Developing Imagery Apps Using the ArcGIS API for JavaScript and WebApp Builder

Naila Khan, Chayanika Khatua
What are we talking about today…

- Custom applications
- Map Viewer
- Web AppBuilder
- ArcGIS API for JavaScript
Image Services

- **Data:**
  - shared through ArcGIS Server
  - accessed by any device that supports connecting to a web service
  - Single image or a collection of images

- **Visualization:**
  - Fast dynamic display

- **Processing**
  - Well-known server-side raster functions
  - Register raster models with image service
  - Client-side processing
ImageServices

Raster Dataset
- Format
- Interpolation
- Rendering Rule
- GetRasterAttributeTable
- GetKeyProperties

Mosaic Dataset (MD)
- MosaicRule
- QueryVisibleRasters

MultiDimensional MD
- MultiDimensional Info
- Dimensional Definition
ArcGIS API for JavaScript
4.x
Overview

- Brand new API to visualize 2D and 3D
- New architecture
- New development pattern for widgets
- New Portal API to provide better integration with WebMap
- AMD only
- Modern browser support: IE11+
Map and View architecture

View
- Renders and interacts with the model
  - MapView
  - SceneView

Model
- Describes the content of the map/scene
  - Map
  - Layers
ImageryLayer

- Renders image data
- Retrieves data as pixels
- Supports LERC, BIL, BSQ, JPG, PNG formats
- View Popups
- Query
- Load webmap
- Apply client side processing using PixelFilters
- Define server side processing using raster functions
- Multi-dimensional API
- Supports 2D & 3D visualization
```javascript
var layer = new ImageryLayer({
    url: "https://myServer.arcgisonline.com/arcgis/rest/services/NLCDLandCover/ImageServer"
});

/******************************
* Add image layer to map
******************************

var map = new Map({
    basemap: "gray",
    layers: [layer]
});

var view = new MapView({
    container: "viewDiv",
    map: map,
    center: [-100, 40],
    zoom: 5
});
```
Imagery layer in 2D and 3D
ImageryLayer: Display

MosaicRule defines how overlapping images are mosaicked together [esri/layers/support/MosaicRule]

• Selection using the where clause
• Method: none | center | nadir | viewpoint | attribute | lock-raster | northwest | seamline
• Sort using sort field and value

```javascript
// Define the way overlapping images are mosaicked together
var mosaicRule = new MosaicRule({
  ascending: true,
  method: "center",
  operation: "last"
});
```
**ImageryLayer: Multidimensional Data**
Filters data based on slices or ranges in one or more dimensions [esri/layers/support/DimensionalDefinition]

- Variables can be temperature, humidity or wind speed
- Dimensions can be time and depth

```javascript
var dimInfo = []; // Define dimensional definition as array
// DEPTH: show only temperatures at sea surface
dimInfo.push(new DimensionalDefinition({
    variableName: "water_temp",
    dimensionName: "StdZ", // Water depth
    values: [0], // Sea surface or 0ft
    isslice: true
}));
// TIME: only show temperatures for the week of April 7, 2014
dimInfo.push(new DimensionalDefinition({
    variableName: "water_temp",
    dimensionName: "StdTIme", // time temp was recorded
    values: [1396828800000], // Week of April 7, 2014
    isslice: true
}));

var mr = new MosaicRule({
    multidimensionalDefinition: dimInfo
});
```
ImageryLayer: Processing

RenderingRule defines how an image should be processed [esri/layers/support/RasterFunction]

- Defines how the image is rendered
- Single function or a chain of functions

```javascript
// Defines a Remap raster function. Remap reclassifies pixel
// values to new values. In this case we want to separate
// two landcover types: forested areas and non-forested areas

var remapRF = new RasterFunction({
  functionName: "Remap",
  functionArguments: {
    inputRanges: [0, 41, 41, 44, 44, 255],
    outputValues: [1, 2, 1],
    // $$((default)) refers to the entire image service,
    // $id refers to a specific image in the image service
    raster: "$$"
  }
});

var layer = new ImageryLayer({
  // apply the defined raster function
  renderingRule: remapRF
});
```
ImageryLayer: Processing

