



# NIAGARA SUMMIT 2026

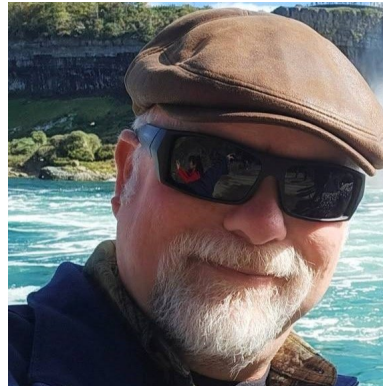
SEAMLESS CONNECTIVITY,  
POWERFUL INTELLIGENCE

TRIDIUM 

# Building Tomorrow : HTML5 UX Development and API Integration

The logo for the Niagara Summit 2026 is a white rounded rectangle with the text "NIAGARA SUMMIT 2026" in a bold, blue, sans-serif font. It is positioned in the lower-left quadrant of the slide, overlaid on a background image of a large, modern glass building at night.

NIAGARA  
SUMMIT  
2026



James Johnson  
Tridium



Charles Johnson  
Lynxspring

*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.*

# Agenda

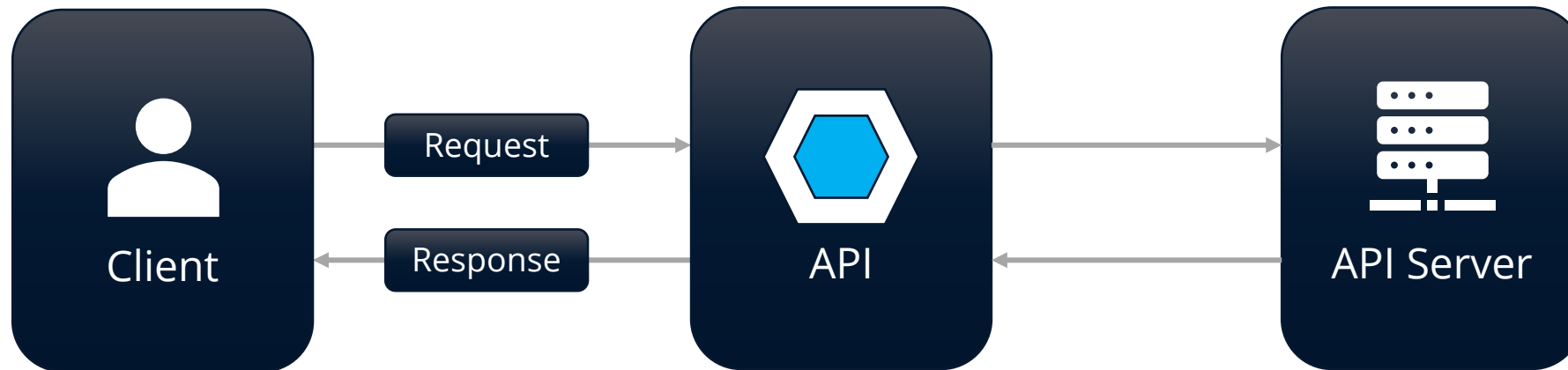
- What are Web APIs?
- Niagara APIs
- API Integration Examples
- HTML User Interface Examples

# What are Web APIs?



# 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.

The screenshot displays a web browser's developer tools interface. At the top, a request bar shows a GET request to the URL `api.openweathermap.org/data/2.5/weather?zip=23233&units=imperial&appid=a9ddd...`. Below this, the 'Params' tab is active, showing a table of query parameters:

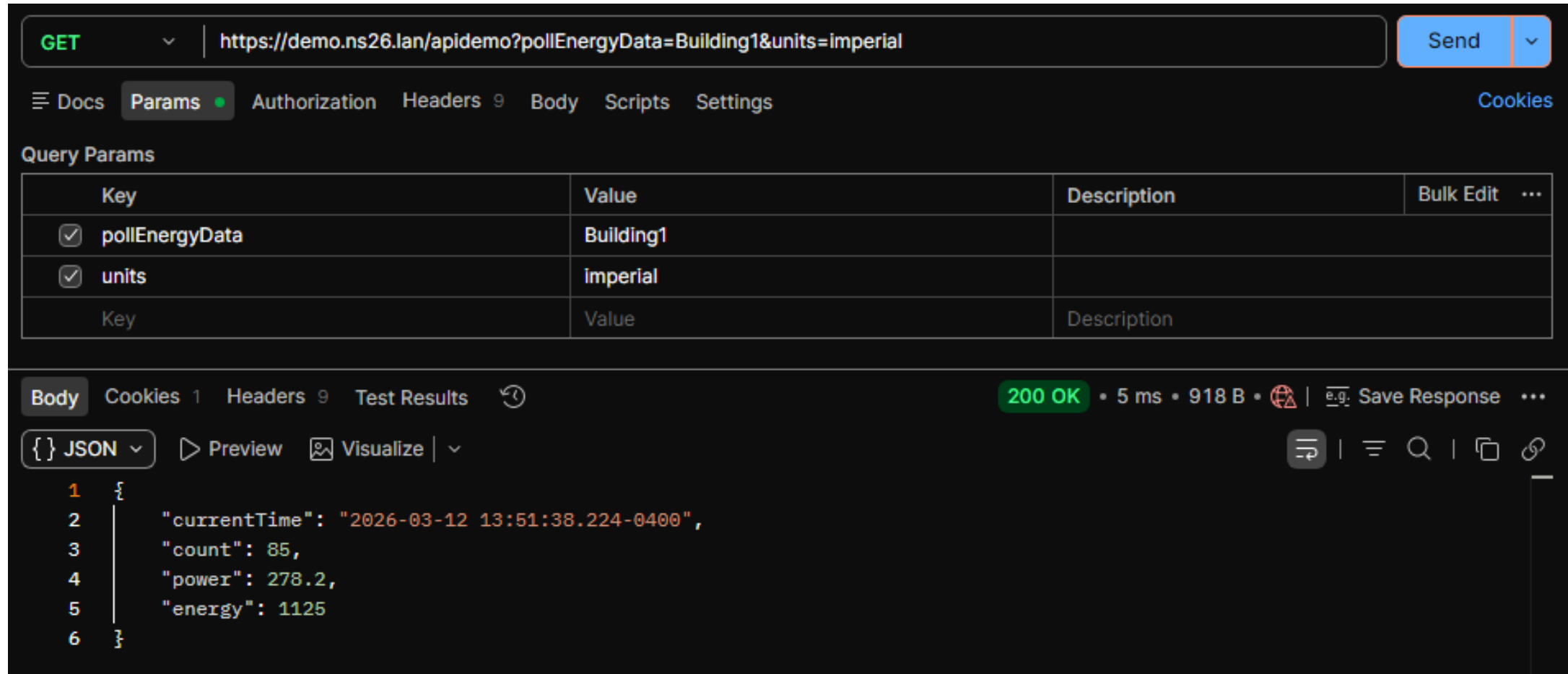
<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	zip	23233			
<input checked="" type="checkbox"/>	units	imperial			
<input checked="" type="checkbox"/>	appid	a9ddd...			
	Key	Value	Description		

