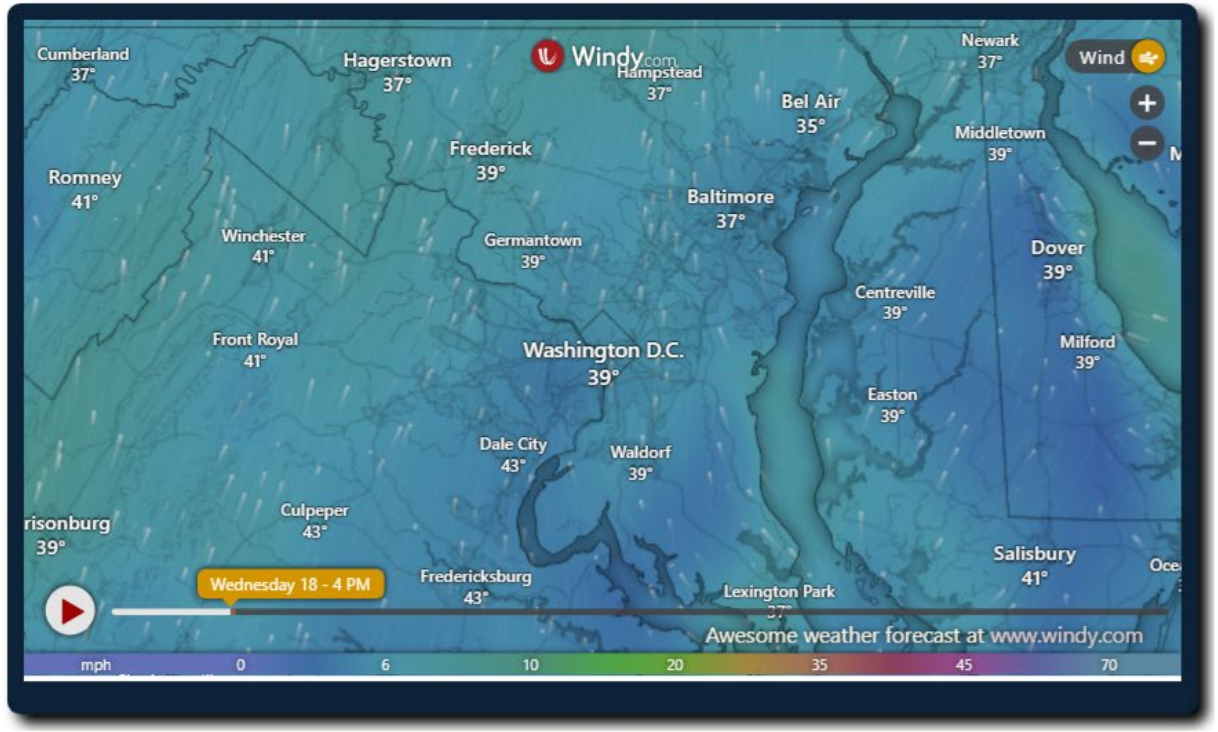
HTTP Client – openweathermap.org

Current Conditions

Summary	Mostly Cloudy
Temperature	40.8 °F
Feels Like	34.1 °F
Humidity	37 %RH
Dew Point	17.8 °F
Pressure	30.39 in Hg
Wind Direction	S
Wind Speed	11.5 mph
Wind Gust	0.0 mph
Visibility	6.21 mile
Cloud Cover	40 %
Observed	18-Mar-26 3:25 PM

