ArcGIS API for JavaScript: Programming Patterns and API Fundamentals

Hugo Campos, Lauren Boyd, Rene Rubalcava

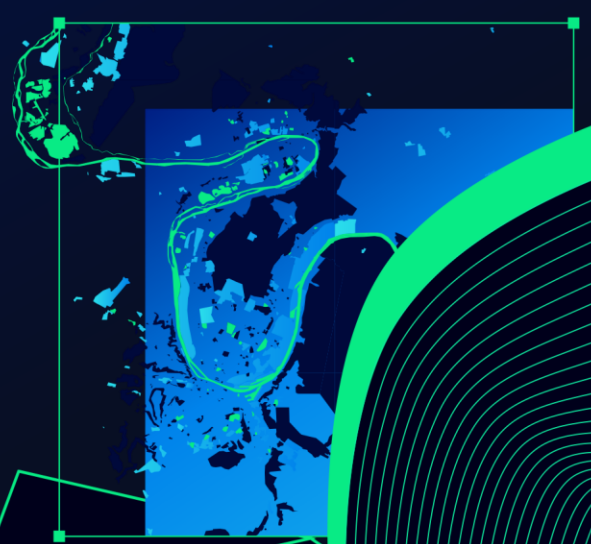
2022 ESRI DEVELOPER SUMMIT

Why choose the ArcGIS API for JavaScript?

- Seamless interfaces to ArcGIS Platform, ArcGIS Online, and ArcGIS Enterprise
- Client-side analysis
- Interactive data visualizations powered by WebGL
- 2D and 3D all in one API
- Widgets to build professional UI

Maps and Views

Lauren Boyd



live by the code

Map and View

```
const map = new Map({
  basemap: "topo"
});

const mapView = new MapView({
  map: map,
  container: "viewDiv"
});
const sceneView = new SceneView({
  map: map,
  container: "viewDiv"
})
```

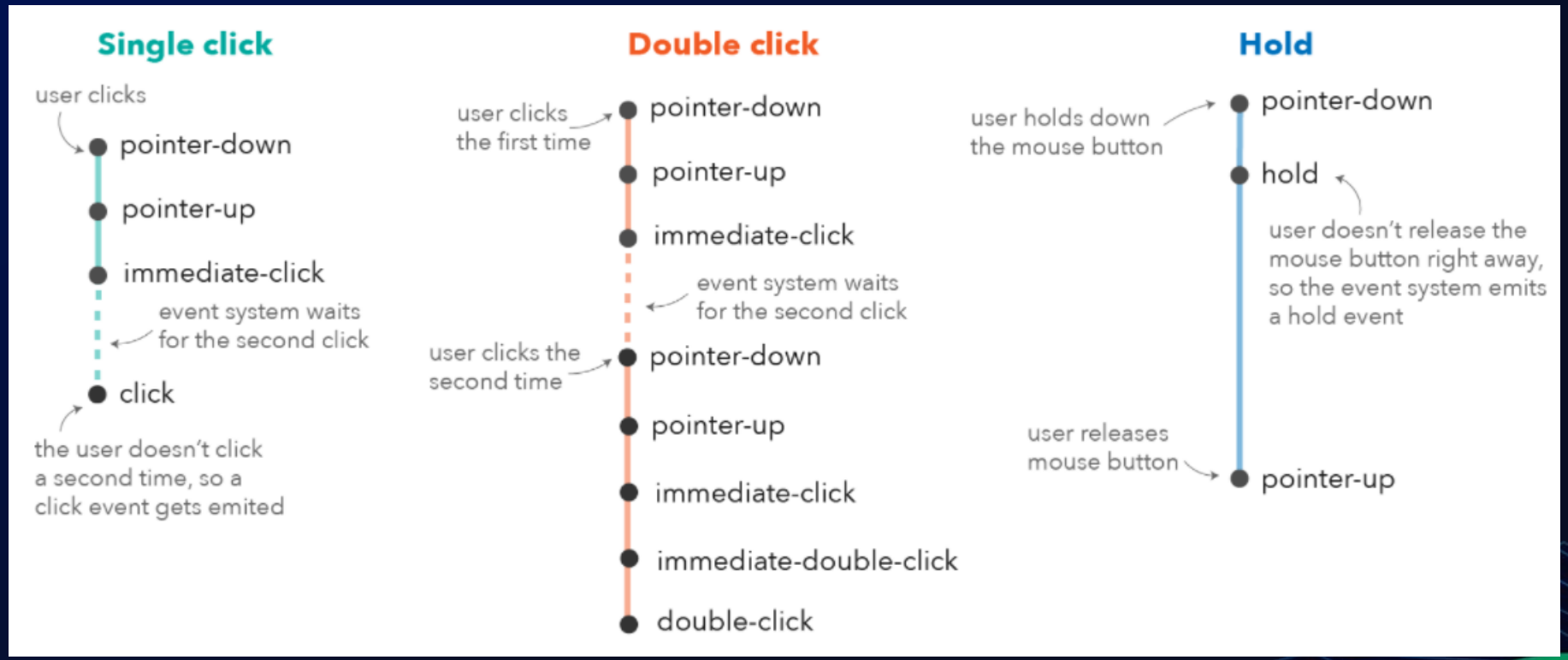
Basemaps

- Basemaps for developers
  - For use with API keys
- Basemaps for ArcGIS Organization users
  - For use with ArcGIS Online/ArcGIS Enterprise users

```
const map = new Map({
  /*
  For ArcGIS organizational usage:
   "satellite", "hybrid", "terrain", "oceans", "osm", "dark-gray-vector", "gray-vector", "streets-vector", "topo-vector", "streets-night-vector", "streets-relief-vector", "streets-navigation-vector"on-vector"
  For API key usage:
  "arcgis-imagery", "arcgis-imagery-standard", "arcgis-imagery-labels", "arcgis-light-gray", "arcgis-dark-gray", "arcgis-navigation", "arcgis-navigation-night", "arcgis-streets", "arcgis-streets-night", "arcgis-streets-relief", "arcgis-topographic", "arcgis-oceans", "osm-standard", "osm-standard-relief", "osm-streets", "osm-streets-relief", "osm-light-gray", "osm-dark-gray", "arcgis-terrain", "arcgis-community", "arcgis-charted-territory", "arcgis-colored-pencil", "arcgis-nova", "arcgis-modern-antique", "arcgis-midcentury", "arcgis-newspaper", "arcgis-hillshade-light", "arcgis-hillshade-dark"

   */
  basemap: "topo-vector"

  /*
   world-elevation
   */
  ground: "world-elevation"
})
```