To the right, the 'Body' tab shows the response status as '200 OK' with a response time of 209 ms and a size of 854 B. The response body is displayed as JSON:

```
1 {
2   "coord": {
3     "lon": -77.398,
4     "lat": 37.4638
5   },
6   "weather": [
7     {
8       "id": 803,
9       "main": "Clouds",
10      "description": "broken clouds",
11      "icon": "04d"
12    }
13  ],
14  "base": "stations",
15  "main": {
16    "temp": 78.66,
17    "feels_like": 79.09,
18    "temp_min": 76.35,
19    "temp_max": 80.35,
20    "pressure": 1019,
```

# HTTP GET

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



The screenshot displays a REST client interface with a GET request to `https://demo.ns26.lan/apidemo?pollEnergyData=Building1&units=imperial`. The 'Params' tab is active, showing a table of query parameters:

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> pollEnergyData	Building1		...
<input checked="" type="checkbox"/> units	imperial		
Key	Value	Description	

The response is shown in the 'Body' tab as a JSON object:

```
1 {
2   "currentTime": "2026-03-12 13:51:38.224-0400",
3   "count": 85,
4   "power": 278.2,
5   "energy": 1125
6 }
```

The status bar indicates a `200 OK` response with a 5 ms duration and 918 B of data.

# HTTP POST

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

The screenshot displays a REST client interface for a POST request to `https://demo.ns26.lan/apidemo`. The request body is a JSON object: `{ "pollEnergyData": "Building1", "units": "imperial" }`. The response is a 200 OK status with a 6 ms latency and 918 B of data. The response body is a JSON object: `{ "currentTime": "2026-03-12 13:51:38.224-0400", "count": 85, "power": 278.2, "energy": 1125 }`.

```
POST https://demo.ns26.lan/apidemo
```

Docs Params Authorization Headers 11 Body Scripts Settings Cookies

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL Text

```
1 {
2   "pollEnergyData": "Building1",
3   "units": "imperial"
4 }
```

Body Cookies 1 Headers 9 Test Results 200 OK • 6 ms • 918 B • e.g. Save Response

{ } JSON Preview Visualize

```
1 {
2   "currentTime": "2026-03-12 13:51:38.224-0400",
3   "count": 85,
4   "power": 278.2,
5   "energy": 1125
6 }
```

# API Documentation Example

[https://openweathermap.org/current?collection=current\\_forecast](https://openweathermap.org/current?collection=current_forecast)

The screenshot shows the OpenWeather API documentation page. The header includes the OpenWeather logo and a 'Login' button. The left sidebar lists various API endpoints under 'Current & Forecast', with 'Current weather data' expanded to show 'Call current weather data'. The main content area is titled 'Call current weather data' and includes a section 'How to make an API call' with a code block containing the URL: `https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}`. Below this is a 'Parameters' section with a table listing 'lat', 'lon', and 'appid' as required parameters.

**OpenWeather** Login

**Current & Forecast**

- Current weather data
  - Product concept
  - Call current weather data
  - API response
- Bulk downloading
  - Other features
- Hourly forecast 4 days
- Daily Forecast 16 Days
- Climatic forecast for 30 days
- Bulk Download
- 5 Day / 3 Hour Forecast
- Road Risk API

## Call current weather data

How to make an API call

```
https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}
```

### Parameters

lat	required	Latitude. If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our <a href="#">Geocoding API</a>
lon	required	Longitude. If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our <a href="#">Geocoding API</a>
appid	required	Your unique API key (you can always find it on your account page under the "API key" tab)

# Niagara APIs

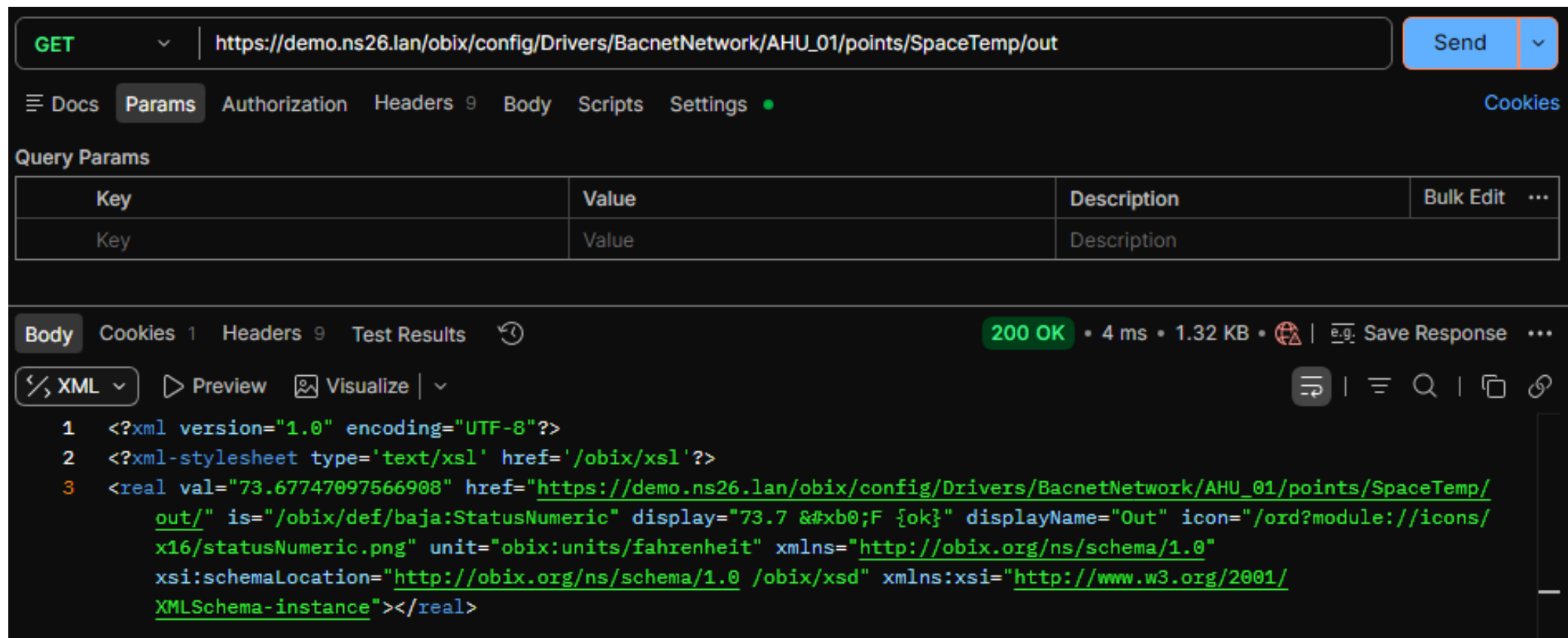


# Niagara – API Data Access

- 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.



