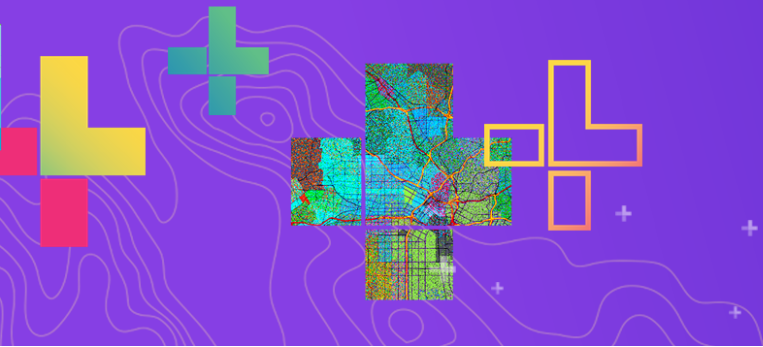



# Interactive Client-Side Mapping with the ArcGIS API for JavaScript

Undral Batsukh [ubatshukh@esri.com] | Richie Carmichael [@kiwiRichie]

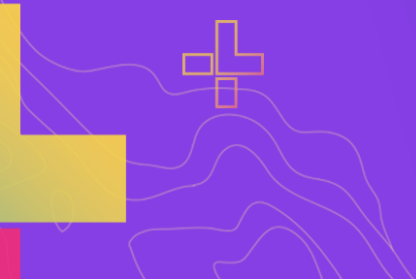
<https://git.io/JvB8u> (printer friendly, pdf)



# Agenda

- Client-side Layers
    - FeatureLayer, CSVLayer, GeoJSONLayer
  - Query
    - Layer vs LayerView
    - Client-side query
  - Filters and Effects
    - Adjusting client-side visuals
  - Geometry Engine, Projection Engine and Geodesic Utils
    - Client-side analysis
- 