Interactivity with View Events

- Use view events to interact with the view
- List of Events



Interactivity with View Events

- Access features on pointer move

```
// get screenpoint from view's pointer-move event
view.on("pointer-move", (event)⇒{
  // Search for graphics on layers at the hovered location
  // exclude view.graphics from the hitTest
  view.hitTest(event, {exclude: view.graphics}).then((response) ⇒ {
    // if graphics are returned, do something with results
    const graphic = response.results[0].graphic;
  })
}
```

Interactivity with View Events

- Stop event propagation to prevent default behavior

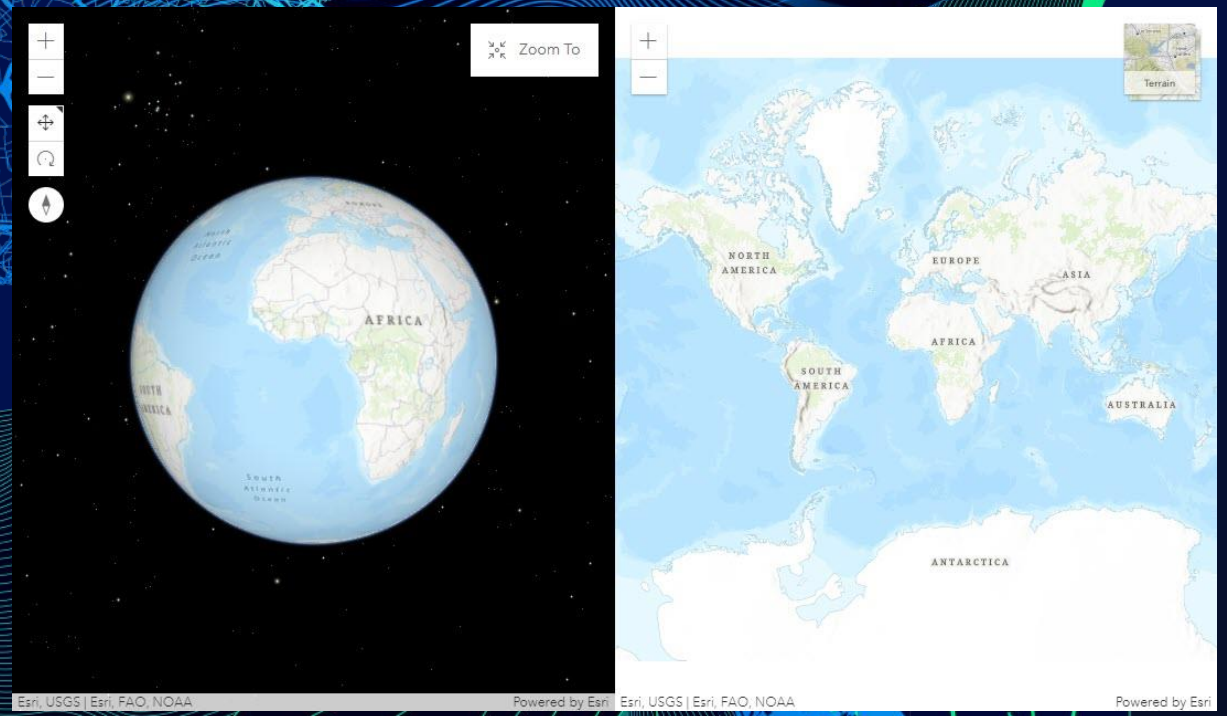
```
1 view.on("drag", event ⟹ {
2   // user won't be able to drag
3   event.stopPropagation();
4 })
```

goTo() with View

Subhead

- Sets the view to given target
  - Navigate to a geometry/feature/location

```
view.goTo({
    center: [-116.538, 33.826],
    zoom: 16
}).catch((error) => {
    console.error(error);
});
```

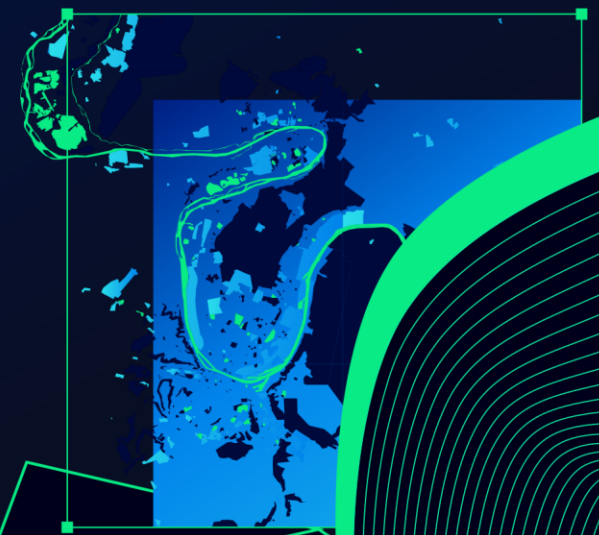


Maps and Views

Demo

Patterns

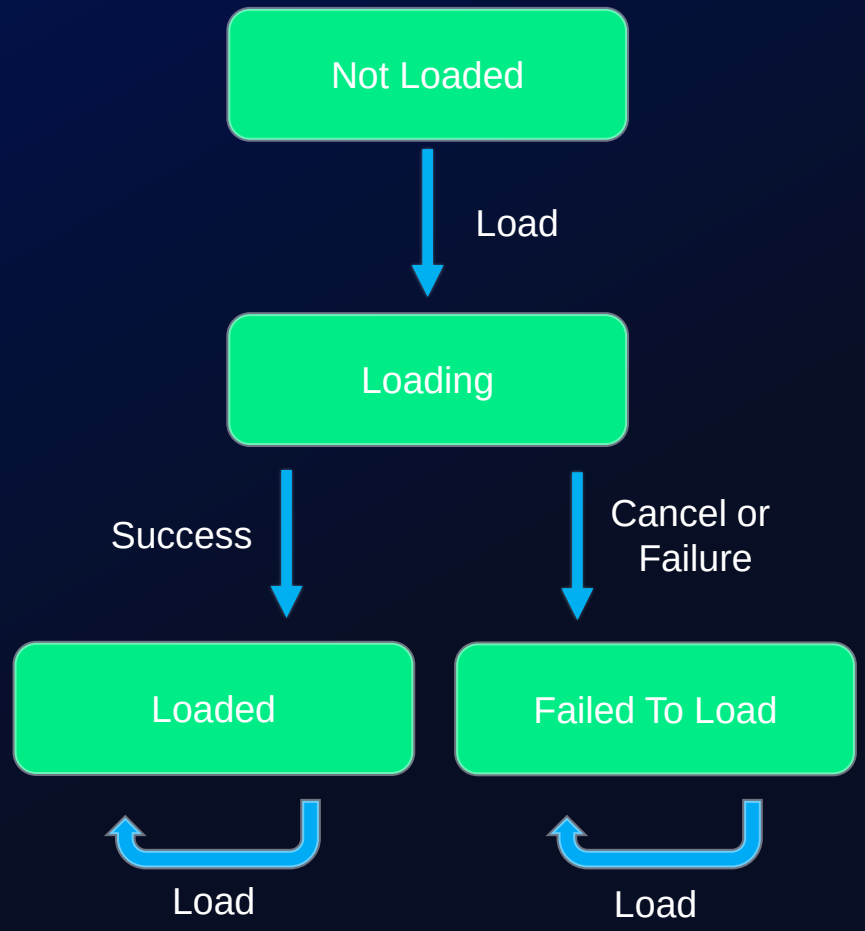
Lauren Boyd



live
by
the
code

Loadable Patterns

- Better control
- Allows scheduling of loading resources
- View automatically loads the map and its layers