The screenshot shows a web browser's developer tools interface. At the top, a GET request is shown for the URL `https://demo.ns26.lan/obix/config/Drivers/BacnetNetwork/AHU_01/points/SpaceTemp/out`. Below the URL bar, there are tabs for Docs, Params, Authorization, Headers, Body, Scripts, and Settings. The Params tab is active, showing a table with columns for Key, Value, Description, and Bulk Edit. The Body tab is also active, showing the response body in XML format. The response status is 200 OK, with a response time of 4 ms and a size of 1.32 KB. The XML response is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type='text/xsl' href='/obix/xsl'?>
3 <real val="73.67747097566908" href="https://demo.ns26.lan/obix/config/Drivers/BacnetNetwork/AHU_01/points/SpaceTemp/out/" is="/obix/def/baja>StatusNumeric" display="73.7 &#xB0;F {ok}" displayName="Out" icon="/ord?module://icons/x16/statusNumeric.png" unit="obix:units/fahrenheit" xmlns="http://obix.org/ns/schema/1.0" xsi:schemaLocation="http://obix.org/ns/schema/1.0 /obix/xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"></real>
```

# Niagara Analytics – Value Request

- Used to read control point values from the station.

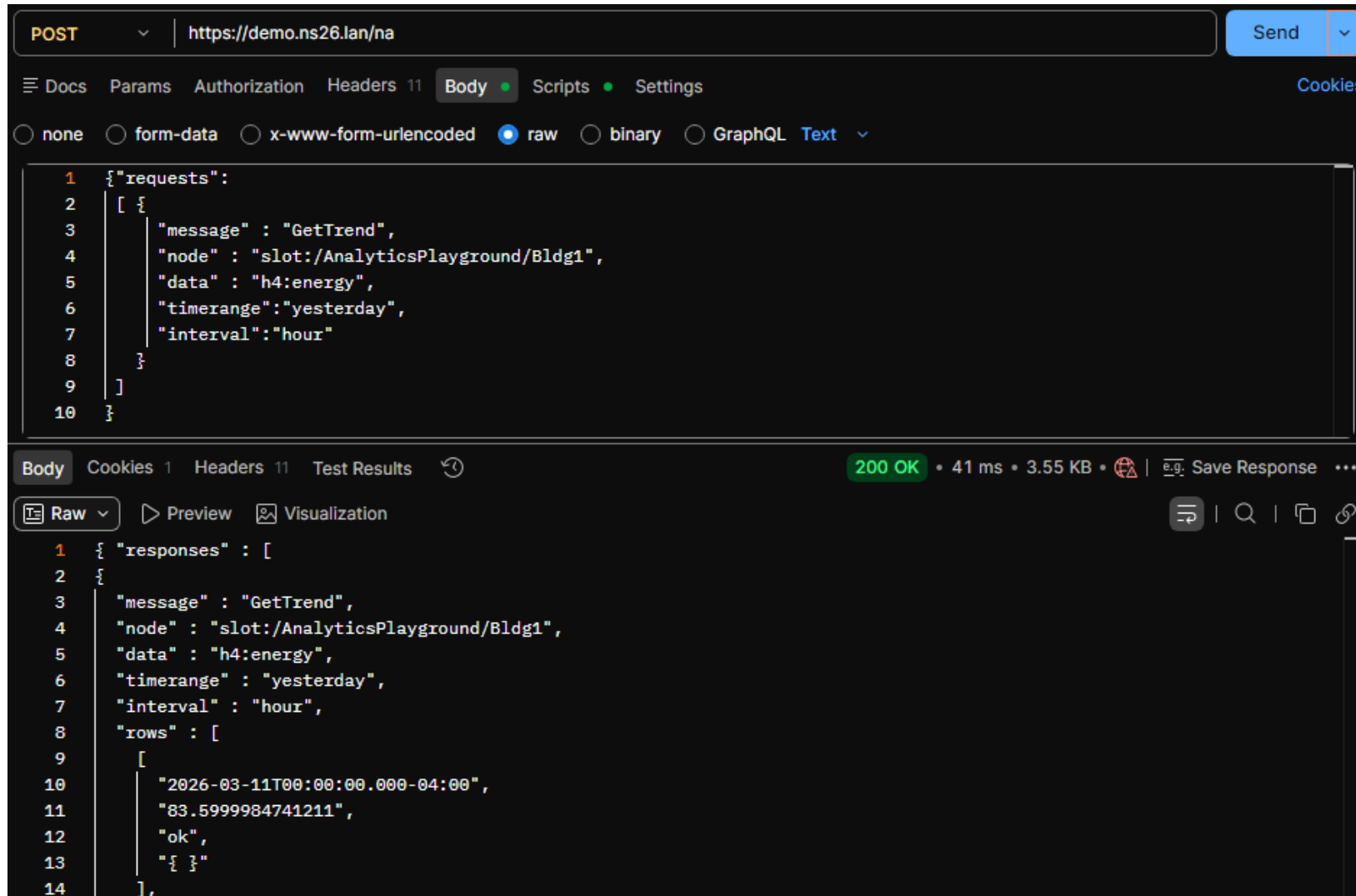
The screenshot displays a REST client interface for a POST request to `https://demo.ns26.lan/na`. The request body is a JSON array containing a single object with the following fields: `message` (GetValue), `node` (slot:/AnalyticsPlayground/Bldg1), and `data` (h4:energy). The response is a 200 OK status with a response body containing a single object with the following fields: `message` (GetValue), `node` (slot:/AnalyticsPlayground/Bldg1), `data` (h4:energy), `value` (1125.0), `status` (ok), and `seriesName` (Bldg1-energy).

