

# ArcGIS API for Javascript

## What's new for 4.0

Yann Cabon - Andy Gup

# 4.0

- Overview
- Features
- New Architecture
- Future work

# 4.0 Overview

# In a nutshell

- Simplified API
- `WebMap` and `WebScene`, first class citizens
- Brings 3D capabilities
- APIs and Widgets redesign
- Lot of common patterns shared with Quartz Runtime SDKs

## In a nutshell

- New architecture
- AMD only
- IE9+ for 2D, IE11+ for 3D
- Classes properly cased: `esri/Map`, `esri/Graphic`, `esri/layers/Layer`
- New folder structure.

# Releases

- Already in production in the WebScene Viewer
- beta1 released in mid-july
- beta2 in the next 2 weeks
- Planned: beta3 in december
- Planned: final in 2016 - devsummit

# Features

# SDK

- New SDK built from scratch
  - beta1
  - beta2
- Simpler, focused samples
- User experience
  - Find the relevant information
- More code snippets

# Layers

- `map.layers`, a collection of the operational layers
  - mix of image AND graphics
- Shorter names: `ArcGIStiledLayer`, `ArcGISDynamicLayer`
- new ones:
  - `ArcGISElevationLayer`
  - `SceneLayer`
  - `GroupLayer`

# GroupLayer

- New layer: GroupLayer
- group layers together
- structure your data visualization
- visibility mode: `exclusive`, `independent`, `inherit`
- listMode: `hide-children`, `hidden`
- DEMO

# GroupLayer

```
map = new Map({
  basemap: 'dark-gray',
  layers: [
    new GroupLayer({
      title: 'USA Tiled Services',
      visibilityMode: 'exclusive',
      //listMode: 'hide-children',
      layers: [
        new ArcGIStiledLayer({
          url: '//server.arcgisonline.com/ArcGIS/rest/services/Demographics/USA_MedHInc_Tile/MapServer',
          title: 'Median Household Income',
          visible: false
        }),
        new ArcGIStiledLayer({
          "url": '//services.arcgisonline.com/ArcGIS/rest/services/Demographics/USA_TapestrySegmentation/MapServer',
          "title": "Tapestry Segmentation",
          visible: true
        })
      ]
    })
  ]
});
```

# VectorTileLayer

- vector data stored in tiles
  - binary format
- styling is down clientside
- consumes new VectorTile service
- new client-side layer

```
new VectorTileLayer({  
  url: ...  
});
```

- vector basemaps available through known names
- DEMO

# Basemap

- full fledge class `esri/Basemap`
- basemap's layers are *not* part of the `map.layers`, but from `map.basemap`
- contains 3 Collections: `baseLayers`, `referenceLayers`, `elevationLayers`
- can be set with
  - string for esri's basemap
  - or custom `Basemap` instance
  - in 2D and 3D

# Basemap

- **basemap** as a string, creation of the appropriated Basemap instance

```
var map = new Map({  
  basemap: 'topo'  
});  
  
map.basemap = 'streets';
```

- **basemap** as an instance of **Basemap**

```
var map = new Map({/*...*/});  
  
var toner = new Basemap({  
  baseLayers: [  
    new WebTiledLayer({  
      urlTemplate: '...'  
    })  
  ]  
});  
  
map.basemap = toner;
```

# Padding

- easier fullscreen view application.
- defines inner margin to make space for UI.
- 2D
- 2D
- 3D

# Animation

- generic function `animateTo(target, options):Promise`

```
querytask.execute(query).then(function(result) {  
    view.animateTo(result.features, {  
        duration: 3000  
    });  
});
```

- customize `easing`, `duration`, `chaining`
- DIY using other libs
- `esri/core/Viewpoint`: common way to share between 2D/3D

# UI/UX - Resizing

- DOM isn't our friend
- automatically measure and position the view
- resize by center, or not
- better integration with responsive design pages
- and frameworks

## UI/UX - CSS

- all existing CSSs are revisited for the views and widget.
- decreased class complexity
  - easily customize the tiniest aspects of each component.
- consistent **kebab** case of CSS class names.
- all classes are prefixed by **esri-** to avoid conflicts with CSS frameworks.