# Client-side Layers

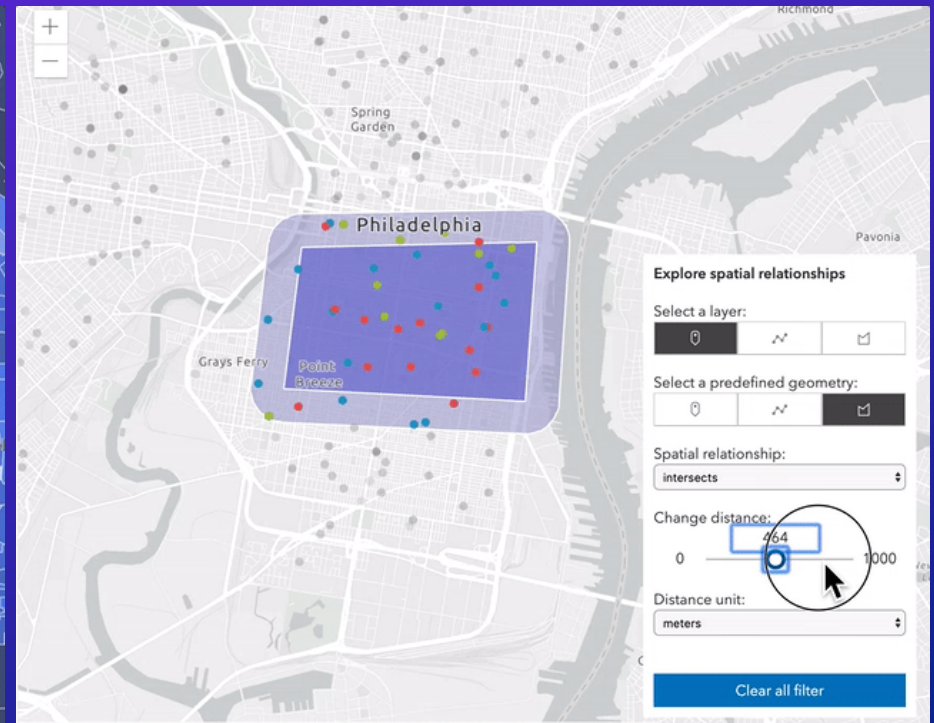
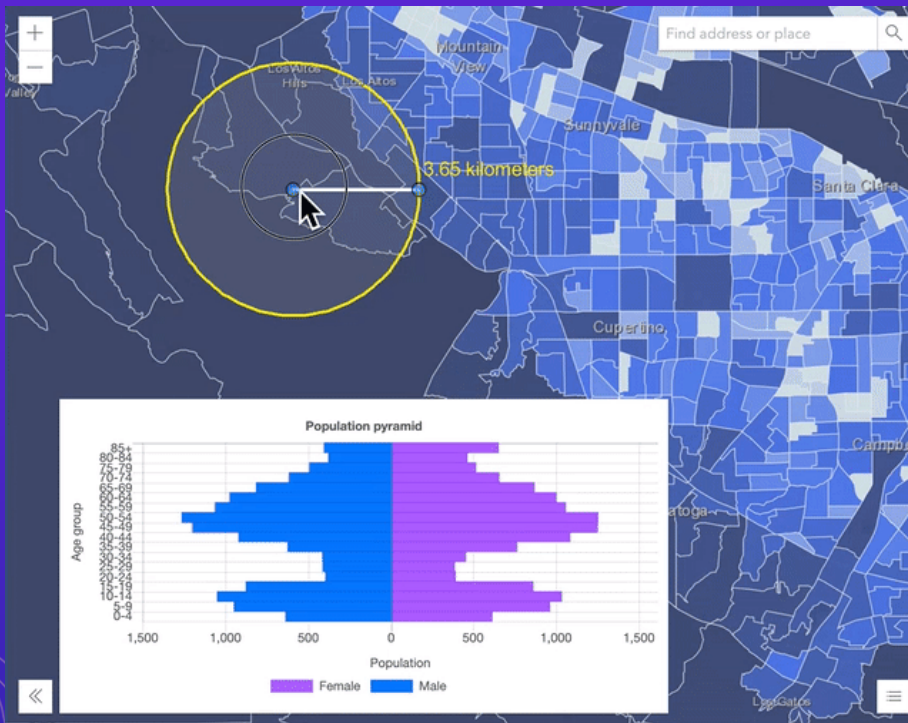


# Client-side Layers

- CSVLayer
- GeoJSONLayer
- FeatureLayer with feature collections.

# Client-side Layers

- Fetch all features at once and store them on the client
- Uniform API
- Responsive and fast performance



# CSVLayer

- Add data from csv/txt file as points

```
const new CSVLayer({
  url: "https://earthquake.usgs.gov/earthquakes/.../2.5_week.csv",
  copyright: "USGS Earthquakes",
  // SR in which the data will be stored
  spatialReference: { wkid: 102100 },
  delimiter: ",",
  latitudeField: "lat",
  longitudeField: "lon",
  // defaults to "__OBJECTID"
  objectIdField: "myOid",
  // create timeInfo for temporal visualization
  timeInfo: {
    startField: "time", // name of the date field
    // set time interval to one day
    interval: { value: 1, unit: "days" },
  }
})
```

API Reference | Sample 1

# CSVLayer - Tips

- X, Y coordinates must be in WGS84 in csv file.
- Specify the layer's spatial reference to improve the performance.
- Not supported:
  - No z-values support.
  - Cannot add, remove or update features.