7:14 AM — 11.89 hr — 7:17 PM

Washington, District of Columbia



Today's Forecast

Summary	Mostly Cloudy
Low	28.3 °F
High	41.2 °F
Precip Chance	0 %

Thursday Forecast

Summary	Overcast
Low	32.0 °F
High	53.7 °F
Precip Chance	0 %

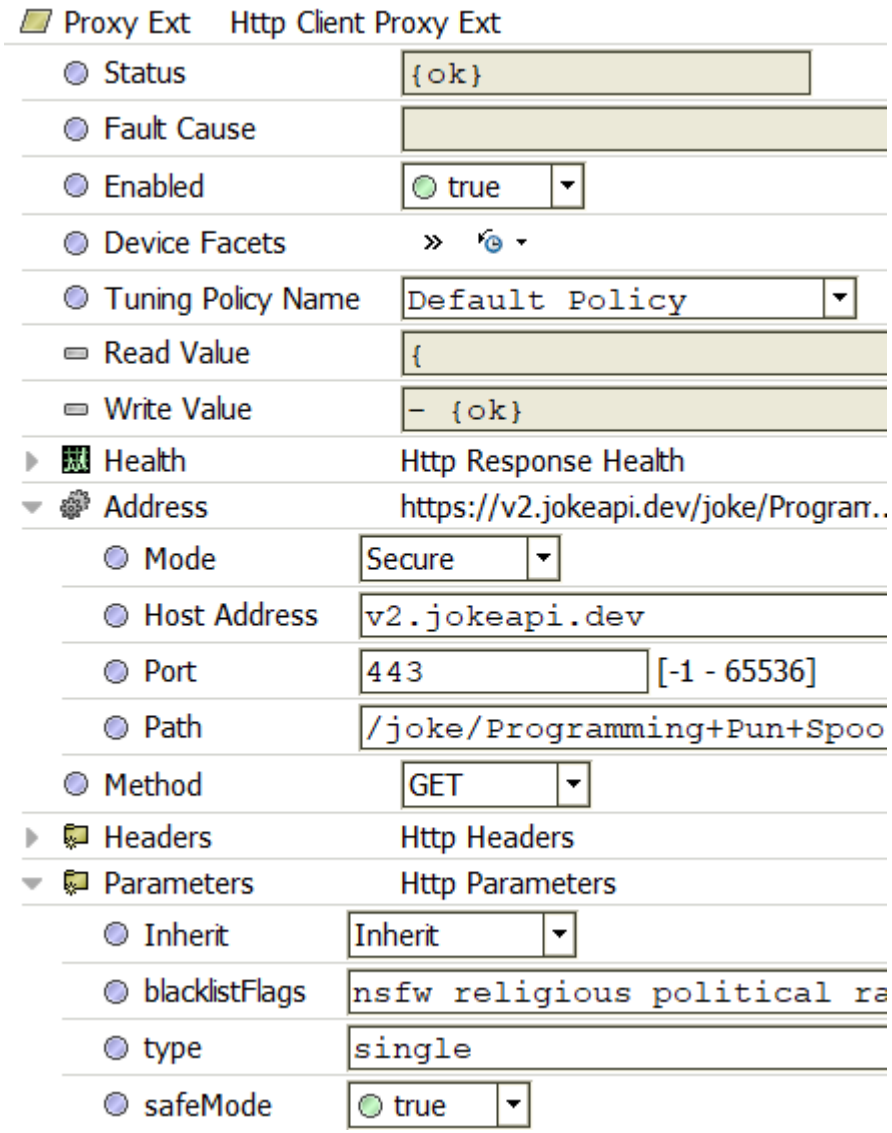
Friday Forecast

Summary	Light Rain
Low	43.2 °F
High	65.2 °F
Precip Chance	0 %

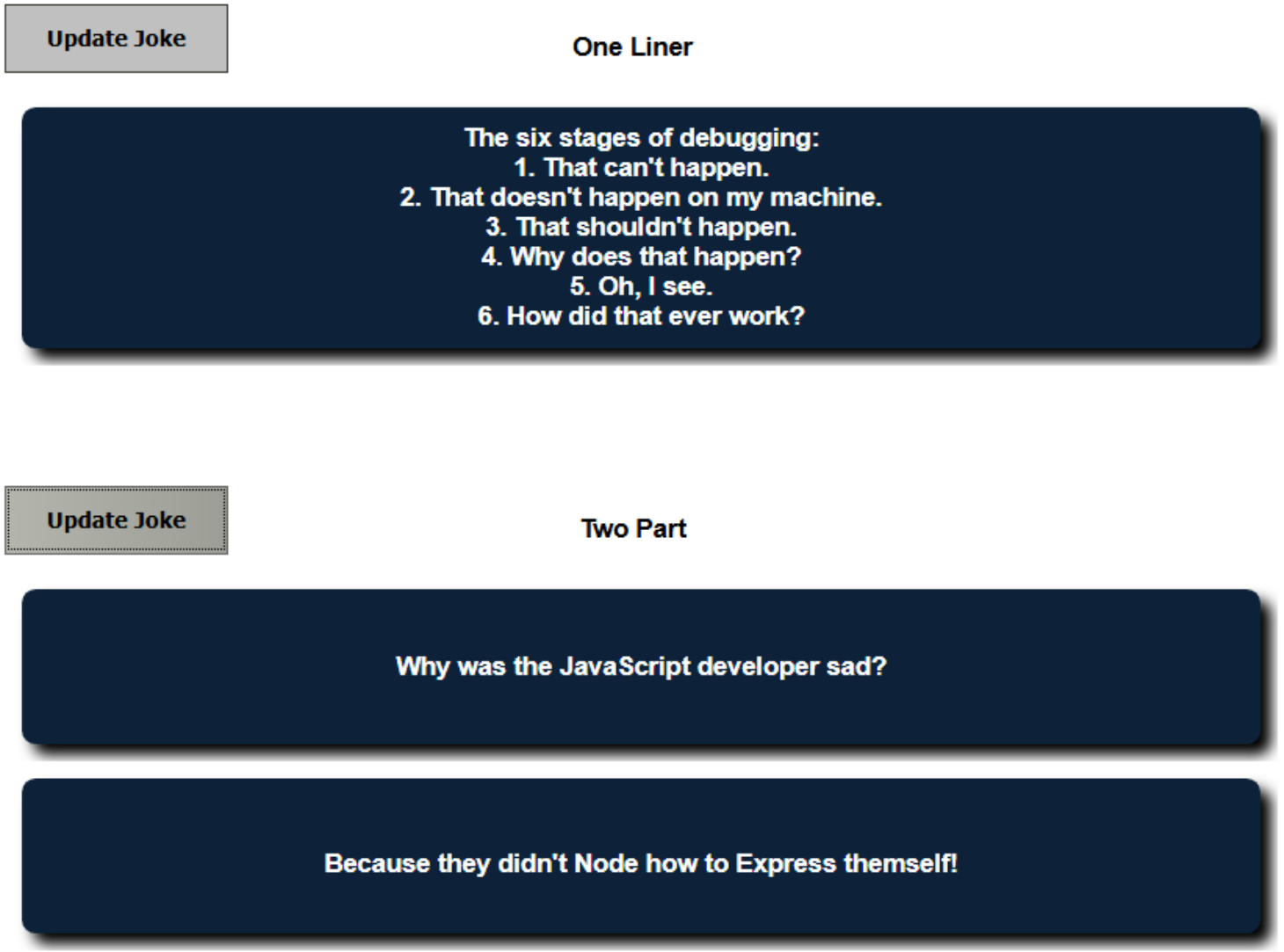
NS26

TRIDIUM

HTTP Client – v2.jokeapi.dev



HTTP Client – v2.jokeapi.dev



credits to https://v2.jokeapi.dev/

Working with HTML files in Niagara

NS26

Serving HTML from the Station's File Space

- Files can be edited by anyone with write access, without any source control or changelogs.
- Files are not versioned, so they cannot easily be upgraded or distributed to multiple stations.
- Files are not relativized, so not easily used as views on components.
- The HTML5 profile automatically provides the latest API improvements.
- HTML files bypass the profile, so they are more vulnerable to breaking when upgrading Niagara versions.

Accessing HTML Files

- The file servlet provides access to files under the station's shared file space.

  https://bldg1f1.ns24.lan/file/index.html

- The file ORD scheme with the File Download View.

  https://bldg1f1.ns24.lan/ord/file:^index.html|view:web:FileDownloadView

noSnoop

- The Niagara web server snoops content being served and potentially modifies the HTML or CSS content such as replacing module ORD schemes with module servlet requests.
- Snooping can be disabled in HTML by adding syntax to the top of the file.

```
<!-- @noSnoop -->
```

- Snooping can be disabled in CSS by adding syntax to the top of the file.

```
/* @noSnoop */
```

Using JavaScript Libraries

- When serving HTML content from the station's file system it may be necessary to load JavaScript libraries such as BajaScript, d3, jQuery, underscore and others.
- RequireJS from https://requirejs.org uses the Asynchronous Module Definition (AMD) method of loading JavaScript code, which is a modular script loader and will likely improve the speed and quality of your code.
- RequireJS can be loaded in an HTML page using script tags.

```
<script src="/requirejs/config.js"></script>
<script src="/module/js/com/tridium/js/ext/require/require.js"></script>
```

