CURB DATA SPECIFICATION

SPEC DRAFT AND IDEAS

Jun-Oct 2021

DRAFT DOC FOR CDS DISCUSSIONS

Draft Curb Data Specification ideas and details to be worked out and commented on here, for future inclusion into the CDS GitHub repo by the Working Group Steering Committee.

Document structure:

1. The first section (CURB DATA SPECIFICATION) shows what will be included in the repo in Markdown in the future.
2. The second section (IDEAS/DISCUSSIONS) is a free-form area to discuss and share things that may go into the first section.

CDS WORK SHIFTED TO GITHUB

Thanks to everyone who contributed comments and suggestions to this working draft document. Now all discussions and spec work are happening on GitHub where you can see this document turned into a spec. Please leave your thoughts as Issues or Discussions, and you can make code/spec suggestions with Pull Requests.

CURB DATA SPECIFICATION

Overview

Urban curb is a valuable, limited, and often under-managed part of the public right of way. Curb demand is growing, including from commercial activity like passenger pickup/drop off, traditional and on-demand delivery services, new mobility programs like scooters, bikeshare, and carshare, and goods and freight delivery. While cities have made some progress in digitizing their curb and using curb data, more tools are needed to proactively manage curbs and sidewalks, and in doing so deliver more public value from this scarce resource. Curb data standards can provide a mechanism for expressing static and dynamic regulations, measuring activity at the curb, and providing access and utilization for curb users.

The Curb Data Specification (CDS) will help cities and companies pilot and scale dynamic curb zones that optimize commercial loading activities of people and goods, and measure the impact of these programs.

In the short term we will produce standards that are necessary to support dynamic curb utilization use cases for commercial loading activities by privately operated companies. In the long term we want to help cities manage their curb zone programs and surrounding areas and measure the utilization and impact for all use cases.

The CDS will be consumed by both cities and transportation providers operating in the public right of way. In many cases, the same mobility providers using curbs with CDS may also be interacting with other OMF MDS Policy, MDS Provider, and MDS Agency data objects within the same MDS Jurisdiction, and MDS Geography. Consistent with the Technology Design Principles codified in the OMF Architectural Landscape Document, the members of this working group are making reasonable best efforts to ensure that work is both *modular* and *interoperable* with other technology managed by the OMF as to avoid duplication and downstream implementation complexity.

Work in Progress

The CDS is a work in progress and is under ongoing development by the community. The goal is to release a beta version of the spec by the end of 2021 and incorporate feedback from pilot programs into future versions.

Working Group

The OMF's Curb Management Working Group, led by a steering committee of individuals representing public agencies and private sector companies, is developing this data specification for curb usage. The CDS will encompass digitized curb regulations, real time and historic event information, and utilization metrics. The specification will allow sharing of this data between cities, commercial companies, and the public.

For more information see our current Curb Management Working Group page.

Curb Data Specification APIs

CDS is at its core a set of Application Programming Interfaces (APIs) and endpoints within those APIs, which allow information to flow between organizations using and managing the curb areas.

CDS is a data exchange format and a translation layer between internal systems and external entities using data feeds. It is not expected that CDS will be the format used internally to store curb regulations in a city. The internal storage format is something different, and a subset of that data should be able to be converted to CDS for publishing out to the public and curb users.

Curbs API

A way for cities to specify areas of interest along the curb that can be connected to events and regulations and shared with companies and the public.

Open Discussion Topics for Curb

Topics that are unresolved, required a decision to be made, or warrant future discussion.

1. Curb regulations should fit the use cases needed for the first draft of CDS, but also be built so they can expand to all curb areas in the long term.
2. Curb areas cannot be defined across multiple blocks, including cross streets, alleys, and service roads. We need to see if some cities have use cases where this is not viable.
3. How should pricing be defined in the rules within the Curb Regulation object? In new Policy object.
4. Unit of measurement. Meters as floating point decimal numbers (eg, 5.28) are used throughout the spec. But are cities using and thinking in feet in some areas and is it a chore to convert integer feet measurements to meters everywhere (eg. 6 feet may have to be 1.8288 meters)? If we need feet, where do we specify the unit, next to each field occurrence (eg, 'length' and 'length_units'), or more globally at a higher level in the spec? Agreed on centimeters.
5. How does CDS / and the Curb API work in conjunction with OMF's MDS, like MDS Policy, Geography, and Jurisdiction APIs. No direct overlap/use yet, except maybe using Geography to define curb Areas, Zones, and Spaces, but we can do that later to avoid complexity with serving and connecting to that API.
6. Should endpoints be GeoJSON format with CDS details in the 'properties', or more freely structured JSON like MDS with GeoJSON objects in them when needed? Decided on JSON.

7. How much do we align field names to MDS standards when talking about Curb Regulation objects, since this borrows a lot from Curb LR and some terms/fields are differently named?
8. How dynamic should the endpoints be? Should the processing be done on the host's side with dynamic parameters and filtering being possible, or should the endpoints be more static and serve larger amounts of data at once with less querying? The former is more complex and likely requires third parties, where the latter may be easier for cities to implement on their own.

Curb Zone objects

A Curb Zone is a geographical entity representing a single region along the curb. What constitutes an individual curb location is determined by the city, but all curb locations MUST meet the following criteria:

1. **Always have a common regulation** along their entire extent (i.e., if at certain times of day, half of a given stretch of curb is a loading zone and the other half is metered parking, that stretch of curb must be divided into at least two curb locations);
2. **Never span multiple blocks** -- an entire curb location must be on the same street and be between the same two cross streets, alleys, or service roads.
3. **Never overlap other Curb Zones** in the same dataset with overlapping validity times. This applies to both polygon geometries and linear references (if used).
4. Be assigned a **unique ID**, in the form of a UUID. This ID SHOULD remain consistent as long as the curb location's extent and regulations remain substantially the same.
5. It SHOULD NOT be possible to legally park a single vehicle in **two different curb locations** at the same time (i.e., a given non-demarcated parking area or loading zone should be represented as a single curb location), unless this conflicts with the requirements above.