# CSVLayer - Tips

- Can pass data by a blob url.

```
const csv = `first_name|Year|latitude|Longitude
Undra1|2020|40.418|20.553
Richie|2018|-118|35`;

const blob = new Blob([csv], {
  type: "plain/text"
});
let url = URL.createObjectURL(blob);

const layer = new CSVLayer({
  url: url
});
await layer.load();

URL.revokeObjectURL(url);
url = null;
```



# FeatureLayer

- Add client-side graphics by setting *FeatureLayer.source*

```
const layer = new FeatureLayer({
  source: [
    new Graphic({ attributes: { myOid: 1 }, geometry: { ... } })
    new Graphic({ attributes: { myOid: 2 }, geometry: { ... } })
    new Graphic({ attributes: { myOid: 3 }, geometry: { ... } })
  ],
  // can be inferred from geometries
  geometryType: "point",
  // can be inferred from geometries
  spatialReference: { wkid: 2154 },
  // can be inferred from fields w/ field.type "oid"
  objectIdField: "myOid",

  fields: [
    new Field({ name: "myOid", type: "oid" })
  ]
})
```

# FeatureLayer - Tips

- Supports data in any spatial reference.
- Specify *source* only at the time of initialization.
- Use `FeatureLayer.applyEdits` to add, remove or update features at runtime.
- Call `FeatureLayer.queryFeatures` to get the updated feature collection.

Sample - add/remove graphics



# GeoJSONLayer

- Add GeoJson data that comply with the RFC 7946 specification

```
const geoJSONLayer = new GeoJSONLayer({  
  url: "https://earthquake.usgs.gov/earthquakes/.../all_month.geojson",  
  copyright: "USGS Earthquakes",  
  // SR in which the data will be stored  
  spatialReference: { wkid: 102100 }  
});
```

API Reference | Time-aware GeoJSONLayer

# GeoJSONLayer - Tips

- Specify layer's spatial reference for performance.
- Support for Feature and FeatureCollection
- `GeoJSONLayer.applyEdits` to add, delete or update features.
- Not supported:
  - Mixed geometry types for consistency with other layers.
  - `crs` object - only geographic coordinates using WGS84 datum (long/lat)
  - No Antimeridian crossing
  - Feature `id` as string

# GeoJSONLayer - Tips

- Create a blob url from GeoJSON object

```
const geojson = `{
  type: "FeatureCollection",
  features: [{
    type: "Feature",
    geometry: { type: "Point", coordinates: [-100, 40] },
    properties: { name: "none" }
  }]
}`;

const blob = new Blob([JSON.stringify(geojson)], {
  type: "application/json"
});
let url = URL.createObjectURL(blob);
const layer = new GeoJSONLayer({ url });

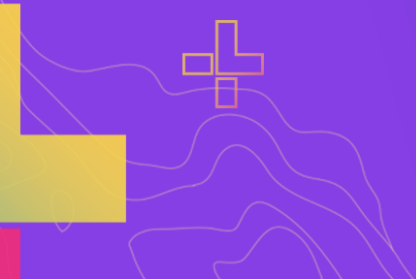
await layer.load();
URL.revokeObjectURL(url);
url = null;
```

# Client-side layers tips

- Each implementation uses the client-side query engine.
- Prefer GeoJSON over CSV.
- Proper attribution using copyright property.



# Query



# Query

- Query expressions are used to select a subset of features and table records.
- Query can be done on the server or on the client-side.
- Different query . . . methods are available on Layers and LayerViews.



# Layers and Layer Views

- Server-side layers
  - Fetch or stream features on demand
  - `FeatureLayer` created from a service, `SceneLayer`
  - A `LayerView` represents the view for a single layer after it has been added to a `View`.
  - `LayerView` API is layer agnostic.
  - Methods and properties used for features available for drawing on the client-side.



# Client-side query

- (CSV|GeoJSON)Layer and FeatureLayer/FeatureCollection
- (CSV|GeoJSON|Feature|Scene|Stream)LayerView
- Query methods on layer and layerView
  - queryFeatures()
  - queryFeatureCount()
  - queryObjectIds()
  - queryExtent()

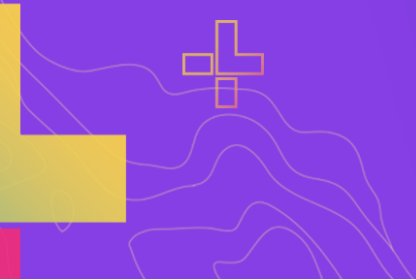
Layer vs LayerView | Age Pyramid | 3D buildings