RenderingRule defines how an image should be processed [esri/layers/support/RasterFunction]
Change detection and histogram using raster functions
ImageryLayer: Processing
PixelFilter manipulates pixels of an image for display or analytics

- Process on the client-side
- Defined using the pixelFilter property of ImageryLayer

```javascript
function maskPixels(pixelData) {
  if (pixelData === null || pixelData.pixelBlock === null ||
      pixelData.pixelBlock.pixels === null) {
    return;
  }

  // The pixelBlock stores the values of all pixels visible in the view
  var pixelBlock = pixelData.pixelBlock;
  var pixels = pixelBlock.pixels;
  var band1 = pixels[0];

  // The mask is an array that determines which pixels are visible to the client
  var mask = pixelBlock.mask;
  var numPixels = pixelBlock.width * pixelBlock.height;

  for (var i = 0; i < numPixels; i++) {
    mask[i] = (band1[i] >= Math.floor(currentMin)) &&
              (band1[i] <= Math.floor(currentMax)) ? 1 : 0;
  }

  var layer = new ImageryLayer({
    url: url,
    pixelFilter: maskPixels,
  });
```

ImageryLayer: PixelBlock

Represents pixels in the view: esri/layers/support/PixelBlock

- Used for accessing pixels
- Decodes the returned data
- Supports LERC, JPG, PNG, BIL, BSQ format
  - Width: number of columns
  - Height: number of rows
  - Pixels: two dimensional array
  - PixelType: s8 | s16 | s32 | u8 | u16 | u32 | f32 | f64
  - Mask: array of 0 or 1
  - Statistics: array of objects containing the min and max values
Pixel Filter
Pie-Chart
ArcGIS API for JavaScript

3.x
Imagery Layers
JS API 3.x

- Renders image data
- Retrieves data as pixels (PNG, JPEG, TIFF, LERC etc.)
- Processing using raster functions
- Processing using pixel filters

...
ArcGISImageServiceLayer and RasterLayer

- Render raster data
- Supports:
  - format, compression, mosaicRule, renderinRule, raster functions, multidimensional API
- Only Raster layer supports pixelFilters
- Define renderers using `setRenderer()`:
  - Unique Value, class breaks, stretch

```javascript
// Add five breaks to the renderer.
// If you have ESRI's ArcMap available, this can be a good way to determine break values.
// You can also copy the RGB values from the color schemes ArcMap applies, or use colors
// from a site like www.colorbrewer.org
//
// Alternatively, ArcGIS Server's generate renderer task could be used
var renderer = new ClassBreaksRenderer(symbol, "POP07_SQM1");
renderer.addBreak(0, 25, new SimpleFillSymbol().setColor(new Color([56, 168, 0, 0.5])));
renderer.addBreak(25, 75, new SimpleFillSymbol().setColor(new Color([139, 209, 0, 0.5])));
renderer.addBreak(75, 175, new SimpleFillSymbol().setColor(new Color([255, 194, 0, 0.5])));
renderer.addBreak(175, 488, new SimpleFillSymbol().setColor(new Color([255, 255, 0, 0.5])));
renderer.addBreak(488, Infinity, new SimpleFillSymbol().setColor(new Color([255, 0, 0.5])));
```
ArcGISImageServiceVectorLayer: Symbology

- Visualize wind, Ocean current
- Draws raster data as vectors using the VectorFieldRenderer:
  - *Single Arrow, Simple Scalar, Wind Barbs, Beaufort Wind, Ocean Current*
- Default pixelFilter for conversion of U-V to Magnitude-Direction

```javascript
var layer = new ArcGISImageServiceVectorLayer(url, {
  imageServiceParameters: params,
  symbolTileSize: 60,
  rendererStyle: "single_arrow"
});
```
WCS Layer

provides access to coverage data in forms that are useful for client-side rendering, as input into scientific models, and for other clients