Loadables

```
const view = new MapView({
  container: "viewDiv"
});

const portal = new Portal({
  url: "https://myportal/"
});

const webmap = new WebMap({
  portalItem: {
    portal: portal,
    id: "f2e9b762544945f390ca4ac3671cfa72"
  }
});

webmap.load()
  .then(() => { view.map = webmap; })
  .catch((error) => {
    console.error("The resource failed to load: ", error);
  });
```

Sublayer to FeatureLayer

- Extract FeatureLayer from MapImageLayer Sublayer
  - Sublayer.createFeatureLayer()
- Utilize FeatureLayer capabilities

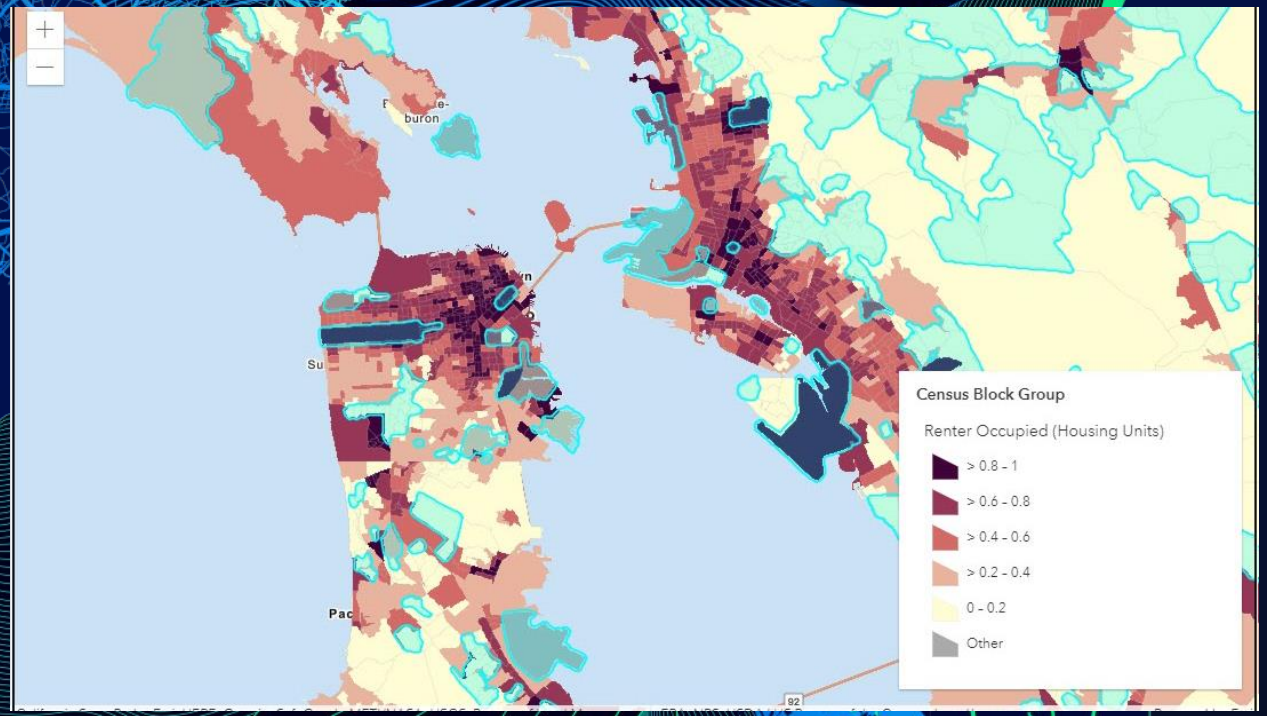
LayerViews

- Responsible for rendering features as graphics in view
- Allows for client-side queries, filtering, effects, etc.

createQuery

- Using layer.createQuery()
  - Query object already has the layers filters and layer definitions
  - more consistent
- Use new Query() when you don't want predefined filters to be applied

```
1 const query = featureLayer.createQuery();
2 query.where = "Field = '1'";
3 query.returnGeometry = true;
4 featureLayer.queryFeatures(query).then((results) => {
5   // do something with the results
6   console.log(results);
7 });
```



LayerViews

Demo

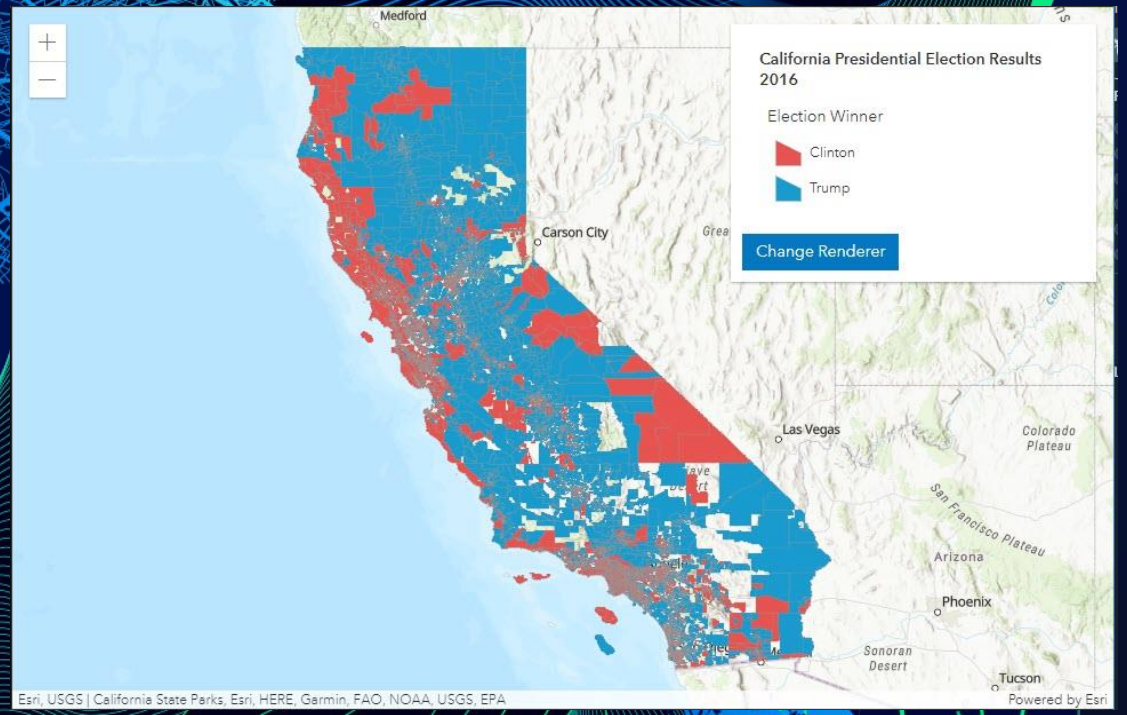
Rendering

- Specifies how a layer is visualized
- Determines what information will be conveyed through the layer's symbology
- What story do you want to tell with this layer's data?