# Query tips

<b>When to use</b>	<b>Layer queries</b>	<b>LayerView queries</b>
Speed and responseviness	No(server layer) Yes(client layers)	Yes. Client-side query
Query all features	Yes	No. Features available for drawing
Geometry precision	Yes	No. Quantized/generalized

# Filters



# What are filters?

- Reduce the number of features shown screen
- Can apply spatial, aspatial, temporal (or any combination)

```
featureLayerView.filter = new FeatureFilter({  
  geometry: myGeometry      // Spatial  
  timeExtent: myTimeExtent, // Temporal  
  where: myWhere           // Aspatial  
});
```

- Client-side
  - Only applied to features currently downloaded
  - Fast.
- FeatureFilter has the same properties as Query

# Spatial Filter

Only show buildings within 10 miles of the mouse cursor

```
mapView.on("pointer-move", (event) => {  
  buildingLayerView.filter = new FeatureFilter({  
    geometry: mapView.toMap({  
      event.x,  
      event.y  
    }),  
    distance: 10,  
    units: "miles"  
  })  
});
```

# Aspatial Filter

Only show earthquakes with a magnitude greater than 7

```
featureLayerView.filter = new FeatureFilter({  
  where: "magnitude >= 7"  
});
```

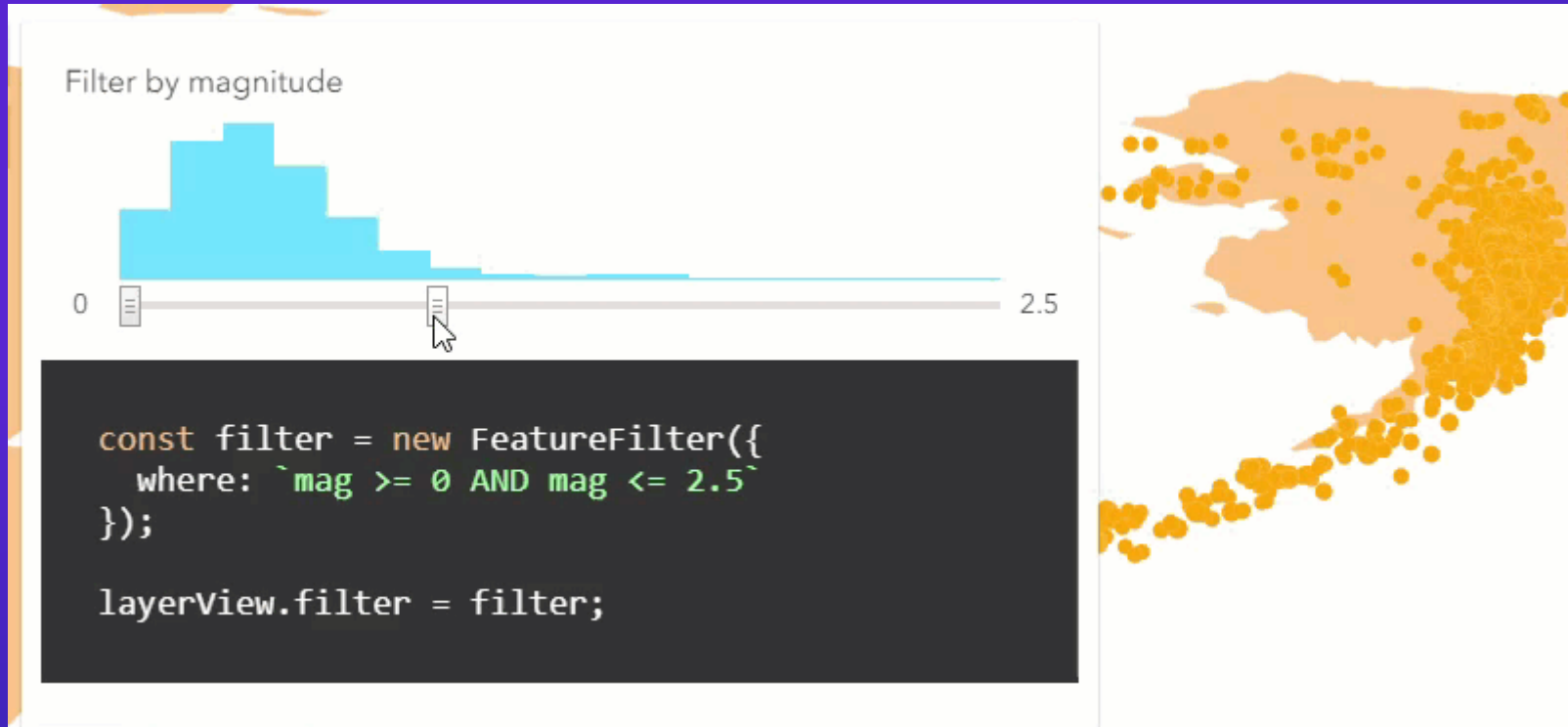
# Temporal Filter

Only show earthquakes that occurred between 2000 and 2007

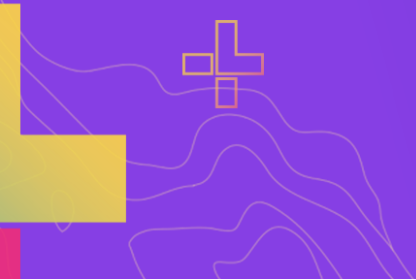
```
featureLayerView.filter = new FeatureFilter({
  timeExtent: new TimeExtent({
    start: new Date(2000, 0, 1),
    end: new Date(2007, 0, 1)
  })
});
```