```
POST https://demo.ns26.lan/na
Body
  none
  form-data
  x-www-form-urlencoded
  raw
  binary
  GraphQL
  Text
  1 {"requests":
  2   [ {
  3     "message" : "GetValue",
  4     "node" : "slot:/AnalyticsPlayground/Bldg1",
  5     "data" : "h4:energy"
  6   }
  7 ]
  8 }
  9

Body Cookies 1 Headers 14 Test Results 200 OK • 125 ms • 1.47 KB • Save Response
Raw Preview Visualize
  1 { "responses" : [
  2   {
  3     "message" : "GetValue",
  4     "node" : "slot:/AnalyticsPlayground/Bldg1",
  5     "data" : "h4:energy",
  6     "value" : "1125.0",
  7     "status" : "ok",
  8     "seriesName" : "Bldg1-energy"
  9   }
  10 ]}
```

# Niagara Analytics – Trend Request

- Used to query history data from the station.



The screenshot displays a REST client interface with a POST request to `https://demo.ns26.lan/na`. The request body is a JSON object with the following structure:

```
1 {"requests":
2   [ {
3     "message": "GetTrend",
4     "node": "slot:/AnalyticsPlayground/Bldg1",
5     "data": "h4:energy",
6     "timerange": "yesterday",
7     "interval": "hour"
8   }
9 ]
10 }
```

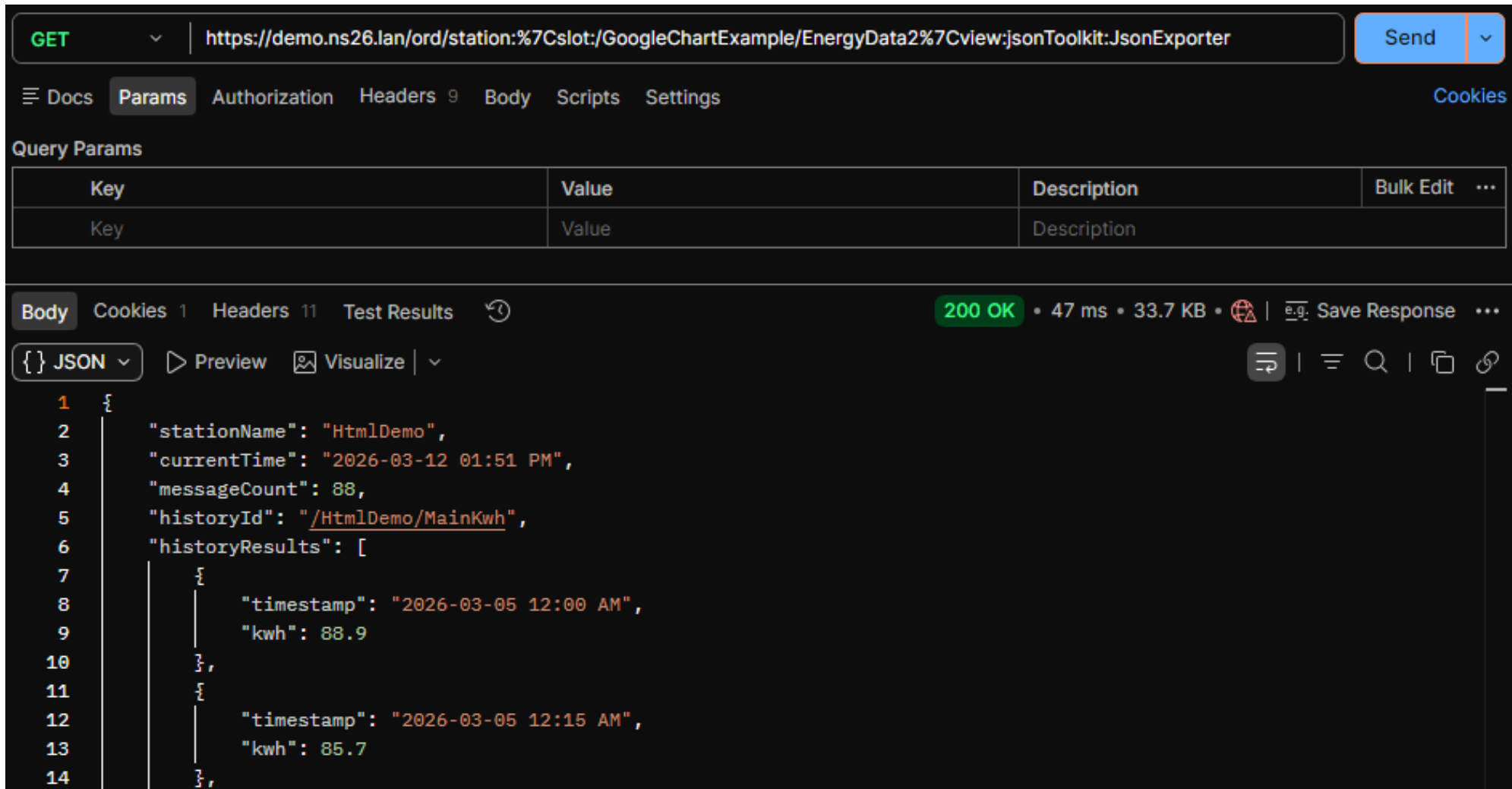
The response status is `200 OK` with a response time of `41 ms` and a size of `3.55 KB`. The response body is a JSON object with the following structure:

```
1 { "responses" : [
2   {
3     "message": "GetTrend",
4     "node": "slot:/AnalyticsPlayground/Bldg1",
5     "data": "h4:energy",
6     "timerange": "yesterday",
7     "interval": "hour",
8     "rows" : [
9       [
10        "2026-03-11T00:00:00.000-04:00",
11        "83.5999984741211",
12        "ok",
13        "{ }"
14      ],

```

# JSON Toolkit – Schema Exporter

- API response returns the schema output as JSON.



The screenshot displays a REST client interface with the following details:

- Method:** GET
- URL:** `https://demo.ns26.lan/ord/station:%7Cslot:/GoogleChartExample/EnergyData%7Cview:jsonToolkit:JsonExporter`
- Response Status:** 200 OK
- Response Time:** 47 ms
- Response Size:** 33.7 KB

The response body is shown in JSON format:

```
1 {
2   "stationName": "HtmlDemo",
3   "currentTime": "2026-03-12 01:51 PM",
4   "messageCount": 88,
5   "historyId": "/HtmlDemo/MainKwh",
6   "historyResults": [
7     {
8       "timestamp": "2026-03-05 12:00 AM",
9       "kwh": 88.9
10    },
11    {
12      "timestamp": "2026-03-05 12:15 AM",
13      "kwh": 85.7
14    },
15  ]
16 }
```

# HTTP String Servlet

**POST** | <https://demo.ns26.lan/apidemo>

Docs Params Authorization Headers 11 **Body** Scripts Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **Text**

```
1 {
2   "pollEnergyData": "Building1",
3   "units": "imperial"
4 }
```

Response Body | Response Folder

- Default Response | Response
- Building1 | Conditional Response
  - Enabled |  true
  - Conditions | Conditions
    - And |  false
    - BodyContains | Body Contains
      - Not |  false
      - Contains | "pollEnergyData": "Building1"
    - ParameterContains | Parameter Contains
  - Response | Response
    - Data | 