Curb Zone Fields

A Curb Zone is represented as a JSON Feature, whose fields are as follows:

- **curb_zone_id**: *String; required.* The UUID for the curb location.
    - *New ID:* When a Curb Zone ceases to be valid, or it needs substantive changes, its ID must not be reused by a Curb Zone with different data, even if it occupies the same location. If data updates reflect changes in the physical world or agency regulations, a new ID MUST be assigned.
    - *No new ID:* data updates that reflect changes in how the same physical locations or regulations are modeled digitally -- e.g., additional metadata, or regulations being modeled more accurately -- should not be implemented by ending a Curb Zone's validity and creating a new one with a new ID. The existing Curb Zone can be updated silently with the new data; callers should not rely on a Curb Zone with the same ID remaining identical over time.
- **geometry**: *Object; required.* A GeoJSON Polygon geometry representing the spatial extent of this curb zone.
- **curb_policy_ids:** *Array of strings, required.* An array of UUIDs, each one of which is the ID of a Policy object. Together, these define the regulations of this Curb Zone.
- **start_date:** *String, required.* An RFC 3339 date/time string defining the earliest time that the data for this curb location is known to be valid. This could be the date on which the data was collected, for instance. This MUST never change for a given id.
- **end_date:** *String, optional.* An RFC 3339 date/time string defining the time at which the data for this curb location ceases to be valid. If not present, the data will be presumed to be valid indefinitely.
- **location_references:** *Array, optional.* An array of one or more Location Reference objects defining linear reference information for this curb location.

- **name:** *String, optional.* A human-readable name for this Curb Zone that identifies it to end users.
- **user_zone_id:** *String, optional.* An identifier that can be used to refer to this Curb Zone on physical signage as well as within mobile applications, typically for payment purposes.
- **street_name:** *String, optional.* The name of the street that this curb location is on.
- **cross_street_start_name**: *String, optional.* The name of the cross street at the start of this curb location (which cross street is at the start and end of the location is defined in the same direction as the linear reference for this curb; if no linear reference is provided, start and end SHOULD be oriented such that start comes before end when moving in the direction of travel for the roadway immediately adjacent to the curb.)
- **cross_street_end_name**: *String, optional.* The name of the cross street at the end of this curb location.
- **length:** *Integer, optional.* The length, in centimeters, of the Curb Zone when projected along the street centerline. Note that this is the definitive length of the curb area, and not the edge length of the geographic polygon.
- **available_space_lengths:** *Array, optional.* If availability information is present, an array of numbers containing the lengths (in centimeters) of all known non-overlapping available spaces within this Curb Zone. In cases where availability is known less precisely, this data MAY be inferred from a model.
- **availability_time:** *String, optional.* If availability information is present, the RFC 3339 timestamp corresponding to the most recent time that availability was computed for this zone.
- **width:** *Integer, optional.* The width, in centimeters, that the Curb Zone occupies from the curb to the roadway lane.
- **parking_angle:** *String, optional.* The angle in which passenger vehicles in this Curb Zone are meant to park. May take one of the following values:
  - parallel

    - perpendicular
    - angled
- **num_spaces:** *Integer, optional.* The number of demarcated spaces within this Curb Zone. Demarcated spaces may also be specified using the **`curb_spaces`** API endpoint.
- **street_side:** *String, optional.* The cardinal or subcardinal direction representing the side of the roadway that this curb is on. Maybe "N", "NE", "E", "SE", "S", "SW", "W", or "NW". For cities with "grid directions", the side MAY be based on the grid direction rather than the closest true-north compass direction, but MUST NOT be more than 90 degrees away from the true compass direction.
- **median:** *Boolean, optional.* If "true", this curb location is on the median of a street, rather than its edge. A median is a strip of land separating two roadways within the same street. Note that, for medians, street_side is interpreted relative to the roadway that the particular curb is on, so the curb along the median of the southern roadway of a divided street would have street_side of "N".
- **entire_roadway**: *Boolean, optional.* If "true", this curb location takes up the entire width of the roadway (which may be impassible for through traffic when the Curb Zone is being used for parking or loading). This is a common condition for alleyways. If entire_roadway is "true", street_side MUST NOT be present.
- **curb_area_id:** *String, optional.* The UUID of the Curb Area that this Curb Zone is a part of. If specified, the area identified MUST be retrievable through the Curb API and its geographical area MUST contain that of the Curb Zone.

Location Reference objects

A Location Reference defines a linear reference for a given Curb Zone. A linear reference defines a Curb Zone's position by reference to a linear feature, like a street centerline or edge-of-pavement line. Linear referencing systems can be global, like SharedStreets linear references or OpenLR, or local to a particular city.

Location Reference Fields

A Location Reference is a JSON object with the following fields:

- **source:** *String, required.* An identifier for the source of the linear reference. This field MUST contain a URL that provides more information about the underlying map or reference system. Values include (but other can be used):
    - https://sharedstreets.io: SharedStreets
    - http://openlr.org: OpenLR
    - https://coord.com: Coord
    - https://yourcityname.gov: custom city LR, direct link if possible
- **ref_id:** *String, required.* The linear feature being referenced (usually a street or curb segment). For OpenLR, this is the Base64-encoded OpenLR line location for the street segment of which this Curb Zone is part, and the start and end offsets below are relative to this segment.
- **start:** *Integer, required.* The distance (in centimeters) from the start of the referenced linear feature to the start of the Curb Zone.
- **end:** *Integer, required.* The distance (in centimeters) from the start of the referenced linear feature to the end of the Curb Zone. *end* MAY be smaller than *start*, implying that the direction of the Curb Zone is opposite to the direction of the referenced linear feature.
- **side:** *String, optional.* If the referenced linear feature is a roadway, the side of the roadway on which the Curb Zone may be found, when heading from the start to the end of the feature in its native orientation. Values are "left" and "right". MUST be absent for features where entire_roadway is true.