local:|module://docDeveloper/doc/requirejs.html

Example HTML File

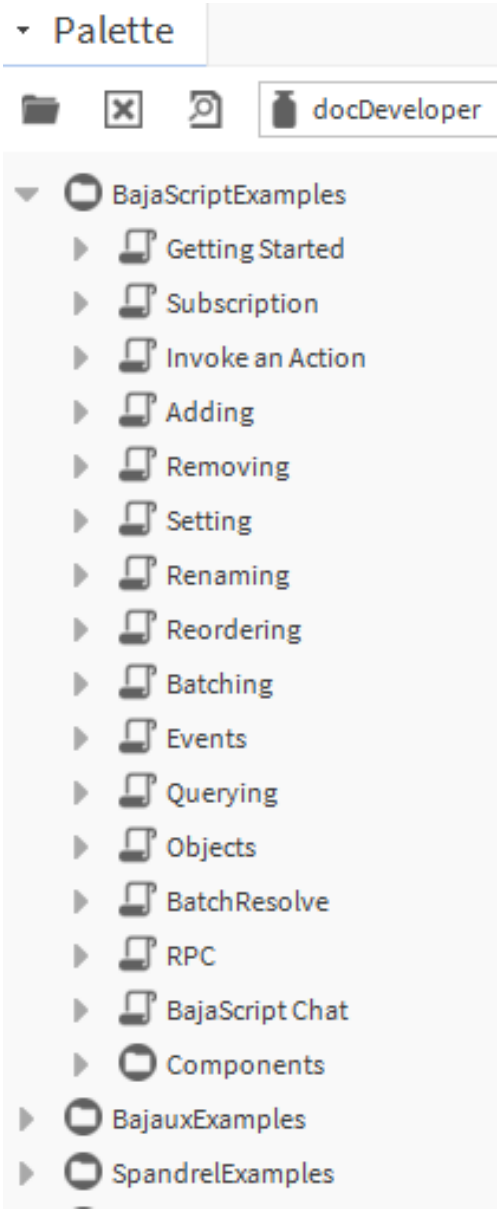
```
<!DOCTYPE html>
<!-- @noSnoop -->
<html>
<head>
  <title>This is a BajaScript test page</title>
  <script type='text/javascript'  src='/requirejs/config.js'></script>
  <script type='text/javascript'  src='/module/js/com/tridium/js/ext/require/require.min.js?'></script>
</head>
<body>
  <h1>BajaScript test page!</h1>
  <script>
    require(["baja!"], function (baja) {
      baja.Ord.make('station:|slot:/').get()
        .then((station) => {
          alert('Station name: ' + station.get('stationName'));
        });
    });
  </script>
</body>
</html>
```

BajaScript

- A framework for building web applications which includes JavaScript APIs for getting data from the station

