

# Hammura.bi for Drones

---

## Summary

This codebase compiles all laws relating to unmanned aerial vehicles (drones) in the U.S. and beyond. It represents the laws computationally in a unified framework, and uses geographic information to apply only the laws that are relevant to the user. The fundamental output is a color coded risk assessment (green, yellow, orange, or red) of the regulatory and environmental risks affecting a particular drone flight. This assessment is based on real-time data obtained from a variety of sources, as well as a database of regulatory and geographic information. The interface to this knowledgebase is a web service that can be connected to a mapping program, such as Google Maps, for a visual display of the risk landscape.

Basic web service inputs:

- \* Drone location
- \* Altitude
- \* Vehicle weight and range
- \* Operator type and flight purpose
- \* Sensing equipment
- \* Other conditional information pertaining to law enforcement

Basic outputs:

- \* Color code + reasons
- \* List of no-fly zones
- \* Preflight checklist items
- \* Response time

Summary of risks assessed:

- \* Regulatory violations
  - FAA drone regulations
  - State drone regulations, which tend to relate to law enforcement
  - FAA no-fly zones such as NOTAMS, airports, and flight ceilings
  - Municipal no-fly zones
  - Laws of other English-speaking countries
- \* Environmental hazards
  - Weather risks
  - Populated areas
  - Time of day

# Assessment

## Define the API

Define the form:

```
api = APIFunction[{
  (* {"VehicleLocation", "Latitude/longitude"} → <|
    "Interpreter" → "GeoCoordinates", "Required" → False |>, *)
  "VehicleAddress" → <|"Interpreter" → "String", "Required" → False |>,
  "Altitude" → <|"Interpreter" → "Number", "Required" → False |>,
  "VehicleRange" → <|"Interpreter" → "Number", "Required" → False |>,
  "OperatorType" → <|
    "Interpreter" → {"Commercial", "Military", "Law enforcement",
      "Government", "Researcher", "Hobbyist"}, "Required" → False |>,
  "FlightPurpose" → <|"Interpreter" → {"Recreation",
    "Surveillance/observation", "Agriculture", "Scientific research",
    "Firefighting", "Mapping/surveying", "Environmental observation",
    "Journalism", "Hunting", "Traffic assessment", "Package delivery",
    "Industrial observation", "Utility maintenance",
    "Real estate sales", "Artistic expression", "Use of weapons",
    "Signal interference"}, "Required" → False |>,
  "SensingEquipment" → <|"Interpreter" → {"Photo", "Video", "Audio",
    "Infrared", "Chemical detection", "Facial recognition",
    "Biometric matching"}, "Required" → False |>,
  "VehicleWeight" → <|"Interpreter" → "Number", "Required" → False |>
},
  analyzeAndReturnJSON[#] &, "JSON"
];
```

Currently using this one due to a possible bug with the interpreter for string options (reported but to Wolfram Research on 10/8/15):

```

api = APIFunction[{
    "VehicleAddress" → <|"Interpreter" → "String", "Required" → False|>,
    "Altitude" → <|"Interpreter" → "Number", "Required" → False|>,
    "VehicleRange" → <|"Interpreter" → "Number", "Required" → False|>,

    "OperatorType" → <|"Interpreter" → "String", "Required" → False|>,
    "FlightPurpose" → <|"Interpreter" → "String", "Required" → False|>,
    "SensingEquipment" → <|"Interpreter" → "String", "Required" → False|>,

    "VehicleWeight" → <|"Interpreter" → "Number", "Required" → False|>
},
analyzeAndReturnJSON[#] &, "JSON"
];

```

## Deploy the API

Deploy the API:

```

CloudDeploy[api, "drone_API", Permissions → "Public"]
CloudObject[
  https://www.wolframcloud.com/objects/user-606699a3-52fc-4528-a492-43b4309e137a/]
  drone_API

CloudDeploy[api, "drone_API_test", Permissions → "Public"]
CloudObject[
  https://www.wolframcloud.com/objects/user-606699a3-52fc-4528-a492-43b4309e137a/]
  drone_API_test

```

## High-level control

JSON output via APIFunction. This function returns:

- \* Vehicle position
- \* Color code + reasons
- \* List of no-fly zones
- \* Preflight checklist items
- \* Response time

```

analyzeAndReturnJSON[input_Association] :=
  Block[{allFacts, geo, lat, long, result, vPos, risks, reasonList,
  text, zoneList, circles, polygons, start, end, color},

```

```

start = AbsoluteTime[];

allFacts = facts[input];

geo = allFacts [["VehicleLocation"]];
lat = allFacts [["VehicleLatitude"]];
long = allFacts [["VehicleLongitude"]];
vPos = <|"lat" → lat, "long" → long|>;

risks = riskFactors[allFacts, <||>];

circles = Catenate[{
    airportDataAPIOutput["AIRPORT", geo, 15, 3],
    airportDataAPIOutput["HELIPORT", geo, 15, 0.1],
    circularTFRDataAPIOutput[nearbyUSTFRs[geo, 35]],
    mlbGameDataAPIOutput[geo, 15, 3] (* See FAA TFR
        4/3621 "http://tfr.faa.gov/save_pages/detail_4_3621.html" *)
}];

polygons = Catenate[{
    polygonTFRDataAPIOutput[nearbyUSTFRs[geo, 25]],
    municipalNoFlyAPIOutput
}] ;

(* Select[nearbyUSTFRs[Here,5900] , ##[["Shape"]] == "Polygon"&]; *)

zoneList = <|
    "circles" → circles,
    "polygons" → polygons
|>;

color = If[factsAreOutOfScope[input], "Black", colorCode[risks]];
text = If[factsAreOutOfScope[input],
    {"Application does not handle that scenario"}, colorCodeText[color]];
reasonList = If[factsAreOutOfScope[input], {}, riskFactorsInOrder[risks]];

end = AbsoluteTime[];

result = <|
    (* "rawFacts" → allFacts, *)
    "vehicleLocation" → vPos,
    "vehicleRangeInMeters" → 1000,
    "assessment" →

```

```

    {"colorCode" → color, "text" → text, "reasons" → reasonList},
    "noFlyZones" → zoneList,
    "checklist" → {"Must maintain visual line of sight",
        "Not under influence of drugs or alcohol",
        "Must follow FAA regulations"} ,
    "processingTimeInSeconds" → end - start
    |>;

Return[result]

(* Lookup[rules,"OtherInfo",{}]*)
];

```

## Get the drone's location, given an address

Given a U.S. street address represented as a string, look up the geocoordinates via Google's API, parse the results, and turn it into a GeoPosition object. It's necessary to use Google because the Wolfram "StreetAddress" interpreter doesn't work very well.

```
Clear[geoPositionFromAddress]
```

```

geoPositionFromAddress[address_] :=
geoPositionFromAddress[address] = Block[{data, lat, long},
  data = listOfRulesToAssociation[Import[
    "https://maps.googleapis.com/maps/api/geocode/json?address=" <>
    StringReplace[address, " " → "+" ] <> "&sensor=false&" ,
    "JSON"]];
  lat = data[["results", "geometry", "location", "lat"]];
  long = data[["results", "geometry", "location", "lng"]];
  GeoPosition[{lat, long}]
]

```

```

geoPositionFromAddress["2122 south street phila pa"]
GeoPosition[{39.945, -75.1778}]

geoPositionFromAddress["1059 cornell ave albany ca"] // AbsoluteTiming
{0.246597, GeoPosition[{37.8851, -122.294}]}
```

```

geoPositionFromAddress["nyc"]
GeoPosition[{40.7128, -74.0059}]
```

```

QuantityMagnitude[Latitude[%]]
37.8854
```

```
listOfRulesToAssociation[rules_] := rules // . List[a__] → Association[a]
```

## Generate list of facts and inferences

Clean up the list of input facts by adding quantity units and looking up real-time data based on the user's location:

```

facts[input_Association] := Block[{
    vehiclePosition =
        geoPositionFromAddress[Lookup[input, "VehicleAddress"]], (* 0.25s *)
    altitude = Quantity[Lookup[input, "Altitude", -1], "Feet"] ,
    lat, long, weather, surfaceElevation
} ,

(* Get weather data *)
lat = Latitude[vehiclePosition];
long = Longitude[vehiclePosition];
weather = weatherData[lat, long, "imperial"]; (* 0.15s *)

(* Alt: consider using Google Geo API for elevation data *)
surfaceElevation = GeoElevationData[vehiclePosition]; (* ~0.35s *)

Return[Join[input, <| 

    (* Spatial position *)
    "VehicleLocation" → vehiclePosition,
    "VehicleLatitude" → QuantityMagnitude[lat],
    "VehicleLongitude" → QuantityMagnitude[long],
    "Altitude" → altitude, (* altitude over sea level *)
    "VehicleHeight" → Round[altitude - surfaceElevation, 0.1],

    (* Weather *)
    "WindSpeed" → weather[["wind", "speed"]],
    "WeatherCode" → weather[["weather", "id"]],
    "WeatherDescription" → ToLowerCase[weather[["weather", "main"]]],
    "AirPressure" → weather[["main", "pressure"]],


    (* Vehicle specs *)
    "VehicleWeight" → Quantity[input[["VehicleWeight"]], "Pounds"] ,
    "VehicleRange" → Quantity[input[["VehicleRange"]], "Miles"] ,


    (* Local times *)
    (* "LocalTime" → LocalTime[vehiclePosition], *) (* ~1.5s!!! *)
    "LocalTime" → FromUnixTime[weather[["dt"]]],
```

```

(* time of last weather update *)
"Sunrise" → Sunrise[vehiclePosition, Today], (* 0.03s *)
"Sunset" → Sunset[vehiclePosition, Today], (* 0.03s *)

(* Local geography *)
"SurfaceElevation" → surfaceElevation,
"UrbanAreaQ" → False (* urbanAreaQ[vehiclePosition] 2-3 sec *)
|>]
]

facts[<|"Altitude" → 2820, "VehicleLocation" → GeoPosition[{42.36, -72.51}],
"VehicleAddress" → "2126+race+st+phila+pa",
"VehicleWeight" → 4, "VehicleRange" → 7|>] // AbsoluteTiming
{0.129662, {Altitude → 2820 ft, VehicleLocation → GeoPosition[{39.9577, -75.175}],
VehicleAddress → 2126+race+st+phila+pa, VehicleWeight → 4 lb, VehicleRange → 7 mi,
VehicleLatitude → 39.9577, VehicleLongitude → -75.175, VehicleHeight → 2770.8 ft,
WindSpeed → 7.78, WeatherCode → 804, WeatherDescription → clouds,
AirPressure → 1020, LocalTime → Mon 6 Jul 2015 13:20:45 GMT-4.,
Sunrise → Mon 6 Jul 2015 05:40 GMT-4., Sunset → Mon 6 Jul 2015 20:34 GMT-4.,
SurfaceElevation → 49.2126 ft, UrbanAreaQ → False}}

```

## Out of scope scenarios

```

factsAreOutOfScope[facts_Association] :=
Block[{opType = facts[["OperatorType"]]},
If [opType == "Military" || opType == "Law enforcement", True, False]
]

```

```
factsAreOutOfScope[<"OperatorType" → "Law enforcement">]
```

```
True
```

## Urban area

Determine whether a position is an urban area (and cache the result):

```

urbanAreaQ[pos_] :=
  urbanAreaQ[pos] = Block[{zip = GeoIdentify["ZIPCode", pos]}, 
    If[zip != {}, Return[
      ZIPCodeData[zip, "PopulationDensity"][[1]] >
        Quantity[5000, Times[Power["Miles", -2], "People"]]
    ]];
    Return[False];
  ]

```

```

urbanAreaQ[Here]
True

urbanAreaQ[GeoPosition[{39.11, -77.86}]] // AbsoluteTiming
{1.99334, False}