Policy objects

A Policy object is a rule that allows or prohibits a particular set of users from using a particular curb at a particular time or times. Multiple Policy objects together define the full extent of regulations within a Curb Zone. Much of this is similar to curbLR.

The "policy" field within the FeatureCollection returned by /curb/zone contains a list of the Policy objects referenced by the returned zones.

A Policy contains the following fields:

- **curb_policy_id:** *String, required.* A UUID that uniquely identifies this exact regulation across Curb Zones. Two Policy objects containing the same curb_policy_id MUST be completely identical. A curb_policy_id MUST never be reused—once created, it must continue to refer to the identical policy forever.
- **priority**: *Integer, required.* Specifies which other regulations on the same Curb Zone this one takes precedence over. If two Policies have overlapping Time Spans and apply to the same user class, the one that applies at a given time is the one with the lowest priority. Two policies that apply to the same Curb Zone with overlapping Time Spans and user classes MUST NOT have the same priority.
- **rules:** *Array of objects, required.* A single policy can include multiple rules, but each of these rules MUST specify disjoint lists of user classes.
    - **activity:** *String, required.* The activity that is forbidden or permitted by this regulation. Value MUST be one of the following:
        - **travel** (to represent curbside lanes intended for moving vehicles, like bus lanes, bike lanes, and rush-hour-only travel lanes; implies that parking, loading, and stopping are prohibited)
        - **parking** (implies that loading and stopping are also permitted)
        - **no parking**
        - **loading** (of goods; implies that stopping is also permitted)
        - **no loading** (implies that parking is also prohibited)
        - **stopping** (stopping briefly to pick up or drop off passengers)
        - **no stopping** (stopping, loading, and parking are all prohibited)

- **max_stay:** *Integer, optional.* The length of time (in minutes) for which the curb may be used under this regulation.
- **no_return:** *Integer, optional.* The length of time (in minutes) that a user must vacate this Curb Zone before allowed to return for another stay.
- **user_classes:** *Array, optional.* If specified, this regulation only applies to users matching the classes contained within. If not specified, this regulation applies to everyone. New user classes MAY be generated by data providers to reflect their regulations, but when possible, the following known values should be used:
  - bicycle
  - bus
  - handicap
  - motorcycle
  - official
  - permit
  - rideshare
  - taxi
  - commercial (applies to "truck"; if both "commercial" and "truck" are specified, trucks use the "truck" rule).
  - truck
- **rate:** *Array, optional.* The cost of using this Curb Zone when this regulation applies. Rates are repeated to allow for prices that change over time. For instance, a regulation may have a price of $1 for the first hour but $2 for every subsequent hour.
  - **per_hour:** *Integer, required.* The rate per hour for this space in cents (or the smallest denomination of local currency).
  - **increment_minutes**: *Integer, optional.* If specified, this is the smallest amount of time a user can pay for (e.g., if increment_minutes is 15, a user can pay for 15, 30, 45, etc. minutes).

    - **increment_amount**: *Integer, optional.* If specified, the rate for this space is rounded *up* to the nearest increment of this amount, specified in the same units as per_hour.
    - **start_minutes**: *Integer, optional.* The amount of time the vehicle must have already been present in the Curb Zone before this rate starts applying. If not specified, this rate starts when the vehicle arrives.
    - **end_minutes:** *Integer, optional.* The amount of time after which the rate stops applying. If not specified, this rate ends when the vehicle departs.
- **time_spans:** *Array, optional.* If specified, this regulation only applies at the times defined by the Time Span objects within.

The JSON response to the object MUST contain a "" property, whose value is a Metadata object with the following properties:

...

Time Span objects

A Time Span defines a period of time (that may occur once or repeatedly) during which a given regulation applies. When multiple fields are combined, *all* criteria must be met in order for a given Time Span to apply. For instance, if days_of_week is ["mon", "tue"], time_of_day_start is "10:00", and time_of_day_end is "13:00", the Time Span contains all times between 10AM and 1PM on Mondays and Tuesdays.

Time Span Fields

A Time Span object contains the following fields:

- **from**: *String, optional.* An RFC 3339 date/time string defining the earliest point in time that this Time Span could apply. If unspecified, the Time Span applies to all matching periods arbitrarily far in the past.
- **to:** *String, optional.* An RFC 3339 date/time string defining the latest point in time that this Time Span could apply. If unspecified, the Time Span applies to all matching periods arbitrarily far in the future.
- **days_of_week:** *Array, optional.* An array of days of the week when this Time Span applies, specified as 3-character strings ("sun", "mon", "tue" ,"wed", "thu", "fri", "sat").
- **days_of_month:** *Array, optional.* An array of days of the month when this Time Span applies, specified as integers (1-31). Note that, in order to specify, e.g., the "2nd Monday of the month", you can use days_of_month combined with days_of_week (in this case, days_of_week = ["mon"] and days_of_month = [8,9,10,11,12,13,14]).
- **months:** *Array, optional.* An array of months as integers (1=January, 12=December). If specified, this Time Span applies only during these months.
- **time_of_day_start:** *String, optional.* An "HH:MM" string representing the 24-hour local time that this Time Span starts to apply. If unspecified, this Time Span starts at midnight.
- **time_of_day_end:** *String, optional.* An "HH:MM" string representing the 24-hour local time that this Time Span stops applying. This is not inclusive, so for instance if time_of_day_end is "17:00", this Time Span goes up to 5PM but does not include it. If unspecified, this Time Span ends at midnight.
- **designated_period:** *String, optional.* A string representing an arbitrarily-named, externally-defined period of time. Any values MAY be specified but the following known values SHOULD be used when possible:
    - snow emergency
    - holidays
    - school days
    - game days

- **designated_period_except:** *Boolean, optional.* If specified and true, this Time Span applies at all times *not* matching the named designated period. (e.g., if designated_period is "snow emergency" and designated_period_except is true, this Time Span does not apply on snow days).