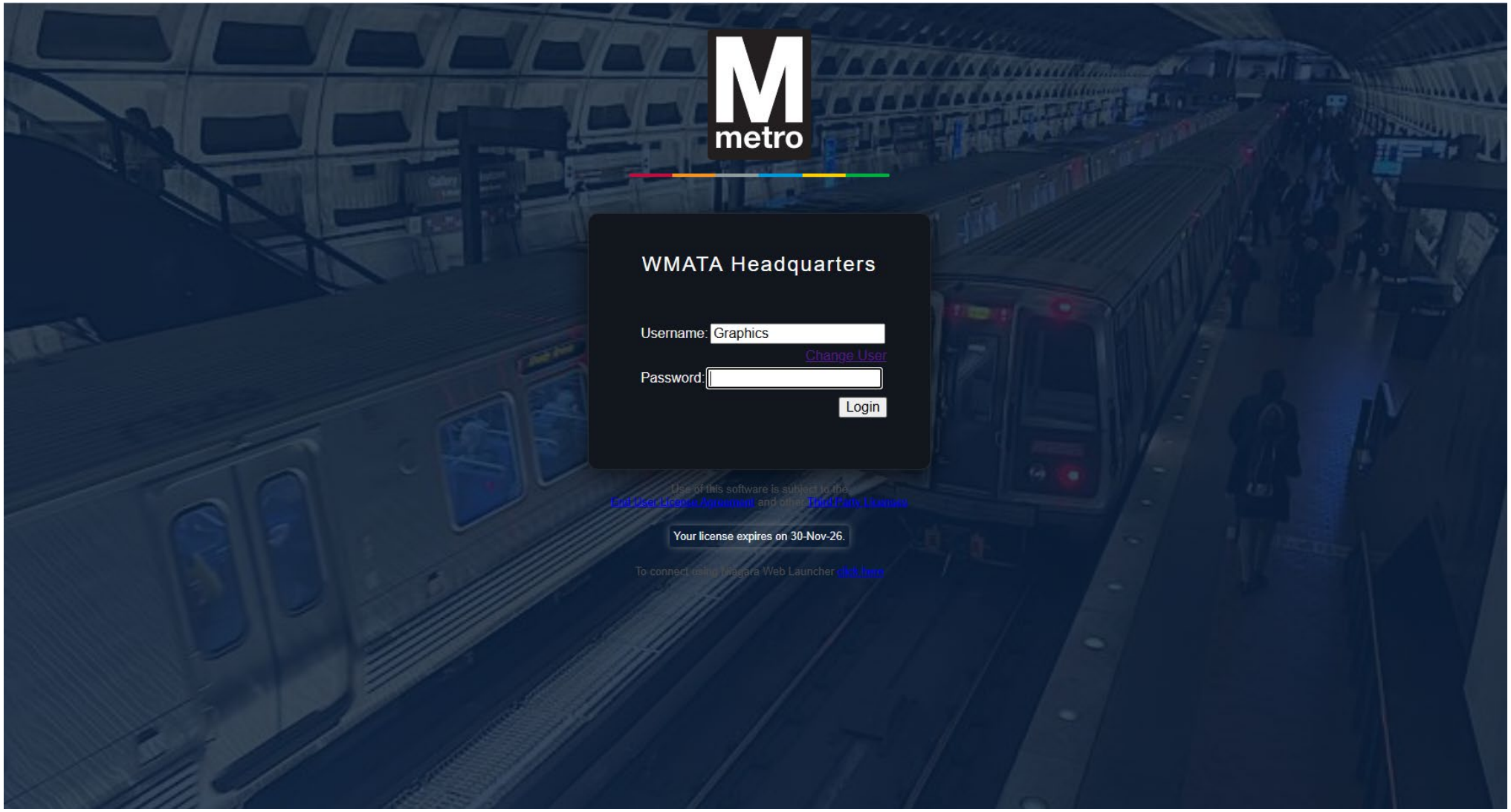
  local:|module://docDeveloper/doc/jsdoc/bajaScript-ux/index.html
- Getting started with BajaScript tutorial

  local:|module://docDeveloper/doc/jsdoc/bajaScript-ux/tutorial-gettingStarted.html
- The docDeveloper palette includes examples



HTML User Interface Examples

Custom Login Screen



Fleet Dashboard

Rail Map


Rail Operations
Live Train Positions
Live · 21:20:48
TRAINS 89
CARS 608
AVG DWELL 1s
HOLDING 0
INCIDENTS 0
OUTAGES 33
21:20:52
TUE 03/24/2026
ALL
RED
BLUE
ORANGE
GREEN
YELLOW
SILVER
Updated 21:20:48
TRAINS 84
ARRIVALS
STATS
ALERTS
OUTAGES 33
BL G05 #118 · Ckt 1403 6 ▶ T1
BL G05 #267 · Ckt 2575 6 ▶ T1
BL G05 #285 · Ckt 2566 6 ◀ T2
BL G05 #436 · Ckt 969 8 ▶ T1
BL G05 #442 · Ckt 2472 6 ▶ T1
BL G05 #464 · Ckt 1060 6 ▶ T1
BL G05 #489 · Ckt 2422 6 ▶ T1
BL J03 #077 · Ckt 1618 6 ◀ T2
BL J03 #109 · Ckt 2744 6 ◀ T2
BL J03 #164 · Ckt 1225 8 ◀ T2
BL J03 #292 · Ckt 2678 8 ◀ T2
BL J03 #431 · Ckt 2603 6 ◀ T2
BL J03 #467 · Ckt 1323 8 ◀ T2
GR E10 #008 · Ckt 2183 8 ▶ T1
GR E10 #025 · Ckt 1796 8 ▶ T1
GR E10 #038 · Ckt 2229 6 ▶ T1
GR E10 #069 · Ckt 1834 8 ▶ T1
Wheaton
Grosvenor-Strathmore
Forest Glen
Medical Center
Silver Spring
Bethesda
Takoma
Greenbelt
College Park-U of Md
Hyattsville Crossing
West Hyattsville
Friendship Heights
Fort Totten
Tenleytown-AU
Van Ness-UDC
Cleveland Park
Georgia Ave-Petworth
Brookland-CUA
Columbia Heights
Woodley Park-Zoo/Adams Morgan
Rhode Island Ave-Brentwood
U Street/African-Amer Civil War Memorial/Cardozo
Shaw-Howard U
Dupont Circle
Mt Vernon Sq/7th St-Convention Center
Gallery Place
Chinatown
Judiciary Square
Archives-Navy Memorial-Penn Quarter
Smithsonian
L'Enfant Plaza
Federal Center SW
Federal Triangle
Stadium-Armory
Potomac Ave
Waterfront
Navy Yard-Ballpark
New Carrollton
Landover
Cheverly
Deanwood
Minnesota Ave
Benning Road
Capitol Heights
Addison Road-Seat Pleasant
Morgan Boulevard
Downtown Largo
Innovation Center
Dulles International Airport
Herndon
Reston Town Center
Wiehle-Reston East
Spring Hill
McLean
Greensboro
West Falls Church
Dunn Loring-Merrifield
Vienna/Fairfax-GMU
East Falls Church
Rosslyn
Court House
Clarendon
Virginia Square-GMU
Ballston-MU
Arlington Cemetery
Pentagon
Pentagon City
Crystal City
Ronald Reagan Washington National Airport
Anacostia
Congress Heights
Southern Avenue
Naylor Road
Suitland
Branch Ave
Potomac Yard
Braddock Road
King St-Old Town
Eisenhower Avenue
Huntington
Van Dorn Street
Franconia-Springfield