# UI/UX - ViewUI

View exposes an API to manipulate the UI elements on top of the layers.

```
view = new MapView( {  
  ui: {  
    components: [ "zoom", "compass", "attribution" ]  
  }  
} );
```

```
// adding UI elements on specific positions  
var ui = view.ui;  
ui.add(search, "top-right");  
ui.add(home, {  
  position: "top-left",  
  index: 0  
});
```

# UI/UX - Widgets

- widgets designed as MVVM
  - one ViewModel, many Views
  - separates the logic from the UI/UX implementation
  - easier to create new views using other frameworks
- ported to 4.0beta: Search, Zoom, Attribution, Popup
- new ones: Compass

D

# UI/UX - Popup

- The Popup in 4 went through a complete redesign.
- The UX was improved in the following areas:
  - It can be docked, through a user action or as an property
  - It is **responsive** and works well in small spaces
  - Allows for easy navigation of multiple features.
  - beta3? includes new large scale view of images and charts to better communicate key information.
- On the developer side:
  - Easily style the popup to match your app
  - **Custom actions** can be configured to expose extra capabilities

# Portal, WebMap, WebScene

- are central classes of the API.
- replace the 3.x `esri/arcgis/utils`
- easy to access, modify, and persist back to portal.

```
var portal = new Portal({
  authMode: 'immediate', // anonymous | auto | immediate
  canSearchPublic: true,
  url: '//jsapi.maps.arcgis.com'
});

portal.load()
  .then(function() {
    return portal.queryItems({
      query: 'Brest'
    })
  })
  .then(function() {
    displayResults(...)
  });

```

# WebMap and WebScene

- Map holds
  - basemap
  - operational layers
  - initial state
- WebMap, WebScene hold
  - portal item
  - presentations
  - environment options
- Scene can be global or local

```
var scene = new WebScene({
  portalItem: new PortalItem({
    id: 'affa021c51944b5694132b2d61fe1057'
  })
});

var view = new SceneView({
  map: scene,
  container: 'viewDiv'
});
```

loading a webscene  
scene local  
scene slides

## Plans

- beta2:
  - new Portal API
  - reading WebScene
- beta3:
  - reading WebMap
  - saving
  - create a layer from a item id.