Curb Area objects

Defines curb areas in a city and their properties. A Curb Area is a particular neighborhood or area of interest that includes one or more Curb Zones. Important notes about Curb Areas:

- Curb Areas may overlap with other Curb Areas
- Curb Areas are not meant to be city-wide, and instead should be an area of interest around a Curb Zone that is no bigger than a neighborhood.

Curb Area Fields

A Curb Area is represented as a JSON Feature, whose fields are as follows:

- **curb_area_id**: *String; required.* The UUID for the curb area.
- **geometry**: *Object; required.* A GeoJSON Polygon geometry representing the spatial extent of this curb location.
- **name:** *String; required.* The name of this curb area.
- **curb_zone_ids**: *Array, required.* The UUIDs of all the Curb Zones included within this Curb Area at the requested time.

Curb Space objects

Defines individual demarcated spaces within a Curb Zone. Important notes about Curb Spaces:

- Curb Spaces may NOT overlap with other Curb Spaces
- Curb Spaces must be wholly contained within a single Curb Zone

Curb Space Fields

A Curb Space is represented as a JSON Feature, whose fields are as follows:

- **curb_space_id**: *String; required.* The UUID for the curb space.
- **geometry**: *Object; required.* A GeoJSON Polygon geometry representing the spatial extent of this curb location.
- **curb_zone_id:** *String; required.* The UUID of the Curb Zone this space is within. The geometry of the specified Curb Zone MUST contain the geometry of this space.
- **space_number:** *Integer; optional.* The sequence number of this space within its Zone. If specified, two spaces within the same Curb Zone MUST NOT share a space number, and space numbers SHOULD be consecutive positive integers starting at 1.
- **length**: *Integer; required.* Length in centimeters of this Space. Note vehicles may have to account for a buffer for doors, mirrors, bumpers, ramps, etc.
- **width**: *Integer; required.* Width in centimeters of this Space. Note vehicles may have to account for a buffer for doors, mirrors, bumpers, ramps, etc.
- **available**: *Boolean; optional.* Whether this space is available for vehicles to park in at the specified time ('True' means the Space is available). Provided if agencies have the capability to track this in near real-time.
- **availability_time:** *String, optional.* If availability information is present, the RFC 3339 timestamp corresponding to the most recent time that availability was computed for this space.

Metadata objects

Every JSON Curbs API object MUST contain a "metadata" property, whose value is a Metadata object with the following fields:

- **time_zone**: *String, required.* The time zone that applies to parking regulations in this dataset. MUST be a valid TZ database time zone name (e.g. "US/Eastern" or "Europe/Paris").
- **currency**: *String, required.* The ISO 4217 3-letter code for the currency in which rates for curb usage are denominated.
- **author**: *String, optional.* The name of the organization that produces and maintains this data.
- **license_url**: *String, optional.* URL to the licensing terms under which this data is provided.

Examples

Get All Curb Zones

Below is an example Curb Zone object. Its regulations specify no stopping 7am-8:30am every 2nd Monday of the month (for street sweeping); 30-minute commercial loading only from 7am to 1pm on weekdays; two-hour, $4-per-hour parking 1pm to 6pm on weekdays; and unlimited free parking all other times.

GET /curb/zone

```
{
   "id": "24a025d3-01d0-4a1f-aed1-6554921720ea",
   "geographies": [ {
      "type": "Feature",
      "geometry": {
         "type": "Polygon",
         "coordinates": [ [
            [-85.76236289,38.25727805], [-85.76301090,38.25735176],
            [-85.76299281,38.25742378], [-85.76235544,38.25735811],
            [-85.76236289,38.25727805]
         ] ]
      } } ],
   "regulations": [ {
      // Street cleaning 7-8:30 2nd Monday of every month
      "rule": {"activity": "no stopping"},
      "time_spans": [ {
         "time_of_day_start": "07:00",
         "time_of_day_end": "08:30",
         "days_of_week": ["mon"],
         "days_of_month": [8, 9, 10, 11, 12, 13, 14]
      } ]
   }, {
      // 30 min commercial loading 7am-1pm weekdays
      "rule": {"activity": "loading", "max_stay": 30},
      "user_classes": [ "commercial" ],
      "time_spans": [ {
```

```
      "time_of_day_start": "07:00",
      "time_of_day_end": "13:00",
      "days_of_week": ["mon", "tue", "wed", "thu", "fri"]
   } ]
}, {
   // All others no stopping during loading hours
   "rule": {"activity": "no standing"},
   "time_spans": [ {
      "time_of_day_start": "07:00",
      "time_of_day_end": "13:00",
      "days_of_week": ["mon", "tue", "wed", "thu", "fri"]
   } ]
}, {
   // 2 hour $4/hour parking 1pm-8pm weekdays
   "rule": {"activity": "parking", "max_stay": 120},
   "time_spans": [ {
   "time_of_day_start": "13:00",
   "time_of_day_end": "18:00",
   "days_of_week": ["mon", "tue", "wed", "thu", "fri"]
} ],
"rate": [ {"per_hour": 400} ]
}, {
   // Unrestricted parking by default
   "rule": {"activity": "parking"}
}],
"valid_as_of": "2021-03-21T12:00:00.000Z",
"location_references": [ {
   // SharedStreets linear reference
   "source": "https://sharedstreets.io",
   "ref_id": "592c2106e6553d3c930a372763f10254",
   "start": 5,
   "end": 38.5,
   "side": "right"
} ],
"street_name": "Main St",
"cross_street_start_name": "1st St",
"cross_street_end_name": "2nd St",
"street_side": "W",
  "length": 33.5,
  "parking_angle": "parallel",
  "num_spaces": 4,
  "curb_area": "3c29a043-f8a1-496b-a554-990dd7f7e1e2"
}
```

Endpoints

GET /curbs/zone: Query Curb Zones.