```
{"currentTime": "2026-03-12 13:51:38.224"}
```
    - Response Code | 200 [100 - 599]

Body Cookies 1 Headers 11 Test Results

**200 OK** • 25 ms • 1.05 KB • Save Response

{ } JSON Preview Visualize

```
1 {
2   "currentTime": "2026-03-12 13:51:38.224-0400",
3   "count": 85,
4   "power": 278.2,
5   "energy": 1125
6 }
```

# API Integration Examples

# HTTP Client - openweathermap.org

Proxy Ext Http Client Proxy Ext

Status	{ok}
Fault Cause	
Enabled	<input checked="" type="radio"/> true
Device Facets	>>
Tuning Policy Name	Default Policy
Read Value	{"coord":{"lon":-77.02,"lat":38.78},"wea
Write Value	- {ok}
Health	Http Response Health
Address	https://api.openweathermap.org/da...
Mode	Secure
Host Address	api.openweathermap.org
Port	443 [-1 -
Path	/data/2.5/weather
Method	GET
Headers	Http Headers
Parameters	Http Parameters
Inherit	Inherit
lat	38.780637
lon	-77.019978
units	imperial



**currentConditions**  
String Point  
Out {"coord":{"lon":-77.02,"lat":38.78},"weath


<b>JsonDemuxRouter</b> Json Demux Router	
Route	
visibility	10000
timezone	-14400
main	{"temp":34.05,"temp_min":32.81,"grnd_level":1009,"humidity":92,"pressure":1015,"sea_level":1015,"feels li
clouds	{"all":100}
sys	{"country":"US","sunrise":1773314649,"sunset":1773357100,"id":2074226,"type":2}
dt	1773343447
coord	{"lon":-77.02,"lat":38.78}
weather	[{"icon":"50d","description":"mist","main":"Mist","id":701}, {"icon":"50d","description":"fog","main":"Fo
Name	Oxon Hill
cod	200
id	4364759
base	stations
wind	
rain	{"deg":0,"speed":0}

# 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



Awesome weather forecast at [www.windy.com](http://www.windy.com)

### 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 %



# HTTP Client – v2.jokeapi.dev

Proxy Ext Http Client Proxy Ext

Status	<input type="radio"/> {ok}
Fault Cause	
Enabled	<input checked="" type="radio"/> true
Device Facets	» ↻
Tuning Policy Name	Default Policy
Read Value	{
Write Value	- {ok}
Health	Http Response Health
Address	https://v2.jokeapi.dev/joke/Programr..
Mode	Secure
Host Address	v2.jokeapi.dev
Port	443 [-1 - 65536]
Path	/joke/Programming+Pun+Spoo
Method	GET
Headers	Http Headers