```

## Determine applicable rules (not currently used)

The basic idea is to: (1) identify all applicable jurisdictions (based on the drone's location), (2) lookup the encoded rules for each of those jurisdictions, (3) use the system inputs to filter out rules that don't apply, and (4) merge the rules from all jurisdictions together into one association. Note that in the rule association, every value is a list (e.g. "FlightCeiling" -> {400}).

```

applicableRules[input_Association] :=
  Block[{jurisdictions = GeoIdentify["City" | "USCounty" | "USState" | "Country",
    input[["VehicleLocation"]]}, 
  Return[
    Merge[Map[getApplicableRules[#, input] &, jurisdictions ],
      DeleteDuplicates[Flatten[#]] &]
  ];
]

```

```

applicableRules[<|"VehicleLocation" -> Here, "OperatorType" -> "Hobbyist",
  "VehicleWeight" -> Quantity[33, "Pounds"], "CBOCertified" -> True|>]
<| FlightCeiling -> {400 ft}, ProximityAirport -> {5 mi},
  ProximityNationalBorder -> {0 mi}, VLOSRequired -> {True}, PermitsRequired -> {None} |>

```

Fetch the rules from a jurisdiction and apply the fact pattern to them in order to see the applicable rules on flying a UAV:

```

getApplicableRules[jurisdiction_,
  facts_Association] := ReleaseHold[policy[jurisdiction] /. Normal[facts]]

```

```

getApplicableRules[
  Entity["Country", "UnitedStates"], <|"OperatorType" → "Hobbyist",
  "VehicleWeight" → Quantity[33, "Pounds"], "CBOCertified" → True|>]
<| FlightCeiling → {400 ft}, ProximityAirport → {5 mi},
ProximityNationalBorder → {0 mi}, VLOSRequired → {True}, PermitsRequired → {None} |>

```

## Display preflight checklist

Display a pre-flight checklist:

```

displayPreflightChecklist[rules_] :=
Block[{list = Lookup[rules, "OtherInfo", {}]},
If[Length[list] > 0,
Panel[Column[{Row[{Style["Preflight checklist:", Bold, Medium]}],
Row[{Column[Map[StringJoin[{". ", #}] &, list]]}]
}],
, Null]
]
}

```

```
displayPreflightChecklist[<|
"MustReportAccidentsTo" → {"FAA"}, 
"PreFlightChecklistItems" → {},
"OtherInfo" → {"Vehicle must display FAA registration number",
    "Careless or reckless operation is prohibited",
    "Must observe FAA right-of-way rules in 14 CFR 107.37",
    "Operator must not have medical condition
        that interferes with operation of vehicle",
    "Operator may not be under the influence of drugs or alcohol",
    "May not be operated from moving vehicle (except boat)",
    "May only operate or observe one UAV at a time",
    "May not be less than 2000 feet horizontally from any clouds"}
|>]
```

**Preflight checklist:**

- Vehicle must display FAA registration number
- Careless or reckless operation is prohibited
- Must observe FAA right-of-way rules in 14 CFR 107.37
- Operator must not have medical condition that interferes with operation of vehicle
- Operator may not be under the influence of drugs or alcohol
- May not be operated from moving vehicle (except boat)
- May only operate or observe one UAV at a time
- May not be less than 2000 feet horizontally from any clouds

```
displayPreflightChecklist[<|"OtherInfo" → {"FAA"}|>]
```

## Composite Risk Assessment

### Application of laws to facts

Determines general and regulatory risks and returns the response as a list of risk factors, i.e. {risk, description}.

For weather risks: Determines current weather risks (temperature, wind, pressure, precipitation level, visibility, extreme conditions). Returns the response as a list of risk factors, i.e. {risk, description}. The input to this function is the data structure from weatherData, so this logic is very tied to the OpenWeatherMap API (see <http://openweathermap.org/weather-conditions>).

```
riskFactors[facts_Association, rules_Association] := Block[{
    windSpeed, code, codesMild, codesSerious, codesSevere,
    codesExtreme, weatherDescription, duringDaylightHours,
    applicableFlightCeiling, overFlightCeiling, nearFlightCeiling, inRedZoneQ,
    (* opType = facts[["OperatorType"]], *)
    urbanArea = facts[["UrbanAreaQ"]]
```

```

},
(* Daylight *)
duringDaylightHours =
  facts[["Sunrise"]] < facts[["LocalTime"]] < facts[["Sunset"]];

(* Weather data *)
windSpeed = facts[["WindSpeed"]];
code = facts[["WeatherCode"]];
weatherDescription = facts[["WeatherDescription"]];

(* Weather codes from OpenWeatherMap.org *)
codesMild = {300, 301, 310, 311, 313,
  321, 500, 520, 600, 701, 711, 721, 731, 741, 751, 761, 762};
codesSerious = {302, 312, 314, 501, 502, 521, 531, 601, 615, 616, 620};
codesSevere = {200, 201, 202, 210, 211, 212,
  221, 230, 231, 232, 503, 522, 602, 611, 612, 621, 622, 771};
codesExtreme = {504, 511, 781, 900, 901, 902, 903, 904, 905, 906, 960, 961, 962};

(* Flight ceiling *)
applicableFlightCeiling = Quantity[500, "Feet"];
(* Min[rules[["FlightCeiling"]]], *)  

overFlightCeiling = facts[["VehicleHeight"]] >= applicableFlightCeiling;
nearFlightCeiling = applicableFlightCeiling * 0.9 <
  facts[["VehicleHeight"]] < applicableFlightCeiling;

(* Temporary: in red zone *)
inRedZoneQ = droneInRedRegionQ[facts[["VehicleLocation"]]];

DeleteCases[{  

  (* Time of day *)
  If[! duringDaylightHours, {0.5, "Not daylight hours"},  

    (* Wind risks *)
    If[windSpeed >= 31, {0.8, "Force 7 winds"}],  

    If[24 < windSpeed < 31, {0.6, "Force 6 winds"}],  

    If[18 < windSpeed < 24, {0.3, "Force 5 winds"}],  

    (* Air pressure *)
    If[facts[["AirPressure"]] >= faaAirPressureLimitInhPA ,
      {0.9, "Air pressure over FAA limit"}],  

    (* Weather *)
  ]},  

  (* Red zone *)
  If[inRedZoneQ, {0.9, "Drone in red zone"},  

    If[overFlightCeiling < facts[["VehicleHeight"]], {0.9, "Vehicle height above flight ceiling"},  

      If[nearFlightCeiling < facts[["VehicleHeight"]], {0.9, "Vehicle height near flight ceiling"},  

        If[windSpeed >= 31, {0.8, "Wind speed force 7"}],  

        If[24 < windSpeed < 31, {0.6, "Wind speed force 6"}],  

        If[18 < windSpeed < 24, {0.3, "Wind speed force 5"}],  

        If[faaAirPressureLimitInhPA <= facts[["AirPressure"]], {0.9, "Air pressure below FAA limit"}]
      ]
    ]
  ]
}

```

```

If[MemberQ[codesMild, code], {0.1, "Weather risk: " <> weatherDescription}],
If[MemberQ[codesSerious, code],
  {0.3, "Weather risk: " <> weatherDescription}],
If[MemberQ[codesSevere, code],
  {0.6, "Weather risk: " <> weatherDescription}],
If[MemberQ[codesExtreme, code],
  {0.9, "Weather risk: " <> weatherDescription}],

(* Flight ceiling *)
If[nearFlightCeiling, {0.6, "Near flight ceiling"}],
If[overFlightCeiling, {1, "Over flight ceiling"}],

(* In red zone *)
If[inRedZoneQ, {1, "Currently in a no-fly zone"}]

}, Null]
]

riskFactors[
<|"Altitude" → 220 ft, "VehicleLocation" → GeoPosition[{38.9072, -77.0369}],
"VehicleAddress" → "2126+race+st+phila+pa", "VehicleWeight" → 4 lb,
"VehicleRange" → 7 mi, "VehicleHeight" → 480.8` ft, "WindSpeed" → 6.7`,
"WeatherCode" → 804, "WeatherDescription" → "clouds",
"LocalTime" → Mon 6 Jul 2015 11:17:58 GMT-4., "Sunrise" → Mon 6 Jul 2015 05:40 GMT-4.,
"Sunset" → Mon 6 Jul 2015 20:34 GMT-4., "SurfaceElevation" → 49.21259842519685` ft,
"UrbanAreaQ" → False, "AirPressure" → 1041|>, <||>] // AbsoluteTiming

{1.94304, {{0.6, Near flight ceiling}, {1, Currently in a no-fly zone}}}

```

```

riskFactors[<|"VehicleAddress" → "washington dc",
  "Altitude" → Quantity[600, "Feet"], "VehicleRange" → Quantity[10, "Miles"],
  "OperatorType" → "Hobbyist", "FlightPurpose" → "Recreation",
  "SensingEquipment" → Missing["NoInput"], "VehicleWeight" → Quantity[1, "Pounds"],
  "VehicleLocation" → GeoPosition[{38.9071923, -77.0368707}],
  "VehicleLatitude" → 38.9071923, "VehicleLongitude" → -77.0368707,
  "VehicleHeight" → Quantity[491.7000000000005, "Feet"],
  "WindSpeed" → 4.43, "WeatherCode" → 800,
  "WeatherDescription" → "clear", "AirPressure" → 1023, "LocalTime" →
    DateObject[{2015, 9, 6}], TimeObject[{12, 1, 29.}, TimeZone → -4.], TimeZone → -4.],
  "Sunrise" → DateObject[{2015, 9, 6}], TimeObject[{6, 42}, TimeZone → -4.],
    TimeZone → -4.], "Sunset" → DateObject[{2015, 9, 6},
    TimeObject[{19, 32}, TimeZone → -4.], TimeZone → -4.], "SurfaceElevation" →
    Quantity[108.26771653543307, "Feet"], "UrbanAreaQ" → False|>, <||>]
  {{0.6, Near flight ceiling}, {1, Currently in a no-fly zone}}}

```

## Accumulation of evidence re: risk

Data structure (example):

```

riskFactorsExample =
{{0.1, "Poor visibility"}, {0.2, "Urban area"}, {0.15, "Near no-fly zone"}};

```

Function that determines the cumulative risk based on all risk factors. It uses the formula  $R = 1 - (1 - r_1)(1 - r_2)(1 - r_3) \dots$ , representing the compound risk of independent events.

```

cumulativeRisk[riskFactors_List] :=
  1 - Apply[Times, Map[1 - # &, Map[First[#] &, riskFactors]]]

cumulativeRisk[{{0.6, "Near flight ceiling"}, {1, "Currently in a no-fly zone"}}]
1.