Train Line View

Subscriber Pattern

```
// One reusable function handles every HttpClient subscription.
// It covers both the initial value on load AND live updates.

function subToOrd(ord, cb) {
    var sub = new baja.Subscriber();

    // Called every time the 'out' slot changes (new poll result)
    sub.attach('changed', function(prop) {
        if (prop.getName() === 'out') {
            var v = this.get('out');
            if (v) cb( parseJSON(v.toString()) );
        }
    });

    // Also grab the current value immediately on subscribe
    baja.Ord.make(ord).get({ subscriber: sub })
        .then(function(comp) {
            var v = comp.get('out');
            if (v) cb( parseJSON(v.toString()) );
        })
        .catch(function(err) {
            console.error('[WMATA] Failed to connect:', ord, err);
            setStatus(false); // flip the status indicator to offline
        });
}
```

ORD Declaration

```
// These look just like any other component ORD.
// The HttpClient sitting at this path has an 'out' slot
// that holds the last API response as a BString.

// Weather — two separate HttpClient components
var ORD_CURRENT  = 'station:|slot:/Examples/Weather/CurrentWeather';
var ORD_FORECAST = 'station:|slot:/Examples/Weather/ForecastWeather';

// WMATA — seven HttpClient components, all subscribed the same way
var ORD_TRAINS    = 'station:|slot:/Examples/WMATA/LiveTrainPositions';
var ORD_STATIONS  = 'station:|slot:/Examples/WMATA/StationInformation';
var ORD_PREDICT   = 'station:|slot:/Examples/WMATA/StationPredictions';
var ORD_INCIDENTS = 'station:|slot:/Examples/WMATA/Incidents';
var ORD_ELEVATOR  = 'station:|slot:/Examples/WMATA/EscalatorElevator';
```

parseJSON Helper

```
// HttpClient stores the response body as a Niagara BString, not a JS object.
// You must call .toString() first, then JSON.parse().
//
// BUT — some stations prepend status text before the JSON body.
// This helper skips any leading non-JSON characters defensively.

function parseJSON(val) {
    if (!val) return null;
    if (typeof val === 'object') return val; // already parsed

    try {
        var s = val.toString().trim();

        // Find the first '{' or '[' — the actual start of JSON
        var i = Math.min(
            s.indexOf('{') >= 0 ? s.indexOf('{') : Infinity,
            s.indexOf('[') >= 0 ? s.indexOf('[') : Infinity
        );

        return i < Infinity ? JSON.parse(s.substring(i)) : null;
    } catch(e) {
        return null; // bad response — fail silently, keep last good state
    }
}
```