Visualization Method	Renderer Class
Location only	SimpleRenderer, HeatmapRenderer
Unique values	UniqueValueRenderer
Class breaks	ClassBreaksRenderer
Continuous color or size	SimpleRenderer or UniqueValueRenderer, with visual variables
Multivariate	SimpleRenderer or UniqueValueRenderer, with visual variables

SmartMapping API

- Easy way to generate visualizations
- Focus on:
  - One visualization method
  - Subset of attributes
  - Specific Area
- Handle data normalization
- Provide specified filters

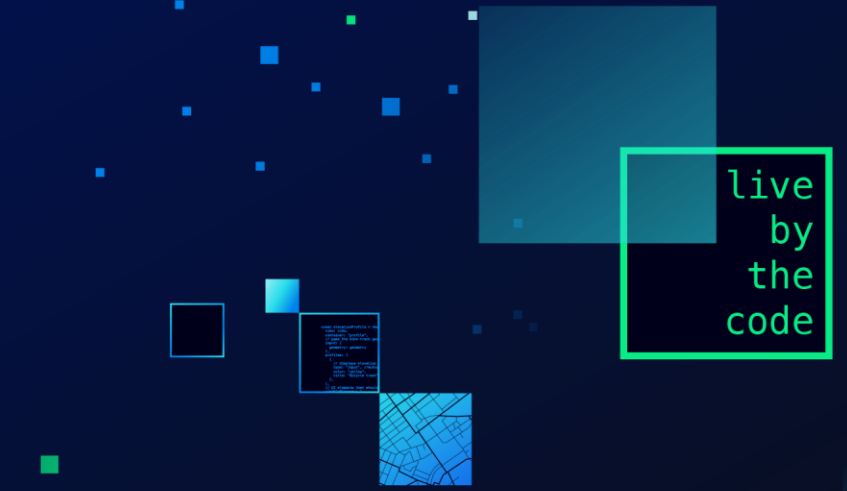


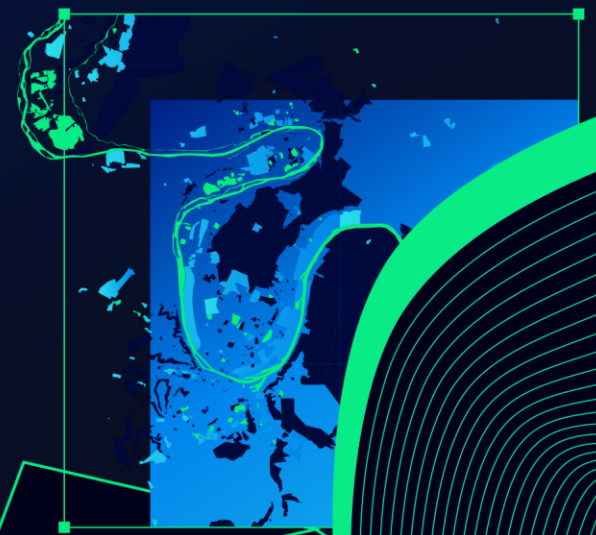
SmartMapping

Demo

Working with Accessor

Hugo Campos





Accessor

- Reactive state management system used across the ArcGIS JS API.
- Inspired by other reactive libs like MobX, SolidJS, Jotai, Recoil.
- Objects are class-based and with properties which can be:
  - Read/set like in normal objects
  - Read-only / computed
  - Watched / observed
- Constructors accept property bags (POJO) and apply them automatically.
- Plain values are auto-cast to the right types when possible.

Accessor – Class definition

```
import Accessor from "esri/core/Accessor";
import { property, subclass } from "esri/core/accessorSupport/decorators";
import { watch } from "esri/core/reactiveUtils";

@subclass("Person")
class Person extends Accessor {
  @property()
  firstName: string = "";

  @property()
  lastName: string = "";

  @property({ readOnly: true })
  get fullName(): string {
    return `${this.firstName} ${this.lastName}`;
  }
}

```

Accessor – Property access

```
const person = new Person({ firstName: "John", lastName: "Doe" });
console.log(person.fullName); // Logs "John Doe";

person.firstName = "Hugo";
person.lastName = "Campos";

console.log(person.fullName); // Logs "Hugo Campos";

// Set multiple properties at once
person.set({
  firstName: "Bruce",
  lastName: "Wayne"
});

console.log(person.fullName); // Logs "Bruce Wayne"

```

Accessor – Reacting to changes

```
1  const person = new Person({ firstName: "John", lastName: "Doe" });
2
3  watch(
4    () ⇒ person.fullName,
5    (fullName) ⇒ console.log(`Hello ${fullName}`),
6    { sync: true }
7  );
8
9  person.firstName = "Hugo"; // Logs "Hello Hugo Doe"
10 person.lastName = "Campos"; // Logs "Hello Hugo Campos"
11
```

Accessor – Reacting to changes

```
// Reacting when a property changes.
reactiveUtils.watch(
  () => view.map.basemap.title,
  (newValue, oldValue) => console.log(`old: ${oldValue}, new: ${newValue}`)
);

// Call the callback initially as well as on any changes.
reactiveUtils.watch(
  () => view.map.basemap.title,
  (newValue) => console.log(`basemap title: ${oldValue}`),
  { initial: true }
);

// React when an expression becomes truthy.
reactiveUtils.when(
  () => view.scale > 1000,
  () => console.log("Scale is greater than 1000")
)

```

Accessor – Reacting to changes

```
// React to multiple properties. Can react to any expression!
reactiveUtils.watch(
  () => [view.stationary, view.scale],
  ({ stationary, scale }) => {
    console.log(view is ${stationary} and scale is ${scale});
  }
)

// We support collections too.
reactiveUtils.watch(
  () => view.layers.map((layer) => layer.title),
  (titles) => console.log(`Layer titles changed! New titles: ${titles}`)
)

// In Typescript, callback types are automatically inferred.
reactiveUtils.watch(() => view.layers, (titles) => {
  // titles inferred as Collection<Layer>
})