```

## Color codes

Determine the applicable color code (green, yellow, orange, red).

```
colorCode[riskFactors_List] := Block[{risk = cumulativeRisk[riskFactors]},  
  Which[  
    risk ≥ 0.8, "Red",  
    risk ≥ 0.5, "Orange",  
    risk > 0, "Yellow",  
    True, "Green"  
  ]]
```

colorCode[riskFactorsExample]

Yellow

Generate text to go along with the color code:

```
colorCodeText[color_] := Switch[color,  
  "Green", "No risks identified",  
  "Yellow", "Moderate risk",  
  "Orange", "Significant risk",  
  "Red", "Out of compliance",  
  "Black", "Unable to assess"]
```

Generate a list of the risk factors in order of most to least risky:

```
riskFactorsInOrder[riskFactors_List] := Map[#[[2]] &, Reverse[Sort[riskFactors]]]  
  
riskFactorsInOrder[riskFactorsExample]  
{Urban area, Near no-fly zone, Poor visibility}  
  
riskFactorsInOrder[{{0.1, "Weather risk: mist"}, {0.5, "Not daylight hours"}}]  
{Not daylight hours, Weather risk: mist}
```

## Regulations

### Default

```
policy[any_] := <| |>;
```

### Federal

```
policy[Entity["Country", "UnitedStates"]] = Hold[Which[
```

```

(* Model aircraft (P.L. 112-95 Sec.336) *)
"OperatorType" == "Hobbyist" &&
("VehicleWeight" ≤ Quantity[55, "Pounds"] ||
 ("VehicleWeight" > Quantity[55, "Pounds"] && "CBOCertified" == True)) ,
<|
"FlightCeiling" → {Quantity[400, "Feet"]},
"ProximityAirport" → {Quantity[5, "Miles"]},
    (* or 3 mi? unless permission from control tower *)
"ProximityNationalBorder" → {Quantity[0, "Miles"]},
"VLOSRequired" → {True},
"PermitsRequired" → {None},
"OtherInfo" → {"You are expected to follow
                the FAA's Model Aircraft Operating Standards"}
|>,

(* Public safety less than 4.4 lbs.(P.L. 112-95 Sec. 334) *)
"OperatorType" == "Law enforcement" &&
    "VehicleWeight" ≤ Quantity[4.4, "Pounds"] ,
<|
"PermittedTimes" → {"Daylight"},
"PermittedAirspace" → {"Class G"},
"ProhibitedAirspace" → {All}, (* everything except Class G *)
"FlightCeiling" → {Quantity[400, "Feet"]},
"ProximityAirport" → {Quantity[5, "Miles"]}, (* includes heliports *)
"ProximityNationalBorder" → {Quantity[0, "Miles"]},
"LineOfSightRequired" → {True},
"PermitsRequired" → {None}
|>,

(* Small UASs (P.L. 112-95 Sec. 34) *)
("OperatorType" == "Commercial" || "OperatorType" == "Researcher") &&
    "VehicleWeight" < Quantity[55, "Pounds"] ,
<|
"PermittedTimes" → {"SunriseToSunset"},
"PermittedAirspace" → {"ClassG"},
"ProhibitedAirspace" →
    {"ClassA", "ClassB", "ClassC", "ClassD", "ExposedPeople"},
        (* Classes B,C,D ok with air traffic prior approval *)
"FlightCeiling" → {Quantity[500, "Feet"]},
"ProximityAirport" → {Quantity[5, "Miles"]},
"ProximityNationalBorder" → {Quantity[0, "Miles"]},
"MaximumSpeed" → {Quantity[100, "mph"]},
"MinimumVisibility" → {Quantity[3, "Miles"]},
"MinimumDistanceBelowClouds" → {Quantity[500, "Feet"]},

```

```

"VLOSRequired" → {True},
"PermitsRequired" →
    {"UAVOperatorCertificate", "CertificateOfAircraftRegistration"},

"MustReportAccidentsTo" → {"FAA"},

"PreFlightChecklistItems" → {},

"OtherInfo" → {"Vehicle must display FAA registration number",
    "Careless or reckless operation is prohibited",
    "Must observe FAA right-of-way rules in 14 CFR 107.37",
    "Operator must not have medical condition that
        interferes with operation of vehicle",
    "Operator may not be under the influence of drugs or alcohol",
    "May not be operated from moving vehicle (except boat)",
    "May only operate or observe one UAV at a time",
    "May not be less than 2000 feet horizontally from any clouds"}
```

|>,

```

(* TODO: All drones w/in 12 NM of U.S. coast -
    maybe put w/ NOTAMs and other ubiquitous checks *)
(*
win12NMofUS == True ,
<|
"BarometricWarningLevel" → {Quantity[31, "inHG"]}
|>,
```

\*

```

(* Default *)
True, <||>];
```

Add later:

```

Hyperlink["14 C.F.R. 107.37",
  "https://www.federalregister.gov/articles/2015/02/23/2015-03544/operation-and-
    certification-of-small-unmanned-aircraft-systems#sec-107-37"]
```

14 C.F.R. 107.37

## State

### Municipal

Northampton, MA:

```

policy[Entity["City", {"Northampton", "Massachusetts", "UnitedStates"}]] =
  Hold[Which[
(* All drones! Effective date: 6/27/13 *)
True,
<|
"ProhibitedAirspace" →
{Entity["City", {"Northampton", "Massachusetts", "UnitedStates"}]}
|>]];

```

**Flatten[<||>]**

Flatten::normal : Nonatomic expression expected at position 1 in Flatten[<| |>]. >>

**Flatten[ <| |> ]**

## Weather information

### Weather data

#### Make API call and process response

Retrieve real-time weather data from OpenWeatherMap.org. Options for units are “imperial” or “metric.” See [http://openweathermap.org/current#current\\_JSON](http://openweathermap.org/current#current_JSON).

```

weatherData[lat_, long_, units_] := Block[{data},
  data = listOfRulesToAssociation[Import[
    "http://api.openweathermap.org/data/2.5/weather?lat=" <>
    ToString[lat] <> "&lon=" <> ToString[long] <> "&units=" <> units
    , "JSON"]];
]

d = weatherData[39.9, -75.1, "imperial"] // AbsoluteTiming
{0.174681, <| main →
  <| temp → 79.93, pressure → 1021, temp_max → 82.99, humidity → 65, temp_min → 77 |>,
  coord → <| lon → -75.09, lat → 39.9 |>, wind → <| speed → 6.7, deg → 140 |>,
  weather → <| id → 804, description → overcast clouds, main → Clouds, icon → 04d |>,
  base → cmc stations, dt → 1436200118, clouds → <| all → 90 |>,
  sys → <| message → 0.0072, sunset → 1436229097, id → 2359, country → US, type → 1,
  sunrise → 1436175513 |>, id → 4500574, name → Audubon Park, cod → 200 |>}

```

```
d = weatherData[21.3069, -157.858, "imperial"]
⟨| main →
  ⟨| temp → 83.89, pressure → 1012, temp_max → 84.2, humidity → 58, temp_min → 82.99 |⟩,
  coord → ⟨| lon → -157.86, lat → 21.31 |⟩, wind → ⟨| speed → 7.78, deg → 150 |⟩,
  weather → ⟨| id → 801, description → few clouds, main → Clouds, icon → 02n |⟩,
  base → cmc stations, dt → 1435887713, clouds → ⟨| all → 20 |⟩,
  sys → ⟨| message → 0.0076, sunset → 1435987058, id → 840, country → US, type → 1,
  sunrise → 1435938823 |⟩, id → 5856195, name → Honolulu, cod → 200 |⟩
```

Temp in F:

```
d[["main", "temp"]]
```

68.27

## Air pressure

Max air pressure allowed by FAA (converted to hPa):

```
faaAirPressureLimitInhPa =
N[QuantityMagnitude[UnitConvert[Quantity[31, "inHG"], "hPa"]]]
```

1049.78

## Daytime/nighttime

Current (unix) time UTC:

```
TimeObject[FromUnixTime[d[["dt"]]]]
```

 21:41:53 GMT-4.

Sunrise and sunset:

```
{TimeObject[FromUnixTime[d[["sys", "sunrise"]]]],
 TimeObject[FromUnixTime[d[["sys", "sunset"]]]]}
```

{ 11:53:43 GMT-4.,  01:17:38 GMT-4.}

# Geographic information and mapping

## Plotting no drone zones (obsolete)

```

redZonesQ[vehiclePosition_, vehicleRange_] :=
  Block[{disk = GeoDisk[vehiclePosition, vehicleRange],
         cities, airports},

    (* Find all cities within the vehicle's radius *)
    (* cities = GeoEntities[disk,"City"]; *)
    cities = nearbyCitiesThatExcludeDrones[disk];

    (* Find all airports within the vehicle's radius *)
    airports = nearbyAirports[disk];

    (* Get list of all no-fly zones *)
    Return[Length[Union[cities, airports]] > 0];
  ]

redZonesQ[GeoPosition[{42.36353, -72.50729}], Quantity[7, "Miles"] ]
Union::heads : Heads nearbyAirports and List at positions 2 and 1 are expected to be the same. >>
True

redZonesQ[GeoPosition[{39.11, -77.83}], Quantity[1, "Miles"] ]
False

Clear[redZonesQ]

```

Generate a map indicating (in red) all no-fly zones that are within the drone's range:

```

generateMap[vehiclePosition_, vehicleRange_] :=
  Block[{disk = GeoDisk[vehiclePosition, vehicleRange],
         cities, cityGraphics, airports, airportGraphics, allZones},

    (* Find all cities within the vehicle's radius *)
    (* cities = GeoEntities[disk,"City"]; *)
    cities = nearbyCitiesThatExcludeDrones[disk];
    cityGraphics = Map[{EdgeForm[Black], FaceForm[Red], Polygon[#]} &, cities];

    (* Find all airports within the vehicle's radius *)
    airports = nearbyAirports[disk];
    airportGraphics =
      Map[{EdgeForm[Black], FaceForm[Red], GeoDisk[#, Quantity[3, "Miles"]]} &,
          airports];

    (* Get list of all no-fly zones *)
    allZones = Union[cityGraphics, airportGraphics, {GeoMarker[vehiclePosition]},
                     {EdgeForm[Black], FaceForm[Gray], disk}
    ];

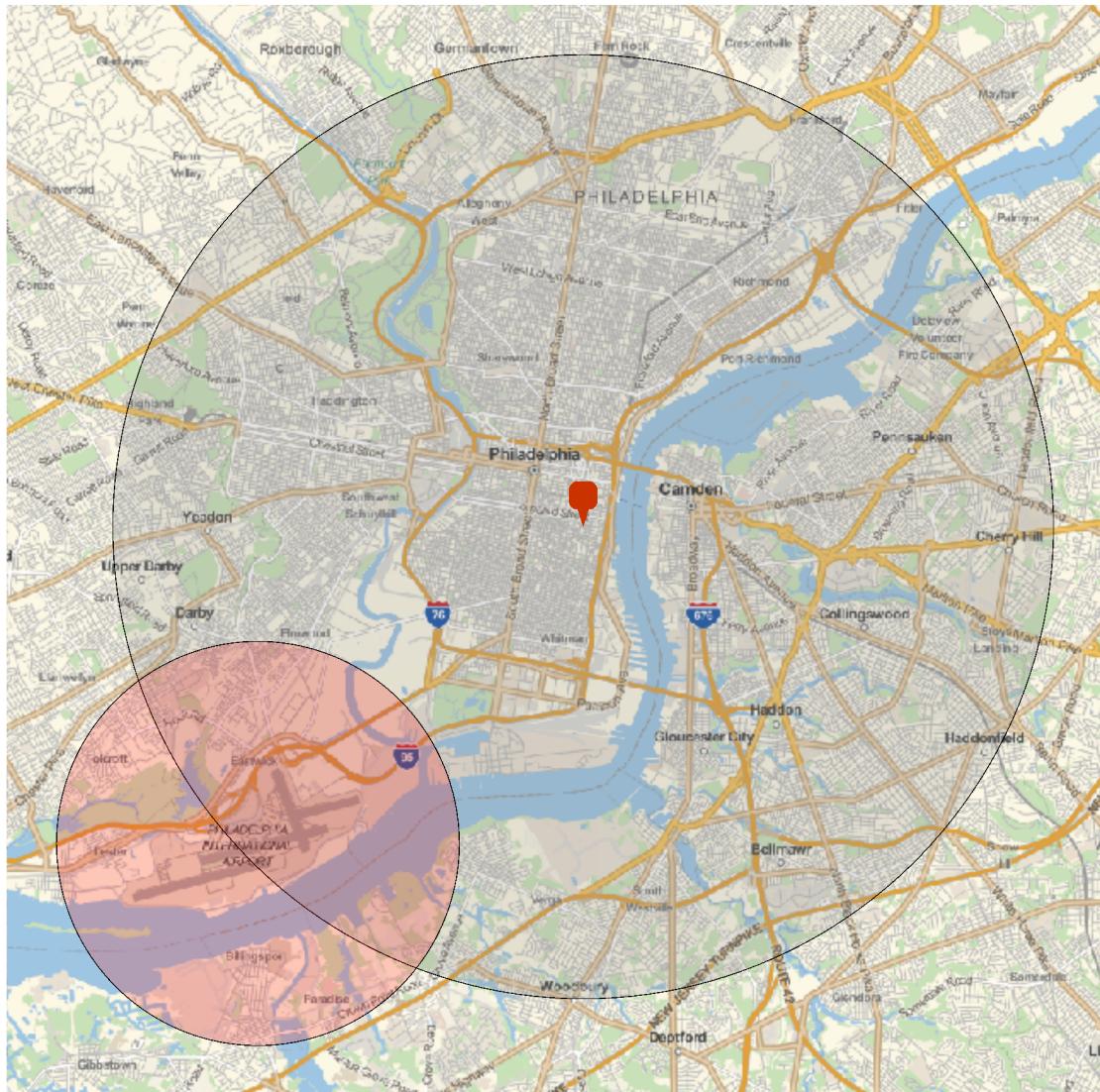
    (* Generate the map *)
    Return[
      GeoGraphics[{allZones}, ImageSize -> Large]
    ];
  ]