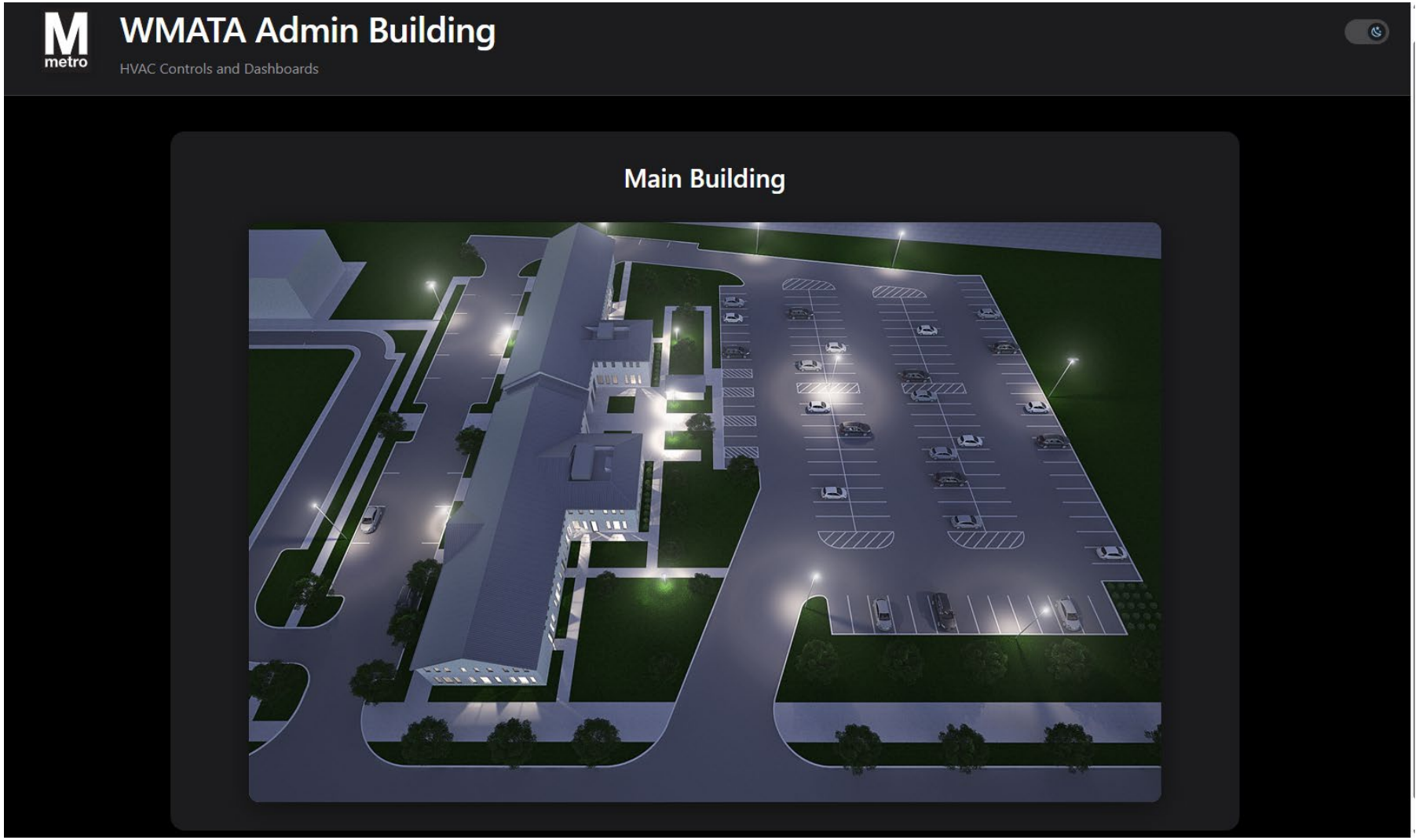
Icon Mapping for Condition Codes

```
// OpenWeatherMap returns a numeric condition code.
// Map it to your icon file set — no external icon library needed.

function iconFile(code, isNight) {
    if (code >= 200 && code < 300) return 'weather-lightning-rainy.svg';
    if (code >= 300 && code < 400) return 'weather-rainy.svg';
    if (code >= 500 && code < 502) return 'weather-rainy.svg';
    if (code >= 502 && code < 600) return 'weather-pouring.svg';
    if (code >= 600 && code < 700) return 'weather-snowy.svg';
    if (code >= 700 && code < 800) return 'weather-fog.svg';
    if (code === 800) return isNight ? 'weather-night.svg' : 'weather-
sunny.svg';
    if (code === 801) return isNight ? 'weather-night-partly-cloudy.svg'
                                     : 'weather-partly-cloudy.svg';
    return 'weather-cloudy.svg';
}

// Usage — determine night from the API's own sunrise/sunset values:
var isNight = (c.dt > c.sys.sunset || c.dt < c.sys.sunrise);
img.src = '/file/icons/' + iconFile(c.weather[0].id, isNight);
```