# Demonstration - filters

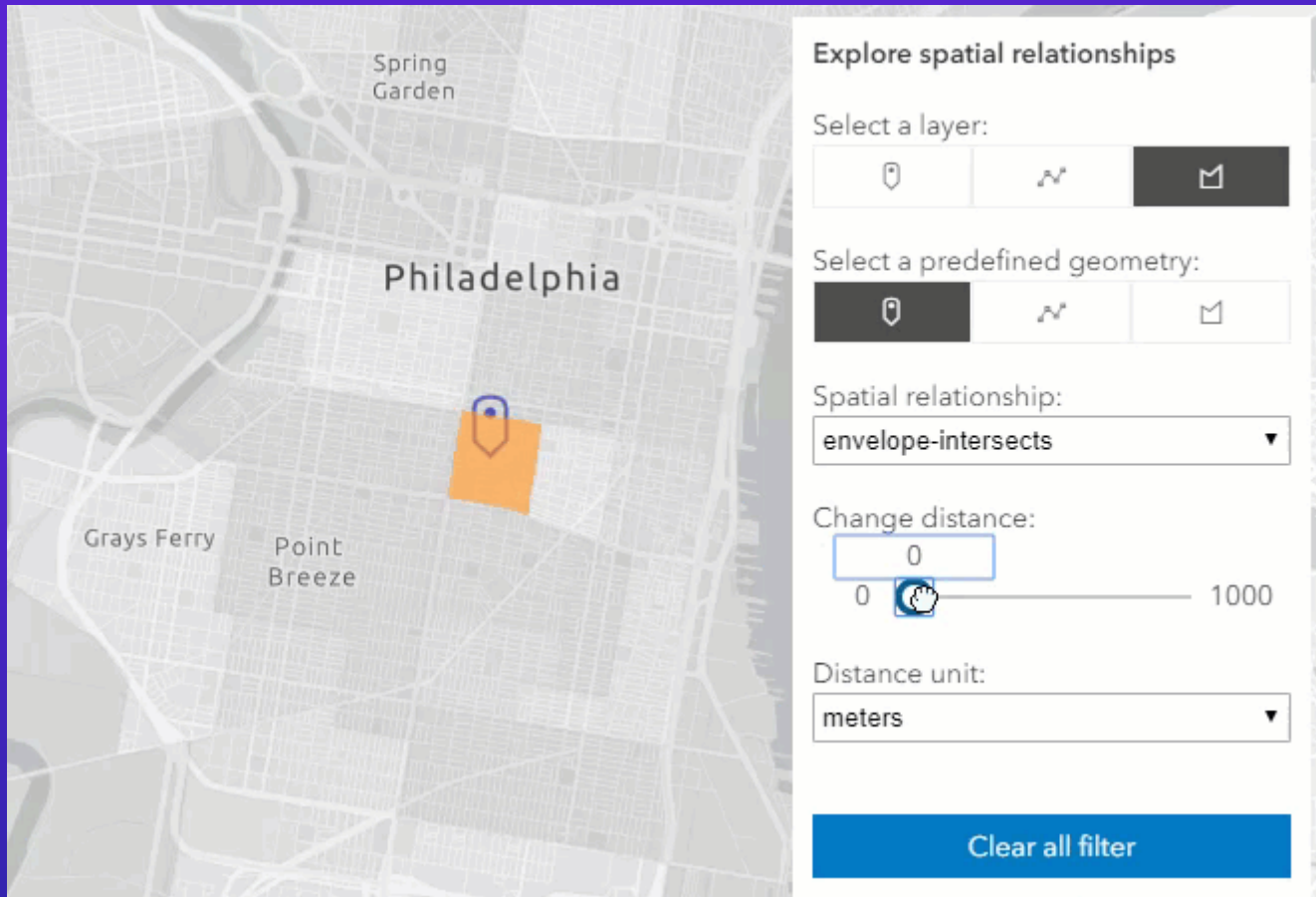


# Effects



# What are effects?

- Visual effects applied to included or excluded features.



# Effect - Snippet

Show earthquakes with a magnitude of 7 or greater as faint shadows

```
// Show quakes less than 7 magnitude as faint shadows.
featureLayerView.effect = new FeatureEffect({
  filter: new FeatureFilter({
    where: "magnitude >= 7"
  }),
  excludedEffect: "grayscale(100%) opacity(0.5)",
  // includedEffect: "saturate(150%)"
});
```

# Demonstration - century of quakes

The screenshot shows a web browser window with the following elements:

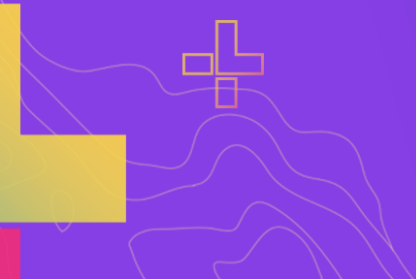
- Browser Tab:** Client-side filters
- Address Bar:** Not secure | tui/GitHub/presentations/2020-developer-su...
- Map:** A map of the Pacific region showing earthquake activity. Labels include Hong Kong, Singapore, and INDONESIA. A grey overlay indicates the current time range.
- Temporal Filter Overlay:**
  - Title:** Choose a Temporal Property
  - Options:**
    - FeatureLayerView.Filter
    - FeatureLayerView.Effect
- Timeline:**
  - Current Range:** 7/28/1900 - 7/28/1910
  - Slider:** A horizontal slider with a white handle positioned at 1910.
  - Axis Labels:** 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990, 2000, 2010
  - Start/End Dates:** 7/28/1900 (left) and 2/8/2019 (right)
  - Navigation:** Play, Previous, and Next buttons.