```

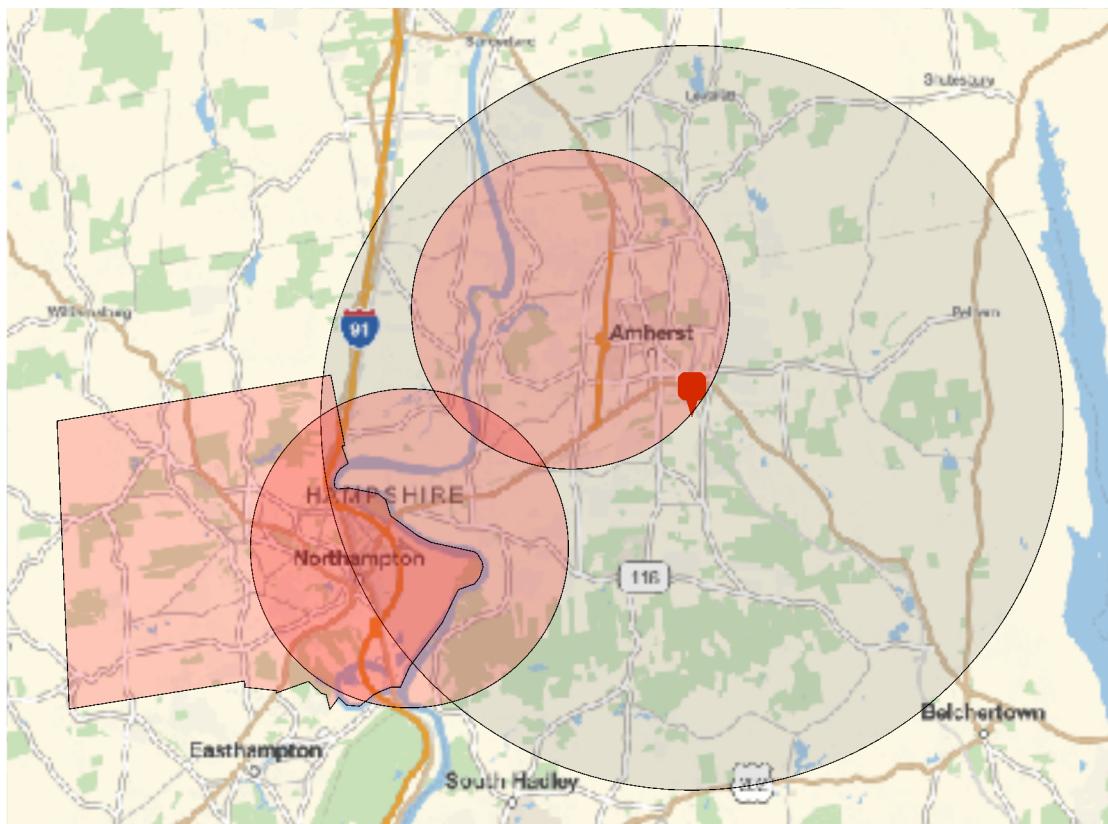
**Here**

```
GeoPosition[{39.94, -75.15}]
```

```
generateMap[Here, Quantity[7, "Miles"] ]
```

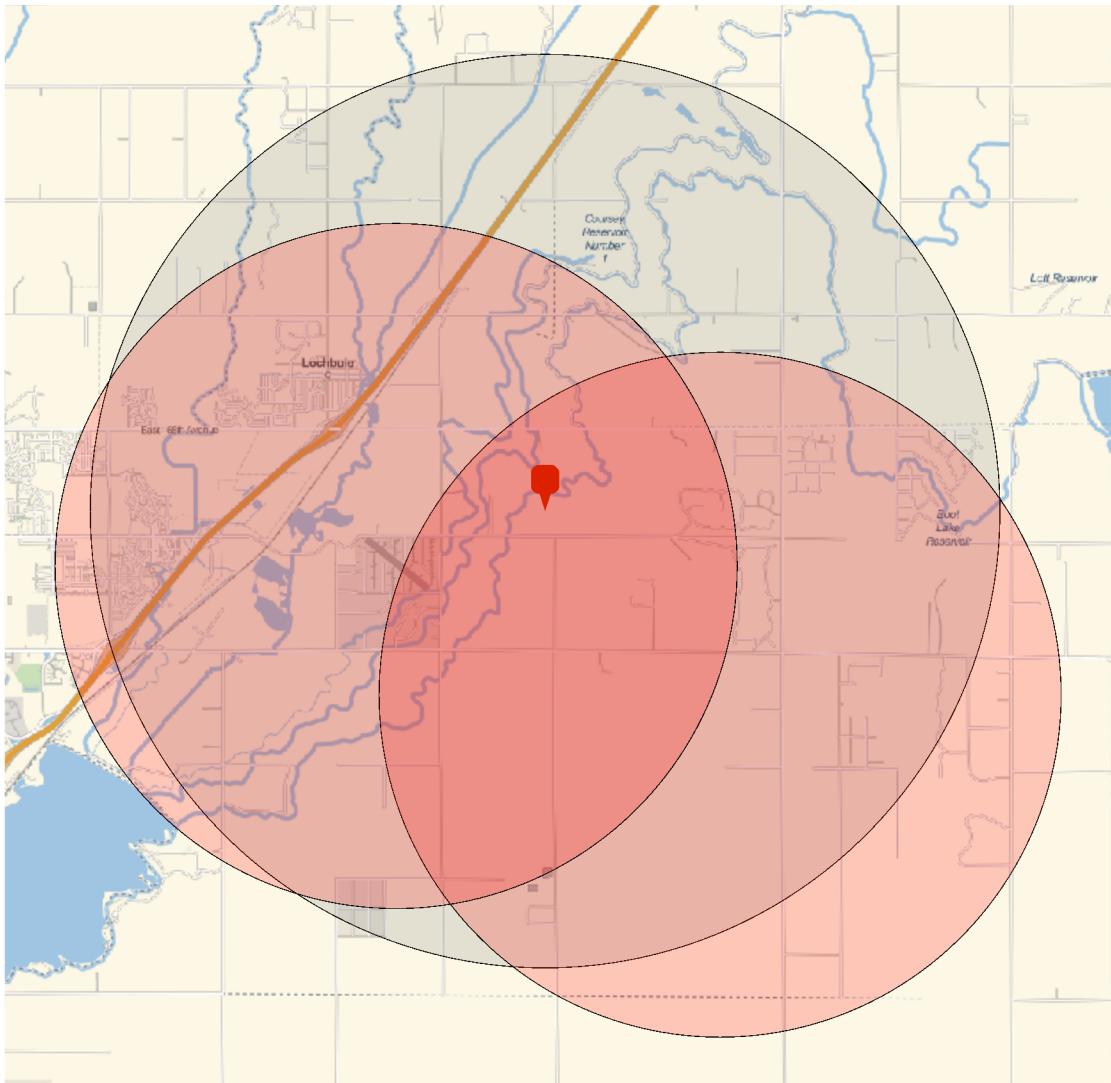


```
generateMap[GeoPosition[{42.36353, -72.50729}], Quantity[7, "Miles"] ]
```



```
GeoPosition[{39.86, 104.67}]
```

```
generateMap[GeoPosition[{39.99, -104.68}], Quantity[4, "Miles"] ] (* Denver *)
```



## Identifying airports (obsolete)

Returns the geoposition of all airports (excluding heliports) in a given geographic radius:

```
nearbyAirports[geoDisk_] := nearbyAirports[geoDisk] = Block[
  localAirportData = Map[AirportData[#, {"Name", "Coordinates"}] &,
    GeoEntities[geoDisk, "Airport"]]],
  Map[#[[2]] &,
    Select[
      localAirportData, ! StringContainsQ[#[[1]], "Heliport" | "Helistop"] &
    ]]
```

```
nearbyAirports[GeoDisk[Here, Quantity[10, "Miles"] ]]
{GeoPosition[{39.859, -75.2996}], GeoPosition[{39.8719, -75.2411}]}
```

## Identifying municipal no-drone zones

Returns a list of cities banning drones within the flight radius - OBSOLETE

```
nearbyCitiesThatExcludeDrones[geoDisk_] :=
nearbyCitiesThatExcludeDrones[geoDisk] =
Intersection[GeoEntities[geoDisk, "City"], citiesThatExcludeDrones]
```

In-memory database of cities that ban all drones:

```
citiesThatExcludeDrones = {
Entity["City", {"Northampton", "Massachusetts", "UnitedStates"}],
Entity["City", {"SaintBonifacius", "Minnesota", "UnitedStates"}],
Entity["City", {"Charlottesville", "Virginia", "UnitedStates"}]
(* Only law enforcement prohibited *)
}
```

{Northampton, Saint Bonifacius, Charlottesville}

Given a place (as a Wolfram entity), get the geographic coordinates defining its polygon, in the format `{{"lat" → 39.9, "long" → -75.1}, ... }`.

```
polygonCoordinatesForPlace[place_] := polygonCoordinatesForPlace[place] =
Map[{"lat" → #[[1]], "long" → #[[2]]} &,
place["Polygon"] /. Polygon[GeoPosition[{stuff_}]] → stuff
]
```

```
polygonCoordinatesForPlace[ Northampton (city) ]
```

```
{ {lat → 42.3168, long → -72.5934},
  {lat → 42.3078, long → -72.5992}, {lat → 42.2958, long → -72.6113},
  {lat → 42.2902, long → -72.6147}, {lat → 42.2902, long → -72.6182},
  {lat → 42.289, long → -72.6222}, {lat → 42.2881, long → -72.6245},
  {lat → 42.285, long → -72.6298}, {lat → 42.2853, long → -72.6347},
  {lat → 42.2874, long → -72.6374}, {lat → 42.2841, long → -72.6407},
  {lat → 42.2882, long → -72.6421}, {lat → 42.2885, long → -72.6438},
  {lat → 42.2918, long → -72.6487}, {lat → 42.2886, long → -72.6605},
  {lat → 42.2893, long → -72.6602}, {lat → 42.2902, long → -72.672},
  {lat → 42.2919, long → -72.6721}, {lat → 42.2842, long → -72.7367},
  {lat → 42.3627, long → -72.7412}, {lat → 42.3752, long → -72.6403},
  {lat → 42.3561, long → -72.6352}, {lat → 42.3559, long → -72.6363},
  {lat → 42.3512, long → -72.634}, {lat → 42.3498, long → -72.6368},
  {lat → 42.3484, long → -72.6382}, {lat → 42.3463, long → -72.6391},
  {lat → 42.3447, long → -72.6389}, {lat → 42.3442, long → -72.638},
  {lat → 42.3445, long → -72.6337}, {lat → 42.343, long → -72.623},
  {lat → 42.3413, long → -72.6201}, {lat → 42.3369, long → -72.6183},
  {lat → 42.3314, long → -72.6082}, {lat → 42.3297, long → -72.6034},
  {lat → 42.3271, long → -72.5875}, {lat → 42.3265, long → -72.5863},
  {lat → 42.3245, long → -72.5847}, {lat → 42.323, long → -72.5842},
  {lat → 42.3201, long → -72.5855}, {lat → 42.3184, long → -72.5874}}
```