Parameters	Http Parameters
Inherit	Inherit
blacklistFlags	nsfw religious political ra
type	single
safeMode	<input checked="" type="radio"/> true

# HTTP Client - v2.jokeapi.dev

Update Joke

One Liner

The six stages of debugging:  
1. That can't happen.  
2. That doesn't happen on my machine.  
3. That shouldn't happen.  
4. Why does that happen?  
5. Oh, I see.  
6. How did that ever work?

Update Joke

Two Part

Why was the JavaScript developer sad?

Because they didn't Node how to Express themself!

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

# Working with HTML files in Niagara

# 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

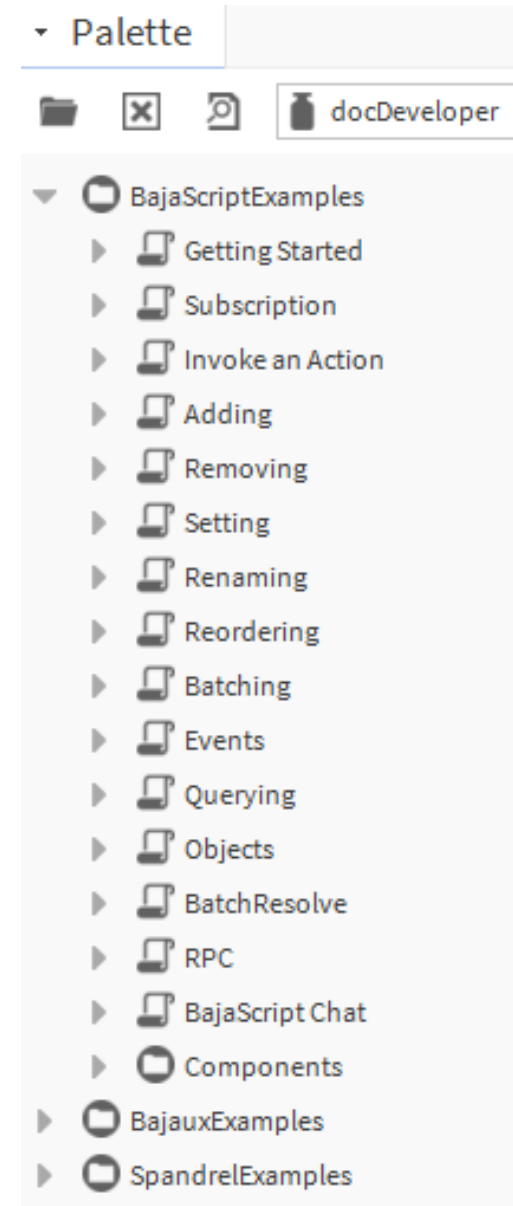
- A framework for building web applications which includes JavaScript APIs for getting data from the station

local: | module://docDeveloper/doc/jsdoc/bajaScript-ux/index.html

- Getting started with BajaScript tutorial

local: | module://docDeveloper/doc/jsdoc/bajaScript-ux/tutorial-gettingStarted.html

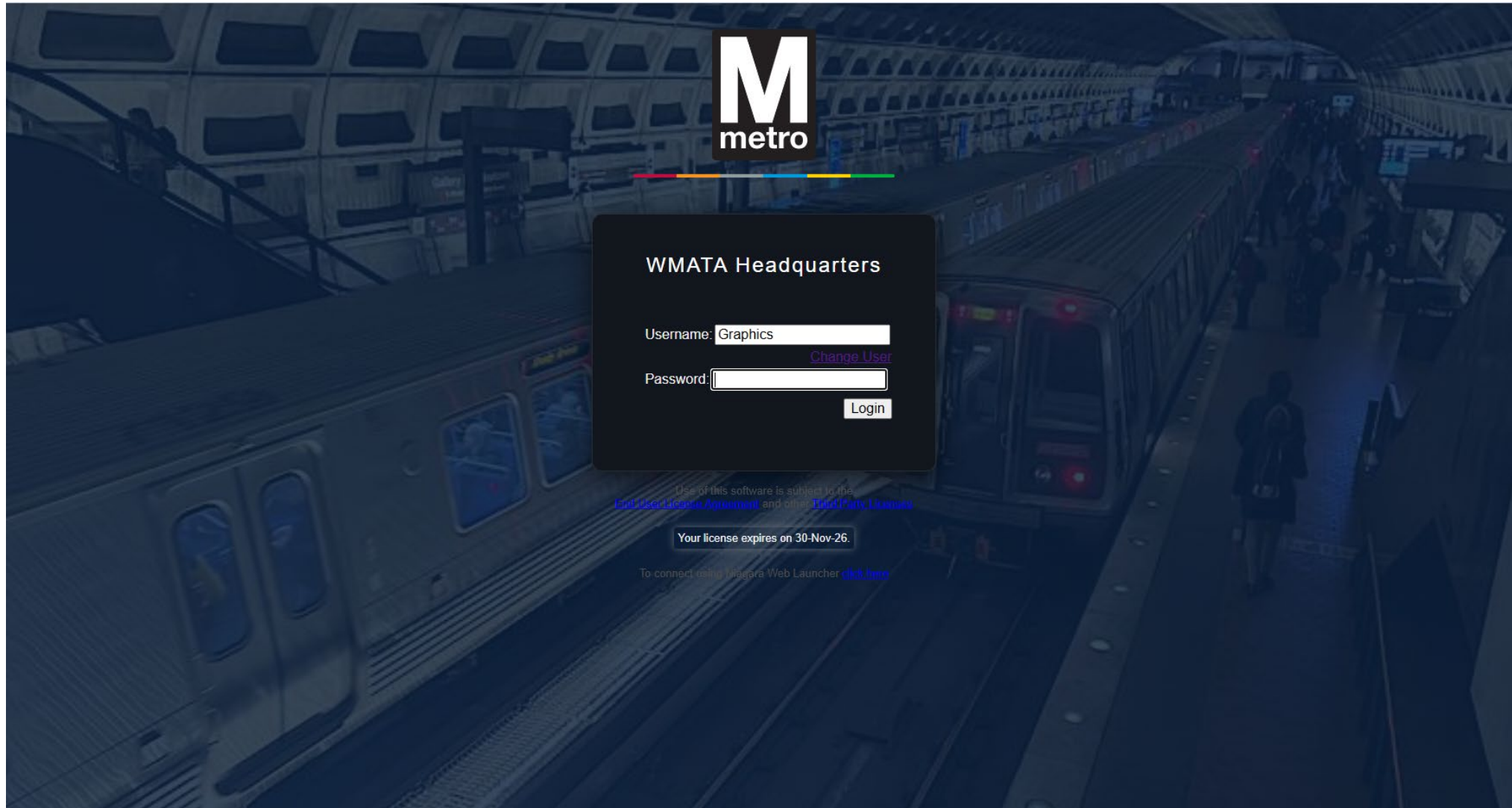
- The docDeveloper palette includes examples



# HTML User Interface Examples



# Custom Login Screen



# Fleet Dashboard

**M** Metro Fleet Dashboard Real-Time Bus Positions ● Live · update 1 21:17:35 Refresh

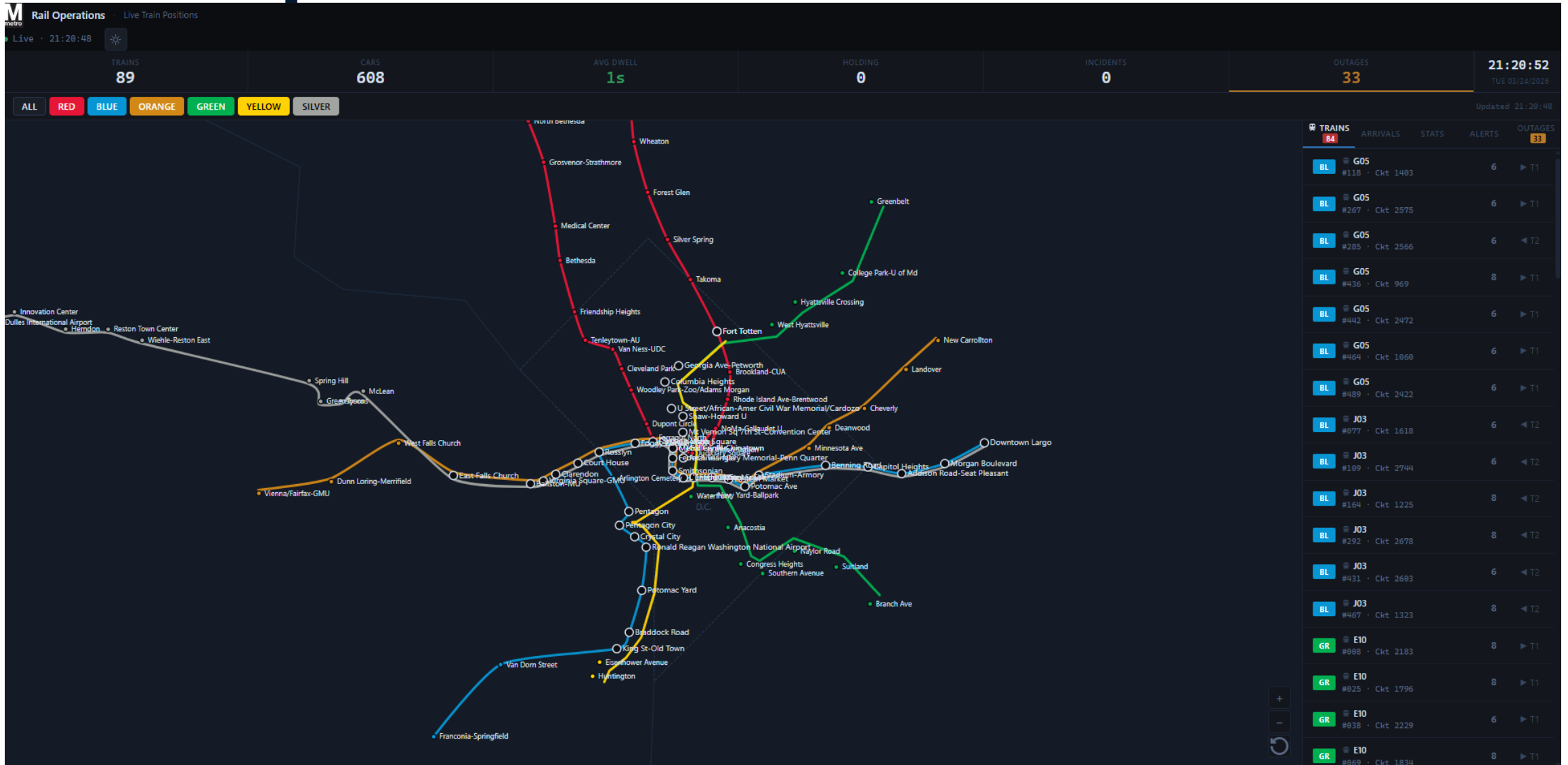
ACTIVE BUSES: **351** (78 routes)    ON TIME: **54** (15%)    RUNNING LATE: **103** (29%)    RUNNING EARLY: **194** (56%)    AVG DEVIATION: **1.7** minutes    WORST DELAY: **-9 min** (C33 (F5408))

Q Route, vehicle, headsign... All On Time Late Early Showing 351 of 351

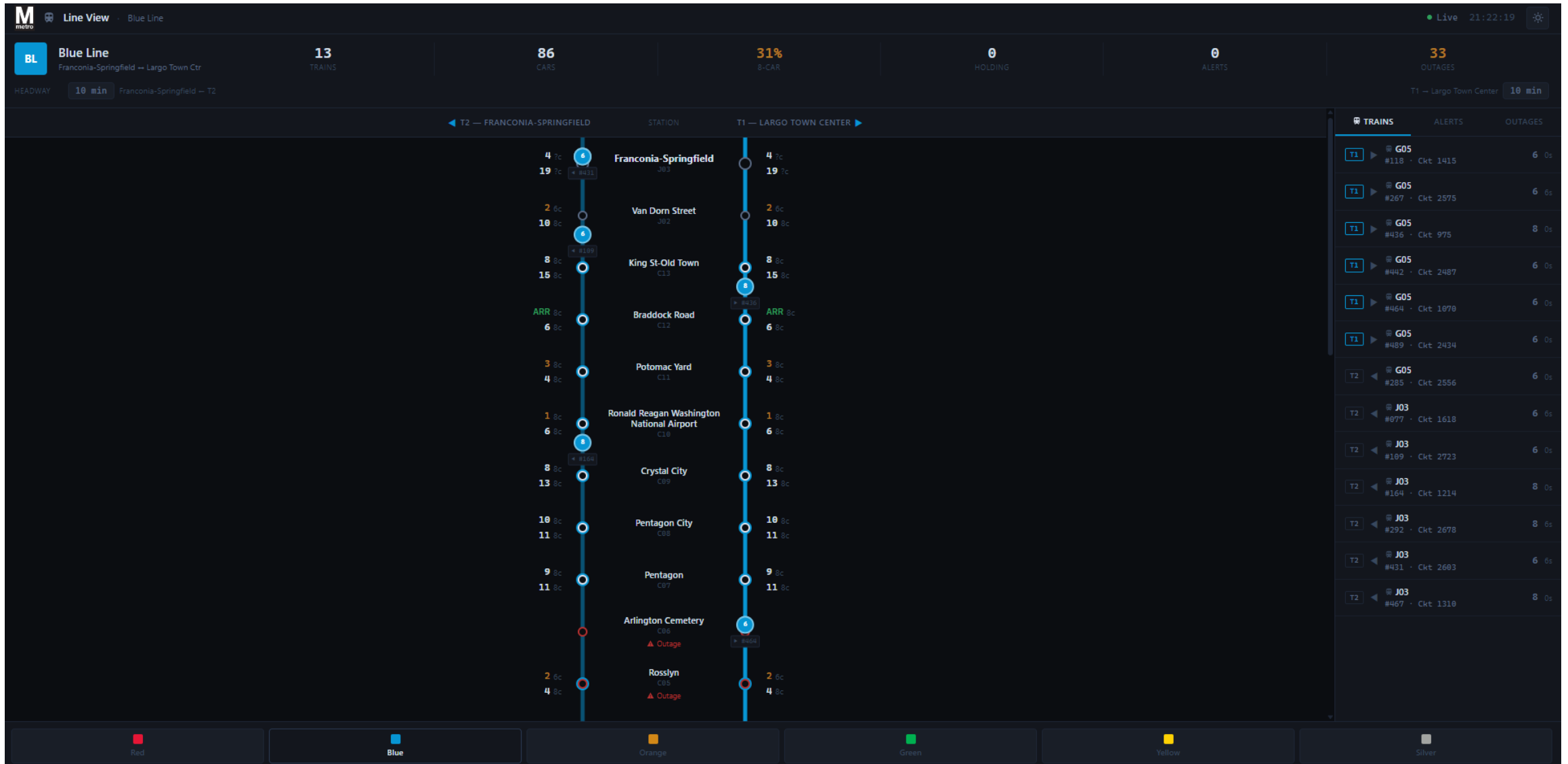
ALL 351 | A11 | A12 | A1X | A27 | A40 | A58 | A66 | A70 | A76 | C11 | C13 | C15 | C17 | C21 | C23 | C26 | C27 | C29 | C31 | C33 | C35 | C37 | C41 | C43 | C51 | C53 | C55 | C61 | C63 | C71 | C75 | C81 | C91 | D10 | D1X | D20 | D24 | D2X | D30 | D32 | D34 | D36 | D40 | D44 | D4X | D50 | D5X | D60 | D6X | D70 | D72 | D74 | D80 | D90 | D94 | D96 | F1X | F20 | F23 | F24 | F44 | F44 | F50 | F60 | F61 | F62 | M12 | M20 | M22 | M44 | M52 | M54 | M60 | M70 | P10 | P12 | P14 | P16 | P1X | P20 | P21 | P30 | P31 | P32 | P33 | P35 | P40 | P41 | P42 | P60 | P61 | P62 | P66 | P72 | P73 | P87 | P90 | P94 | P96