```

Promises

- Asynchronous methods return native Promises

```
layer.queryFeatures(query)
  .then(handleResult)
  .catch(handleError);

// Or with async/await
const doQuery = async (query) => {
  const results = await layer.queryFeatures(query);
  const transformedResults = results.map(transformData);
  return transformedResults;
}

```

Promises

- Asynchronously initialized Layer, WebMap, WebScene, View

```
const map = new Map({...})

view = new SceneView({
  map: map,
  //...
});

view.when(() => {
  // the view is ready to go
});

```

Promises – multiple async operations

```
try {
  // Wait for the view to be ready/loaded.
  await view.when();

  // Get the layer view.
  const layerView = await view.whenLayerView(map.findLayerById("awesomeLayer"));

  // Layer is fully rendered and ready to be queried.
  await reactiveUtils.whenOnce(() => !layerView.updating);

  // Query features on the client.
  const features = await layerView.queryFeatures(/* ... */);

  doSomethingWithFeatures(features);
} catch (e) {
  // Handle errors
}

```

Promises – async module loading

```
async function loadMap(id: string): Promise<WebMap> {
  const { default: WebMap } = await import('@arcgis/core/WebMap')
  return new WebMap({ portalItem: { id } })
}

```

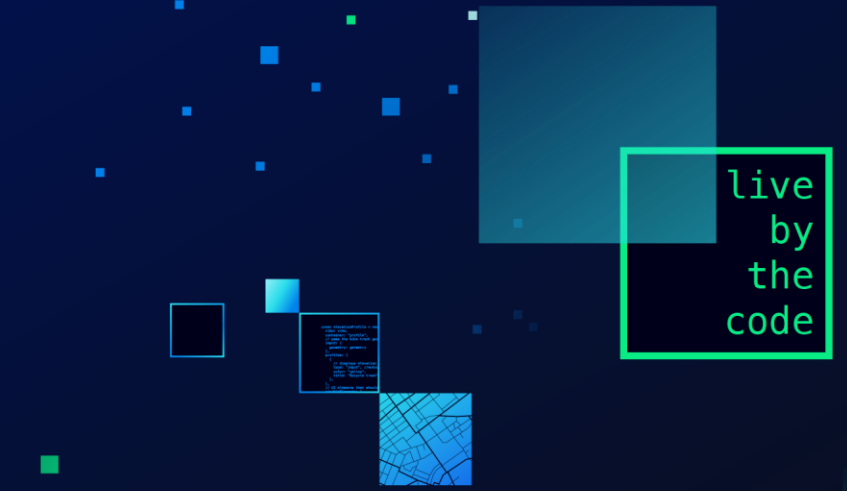
Promises – cancellation

```
let abortController: AbortController = null;

view.on('click', (screenPoint: { x: number, y: number }) => {
  // Cancel previous operation, if any is pending.
  abortController?.abort();
  abortController = new AbortController();
  const { signal } = abortController;

  try {
    const response = await view.hitTest(screenPoint);
    throwIfAborted(signal); // Another click happend, so don't go any further.

    // Pass signal to async operation, which can look at it in order to stop any async work.
    doSomethingAsyncWithGraphic(response.results[0].graphic, signal);
    throwIfAborted(signal);
  } catch (e) {
    throwIfNotAbortError(e); // Only throw for unexpected errors
  }
})
```

```
function throwIfAborted(signal: AbortSignal): void {
  if (signal.aborted) {
    throw new Error("AbortError");
  }
}

function throwIfNotAbortError(error: Error): void {
  if (error?.message !== "AbortError") {
    throw error;
  }
}
```

Widgets

René Rubalcava



Widgets

- Professionally Designed and tested building blocks
- Responsive, Accessible, and Localized
- Out of the box, built for maps-based applications
- Implements Calcite-Components and Design

Widgets

- All follow a similar pattern

```
const widget = new SomeWidget({
  view,
  ...otherOptions
})
```