API data for all municipal no-drone zones:

```
municipalNoFlyAPIOutput :=  
Map[<|"region" → polygonCoordinatesForPlace[#], "label" → "Municipal"|> &,  
citiesThatExcludeDrones]
```

```
municipalNoFlyAPIOutput  
{ <| region → { {lat → 42.3168, long → -72.5934}, {lat → 42.3078, long → -72.5992},
  {lat → 42.2958, long → -72.6113}, {lat → 42.2902, long → -72.6147},
  {lat → 42.2902, long → -72.6182}, {lat → 42.289, long → -72.6222},
  {lat → 42.2881, long → -72.6245}, {lat → 42.285, long → -72.6298},
  {lat → 42.2853, long → -72.6347}, {lat → 42.2874, long → -72.6374},
  {lat → 42.2841, long → -72.6407}, {lat → 42.2882, long → -72.6421},
  {lat → 42.2885, long → -72.6438}, {lat → 42.2918, long → -72.6487},
  {lat → 42.2886, long → -72.6605}, {lat → 42.2893, long → -72.6602},
  {lat → 42.2902, long → -72.672}, {lat → 42.2919, long → -72.6721},
  {lat → 42.2842, long → -72.7367}, {lat → 42.3627, long → -72.7412},
  {lat → 42.3752, long → -72.6403}, {lat → 42.3561, long → -72.6352},
  {lat → 42.3559, long → -72.6363}, {lat → 42.3512, long → -72.634},
  {lat → 42.3498, long → -72.6368}, {lat → 42.3484, long → -72.6382},
```

```

{lat → 42.3463, long → -72.6391}, {lat → 42.3447, long → -72.6389},
{lat → 42.3442, long → -72.638}, {lat → 42.3445, long → -72.6337},
{lat → 42.343, long → -72.623}, {lat → 42.3413, long → -72.6201},
{lat → 42.3369, long → -72.6183}, {lat → 42.3314, long → -72.6082},
{lat → 42.3297, long → -72.6034}, {lat → 42.3271, long → -72.5875},
{lat → 42.3265, long → -72.5863}, {lat → 42.3245, long → -72.5847},
{lat → 42.323, long → -72.5842}, {lat → 42.3201, long → -72.5855},
{lat → 42.3184, long → -72.5874}}, label → Municipal|),
<| region → {{lat → 44.8966, long → -93.753}, {lat → 44.8983, long → -93.7531},
{lat → 44.8983, long → -93.7577}, {lat → 44.9047, long → -93.7577},
{lat → 44.905, long → -93.7586}, {lat → 44.9055, long → -93.7586},
{lat → 44.9055, long → -93.7575}, {lat → 44.9128, long → -93.7579},
{lat → 44.9131, long → -93.7371}, {lat → 44.8978, long → -93.7371},
{lat → 44.8977, long → -93.7457}, {lat → 44.901, long → -93.7421},
{lat → 44.901, long → -93.7413}, {lat → 44.902, long → -93.7415},
{lat → 44.9018, long → -93.7462}, {lat → 44.9011, long → -93.746},
{lat → 44.901, long → -93.7438}, {lat → 44.8979, long → -93.7466},
{lat → 44.894, long → -93.7528}}, label → Municipal|),
<| region → {{lat → 38.0487, long → -78.4707}, {lat → 38.047, long → -78.4699},
{lat → 38.0458, long → -78.4661},
{lat → 38.0481, long → -78.4653}, {lat → 38.0495, long → -78.464},
{lat → 38.0495, long → -78.4573}, {lat → 38.0489, long → -78.457},
{lat → 38.0503, long → -78.4572}, {lat → 38.0509, long → -78.4587},
{lat → 38.0556, long → -78.4566}, {lat → 38.0534, long → -78.4536},
{lat → 38.0547, long → -78.4547}, {lat → 38.0587, long → -78.4491},
{lat → 38.0572, long → -78.4478}, {lat → 38.0511, long → -78.4472},
{lat → 38.0494, long → -78.4463}, {lat → 38.0461, long → -78.4473},
{lat → 38.0452, long → -78.4491}, {lat → 38.0458, long → -78.4536},
{lat → 38.0454, long → -78.4546}, {lat → 38.0439, long → -78.4548},
{lat → 38.0412, long → -78.454}, {lat → 38.0387, long → -78.4544},
{lat → 38.0363, long → -78.4562}, {lat → 38.0332, long → -78.4599},
{lat → 38.029, long → -78.4611}, {lat → 38.0278, long → -78.4608},
{lat → 38.0274, long → -78.46}, {lat → 38.0276, long → -78.4587},
{lat → 38.0291, long → -78.4572}, {lat → 38.0299, long → -78.4555},
{lat → 38.0296, long → -78.4541}, {lat → 38.0285, long → -78.4528},
{lat → 38.0277, long → -78.4524}, {lat → 38.0265, long → -78.4526},
{lat → 38.0221, long → -78.4547}, {lat → 38.0211, long → -78.4567},
{lat → 38.0224, long → -78.4593}, {lat → 38.0225, long → -78.4621},
{lat → 38.0206, long → -78.4635}, {lat → 38.0184, long → -78.4669},
{lat → 38.0147, long → -78.4692}, {lat → 38.014, long → -78.4684},
{lat → 38.0114, long → -78.4697}, {lat → 38.013, long → -78.4746},
{lat → 38.0135, long → -78.4751}, {lat → 38.0142, long → -78.4747},
{lat → 38.0147, long → -78.4756}, {lat → 38.0133, long → -78.4803},

```

```

{lat → 38.0118, long → -78.482}, {lat → 38.012, long → -78.4832},
{lat → 38.0145, long → -78.4833}, {lat → 38.0154, long → -78.4841},
{lat → 38.0173, long → -78.4906}, {lat → 38.015, long → -78.4935},
{lat → 38.0128, long → -78.5}, {lat → 38.0102, long → -78.5033},
{lat → 38.0101, long → -78.5046}, {lat → 38.0114, long → -78.5086},
{lat → 38.0098, long → -78.5103}, {lat → 38.0096, long → -78.5129},
{lat → 38.0109, long → -78.5193}, {lat → 38.0132, long → -78.5187},
{lat → 38.017, long → -78.5194}, {lat → 38.0177, long → -78.5205},
{lat → 38.0184, long → -78.5204}, {lat → 38.0189, long → -78.5209},
{lat → 38.0184, long → -78.5221}, {lat → 38.0195, long → -78.5234},
{lat → 38.0223, long → -78.5236}, {lat → 38.0263, long → -78.5231},
{lat → 38.026, long → -78.52}, {lat → 38.0298, long → -78.5202},
{lat → 38.0298, long → -78.5183}, {lat → 38.0286, long → -78.5183},
{lat → 38.0286, long → -78.5148}, {lat → 38.0302, long → -78.5119},
{lat → 38.0322, long → -78.5096}, {lat → 38.0321, long → -78.502},
{lat → 38.0333, long → -78.4978}, {lat → 38.0343, long → -78.5001},
{lat → 38.0361, long → -78.5016}, {lat → 38.0367, long → -78.5035},
{lat → 38.0404, long → -78.5012}, {lat → 38.0429, long → -78.5043},
{lat → 38.0421, long → -78.5053}, {lat → 38.0413, long → -78.5058},
{lat → 38.041, long → -78.5052}, {lat → 38.0394, long → -78.5061},
{lat → 38.0393, long → -78.5068}, {lat → 38.0358, long → -78.5081},
{lat → 38.0371, long → -78.5133}, {lat → 38.0367, long → -78.5155},
{lat → 38.0399, long → -78.5169}, {lat → 38.0403, long → -78.5164},
{lat → 38.0412, long → -78.5174}, {lat → 38.0441, long → -78.5157},
{lat → 38.0442, long → -78.514}, {lat → 38.0448, long → -78.5146},
{lat → 38.0438, long → -78.5114}, {lat → 38.0455, long → -78.5059},
{lat → 38.0473, long → -78.5048}, {lat → 38.0513, long → -78.5103},
{lat → 38.0563, long → -78.5076}, {lat → 38.0577, long → -78.5013},
{lat → 38.0584, long → -78.5009}, {lat → 38.0605, long → -78.5023},
{lat → 38.0653, long → -78.495}, {lat → 38.0636, long → -78.4926},
{lat → 38.0619, long → -78.4916}, {lat → 38.069, long → -78.4852},
{lat → 38.0667, long → -78.4811}, {lat → 38.0656, long → -78.4809},
{lat → 38.0656, long → -78.4789}, {lat → 38.0647, long → -78.4772},
{lat → 38.0667, long → -78.476}, {lat → 38.0677, long → -78.4746},
{lat → 38.0685, long → -78.4752}, {lat → 38.0701, long → -78.473},
{lat → 38.0706, long → -78.4714}, {lat → 38.0697, long → -78.4707},
{lat → 38.0675, long → -78.4723}, {lat → 38.0681, long → -78.4685},
{lat → 38.0676, long → -78.4684}, {lat → 38.0678, long → -78.4669},
{lat → 38.0646, long → -78.4643}, {lat → 38.0558, long → -78.4696},
{lat → 38.0526, long → -78.4722}, {lat → 38.0504, long → -78.4714},
{lat → 38.049, long → -78.4723}}, label → Municipal|>}

```

```
municipalNoFlyGeoRegions = Map[#["Polygon"] &, citiesThatExcludeDrones];
```

## In-memory database of U.S. airports

### Convert from FAA lat/long format to Wolfram format

Logan airport ( $42.3631^{\circ}$  N,  $71.0064^{\circ}$  W) is represented as (152506.6741N, 255623.1257W) in the FAA airport data table. The FromDMS function doesn't convert this properly, so a new conversion function is necessary.

```
{FromDMS["42-21-46.6741N"], FromDMS["071-00-23.1257W"]}
{41.637, -70.9936}
```

Note: This needs to be evaluated before the data is obtained and cleaned (below). If not, the airport query will take **30x longer**.

```
fromFAALatLong[latLong_String] := FromDMS[
  ToExpression[StringSplit[
    StringReplace[latLong, {"-" → ",", "N" → "", "W" → "", "S" → ""}], ","]]]
```

```
fromFAALatLong["42-21-46.6741N"]
```

```
42.363
```

```
fromFAALatLong["071-00-23.1257W"]
```

```
71.0064
```

### Obtain and clean the data

Import an abbreviated version of the FAA airport database, available at [http://www.faa.gov/airports/airport\\_safety/airportdata\\_5010/](http://www.faa.gov/airports/airport_safety/airportdata_5010/) (last updated 7/1/15).

```
data = Import["/Users/mpoulshock/Dropbox/Projects/Drones/Airport
    data/FAA_Airport_Database_2.csv", "CSV"]
```

```
{ {AIRPORT, 33-38-12.1186N, 084-25-40.3104W} ,
  {AIRPORT, 33-56-33.0011N, 118-24-29.0651W} ,
  {AIRPORT, 41-58-45.6000N, 087-54-26.6000W} ,
  {AIRPORT, 32-53-50.0336N, 097-02-15.7018W} ,
  {AIRPORT, 39-51-42.0000N, 104-40-23.4000W} ,
  {AIRPORT, 44-52-55.1000N, 093-13-18.4000W} ,
  {AIRPORT, 42-03-52.3000N, 104-09-09.7000W} ,
  {AIRPORT, 44-05-25.9320N, 104-38-26.8590W} ,
  {AIRPORT, 42-18-24.8900N, 104-56-11.8940W} ,
  {AIRPORT, 41-55-58.8950N, 105-05-05.9120W} ,
  {HELIPORT, 43-57-55.8440N, 107-57-01.3360W} }
```