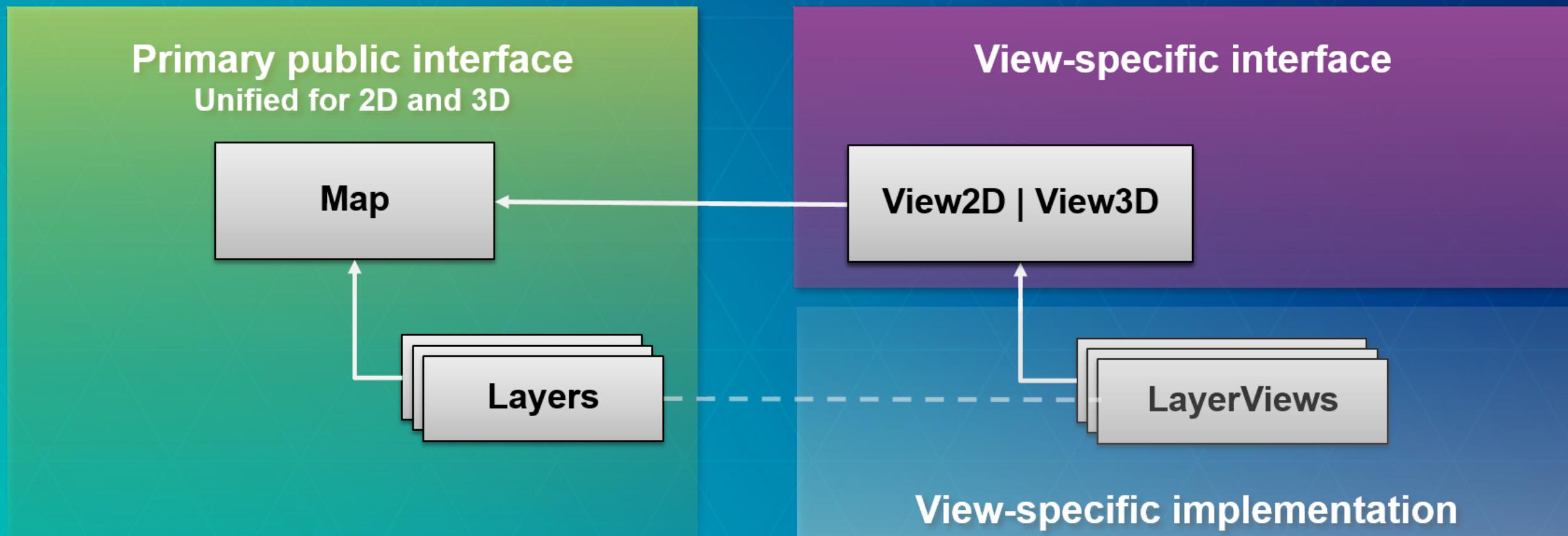
# Architecture

# Why a new architecture?

- One of the starting point of 4.0: bring 3D
- currently in 3.x:
  - Map, many DOM nodes
  - Each Layer, 1 DOM Node
- rendering 3D with WebGL which is completely different

# New architecture

- Separate the business logic from the drawing logic.



```
var map = new Map( {  
    basemap: 'topo',  
    layers: [  
        new ArcGISDynamicLayer(...)  
    ]  
} );  
  
var mapView = new MapView({  
    map: map,  
    container: 'mapDiv'  
} );  
  
var sceneView = new SceneView({  
    map: map,  
    container: 'sceneDiv'  
} );
```

side by side views

# 2D/3D

- For the rest, one API
- DEMO

# 2D

- new "engine" in the work.
- faster, more future proof
  - abstraction to draw tiles and dynamic images to ease custom layers/layerviews
  - abstraction to draw in DOM or Canvas, possibly WebGL
- display graphics while zooming.
- rotation
- continuous zoom
- vector map tiles, basemaps

# 3D

- WebGL engine to display the globe.
- z/m support in the API, tasks, layers...
- support for simple symbols
- new 3D Symbols

# Architecture

## new core components

- esri/core/Accessor
- esri/core/Promise
- esri/core/Loadable
- esri/core/Collection

# **esri/core/Accessor**

- Mixin similar to [dojo/Stateful](#)
- Consistent pattern:
  - getting and setting properties value
  - watching properties change
- Single object constructor
- Computed properties
- Autocast

# Properties watching

- Direct benefits:
  - remove inconsistancies between constructor, getter, setter functions, events
  - one convention everywhere. "*just need to know what properties for a class*"
  - Single object constructor, no more 3+ constructors
  - Leaner SDK: we doc only the properties, the rest is convention
- Changes:
  - no more *property*-change events, use `watch()`
  - in 3.x, listen for `extent-change` event.
  - in 4.0 `extent` watchers will be call very often
  - new events and properties for animation.

# Accessor - Properties watching

```
var map = new Map(...);
var view = new MapView({ map: map });

// watch for viewpoint updates
view.watch('viewpoint', function(newValue, oldValue, property, target) {
  console.log(newValue, oldValue, property, target);
})

// chain watching
map.watch('basemap.title', function(value) {
  console.log(value);
});
map.basemap = 'topo';
```

# Accessor - Properties watching

- Frameworks integration
  - properties are framework agnostic
  - better/easier integration
- Examples
  - dbind
  - React
  - camera recorder

# Accessor - Single object constructor

```
require([
  'esri/Map',
  'esri/Basemap',
  'esri/core/Collection',
  'esri/layers/ArcGISTiledLayer'
],
function(
  Map,
  Basemap,
  Collection,
  ArcGISTiledLayer
) {
  var map = new Map({
    basemap: new Basemap({
      baseLayers: new Collection([
        new ArcGISTiledLayer(url)
      ])
    })
  });
});
```