Widgets

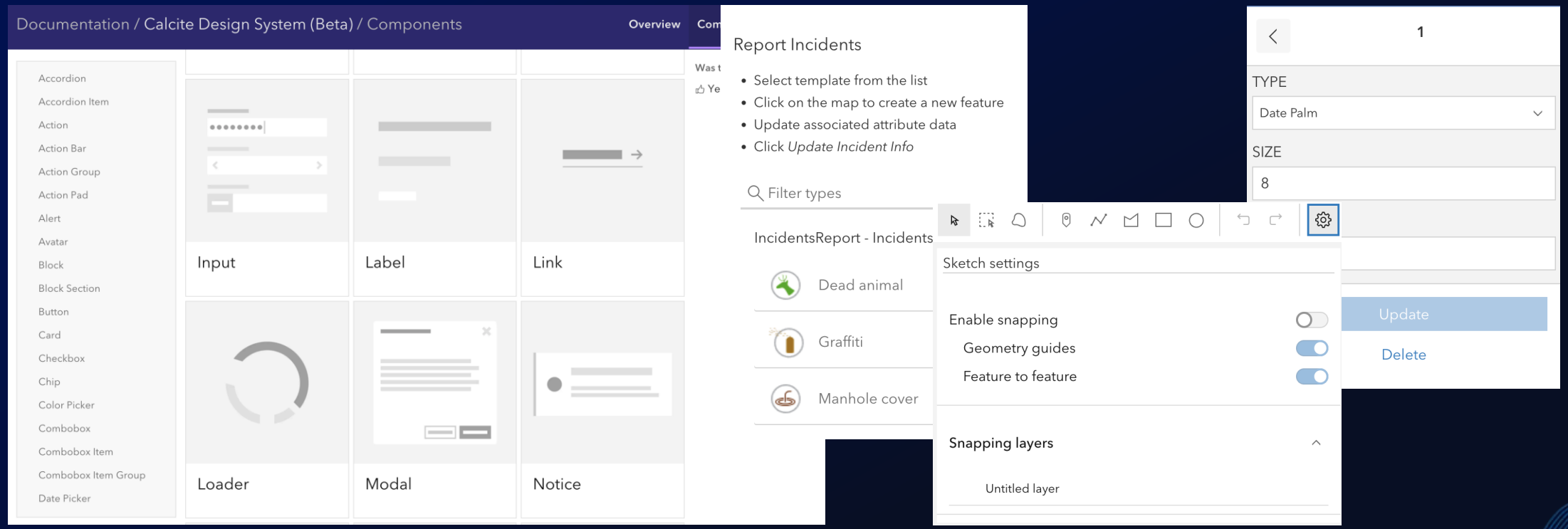
ViewModels

Widgets

ViewModels

Widgets

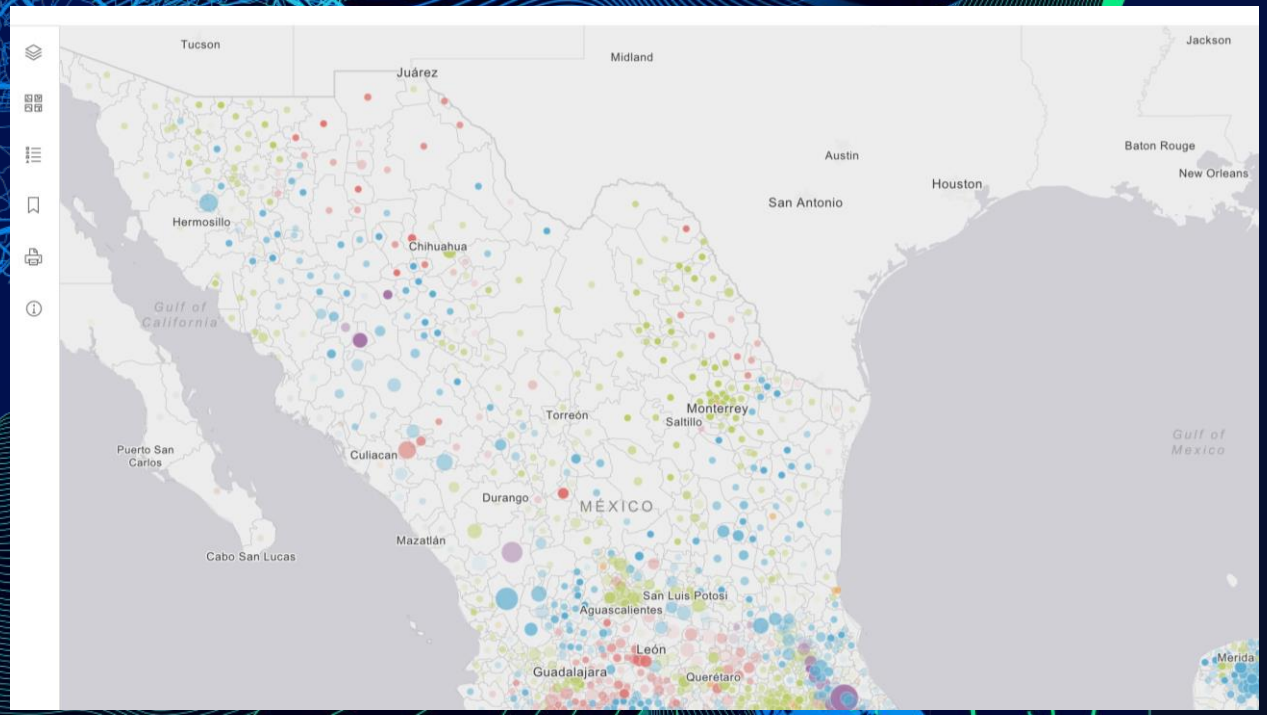
Integrates Calcite Components



Widgets

Calcite Components

- Widgets built with Web Components
- Design System available for you
- Provides consistent user experience



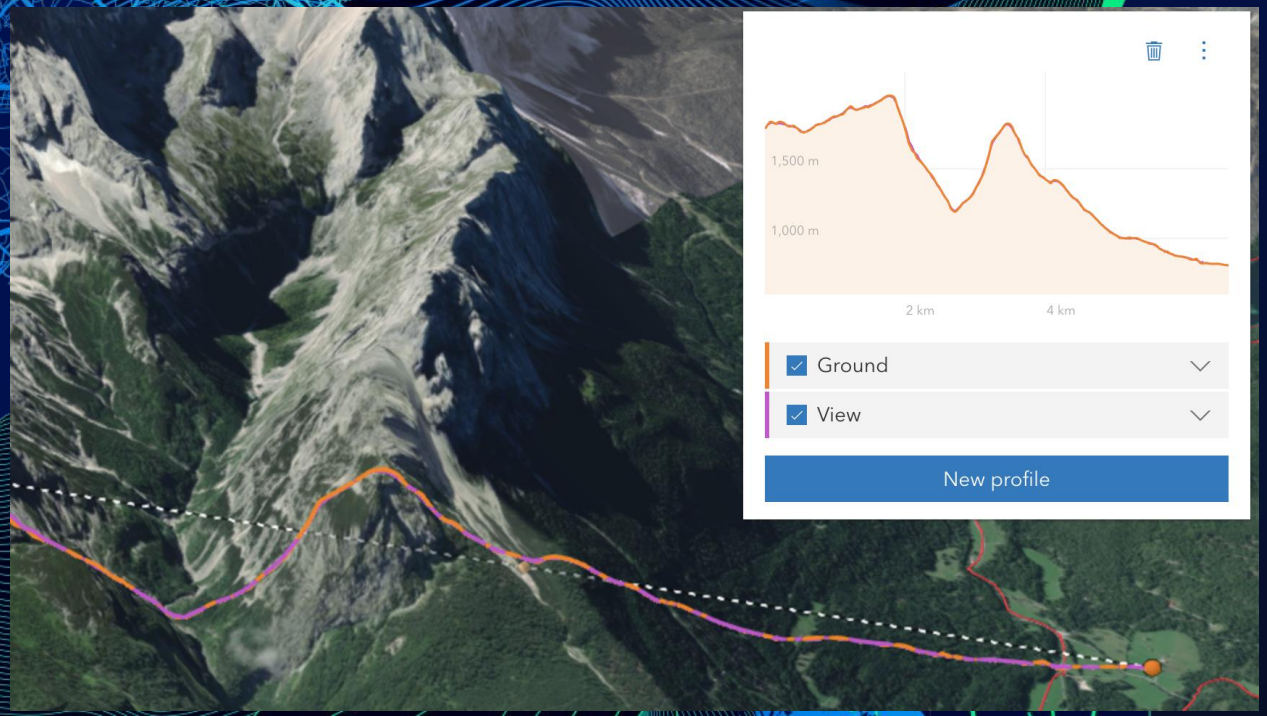
Calcite Components

Creating a mapping app

Widgets

Elevation Profile

- Works in 2D and 3D
- Supports multiple profiles
- Seriously, this type of work was a full day task 15 years ago