Required.

Parameters (all optional):

- **area**: The UUID of a Curb Area. If specified, the provider MAY only return Curb Zones contained within this area.
- **min_lat/min_lng/max_lat/max_lng**: Specifies a bounding box; if one of these parameters is specified, all four MUST be. If specified, the provider MAY only return Curb Zones that intersect the supplied bounding box.
- **lat/lng/radius**: If one of these parameters is specified, all three MUST be. If specified, returns only Curb Zones that are within radius centimeters of the point identified by lat/lng. Curb Zones in the returned feature collection MUST be ordered ascending by distance from the center point.
- **include_geometry**: If the value is "false", the API provider MAY choose not to include the "geometry" field within the Curb Zone feature object.
- **include_policy**: If the value is "false", the API provider may choose not to include the "policy" field within the returned FeatureCollection; policy IDs MUST still be returned within Curb Zone objects.
- **time**: An RFC 3339 date/time string that specifies a particular moment in time. If not present, defaults to the current time. The provider MAY choose not to return Curb Zone objects whose validity period does not include this time. This is also used for the timing of availability data (if supplied).

Returns: a JSON file whose features are Curb Zones, also containing `metadata` and `policy` fields.

GET /curbs/zone/<id>: Fetch a single Curb Zone.

Optional.

Parameters (optional):

- **time**: An RFC 3339 date/time string that specifies a particular moment in time. If not present, defaults to the current time, or (if the requested zone's end_date is before the current time) the effective_date of the requested zone). This is used for the timing of availability data (if supplied). If a time is specified that is before the requested zone's effective_time or at or after its end_time, the server SHOULD return a 400 status code.
- **include_geometry**: If the value is "false", the API provider MAY choose not to include the "geometry" field within the Curb Zone feature object.

Returns: a single JSON Curb Zone feature.

GET /curbs/policy: Query Curb Policies.

Optional.

Parameters (optional):

- **ids**: A comma-separated list of policy IDs, the policies to return. If not present, returns all policies known by the provider. The provider MAY choose to return additional policies not listed.

Returns: A JSON object with a single `policy` field, containing a list of Policy objects.

GET /curbs/policy/<id>: Fetch a single Curb Policy.

Optional.

Parameters: none

Returns: a single Policy object matching the requested ID.

GET /curbs/area: Query Curb Areas

Optional (if not implemented, server should reply with HTTP 501).

Parameters (all optional):

- **min_lat/min_lng/max_lat/max_lng**: Specifies a bounding box; if one of these parameters is specified, all four MUST be. If specified, returns Curb Areas that intersect the supplied bounding box.
- **lat/lng/radius:** If one of these parameters is specified, all three MUST be. If specified, returns only Curb Areas that are within radius centimeters of the point identified by lat/lng. Curb Areas in the returned feature collection MUST be ordered ascending by distance from the center point.
- **time**: An RFC 3339 date/time string that specifies the time for which returned Curb Zones will be valid; if not specified, defaults to the current time.

Returns: a JSON file whose features are Curb Areas, also containing `metadata` and `policy` fields.

GET /curbs/area/<id>: Fetch a single Curb Area

Optional (if not implemented, server should reply with HTTP 501).

Parameters:

- **time**: An RFC 3339 date/time string that specifies the time for which returned Curb Zones will be valid; if not specified, defaults to the current time.

Returns: a single JSON Curb Area feature.

GET /curbs/space: Query Curb Spaces

Optional (if not implemented, server should reply with HTTP 501).

Parameters (all optional):

- **area**: The UUID of a Curb Area. If specified, will only return Curb Spaces contained within this area.
- **zone**: The UUID of a Curb Zone. If specified, will only return Curb Spaces contained within this area.
- **min_lat/min_lng/max_lat/max_lng**: Specifies a bounding box; if one of these parameters is specified, all four MUST be. If specified, returns Curb Spaces that intersect the supplied bounding box.
- **lat/lng/radius:** If one of these parameters is specified, all three MUST be. If specified, returns only Curb Spaces that are within radius centimeters of the point identified by lat/lng. Curb Spaces in the returned feature collection MUST be ordered ascending by distance from the center point.
- **time**: An RFC 3339 date/time string that specifies a particular moment in time, to be used for availability data (if supplied) If not present, defaults to the current time.
- **available**: If present, may be `true` or `false`, and the API will return only available or unavailable spaces (respectively). Spaces without availability information will not be returned. If the available parameter is set but the API being queried has no availability information whatsoever, the server MAY return an HTTP 400 error code in response to the request.

Returns: a JSON file whose features are Curb Spaces, also containing a `metadata` property

GET /curbs/space/<id>: Fetch a single Curb Space

Optional (if not implemented, server should reply with HTTP 501).

Parameters:

- **time**: An RFC 3339 date/time string that specifies a particular moment in time, to be used for availability data (if supplied) If not present, defaults to the current time.

Returns: a single JSON Curb Space feature.