# Supported Effects

```
/* effect(default-value) */  
brightness (0.4);  
contrast (200%);  
grayscale (50%);  
hue-rotate (90deg);  
invert (75%);  
opacity (25%);  
saturate (30%);  
sepia (60%);
```

CSS reference

# Geometry Engine



# One Engine - Thirty Methods

- Compute new geometries  
*e.g. buffer, clip, densify, generalize*
- Testing spatial relationship  
*e.g. contains, crosses, intersects*
- Advanced computations  
*e.g. geodesicArea, planarArea*





# One Engine - Two Flavors

- geometryEngine

```
const geometryEngine = new GeometryEngine();
const length = geometryEngine.planarLength(myPolyline, "meters");
console.log("length(m)", length);
```

- geometryEngineAsync

```
const geometryEngine = new GeometryEngineAsync();
geometryEngine.planarLength(myPolyline, "meters").then((length) => {
  console.log("length(m)", length);
});
```

# Demonstration - Sketch Validation



# Wait, what about GeometryService?

- Same methods as GeometryEngine/GeometryEngineAsync
- Sends request to remote server for computation
- **Pro:**
  - May be faster for complex computations
- **Con:**
  - May be slower due to network latency



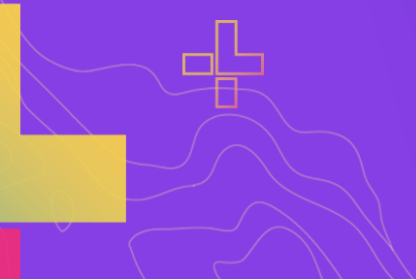
# GeometryService - Snippet

```
const parameters = new LengthsParameters({
  calculationType: "planar",
  lengthUnit: "esriSRUnit_Meter",
  polylines: [myPolyline]
});

const geometryService = new GeometryService({
  url: "http://sampleserver6.arcgisonline.com/arcgis/rest/services/" +
    "Utilities/Geometry/GeometryServer"
});

geometryService.lengths(parameters).then((response) => {
  console.log("length(m)", response.lengths[0]);
});
```

# Projection Engine



# What is Projection Engine

- Client-side module for projecting geometries between coordinate systems
- Based on the engine used by Runtime
- Uses *WebAssembly*
  - Not support by Internet Explorer (see caniuse)

# Loading the Projection Engine

```
const projection = new Projection();

if (!projection.isSupported()) {
  console.error("projection is not supported");
  return;
}

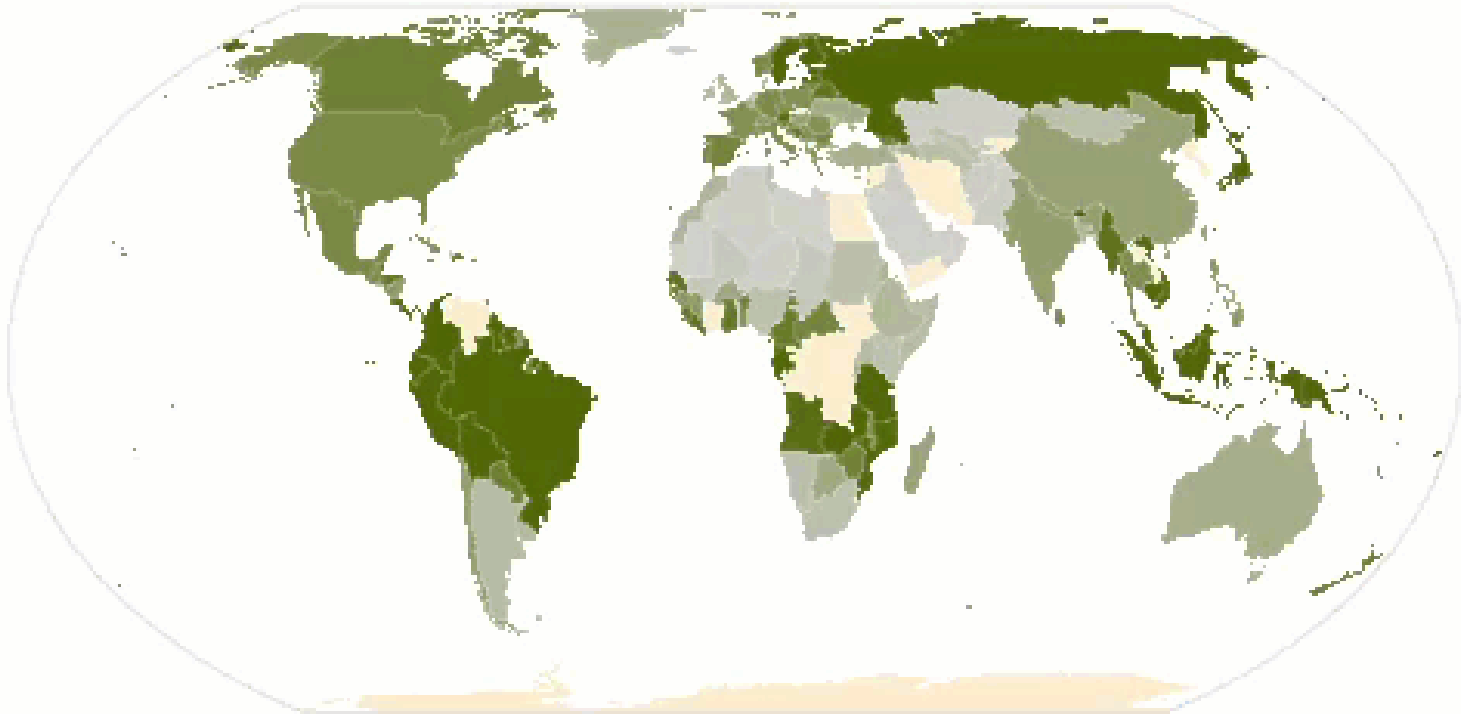
if (projection.isLoaded()) {
  console.log("projection already loaded");
  return;
}

projection.load().then(() => {
  console.log("projection loaded");
});
```