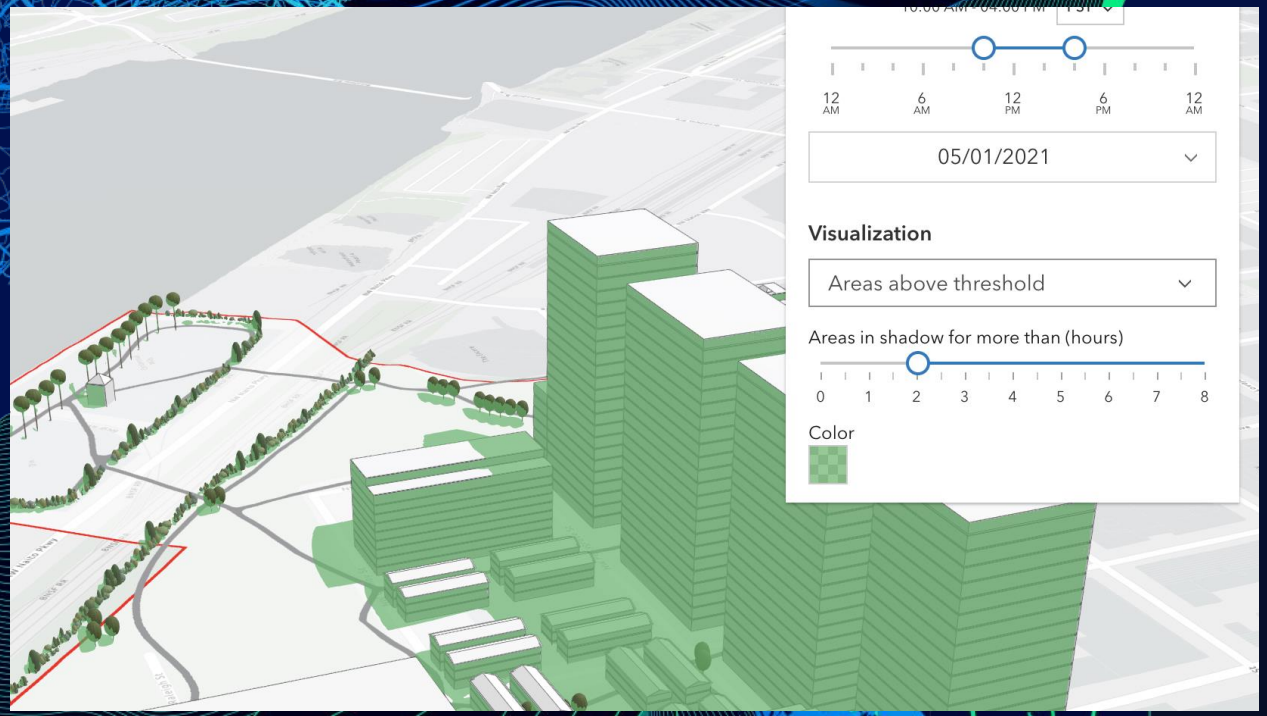
Elevation Profile

Demo

Widgets

Shadow Cast

- Display shadows by day and time
- Combine with the Daylight widget
- Great for construction/redevelopment scenarios



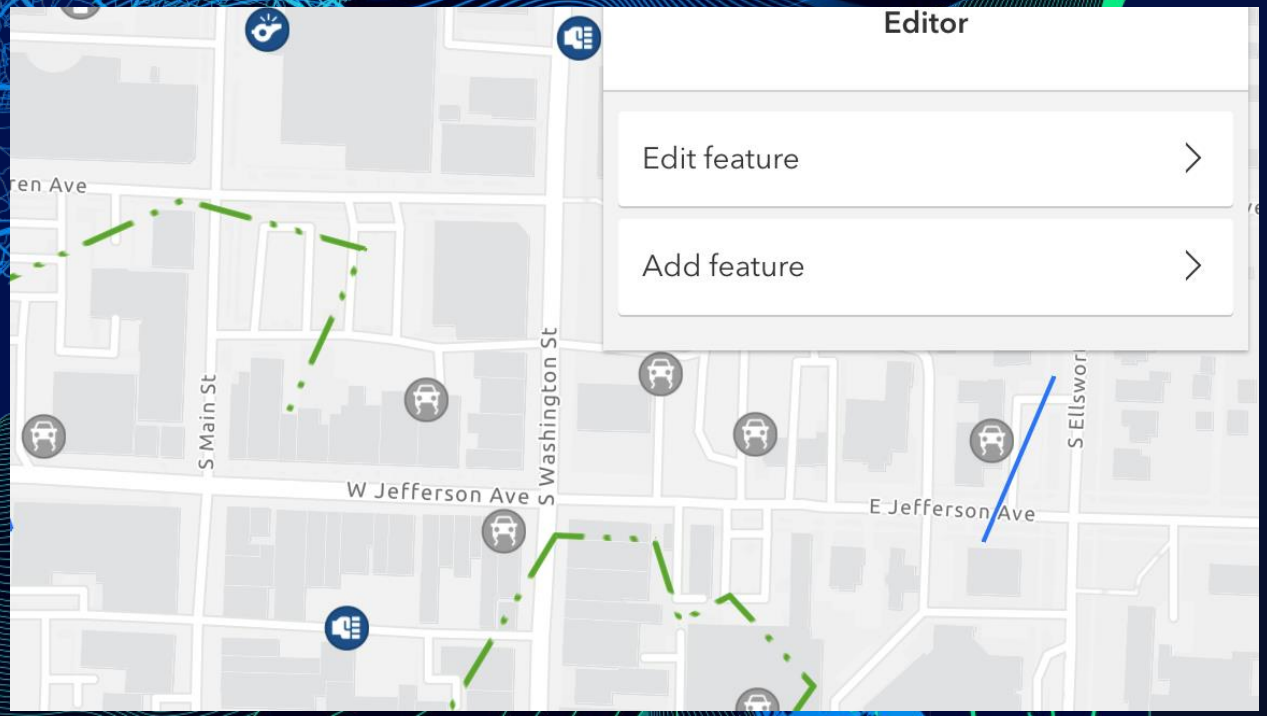
ShadowCast

[Demo]

Widgets

Editing

- Supported in 2D and 3D
- Snapping support
- Custom workflows
- Supported in FeatureTable


Editor
Edit feature
Add feature
S Main St
S Washington St
W Jefferson Ave
E Jefferson Ave