Status Codes

The following HTTP status codes SHOULD be returned by API servers under certain conditions. When a non-200 status code is returned, the response body is implementation-defined, and may or may not be a valid JSON object. Callers should use the Content-Type header to determine how to process the response. Other status codes MAY be returned to reflect additional error conditions per the HTTP protocol and RFC 7231.

- **200 OK:** the request was processed successfully.
- **400 Bad Request:** when a request is malformed, including the following conditions:
  - Invalid parameters (e.g., including radius without lat or lng, or including an improperly-formatted time parameter)
  - A request whose bounds are too large for the server to handle (either because of sheer size or because the returned data would span multiple time zones)
- **404 Not Found:** when a request is made for an endpoint or for an individual area, zone, or space that does not exist.
- **501 Not Implemented:** when a server does not implement a given optional API endpoint.

Events API

The Events API is a way for real-time and historic events at the curb to be sent to cities. Events can come from company data feeds, sensors, payments, check-ins, enforcement, and/or other city data.

Each recorded event is represented by a Curb Event object in an authenticated JSON endpoint.

Curb Event Objects

A Curb Event object contains the following fields:

- **event_id**: *UUID, required*. The globally unique identifier of the event that occurred.
- **event_type**: *Curb Event Type, required*. The event_type happening for this event.
- **event_source_category:** *Enum, required.* General category of the source creating the event.
    - Sensor, company feed, payment, enforcement, monitor, camera, app, ALPR
- **source_operator_id:** *UUID, required.* Unique identifier of the entity responsible for operating the event source.
- **source_id:** *UUID, required.* Unique identifier of this event source, whether sensor, vehicle, camera, etc. Allows agencies to connect related Events as they are recorded.
- **event_location**: *GeoJSON, required*. The geographic point location where the event occurred.
- **event_time**: *Timestamp, required*. Time at which the event occurred.
- **publication_time**: *Timestamp, required*. Time at which the event became available for consumption by this API.
- **curb_zone_id**: *UUID, conditionally required*. Unique ID of the Curb Zone where the event occurred. Required for events that occurred at a known Curb Zone for ALL *event_types*.

- **curb_area_ids**: Array of *UUIDs, conditionally required.* Unique IDs of the Curb Area where the event occurred. Since Curb Areas can overlap, an event may happen in more than one. Required for events that occurred in a known Curb Area for these *event_types: enter_area, exit_area, park_start, park_end*
- **curb_space_id**: *UUID, conditionally required.* Unique ID of the Curb Space where the event occurred. Required for events that occurred at a known Curb Space for these *event_types: park_start, park_end, enter_area, exit_area*
- **provider_id**: *UUID, optional.* Unique ID of the provider responsible for operating the vehicle at the time of the event, if any.
- **provider_name**: *String, optional.* Name of the provider responsible for operating the vehicle, device, or sensor at the time of the event, if any.
- **sensor_id**: *UUID, optional.* If a sensor was used, the globally unique identifier of the sensor that recorded the event.
- **sensor_status**: *Object, optional.* The status of the sensor that reported the event at the time that the event was reported.
  - is_commissioned: Boolean, required. Indicates whether the sensor is currently in a state where it should be reporting data.
  - is_online: Boolean, required. Indicates whether the sensor is currently online and reporting data
- **vehicle_id**: *String, Optional.* Vehicle Identification Number visible on the vehicle itself
- **vehicle_length**: *Integer, conditionally required.* Approximate length of the vehicle that performed the event, in centimeters. Required for sources capable of determining vehicle length.
- **vehicle_type**: *Vehicle Type, conditionally required.* Type of the vehicle that performed the event. Required for sources capable of determining vehicle type
- **propulsion_types**: *Array of Propulsion Type, conditionally required.* List of propulsion types used by the vehicle that performed the event. Required for sources capable of determining vehicle propulsion type.

- **blocked_lane_types**: *Array of Lane Type, conditionally required.* Type(s) of lane blocked by the vehicle performing the event. If no lanes are blocked by the vehicle performing the event, the array should be empty. Required for the following *event_types: park_start*
- **curb_occupants**: *Array of Curb Occupant, conditionally required.* Current occupants of the Curb Zone. If the sensor is capable of identifying the linear location of the vehicle, then elements are sorted in ascending order according to the start property of the linear reference. Otherwise, elements appear in no particular order. Required for the following *event_types*: *park_start, park_end, scheduled_report*

Curb Event Type

event_type. Curb Event Type enumerates the set of possible types of Curb Event. The values that it can assume are listed below:

- **comms_lost**: communications with the event source were lost
- **comms_restored**: communications with the event source were restored
- **decommissioned**: event source was decommissioned
- **park_start**: a vehicle stopped, parked, or double parked
- **park_end**: a parked vehicle leaving a parked or stopped state and resuming movement
- **scheduled_report**: event source reported status status at a scheduled interval
- **enter_area:** vehicle enters the relevant geographic area
- **exit_area:** vehicle exits the relevant geographic area

Curb Occupant Objects

A Curb Occupant object represents a specific vehicle's occupancy in a curb region at a specific point in time. Curb Occupant objects contain the following fields:

- **type**: *Vehicle Type, required.* The type of the occupant. When the event source is not capable of distinguishing vehicle type, this property must take the value "unspecified."

- **length**: *Float, conditionally required*. The approximate length in centimeters of the vehicle. Required when the event source is capable of determining vehicle length.
- **linear_location**: *Array of Float, conditionally required*. A two-element array that specifies the start and end of the occupant's linear location relative to the start of the Curb Zone in that order. Required when the event source is capable of determining the linear location of occupants.