# Demonstration - Projection

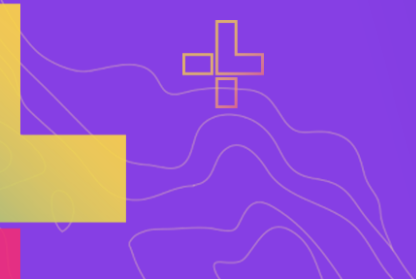
Select a projection

Equal Earth Greenwich ▼





# Geodesic Utils



# What is Geodesic Utils?

- Lightweight module for performing geodesic computations
- Some redundancy with GeometryEngine (GE)
  - `GU.geodesicAreas()` vs. `GE.geodesicArea()`
  - `GU.geodesicDensify()` vs. `GE.geodesicDensify()`
  - `GU.geodesicLengths()` vs. `GE.geodesicLength()`
- Works on any supported geographic coordinate system include 70+ non-terrestrial systems (e.g. Mars and Moon)



# Geodesic Utils - Example

What is the area of the Bermuda Triangle?

```
const MIAMI      = { lat: 25.775278, lon: -80.208889 }; // Florida
const HAMILTON  = { lat: 32.293, lon: -64.782 }; // Bermuda
const SANJUAN   = { lat: 18.406389, lon: -66.063889 }; // Puerto Rico
const polygon = new Polygon({
  rings: [[
    [MIAMI.lon, MIAMI.lat],
    [HAMILTON.lon, HAMILTON.lat],
    [SANJUAN.lon, SANJUAN.lat],
    [MIAMI.lon, MIAMI.lat]
  ]]
});
const areas = geodesicUtils.geodesicAreas([polygon], "square-kilometers");
const area = Math.round(areas[0]);
console.log(`Area: ${area} km2`); // Area: 1150498 km2
```

# Geodesic Utils - Mars Rovers

How far is Curiosity (active) from Opportunity on Mars?

```
const polyline = new Polyline({
  spatialReference: {
    wkid: 104905 // Mars 2000 Coordinate System
  }
});
polyline.addPath([ curiosity.geometry, opportunity.geometry ]);

const distance = geodesicUtils.geodesicLengths([polyline], "kilometers")[0];
const densified = geodesicUtils.geodesicDensify(polyline, 100);

const path = new Graphic({
  attributes: {
    oid: 1,
    distance
  },
  geometry: densified
});
```



**esri**

**THE  
SCIENCE  
OF  
WHERE**