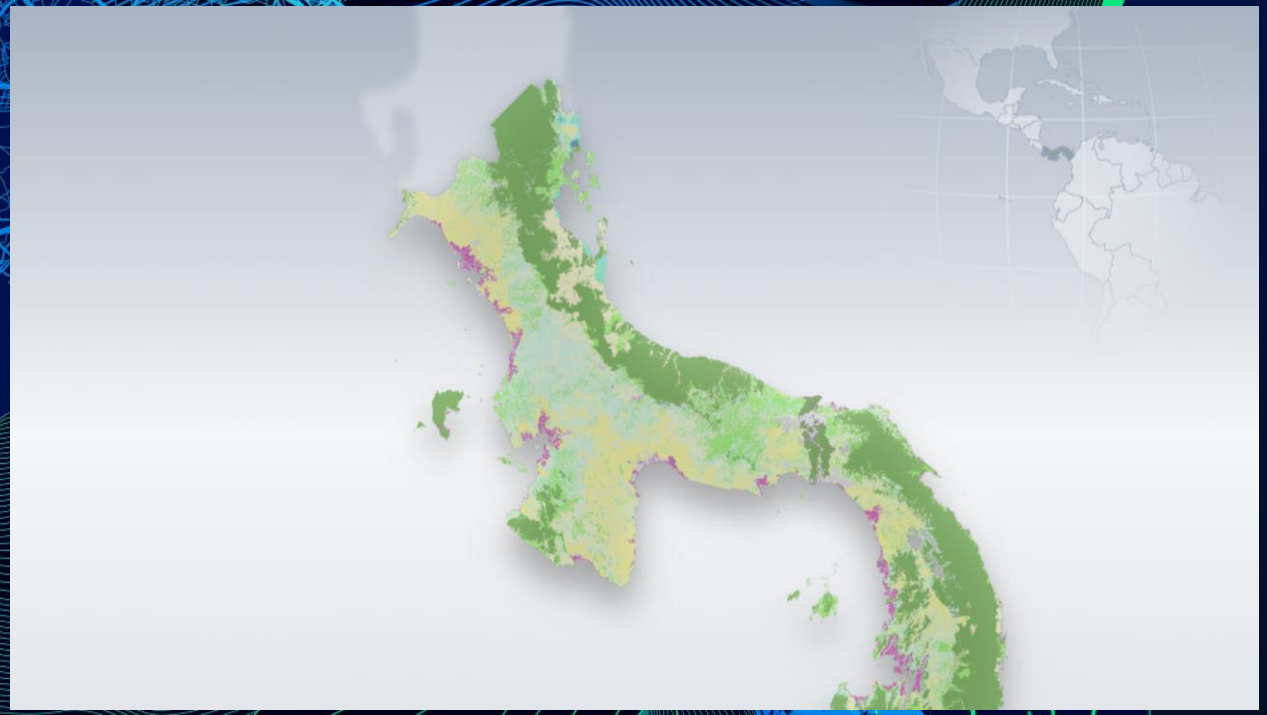
Editor

Demo

View Models

Using Custom Views

- Using View Model methods
- View UI and Interactivity
- MapView and SceneView



Panama Map

Demo

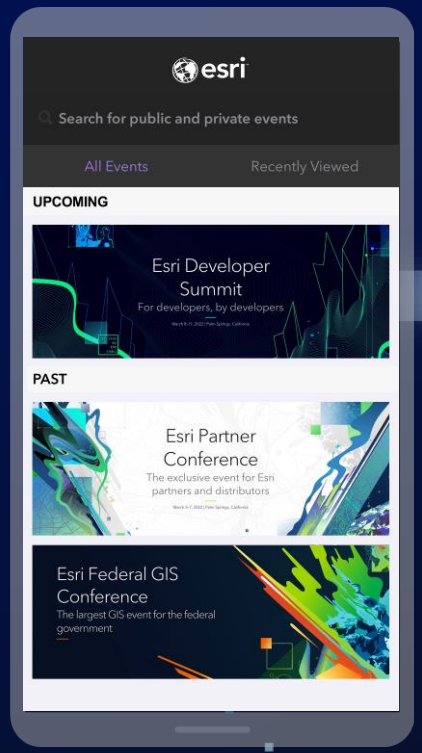
Where can I learn more?

Resources

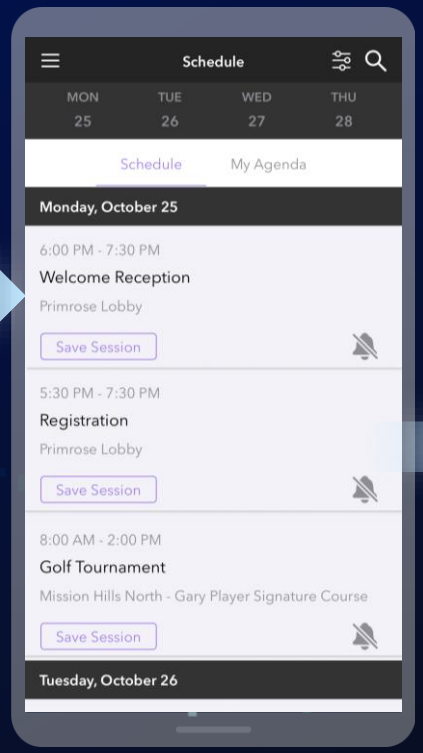
- SDK Documentation
- Programming Patterns Guide
- Calcite Components
- Esri Blogs

Please Share Your Feedback in the App

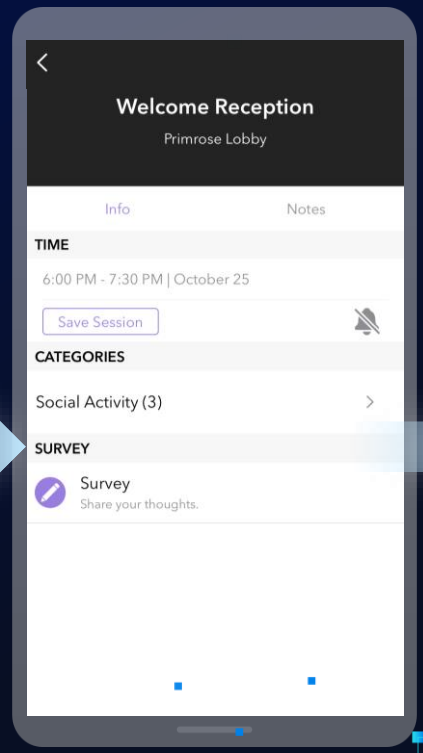
Download the Esri Events app and find your event



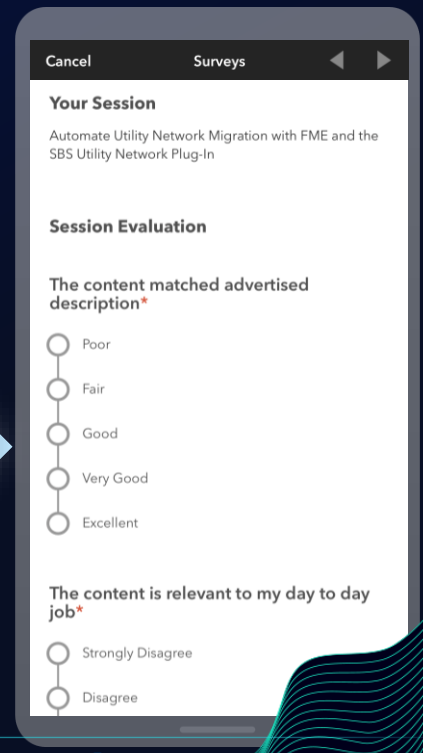
Select the session you attended



Scroll down to “Survey”



Log in to access the survey



Copyright © 2022 Esri. All rights reserved.