ROUTE	VEHICLE	HEADSIGN	DIR	DEVIATION	LAT	LOD	LAST REPORT	TRIP START	TRIP END	TRIP ID	BLOCK
A11	3203	PENTAGON	NORTH	+3 min	38.8683	-77.8548	22:15:57	21:28:00	22:12:00	34079020	F220
A11	3201	HUNTINGTON	SOUTH	+9 min	38.8857	-77.8469	22:17:18	21:29:00	22:18:00	9666020	F232
A11	3344	HUNTINGTON	SOUTH	ON TIME	38.8932	-77.8666	22:16:58	21:59:00	22:48:00	27138020	F045
A12	3129	HUNTING POINT	SOUTH	+1 min	38.8469	-77.8838	22:17:05	21:59:00	22:53:00	11892020	F207
A12	2938	BALLSTON	NORTH	+6 min	38.8842	-77.8471	22:16:54	22:06:00	22:55:00	2250020	F172
A1X	2990	PENTAGON CITY	NORTH	ON TIME	38.8433	-77.8542	22:16:57	21:48:00	22:04:00	26691020	F035
A1X	2965	PENTAGON CITY	NORTH	-1 min	38.8998	-77.8492	22:17:09	22:00:00	22:24:00	1227020	F084
A1X	2993	BRADDOCK RD	SOUTH	ON TIME	38.8429	-77.8586	22:17:10	22:00:00	22:24:00	15229020	F091
A27	2933	PENTAGON	NORTH	+12 min	38.8692	-77.8536	22:17:02	21:20:00	22:04:00	3307020	F117
A27	3102	VAN DORN ST	SOUTH	+2 min	38.8238	-77.1236	22:17:15	21:48:00	22:28:00	10734020	F166
A27	3138	PENTAGON	NORTH	-2 min	38.8680	-77.8536	22:16:52	21:48:00	22:24:00	3630020	F190
A27	3205	VAN DORN ST	SOUTH	+1 min	38.8430	-77.8854	22:17:10	22:00:00	22:48:00	30305020	F235
A27	2944	PENTAGON	NORTH	+2 min	38.8119	-77.1267	22:16:55	22:10:00	22:54:00	4565020	F109
A40	3186	CRYSTAL CITY	EAST	+10 min	38.8565	-77.1094	22:16:49	22:00:00	22:30:00	3048020	F196
A58	2991	FARRAGUT SQ	EAST	+4 min	38.9023	-77.8385	22:13:55	21:16:00	22:09:00	2913020	F163
A58	3169	SEVEN CORNERS	WEST	+10 min	38.8871	-77.8955	22:17:15	21:48:00	22:37:00	6101020	F233
A58	3155	FARRAGUT SQ	EAST	ON TIME	38.8915	-77.8832	22:17:05	21:46:00	22:39:00	11369020	F231
A58	3122	SEVEN CORNERS	WEST	+2 min	38.9029	-77.8508	22:17:18	22:18:00	23:07:00	15230020	F148
A58	3198	FARRAGUT SQ	EAST	+1 min	38.8786	-77.1535	22:17:08	22:16:00	23:09:00	30762020	F169
A66	2946	CULMORE	WEST	+1 min	38.8633	-77.8543	22:17:06	22:11:00	22:55:00	645020	F174
A70	2984	POTOMAC YARD	EAST	+2 min	38.8348	-77.8487	22:17:12	21:18:00	22:16:00	28713020	F168

# Rail Map



# 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