[large output](#)

[show less](#)

[show more](#)

[show all](#)

[set size limit...](#)

Convert latitude and longitude to numeric format and define an in-memory data structure to hold the airport data:

```
usAirportDatabase =
Map[{#[[1]], fromFAALatLong#[[2]], fromFAALatLong#[[3]] * -1} &, data]
```

```
{ {AIRPORT, 33.6367, -84.4279}, {AIRPORT, 33.9425, -118.408} ,
  {AIRPORT, 41.9793, -87.9074}, {AIRPORT, 32.8972, -97.0377} ,
  {AIRPORT, 39.8617, -104.673}, {AIRPORT, 42.0645, -104.153} ,
  {AIRPORT, 44.0905, -104.641}, {AIRPORT, 42.3069, -104.937} ,
  {AIRPORT, 41.933, -105.085}, {HELIPORT, 43.9655, -107.95} }
```

[large output](#)

[show less](#)

[show more](#)

[show all](#)

[set size limit...](#)

Convert miles to degrees of longitude and latitude. Assuming 1 degree = 69 mi, even though this is not completely correct, particularly for longitude.

```
milesToDegrees[miles_] := N[QuantityMagnitude[miles] / 69]
```

```
milesToDegrees[1]
```

0.0144928

## Query the airport database

Query the in-memory airport database to get all airports (of a particular type) within N miles of a location. Possible values for airportType are AIRPORT, HELIPORT, BALLOONPORT, GLIDERPORT, SEAPLANE BASE, and ULTRALIGHT.

```

nearbyUSAirports[airportType_String, position_GeoPosition, rangeInMiles_] :=
  Block[{lat, long, offset, a, b, c, d},
    lat = QuantityMagnitude[Latitude[position]];
    long = QuantityMagnitude[Longitude[position]];
    offset = milesToDegrees[rangeInMiles];
    a = lat - offset;
    b = lat + offset;
    c = long - offset;
    d = long + offset;

    Select[usAirportDatabase,
      #[[1]] == airportType && a < #[[2]] < b && c < #[[3]] < d &
    ]
  ]

nearbyUSAirports["HELIPORT", Here, 10] // AbsoluteTiming
{0.050068, {{HELIPORT, 39.9404, -75.1066}, {HELIPORT, 39.9411, -75.1164},
  {HELIPORT, 39.9523, -75.0841}, {HELIPORT, 39.9464, -75.1039},
  {HELIPORT, 39.9501, -75.0496}, {HELIPORT, 39.9902, -75.0448},
  {HELIPORT, 40.0168, -75.2413}, {HELIPORT, 40.0096, -75.2138},
  {HELIPORT, 40.0368, -75.1435}, {HELIPORT, 39.9198, -75.1966},
  {HELIPORT, 39.9482, -75.1941}, {HELIPORT, 39.9584, -75.1913},
  {HELIPORT, 39.9901, -75.1305}, {HELIPORT, 39.9571, -75.1627},
  {HELIPORT, 39.9459, -75.1941}, {HELIPORT, 39.9881, -75.2596},
  {HELIPORT, 39.9307, -75.2441}, {HELIPORT, 39.9126, -75.1399},
  {HELIPORT, 40.0357, -75.0163}, {HELIPORT, 39.9373, -75.1413},
  {HELIPORT, 39.9548, -75.1782}, {HELIPORT, 39.9534, -75.1496},
  {HELIPORT, 39.8793, -75.2288}, {HELIPORT, 40.0065, -75.1257},
  {HELIPORT, 39.9965, -75.2182}, {HELIPORT, 40.0057, -75.1501},
  {HELIPORT, 39.949, -75.1585}, {HELIPORT, 40.0073, -75.2155}}}

```

## Convert data to output format for API

```

airportDataAPIOutput[airportType_, position_,
  plotRadiusInMiles_, airportRadiusInMiles_] := Block[{meters},
  meters = Round[N[QuantityMagnitude[
    UnitConvert[Quantity[airportRadiusInMiles, "Miles"], "Meters"]]], 1];

  Map[<|"lat" → #[[2]], "long" → #[[3]],
    "radiusInMeters" → meters, "label" → airportType|> &,
  nearbyUSAAirports[airportType, position, plotRadiusInMiles]]
]

airportDataAPIOutput["AIRPORT", Here, 15, 2] // AbsoluteTiming
{0.124757, {<| lat → 39.8722, long → -75.2409, radiusInMeters → 3219 |>,
  <| lat → 39.7784, long → -74.9478, radiusInMeters → 3219 |>,
  <| lat → 39.7893, long → -75.2291, radiusInMeters → 3219 |>,
  <| lat → 40.0819, long → -75.0106, radiusInMeters → 3219 |>,
  <| lat → 40.1376, long → -75.2651, radiusInMeters → 3219 |> } }

```

## In-memory database of FAA Temporary Flight Restrictions (TFRs)

### Database

```

faaTFRDatabase = {
  (* FDC 5/1329 *)
  <|"XML" → "http://tfr.faa.gov/save_pages/detail_5_1329.xml",
    "Shape" → "Circle", "EffectiveStartUTC" → Thu 9 Jul 2015 15:30:00 GMT,
    "EffectiveEndUTC" → Thu 9 Jul 2015 16:30:00 GMT,
    "MinAltitudeMSL" → 0 ft, "MaxAltitudeMSL" → 4500 ft ,
    "Center" → GeoPosition[{47.46777778°, -92.91805556°}],
    "CenterLat" → 47.46777778, "CenterLong" → -92.91805556,
    "Radius" → 3 nmi , "RadiusInMeters" → 5556 |>,

  (* FDC 5/1326 *)
  <|"XML" → "http://tfr.faa.gov/save_pages/detail_5_1326.xml",
    "Shape" → "Circle", "EffectiveStartUTC" → Tue 7 Jul 2015 18:30:00 GMT ,

```

```

"EffectiveEndUTC" → Thu 1 Oct 2015 18:30:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 6000 ft ,
"Center" → GeoPosition[{47.76583333` , -117.5075`}], "CenterLat" → 47.7658333 ,
"CenterLong" → -117.5075, "Radius" → 3.5` nmi , "RadiusInMeters" → 6482 |〉 ,
```

(\* FDC 5/1274 \*)

```

<|"XML" → "http://tfr.faa.gov/save_pages/detail_5_1274.xml",
"Shape" → "Polygon", "EffectiveStartUTC" → Tue 7 Jul 2015 17:00:00 GMT ,
"EffectiveEndUTC" → Sat 1 Aug 2015 00:00:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 12 000 ft ,
"Center" → GeoPosition[{34.22722222` , -116.95`}],
"CenterLat" → 34.22722222, "CenterLong" → -116.95,
"BoundedRegion" → {GeoPosition[{34.22722222` , -116.95`}],
GeoPosition[{34.21222222` , -116.61666667`}],
GeoPosition[{34.05` , -116.66666667`}], GeoPosition[{34.0975` , -116.95`}],
GeoPosition[{34.22722222` , -116.95`}]} |〉 ,
```

(\* FDC 5/1201 \*)

```

<|"XML" → "http://tfr.faa.gov/save_pages/detail_5_1201.xml",
"Shape" → "Circle", "EffectiveStartUTC" → Wed 8 Jul 2015 12:30:00 GMT ,
"EffectiveEndUTC" → Wed 8 Jul 2015 15:30:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 15 000 ft ,
"Center" → GeoPosition[{30.32666667` , -87.13972222`}],
"CenterLat" → 30.32666667, "CenterLong" → -87.13972222,
"Radius" → 5 nmi , "RadiusInMeters" → 9260 |〉 ,
```

```

<|"XML" → "http://tfr.faa.gov/save_pages/detail_5_1201.xml",
"Shape" → "Circle", "EffectiveStartUTC" → Thu 9 Jul 2015 18:00:00 GMT ,
"EffectiveEndUTC" → Thu 9 Jul 2015 20:30:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 15 000 ft ,
"Center" → GeoPosition[{30.32666667` , -87.13972222`}],
"CenterLat" → 30.32666667, "CenterLong" → -87.13972222,
"Radius" → 5 nmi , "RadiusInMeters" → 9260 |〉 ,
```

(\* FDC 5/1200 \*)

```

<|"XML" → "http://tfr.faa.gov/save_pages/detail_5_1200.xml",
```

```

"Shape" → "Circle", "EffectiveStartUTC" →  Fri 10 Jul 2015 16:45:00 GMT ,
"EffectiveEndUTC" →  Fri 10 Jul 2015 20:15:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 15 000 ft ,
"Center" → GeoPosition[{30.32666667` , -87.13972222`}],
"CenterLat" → 30.32666667, "CenterLong" → -87.13972222,
"Radius" → 5 nmi , "RadiusInMeters" → 9260 |> ,
```

```

<|"XML" → "http://tfr.faa.gov/save_pages/detail_5_1200.xml",
"Shape" → "Circle", "EffectiveStartUTC" →  Sat 11 Jul 2015 16:45:00 GMT ,
"EffectiveEndUTC" →  Sat 11 Jul 2015 20:15:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 15 000 ft ,
"Center" → GeoPosition[{30.32666667` , -87.13972222`}],
"CenterLat" → 30.32666667, "CenterLong" → -87.13972222,
"Radius" → 5 nmi , "RadiusInMeters" → 9260 |> ,
```

```

<|"XML" → "http://tfr.faa.gov/save_pages/detail_5_1200.xml",
"Shape" → "Circle", "EffectiveStartUTC" →  Sun 12 Jul 2015 16:45:00 GMT ,
"EffectiveEndUTC" →  Sun 12 Jul 2015 20:15:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 15 000 ft ,
"Center" → GeoPosition[{30.32666667` , -87.13972222`}],
"CenterLat" → 30.32666667, "CenterLong" → -87.13972222,
"Radius" → 5 nmi , "RadiusInMeters" → 9260 |> ,
```

```

(* FDC 5/0192 *)
<|"URL" → "http://tfr.faa.gov/save_pages/detail_5_0192.html",
"XML" → "http://tfr.faa.gov/save_pages/detail_5_0192.xml",
"Shape" → "Circle", "EffectiveStartUTC" →  Sat 20 Jun 2015 22:00:00 GMT ,
"EffectiveEndUTC" →  Fri 31 Jul 2015 22:00:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 1000 ft ,
"Center" → GeoPosition[{34.46194444` , -120.06388889`}],
"CenterLat" → 34.46194444` , "CenterLong" → -120.06388889` ,
"Radius" → 5 nmi , "RadiusInMeters" → 9260 |> ,
```

```

(* FDC 5/7942 *)
<|"URL" → "http://tfr.faa.gov/save_pages/detail_5_7942.html",
"XML" → "http://tfr.faa.gov/save_pages/detail_5_7942.xml",
```

```

"Shape" → "Circle", "EffectiveStartUTC" → Sat 13 Jun 2015 11:00:00 GMT ,
"EffectiveEndUTC" → Fri 11 Sep 2015 21:00:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 2499 ft ,
"Center" → GeoPosition[{42.93861111°, -71.64083333°}],
"CenterLat" → 42.93861111°, "CenterLong" → -71.64083333°,
"Radius" → 1 nmi , "RadiusInMeters" → 1852 |⟩ ,
(* FDC 5/0988 *)
⟨|"URL" → "http://tfr.faa.gov/save_pages/detail_5_0988.html",
"XML" → "http://tfr.faa.gov/save_pages/detail_5_0988.xml",
"Shape" → "Circle", "EffectiveStartUTC" → Tue 2 Jun 2015 18:00:00 GMT ,
"EffectiveEndUTC" → Sun 2 Aug 2015 22:00:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 2000 ft ,
"Center" → GeoPosition[{33.43111111°, -94.24555556°}],
"CenterLat" → 33.43111111°, "CenterLong" → -94.24555556°,
"Radius" → Quantity[2, "NM"], "RadiusInMeters" → 1852 |⟩ ,
(* FDC 5/4058 Andersen AFB Guam *)
⟨|"URL" → "http://tfr.faa.gov/save_pages/detail_5_4058.html",
"XML" → "http://tfr.faa.gov/save_pages/detail_5_4058.xml",
"Shape" → "Polygon", "EffectiveStartUTC" → Thu 30 Apr 2015 03:00:00 GMT ,
"EffectiveEndUTC" → Thu 30 Jul 2015 03:00:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 220 ft ,
"Center" → GeoPosition[{13.64833333°, 144.79916667°}],
"CenterLat" → 13.64833333°, "CenterLong" → 144.79916667°,
"BoundedRegion" → {GeoPosition[{13.64833333°, 144.79916667°}],
GeoPosition[{13.69305556°, 144.88916667°}], GeoPosition[
{13.64°, 144.91583333°}], GeoPosition[{13.59583333°, 144.82611111°}],
GeoPosition[{13.64833333°, 144.79916667°}]} |⟩ ,
(* FDC 5/1128 Vieques, Puerto Rico *)
⟨|"URL" → "http://tfr.faa.gov/save_pages/detail_5_1128.html",
"XML" → "http://tfr.faa.gov/save_pages/detail_5_1128.xml",
"Shape" → "Circle", "EffectiveStartUTC" → Mon 19 Jan 2015 12:00:00 GMT ,
"EffectiveEndUTC" → Thu 31 Dec 2015 22:00:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 5000 ft ,
"Center" → GeoPosition[{18.13833333°, -65.33666667°}],