Vehicle Type

Type of vehicle, similar to vehicle_type in MDS. For this CDS release the list will be developed independently here to accommodate CDS and MDS use cases, while still aligning to the MDS design principles. In the next major MDS 2.0 release and next CDS release, alignment between CDS and MDS vehicle types can occur. List:

`vehicle_type`	**Description**
bicycle	A two-wheeled mobility device intended for personal transportation that can be operated via pedals, with or without a motorized assist (includes e-bikes, recumbents, and tandems)
cargo_bicycle	A two- or three-wheeled bicycle intended for transporting larger, heavier cargo than a standard bicycle (such as goods or passengers), with or without motorized assist (includes bakfiets/front-loaders, cargo trikes, and long-tails)
car	A passenger car or similar light-duty vehicle
scooter	A standing or seated fully-motorized mobility device intended for one rider, capable of travel at low or moderate speeds, and suited for operation in infrastructure shared with motorized bicycles

moped	A seated fully-motorized mobility device capable of travel at moderate or high speeds and suited for operation in general urban traffic
other	A device that does not fit in the other categories

Propulsion Type

Propulsion type of the vehicle, similar to propulsion_type in MDS. For this CDS release the list will be developed independently here to accommodate CDS and MDS use cases, while still aligning to the MDS design principles. In the next major MDS 2.0 release and next CDS release, alignment between CDS and MDS propulsion types can occur. List:

`propulsion`	Description
`human`	Pedal or foot propulsion
`electric_assist`	Provides power only alongside human propulsion
`electric`	Contains throttle mode with a battery-powered motor
`combustion`	Contains throttle mode with a gas engine-powered motor

A vehicle may have one or more values from the `propulsion`, depending on the number of modes of operation. For example, a scooter that can be powered by foot or by electric motor would have the `propulsion` represented by the array `['human', 'electric']`. A bicycle with pedal-assist would have the `propulsion` represented by the array `['human', 'electric_assist']` if it can also be operated as a traditional bicycle.

Lane Type

Type(s) of lane blocked by the vehicle performing the event. List:

- ...

Endpoints

GET /events/event: Query a source for their events

...

GET /events/status: Query a source for their current status

...

Examples

...

Metrics API

Calculations to determine historic dwell time, occupancy, usage, and other aggregated statistics in curb areas.

An event/transaction at the curb is defined as…

Unit of measure, time, length, etc…

Metrics Methodology

Cities are facilitating access to the curb for different users based on the curb access priorities of that particular area. The following metrics can be useful in understanding how curb usage aligns with priorities.

Total Events

Methodology

count[events] for a specific time period

Use Case

Cities use this to determine 'demand' for curb space and understand how much activity is happening at the curb. Seems pretty basic but a lot of cities don't have this insight or if they do it's not current or comprehensive.

Turnover

Methodology

count[events]/hour for a specific time period

Use Case

Used together with Average Dwell Time by cities to understand how long vehicles are parked at the curb. When evaluated alongside a vehicle type breakdown, cities can see if the curb space is being used as intended and design better rules for compliance.

Average Dwell Time

Methodology

sum[dwell time] / count[events] for a specific time period

Use Case

Turnover and Average Dwell Time are used together by cities to understand how long vehicles are parked at the curb. When evaluated alongside a vehicle type breakdown, cities can see if the curb space is being used as intended and design better rules for compliance.

For example, a city may want to impose a 5 minute time limit for a restaurant pick-up and drop-off zone if they see that a lot of vehicles are in that space for 30 minutes to help prioritize the quick pick-up and drop-offs.

Another example would be if a city sees that a space has large vehicles with an average dwell time of 90 minutes or more they may want to make sure their time limits and operating hours can accommodate these deliveries so that companies aren't having to leave mid-delivery to move their vehicle.

Occupancy Percent

Methodology

sum[dwell time] / total duration for a specific time period

Use Case

Occupancy is a metric from parking that cities would like to apply to curbs. With parking there's an optimal occupancy where most of the parking spaces are in-use but there are enough open spaces for any vehicle to be able to drive up and find a space. Cities are interested to learn what the optimal occupancy would be for curbs. With so much competition at the curb, cities want to make sure that any loading zones they create are being used a lot, but not so much that they're full and drivers have to double-park instead.

Curb Productivity Index

Methodology

sum [(vehicle ft x dwell time)] / (curb length x total duration) for a specific time period

- end result = number of events per hour for a curb space (or events per day)
- normalizes events across vehicle length and dwell time
- helps understand 'occupancy' as a function of both time and space
- where CPI=1 means 100% time and space occupancy of the curb space

Examples of equivalent hourly CPI for a loading zone 100 ft in length

- a single 80 ft heavy duty vehicle for 60 minutes, CPI = 0.8
- two 40 ft medium duty vehicles for 60 minutes, CPI = 0.8
- four 40 ft medium duty vehicles for 30 minutes, CPI = 0.8
- 48 x 20ft light duty vehicles with food pickup/drop of 5 minutes each, CPI = 0.8

Use Case

City streets are congested and demand for access to curb space is higher than ever before. With so many different types of curb users and spaces cities oftentimes have no idea how to determine curb “performance”. How does a beverage delivery of 30 minutes compare to a on-demand pickup of 5 minutes?

The Curb Productivity Index is based on a Fehr & Peers study for Uber and SFMTA which was also used by the Urban Freight Lab for a TNC study in Seattle. We’ve modified the Curb Productivity Index to normalize events across vehicle types and dwell time so that cities can review and compare curb “performance” across a diverse range of use cases.

We’ve discussed a “productivity index” with cities but haven’t gotten into details as most cities are still looking for basic events and occupancy. We’d be curious to hear any feedback on this concept and ways to improve it.

Metrics Objects

Endpoint: /usage

...

Examples

...

Authentication

How CDS API endpoints are authenticated.

...

IDEAS/DISCUSSIONS

Area to put rough draft ideas, images, and discussions.