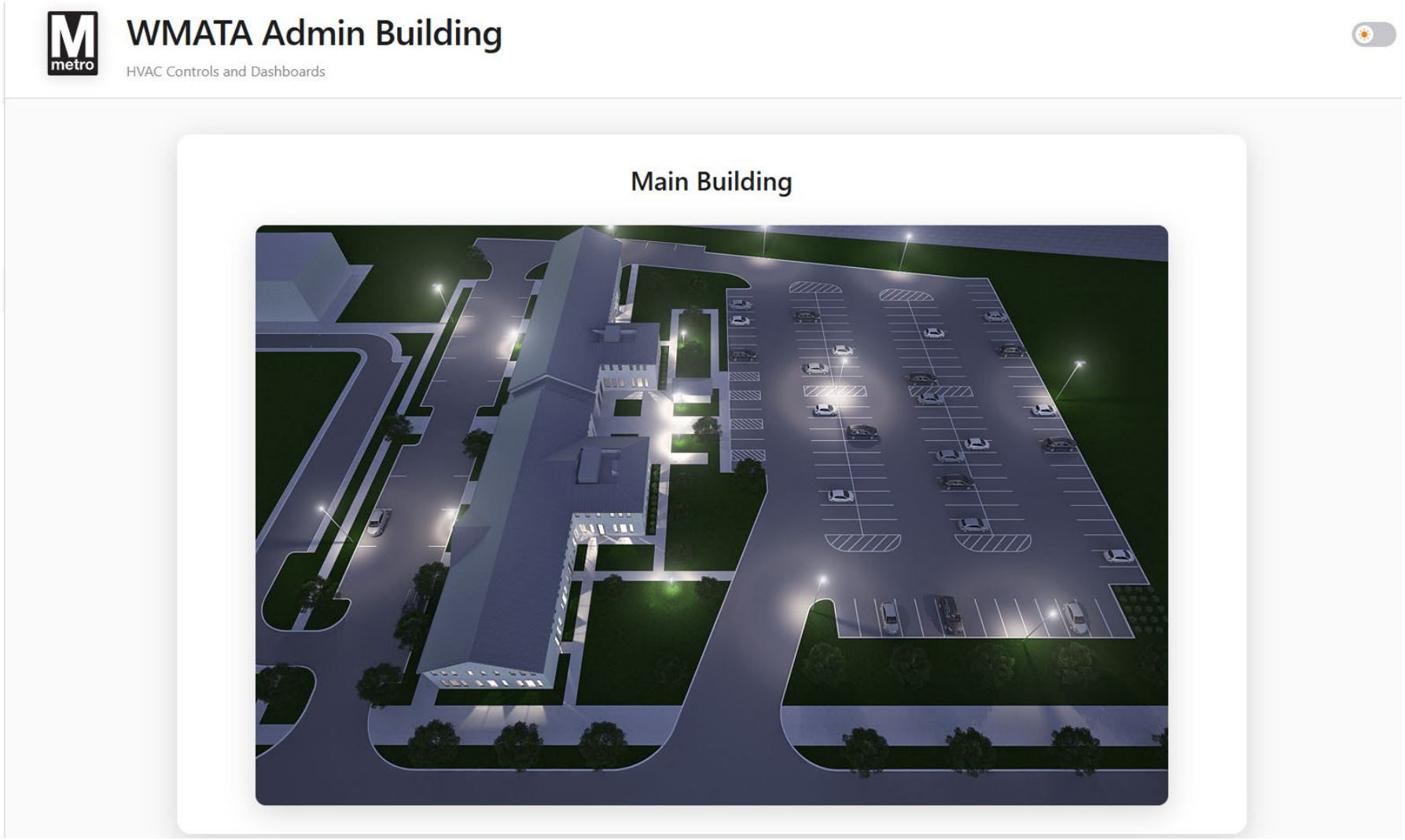
Light/Dark Mode – CSS Variable Cascade

```
// Toggle button handler — state persists across page reloads
document.getElementById('themeToggle').addEventListener('click', function() {
    document.body.classList.toggle('light');
    localStorage.setItem('theme',
        document.body.classList.contains('light') ? 'light' : 'dark'
    );
});

// On load — restore user's last preference
if (localStorage.getItem('theme') === 'light') {
    document.body.classList.add('light');
}
```

Light/Dark Mode – Graphics



Light/Dark Mode – Graphics



Questions