```

```

"CenterLat" → 18.13833333` , "CenterLong" → -65.33666667` ,
"Radius" → Quantity[4.1, "NM"] , "RadiusInMeters" → 7593|⟩ ,


(* FDC 4/3635 Disneyland *)
⟨|"URL" → "http://tfr.faa.gov/save_pages/detail_4_3635.html",
"XML" → "http://tfr.faa.gov/save_pages/detail_4_3635.xml",
"Shape" → "Circle", "EffectiveStartUTC" → Mon 27 Oct 2014 15:00:00 GMT ,
"EffectiveEndUTC" → Fri 1 Jan 2100 12:00:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 3000 ft ,
"Center" → GeoPosition[{33.80138889`, -117.92138889`}],
"CenterLat" → 33.80138889` , "CenterLong" → -117.92138889` ,
"Radius" → Quantity[3, "NM"] , "RadiusInMeters" → 5556|⟩ ,


(* FDC 4/3634 Disney World *)
⟨|"URL" → "http://tfr.faa.gov/save_pages/detail_4_3634.html",
"XML" → "http://tfr.faa.gov/save_pages/detail_4_3634.xml",
"Shape" → "Circle", "EffectiveStartUTC" → Mon 27 Oct 2014 15:00:00 GMT ,
"EffectiveEndUTC" → Fri 1 Jan 2100 12:00:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 3000 ft ,
"Center" → GeoPosition[{28.4125`, -81.57222222`}],
"CenterLat" → 28.4125` , "CenterLong" → -81.57222222` ,
"Radius" → Quantity[3, "NM"] , "RadiusInMeters" → 5556|⟩ ,


(* FDC 4/9152 Leesburg Airport (JY0) *)
⟨|"URL" → "http://tfr.faa.gov/save_pages/detail_4_9152.html", "XML" → "",
"Shape" → "Polygon", "EffectiveStartUTC" → Sun 15 Jun 2014 04:01:00 GMT ,
"EffectiveEndUTC" → Wed 15 Jun 2016 03:59:00 GMT ,
"MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 15000 ft ,
"Center" → GeoPosition[{13.64833333`, 144.79916667`}],
"CenterLat" → 13.64833333` , "CenterLong" → 144.79916667` ,
"BoundedRegion" → {GeoPosition[{39.0275`, -77.64055555555555`}],
GeoPosition[{39.21166666666666`, -77.49166666666666`}],
GeoPosition[{39.0508333333334`, -77.47694444444444`}],
GeoPosition[{39.0275`, -77.55694444444444`}],
GeoPosition[{39.0275`, -77.64055555555555`}]}|⟩ ,


(* FDC 9/3124 *)

```

```

<|"URL" → "http://tfr.faa.gov/save_pages/detail_9_3124.html",
 "XML" → "http://tfr.faa.gov/save_pages/detail_9_3124.xml",
 "Shape" → "Circle", "EffectiveStartUTC" → Tue 27 Jan 2009 23:34:00 GMT ,
 "EffectiveEndUTC" → Fri 1 Jan 2100 12:00:00 GMT ,
 "MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 1500 ft ,
 "Center" → GeoPosition[{39.76805556`,-75.61777778`}],
 "CenterLat" → 39.76805556` , "CenterLong" → -75.61777778` ,
 "Radius" → Quantity[1, "NM"] , "RadiusInMeters" → 1852 |>,

(* FDC 0/8326 DC FRZ and ADIZ (see http://
   www.faa.gov/news/updates/adiz_frz/media/DC_ADIZ-FRZ_Advisory.pdf) *)
<|"URL" → "http://tfr.faa.gov/save_pages/detail_0_8326.html",
 "XML" → "http://tfr.faa.gov/save_pages/detail_0_8326.xml",
 "Shape" → "Circle", "EffectiveStartUTC" → Thu 18 Nov 2010 19:15:00 GMT ,
 "EffectiveEndUTC" → Fri 1 Jan 2100 12:00:00 GMT ,
 "MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 15000 ft ,
 "Center" → GeoPosition[{38.8522,-77.0378}], "CenterLat" → 38.8522,
 "CenterLong" → -77.0378, "Radius" → 30 nmi , "RadiusInMeters" → 55560 |>,

(* FDC 9/2934 *)
<|"URL" → "http://tfr.faa.gov/save_pages/detail_9_2934.html",
 "XML" → "http://tfr.faa.gov/save_pages/detail_9_2934.xml",
 "Shape" → "Circle", "EffectiveStartUTC" → Mon 26 Jan 2009 22:31:00 GMT ,
 "EffectiveEndUTC" → Sat 26 Jan 2109 22:31:00 GMT ,
 "MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 1500 ft ,
 "Center" → GeoPosition[{32.88916667`,-96.80972222`}],
 "CenterLat" → 32.88916667` , "CenterLong" → -96.80972222` ,
 "Radius" → Quantity[1, "NM"] , "RadiusInMeters" → 1852 |>,

(* FDC 8/9697 KILAUEA SUMMIT,Hawaii *)
<|"URL" → "http://tfr.faa.gov/save_pages/detail_8_9697.html",
 "XML" → "http://tfr.faa.gov/save_pages/detail_8_9697.xml",
 "Shape" → "Circle", "EffectiveStartUTC" → Tue 25 Mar 2008 21:31:00 GMT ,
 "EffectiveEndUTC" → Sun 25 Mar 2108 21:31:00 GMT ,
 "MinAltitudeMSL" → 0 ft , "MaxAltitudeMSL" → 4000 ft ,
 "Center" → GeoPosition[{19.40555556`,-155.29055556`}],

```

```

    "CenterLat" → 19.40555556` , "CenterLong" → -155.29055556` ,
    "Radius" → Quantity[3, "NM"] , "RadiusInMeters" → 5556` |
};

}

```

## Query

Search for TFRs that are currently in effect near a given position:

```

nearbyUSTFRs[position_GeoPosition, rangeInMiles_] :=
Block[{lat, long, offset, a, b, c, d},
  lat = QuantityMagnitude[Latitude[position]];
  long = QuantityMagnitude[Longitude[position]];
  offset = milesToDegrees[rangeInMiles];
  a = lat - offset;
  b = lat + offset;
  c = long - offset;
  d = long + offset;

Select[faaTFRDatabase, a < #[["CenterLat"]] < b && c < #[["CenterLong"]] < d
  && #[["EffectiveStartUTC"]] <= Now && #[["EffectiveEndUTC"]] >= Now &]
]

```

```

nearbyUSTFRs[Here, 900] // AbsoluteTiming
{0.037123, {({URL → http://tfr.faa.gov/save_pages/detail_5_7942.html,
XML → http://tfr.faa.gov/save_pages/detail_5_7942.xml,
Shape → Circle, EffectiveStartUTC → Sat 13 Jun 2015 11:00:00 GMT , ,
EffectiveEndUTC → Fri 11 Sep 2015 21:00:00 GMT , MinAltitudeMSL → 0 ft ,
MaxAltitudeMSL → 2499 ft , Center → GeoPosition[{42.9386, -71.6408}] ,
CenterLat → 42.9386, CenterLong → -71.6408, Radius → 1 nmi }|,
({URL → http://tfr.faa.gov/save_pages/detail_4_3634.html,
XML → http://tfr.faa.gov/save_pages/detail_4_3634.xml,
Shape → Circle, EffectiveStartUTC → Mon 27 Oct 2014 15:00:00 GMT , ,
EffectiveEndUTC → Fri 1 Jan 2100 12:00:00 GMT , MinAltitudeMSL → 0 ft ,
MaxAltitudeMSL → 3000 ft , Center → GeoPosition[{28.4125, -81.5722}] ,
CenterLat → 28.4125, CenterLong → -81.5722, Radius → 3 nmi }|,
({URL → http://tfr.faa.gov/save_pages/detail_9_3124.html,
XML → http://tfr.faa.gov/save_pages/detail_9_3124.xml,
Shape → Circle, EffectiveStartUTC → Tue 27 Jan 2009 23:34:00 GMT , ,
EffectiveEndUTC → Fri 1 Jan 2100 12:00:00 GMT , MinAltitudeMSL → 0 ft ,
MaxAltitudeMSL → 1500 ft , Center → GeoPosition[{39.7681, -75.6178}] ,
CenterLat → 39.7681, CenterLong → -75.6178, Radius → 1 nmi }|,
({URL → http://tfr.faa.gov/save_pages/detail_0_8326.html,
XML → http://tfr.faa.gov/save_pages/detail_0_8326.xml,
Shape → Circle, EffectiveStartUTC → Thu 18 Nov 2010 19:15:00 GMT , ,
EffectiveEndUTC → Fri 1 Jan 2100 12:00:00 GMT , MinAltitudeMSL → 0 ft ,
MaxAltitudeMSL → 15000 ft , Center → GeoPosition[{38.8522, -77.0378}] ,
CenterLat → 38.8522, CenterLong → -77.0378, Radius → 30 nmi }])}

```

```
Select[nearbyUSTFRs[Here, 5900], #["Shape"] == "Polygon" &]
{<| XML → http://tfr.faa.gov/save_pages/detail_5_1274.xml,
  Shape → Polygon, EffectiveStartUTC → Tue 7 Jul 2015 17:00:00 GMT,
  EffectiveEndUTC → Sat 1 Aug 2015 00:00:00 GMT,
  MinAltitudeMSL → 0 ft, MaxAltitudeMSL → 12 000 ft,
  Center → GeoPosition[{34.2272, -116.95}], CenterLat → 34.2272,
  CenterLong → -116.95, BoundedRegion → {GeoPosition[{34.2272, -116.95}],
  GeoPosition[{34.2122, -116.617}], GeoPosition[{34.05, -116.667}],
  GeoPosition[{34.0975, -116.95}], GeoPosition[{34.2272, -116.95}]}|>}
```