- Enable Web Coverage Service on image services
- Get server info: coverages, formats, interpolations, versions, etc.
- Support for identify
- Client side pixel filters too!
- Fully documented SDK

```javascript
var wcsUrl = "//sampleserver6.arcgisonline.com/arcgis/services/ScientificData/MODIS_Landcover/ImageServer/WCSServer";
var wcsLayer = new WCSLayer(wcsUrl, {
  version: "1.0.0",
  pixelFilter: colorizer
});
```
Documentation Resources

- ArcGIS API for JavaScript:
  https://developers.arcgis.com/javascript/3/

- ArcGIS API for JavaScript 4.x
  https://developers.arcgis.com/javascript
What are we talking about today…

Custom applications

Map Viewer

Web AppBuilder

ArcGIS API for JavaScript
Map Viewer capabilities

- **Symbology**
  - Stretch, Classify and Unique Value renderers
  - Leverages the rendering capability added in the ArcGIS API for JavaScript 3.x
  - Through the image display pane

- **Image Processing**
  - Dynamic processing (on the fly): Server processing templates
  - Persisted output: Raster Analysis (Enterprise only)
Raster Analysis

- Quickly extract information using spatial analysis and raster models by leveraging distributed processing and storage
- Optimized raster format: CRF (Cloud Raster Format)
  - Multi-band, block based, multiple readers and writers, fast
- Can be accessed by different clients (Pro and Map viewer) and have a REST API to access services
- Once finished, results immediately available in your Web GIS
Raster Analysis using ArcGIS Enterprise map viewer

• 11 out of the box tools added in 10.5
• Ability to share and apply custom processing models (RFT) added in 10.6
• Build and edit custom templates, later this year
Custom Analysis Tools

Select a custom analysis tool.

BurnScar

Added at 10.6
Upcoming: Raster Function Editor in Map Viewer

- View, build and edit raster function templates in Portal.
- Save and share new or updated templates.
- Run it as an analysis operation to produce a persisted output.
- Scheduled for release later this year.
Image Services in
Web AppBuilder for ArcGIS
Web AppBuilder for ArcGIS

- Quickly author customized apps, no coding required
- Fully integrated with the ArcGIS platform
- Extensible framework with the developer edition
- Built using the ArcGIS API for JavaScript
- Runs in a browser on tablets, smartphones or desktops
Imagery support in Web AppBuilder for ArcGIS

- Layer List
- Legend
- Query
- Attribute Table
- Popups
- Image Measurement*
- Oblique viewer*

* Specifically designed to work with image services
Oblique Viewer

- Displays oblique images without distortion
- Provides a natural view from the camera location
- View features from different perspectives
- Available through the ArcGIS API for JavaScript 3.x and as a Web AppBuilder widget

```javascript
oblique = new ObliqueViewer({
  map: map,
  imageServiceLayer: imageServiceLayer,
  rotationDiv: document.getElementById("obliqueRotationDiv")
});
```
Oblique Viewer

Web AppBuilder
ArcGIS API for JavaScript

http://esriurl.com/ObliqueViewer
Custom Imagery Widgets

http://landsatexplorer.esri.com/
Web AppBuilder widgets for Image Services (WABIS)

- Widgets specifically designed for imagery workflows.
- Change detection, image selection, mask.
- Process imagery, export and share your results!
- Web AppBuilder widgets
- Imagery Interpretation template (beta)

https://github.com/Esri/WAB-Image-Services-Widgets
Summary

- **ArcGIS API for JavaScript:**
  - Different layers in 4.x and 3.x for displaying raster data
  - How to manipulate display using mosaicRule and renderers
  - Server-side and Client-side processing using raster functions and pixel filters
  - Display vectors using ArcGISImageServiceVector layer
  - Visualize raster data using WCS layer

- **Map viewer:**
  - Define symbology and process data using the power of Raster analytics

- **Web AppBuilder:**
  - Built-in and custom imagery widgets
Please Take Our Survey on the Esri Events App!

Download the Esri Events app and find your event

Select the session you attended

Scroll down to find the survey

Complete Answers and Select “Submit”

Download the Esri Events app and find your event

Select the session you attended

Scroll down to find the survey

Complete Answers and Select “Submit”