```
function parseJSON(val) {  
    // 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.  
  
    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);
```

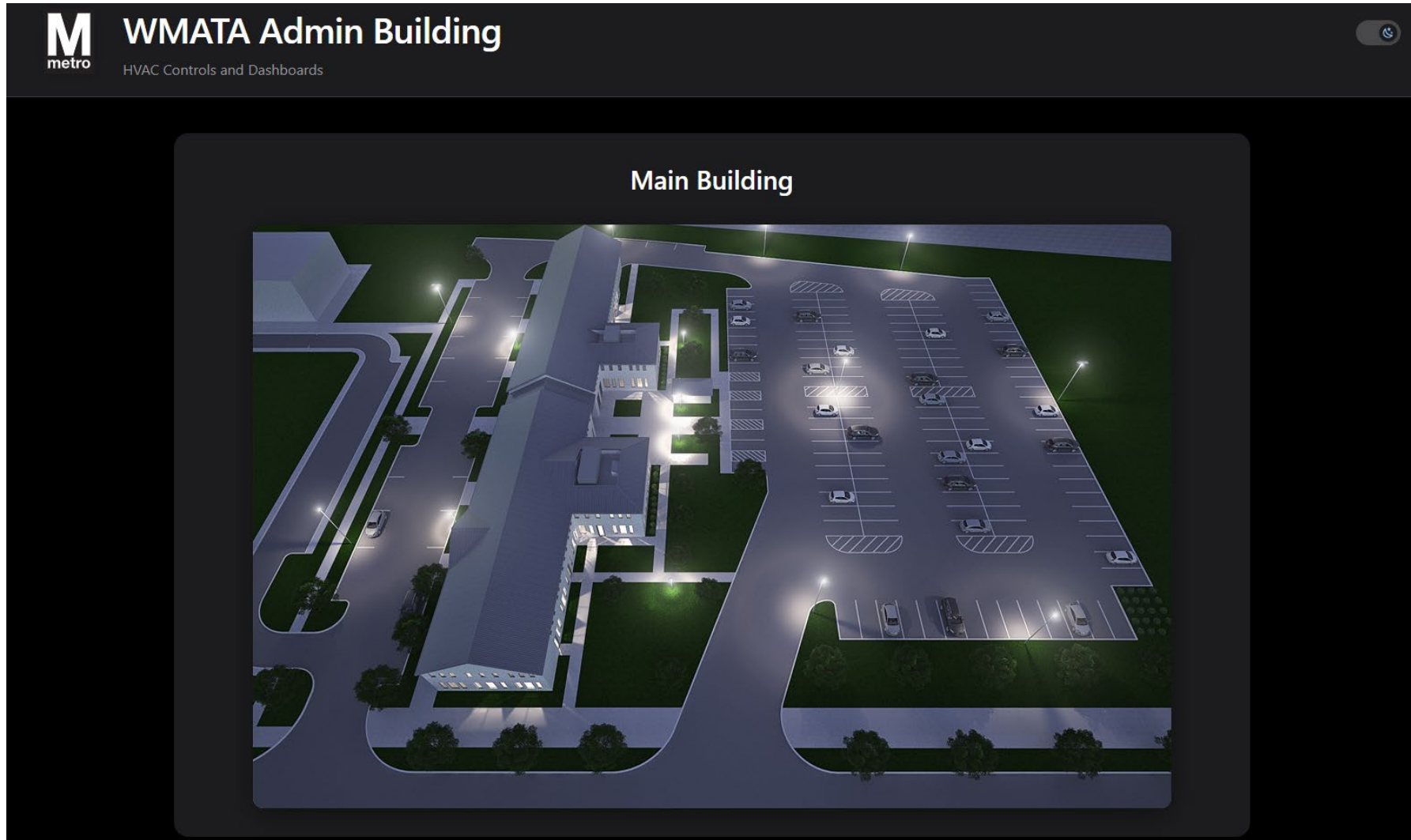
# Light/Dark Mode – CSS Variable Cascade



```
// 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



WMATA Admin Building

HVAC Controls and Dashboards



Main Building



# Questions