## Prepare data for API output

Given a list of TFRs in effect near the vehicle, identify ones with circular shapes and convert the data to the API output format ({{"lat" → 39.9, "long" → -75.1, "radius" → 4},...}).

```
circularTFRDataAPIOutput[tfrList_] := Block[{circles},
  circles = Select[tfrList, #["Shape"] == "Circle" &];

  Map[<|"lat" → #["CenterLat"], "long" → #["CenterLong"],
  "radiusInMeters" → #["RadiusInMeters"], "label" → "TFR"|> &, circles]
]
```

```
circularTFRDataAPIOutput[nearbyUSTFRs[Here, 9900]] // AbsoluteTiming
{0.097057, {<| lat → 47.7658, long → -117.508, radiusInMeters → 6482 |>,
  <| lat → 34.4619, long → -120.064, radiusInMeters → 9260 |>,
  <| lat → 42.9386, long → -71.6408, radiusInMeters → 1852 |>,
  <| lat → 33.4311, long → -94.2456, radiusInMeters → 3704 |>,
  <| lat → 18.1383, long → -65.3367, radiusInMeters → 7593 |>,
  <| lat → 33.8014, long → -117.921, radiusInMeters → 5556 |>,
  <| lat → 28.4125, long → -81.5722, radiusInMeters → 5556 |>,
  <| lat → 39.7681, long → -75.6178, radiusInMeters → 1852 |>,
  <| lat → 38.8522, long → -77.0378, radiusInMeters → 55560 |>,
  <| lat → 32.8892, long → -96.8097, radiusInMeters → 1852 |>,
  <| lat → 19.4056, long → -155.291, radiusInMeters → 5556 |>}}
```

```

nearbyUSTFRs[GeoPosition[{38.9072, -77.0369}], 35] (* Washington, DC *)
{⟨ URL → http://tfr.faa.gov/save_pages/detail_0_8326.html,
  XML → http://tfr.faa.gov/save_pages/detail_0_8326.xml,
  Shape → Circle, EffectiveStartUTC →  Thu 18 Nov 2010 19:15:00 GMT ,
  EffectiveEndUTC →  Fri 1 Jan 2100 12:00:00 GMT ,
  MinAltitudeMSL → 0 ft , MaxAltitudeMSL → 15000 ft ,
  Center → GeoPosition[{38.8522, -77.0378}] , CenterLat → 38.8522,
  CenterLong → -77.0378, Radius → 30 nmi , RadiusInMeters → 55560 ⟩}
circularTFRDataAPIOutput[nearbyUSTFRs[GeoPosition[{38.9072, -77.0369}], 35]]
{⟨| lat → 38.8522, long → -77.0378, radiusInMeters → 55560, label → TFR |⟩}

```

Convert nautical miles to meters (and round):

```

convertNMtoMeters[nm_] :=
  Round[N[QuantityMagnitude[UnitConvert[nm, "NM"], "Meters"]], 1]

```

```

convertNMtoMeters[Quantity[30, "NM"]]
55560

```

Do the same for polygons:

```

polygonTFRDataAPIOutput[tfrList_] := Block[{polys},
  polys = Select[tfrList, #[["Shape"]] == "Polygon" &];

  Map[polygonCoordinatesForAPIOutput[##[["BoundedRegion"]]] &, polys]
]

polygonTFRDataAPIOutput[nearbyUSTFRs[Here, 9900]] // AbsoluteTiming
{0.050805, {{lat → 34.2272, long → -116.95},
  {lat → 34.2122, long → -116.617}, {lat → 34.05, long → -116.667},
  {lat → 34.0975, long → -116.95}, {lat → 34.2272, long → -116.95}}}

```

Prepare polygon data for API output:

```

Clear[polygonCoordinatesForAPIOutput]

polygonCoordinatesForAPIOutput[region_] :=
  polygonCoordinatesForAPIOutput[region] =
    Map[{"lat" → QuantityMagnitude[Latitude[#]],
      "long" → QuantityMagnitude[Longitude[#]]} &, region]

```

```

polygonCoordinatesForAPIOutput[{GeoPosition[{34.22722222`,-116.95`}],
GeoPosition[{34.21222222`,-116.61666667`}],
GeoPosition[{34.05`,-116.66666667`}],GeoPosition[{34.0975`,-116.95`}],
GeoPosition[{34.22722222`,-116.95`}]}]

{{lat→34.2272, long→-116.95},
{lat→34.2122, long→-116.617}, {lat→34.05, long→-116.667},
{lat→34.0975, long→-116.95}, {lat→34.2272, long→-116.95}}

```

## Stadiums

### Data

Import data on U.S. sports stadiums (MLB, NFL, Nascar, NCAA football). Data fields: lat, long, sport, team, stadium, capacity.

```

usStadiumData =
Import["/Users/mpoulshock/Dropbox/Projects/Drones/Misc/Stadiums.csv", "CSV"];

```

### Query

Query the in-memory airport database to get all stadiums (of a particular type) within N miles of a location. Possible values for type are MLB, NFL, NCAAFootball, and AUTO.

```

nearbyUSStadiums[position_GeoPosition, rangeInMiles_, type_] :=
Block[{lat, long, offset, a, b, c, d},
lat = QuantityMagnitude[Latitude[position]];
long = QuantityMagnitude[Longitude[position]];
offset = milesToDegrees[rangeInMiles];
a = lat - offset;
b = lat + offset;
c = long - offset;
d = long + offset;

Select[usStadiumData, #[[3]] == type && a < #[[1]] < b && c < #[[2]] < d &]
]
```

```
nearbyUSStadiums[Here, 500, "MLB"] // Column
{39.9058, -75.1664, MLB, Phillies, Philadelphia Phillies, Citizens Bank Park, 43 651}
{41.2634, -81.5596, MLB, Red Sox, Boston Red Sox, Fenway Park, 37 499}
{36.2274, -80.8334, MLB, Rangers,
 Texas Rangers, Globe Life Park in Arlington, 48 114}
{42.5659, -75.9553, MLB, Reds, Cincinnati Reds, Great American Ball Park, 42 319}
{38.0357, -81.8315, MLB, Nationals, Washington Nationals, Nationals Park, 41 418}
{40.6418, -73.1707, MLB, Braves, Atlanta Braves, Turner Field, 49 586}
{37.0446, -80.228, MLB, White Sox, Chicago White Sox, U.S. Cellular Field, 40 615}
{37.0446, -80.228, MLB, Cubs, Chicago Cubs, Wrigley Field, 41 019}
```

## MLB games today

This queries Wolfram|Alpha to identify today's MLB games, and then restructures the results so only the game time and home team are returned. It takes about 3s the first time it's used each day.

```
mlbGamesToday[date_] := mlbGamesToday[date] = Block[{d, games},
  games = WolframAlpha["MLB games today", "DataRules"];
  Map[{DateObject#[[2]], StringSplit#[[3]], " at "][[2]]} &,
  Drop[games[[1, 2]], 1]]
]

mlbGamesToday[Today]
{{{"Wed 8 Jul 2015 19:10 GMT-4.", Dodgers}, {"Wed 8 Jul 2015 19:10 GMT-4.", Royals}, {"Wed 8 Jul 2015 19:10 GMT-4.", Indians}, {"Wed 8 Jul 2015 19:10 GMT-4.", White Sox}, {"Wed 8 Jul 2015 19:10 GMT-4.", Red Sox}, {"Wed 8 Jul 2015 19:05 GMT-4.", Nationals}, {"Wed 8 Jul 2015 19:05 GMT-4.", Pirates}, {"Wed 8 Jul 2015 19:05 GMT-4.", Yankees}, {"Wed 8 Jul 2015 19:05 GMT-4.", Cubs}, {"Wed 8 Jul 2015 19:05 GMT-4.", Rangers}, {"Wed 8 Jul 2015 18:40 GMT-4.", Rockies}, {"Wed 8 Jul 2015 13:10 GMT-4.", Brewers}, {"Wed 8 Jul 2015 12:45 GMT-4.", Giants}, {"Wed 8 Jul 2015 12:40 GMT-4.", Mariners}, {"Wed 8 Jul 2015 12:10 GMT-4.", Twins}}
```

Given a team, this returns the time that team is playing today:

```
mlbGameTime[team_] := Block[{games = Cases[mlbGamesToday[Today], {_, team}]},
  If[games == {}, Yesterday, games[[1, 1]]]
]
```

## Convert data to output format for API

This function identifies all MLB games in progress near the vehicle:

```

mlbGameDataAPIOutput[position_, plotRadiusInMiles_, stadiumRadiusInNM_] :=
  Block[{meters, nearbyStadiums, latLongTimeTeam, latLongTimeTeamLiveGames},
    (* meters =Round[N[QuantityMagnitude[
      UnitConvert[Quantity[stadiumRadiusInNM,"NM"],"Meters"]]],1]; - BUG *)
    (* Identify nearby MLB stadiums by doing a database lookup *)
    nearbyStadiums = nearbyUSStadiums[position, plotRadiusInMiles, "MLB"];

    (* Get the latitude, longitude, gametime, and team of each nearby game *)
    latLongTimeTeam =
      Map[{#[[1]], #[[2]], mlbGameTime#[[4]], #[[4]]} &, nearbyStadiums];

    (* Identify games where it's
       currently 1 hour before to 1 hour after gametime *)
    latLongTimeTeamLiveGames = Select[latLongTimeTeam,
      DatePlus#[[3]], {-1, "Hour"}] ≤ Now ≤ DatePlus#[[3]], {5, "Hour"}] &;

    (* Convert data to API output format *)
    Map[<|"lat" → #[[1]], "long" → #[[2]],
      "radiusInMeters" → 5556, "label" → "MLB"|> &, latLongTimeTeamLiveGames]
  ]
}

mlbGameDataAPIOutput[Here, 1000, 3]
{ }

mlbGameDataAPIOutput[GeoPosition[{40.8292, -73.9264}], 10, 3]

{<|lat → 40.8292, long → -73.9264, radiusInMeters → 5556|>}

```

## Determine whether the drone is in a no-fly zone

Temporary hack: Determine whether a location is in a 3 mi. radius of an airport, in a circular TFR, or in a municipal no-fly zone. Later, a deeper and more efficient method will have to be used for this analysis.

```

droneInRedRegionQ[geo_] :=
  Block[{airportZones, tfrZones, inCircularZone, inMunicipalNoFlyZone},
    (* Is drone within 3 mi. of any airport? *)
    airportZones = Map[
      GeoDisk[GeoPosition[{#[["lat"]], #[["long"]]}], Quantity[3, "Miles"]] &,
      airportDataAPIOutput["AIRPORT", geo, 15, 3]
    ];
    tfrZones = Map[
      GeoDisk[GeoPosition[{#[["lat"]], #[["long"]]}],
        Quantity[#[["radiusInMeters"]], "Meters"]] &,
      circularTFRDataAPIOutput[nearbyUSTFRs[geo, 35]]
    ];
    inCircularZone =
      MemberQ[GeoWithinQ[Catenate[{airportZones, tfrZones}], geo], True];
    inMunicipalNoFlyZone =
      MemberQ[GeoWithinQ[municipalNoFlyGeoRegions, Here], True];
    Return[inCircularZone || inMunicipalNoFlyZone];
  ]
droneInRedRegionQ[Here] // AbsoluteTiming
{0.763566, False}
droneInRedRegionQ[GeoPosition[{38.9072, -77.0369}]] // AbsoluteTiming (* DC *)
{0.110736, True}
droneInRedRegionQ[GeoPosition[{42.3333, -72.65}]] //
  AbsoluteTiming (* Northampton *)
{0.084629, True}

```