# Accessor - Autocast

```
require([
  'esri/Map',
  'esri/layers/ArcGISTiledLayer'
],
function(
  Map,
  ArcGISTiledLayer
) {
  var map = new Map({
    basemap: {
      baseLayers: [
        new ArcGISTiledLayer(url)
      ]
    },
    initialExtent: {
      xmin: -180, xmax: 180,
      ymin: -80, ymax: -80,
      spatialReference: { wkid: 4326 }
    }
  });
});
```

# Accessor - Autocast

```
// 3.x
new SimpleMarkerSymbol(SimpleMarkerSymbol.STYLE_SQUARE, 10,
  new SimpleLineSymbol(SimpleLineSymbol.STYLE_SOLID,
    new Color([255,0,0]), 4),
  new Color([255,255,255,0.25]));

// 4.0
new SimpleMarkerSymbol({
  style: 'square',
  color: 'red',
  size: 10,

  outline: {
    color: 'rgba(255, 255, 255, 0.5)'
    width: 4
  }
});
```

# Promises

- 4.0 is a promises-based architecture!
- The basic pattern looks like this:

```
someAsyncFunction().then(callback, errback);
```

- And, it's implemented like this:

```
someAsyncFunction().then(  
  function(resolvedVal){  
    //This is called when the promise resolves  
    console.log(resolvedVal); //logs the value the promise resolves to  
  },  
  function(error){  
    //This function is called when the promise is rejected  
    console.error(error); //logs the error message  
  }  
) ;
```

- Getting Started with Dojo
- all async fonctions and methods are returning to **Promise**. no more **events**

# Promises

- Classes may be Promise
  - because they might have to load resources
  - because they might be asynchronously initialized

Map, Layer, View

```
var map = new Map({...})  
  
view = new SceneView({  
  map: map,  
  //...  
});  
  
view.then(function() {  
  // the view is ready to go  
});
```

# Loadable - 4.0beta2

- brings better control, and scheduling of loading resources.
- extension of `esri/core/Promise`
- in 3.x, instanciating a layer loads it. in 4.0, it's an explicit call
- the views automatically loads the map and its layers
- use cases:
  - `WebMap` / `WebScene` / `Basemap` need to load:
    - the portal item
    - the layer modules
    - the layer portal items
  - `MapView` / `SceneView` need to load:
    - the map
    - the layers
  - `BasemapGallery` need to load
    - basemaps from the portal, not the layers

# In a single page application, get a feature from a FeatureLayer from a WebMap without displaying it, ASAP!

```
var webmap = new WebMap({
  portalItem: {
    id: 'affa021c51944b5694132b2d61fe1057'
  }
});

webmap.load()
  .then(function() {
    return webmap.getLayer('myFeatureLayerId').load();
  })
  .then(function(featureLayer) {
    return featureLayer.queryFeatures({
      where: 'OBJECTID = 1'
    });
  })
  .then(function(result) {
    displayDetails(result.features[0]);
  })
  .otherwise(function(error) {
    console.error(error);
  });
}
```

# Collection

- More or less like an Array
- `add` / `remove` / `forEach` / `map` / `find` / `findIndex`...
- emit "change" events when something is added/removed/moved
- used for layers, used for layers in Basemap, used for graphics...

# Future work

# beta3

- more Portal APIs
  - saving items
- loading layers from portal items
- WebMap
  - reading
- WebScene
  - writing

# Conclusion

- One API
- 3D, and better 2D
- New concepts
  - to observe properties with Accessor
  - to handle asynchronous operations with Promise
- Simplified and consistent API

# Questions

@yanncabon  
ycabon@esri.com

@agup  
agup@esri.com



esri®

BERLIN

Prenzlauer Berg

Fennpfuhl

Friedrichshain

Rummelsburg