Production of CDS Interface and Data Publishing

Need to describe the method of taking data from internal systems and getting it into a CDS-capable publishing and API platform. How does it affect each party, and what is the complexity of the tooling needed?

Double-parking / representing travel lanes

One of the use cases we've discussed is identifying double-parked vehicles. Would we recommend doing this by having two adjacent Curb Zones -- one in the travel lane and one in the curb lane -- or one that's double-wide? (Maybe this is a use case for CurbSpaces)? In either case, should we have metadata that defines which lane(s) a given Curb Zone covers?

Should regulations and areas be separate endpoints?

It is possible that cities will want to refer to curb areas without defining regulations for them, e.g., if their goal is just capturing events for an investigative study or if they are only defining a list of loading zones that all have the same regulations. In this case, it might be easier if we separated the endpoints for curb areas, which defines the geography and metadata of a particular stretch of curb, from the regulations endpoints which define who's allowed to do what there, when. The downside is more endpoints and potentially more work for callers, but it might make the spec cleaner and would make it easier for those who don't want to use regulations. Direction is now, yes, they are separate endpoints.

Should CDS contain a full curb regulation spec?

For the initial use case, it is not necessary to be able to define all possible curb regulations in CDS. However, we know that defining regulations is something that many cities plan to use CDS for going forward, even after our initial launch. So there are two questions here:

- How much regulations stuff do we put into the initial spec?
- How much do we pay attention to future regs extensibility when designing the initial spec?

For example, consider pricing. There are many pricing schemes people use for curbs -- e.g. some meters are $4 for the first hour, $5 for the second, etc.; cities have different minimum time/price increments; sometimes the amount you pay varies based on the payment method or other factors; etc. So if we put in a single field called "price_per_hour", this may make it hard for us to add additional pricing detail in the future without making a breaking change to the spec. So do we make pricing more complicated, knowing we won't be using all that stuff initially? Or not?

It is not expected that CDS will be the format used internally to store curb regulations in a city. The internal storage format is something different, and a subset of that data should be able to be converted to CDS for publishing out to the public and curb users. CDS is a data exchange format and a translation layer between internal systems and external entities using data feeds.

Polygons vs Linear Referencing

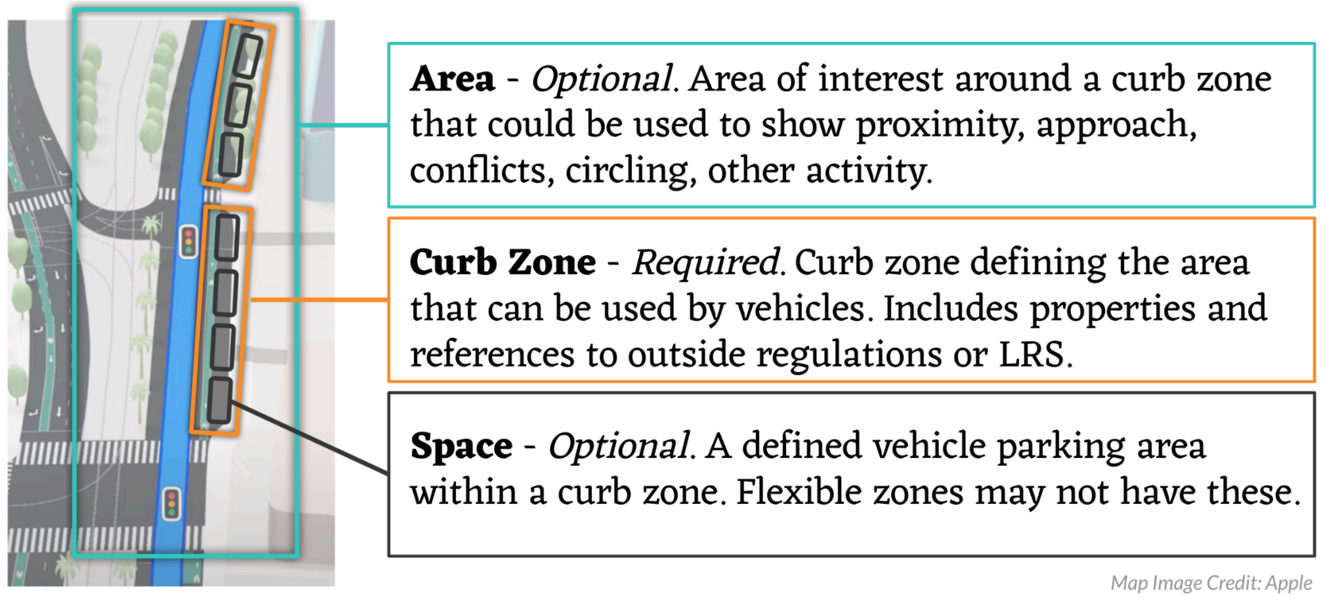
Text

Open API

Text

Regulations Hierarchy and Geometry

Draft ideas on the polygon geometry for CDS and how it relates to LRS.



Area - *Optional*. Area of interest around a curb zone that could be used to show proximity, approach, conflicts, circling, other activity.

Curb Zone - *Required*. Curb zone defining the area that can be used by vehicles. Includes properties and references to outside regulations or LRS.

Space - *Optional*. A defined vehicle parking area within a curb zone. Flexible zones may not have these.

Map Image Credit: Apple

Note for these 2D curb Zones, drawing an approximation of the asphalt curb area is the goal. While GPS will be used for showing if a vehicle is in a curb area, GPS can't be relied on for details, so it's up to the data coming through the Events API (from sensors, company data feeds, etc) to align curb activity with a curb area. Polygon width should not be used to determine lane width and instead width is a field in the curb Area part of the spec.