Oriented Imagery API

User Documentation | July 26, 2019

esri® | THE SCIENCE OF WHERE™

This document applies to Oriented Imagery, currently provided as a prototype and for testing only. The functionality has not been exhaustively tested and is not currently covered under ArcGIS Support. Please address questions or suggestions related to this workflow to the Oriented Imagery GitHub repo or to ImageManagementWorkflows@esri.com

Contents

Overview

The Oriented Imagery API lets you build and customize Oriented Imagery applications.

Oriented Imagery from Esri is a collection of tools that provide a solution for managing, visualizing, and exploring imagery that's taken from any angle. Visit Imagery Workflows to learn more about Oriented Imagery.

The API requires that imagery is first managed as an oriented imagery catalog (OIC). The API can be used to find and display all the images in an OIC that contain the asset being inspected. It also generates coverage polygons of images in an OIC, which can be displayed on the map.

The API is built using JavaScript, WebGL, HTML5 and CSS3. It uses external libraries, including the ArcGIS API for JavaScript 4.10 and Pannellum.

Types of images supported

The following types of imagery are supported by Oriented Imagery:
1. Streetview images (where the camera pitch is 90 degrees, towards the horizon)
2. Wide-angle images (where the horizontal field of view (HFOV) is less than 180 degrees)
3. Panorama images
4. Oblique or drone images
5. Inspection images (taken at close-range, where the camera pitch can be greater than 90 degrees)
6. Video
7. Bubble images, including the following:
    a. A single, stitched image
    b. Six separate images stitched together
    c. Multiple images taken from the same location pointing in different directions, but considered a group

Image and video formats supported

The following file formats are compatible with Oriented Imagery:

- JPEG
- PNG
- MRF (using either LERC or JPEG compression)
- LERC
- MP4 (Codec – h.264)

What's new in the Oriented Imagery API 2.0

The following enhancements and bug fixes were made in the 2.0 release.

3D support

- A new mode has been added to the API to display the image in the mesh instead of canvas. There are now two modes:
    - 2D mode (The image is drawn on the canvas)
    - 3D mode (The image is drawn in mesh, which can be added to the web scene)
- The API now passes frustums along with polygons, which can be displayed in web scenes.

API enhancements

- The API now passes a Camera object (heading, pitch, field of view, position) which can be used to set the camera position in a web scene.
- The best image computation has been updated. The best image is selected based on suitability, which is determined from distance, heading, and pitch. Weights are assigned for each three parameters and changed according to different modes.
- Graphics in panorama images are now drawn on the graphic canvas instead of added as svg elements.
- The user can now create multiple viewers (multiple instances of the viewer).
- The API now supports setting the client type to web or pro.
- Fixed an existing issue with coverage polygons and ground to image transformation if CamPitch + VFOV/2 > 90 and CamRoll is not 0.

Styling

- API styling updated to match Calcite.
- The API now supports light and dark theme for both web and pro clients.
- New tool icons for the viewer in the web client were added.

Enhancements to the viewer

- The focus point in the viewer can now be hidden, either programmatically or by clicking the red cross in the navigation tool.
- Editing capability in the viewer has been added. A user can now do the following with vectors in the viewer:
    - Add vector layers as points (circle, cross, x, diamond, triangle, square), polylines or polygons and overlay them on top of the image.

- o Add new features or delete existing records using the editing tool in the viewer.
- o Select features by clicking on them in the viewer.
- o Open an image by selecting features on the map.
- o View a vector only in the image in which it was recorded.
- The viewer supports vector layers with multiple renderers. Support has been added for feature services with two types of renderers—simple and uniqueValue.

Oriented Imagery API

Overview

The API relies on the following:

viewer
create a new oriented imagery viewer
Parameters
- container (string | HTMLElement) The container (div) element for the viewer, or its ID.

setTheme
Set the theme of the viewer. Themes available – light and dark
Parameters
- theme (string) Valid values are light and dark.

setPlatform
Set the application platform or environment.
Parameters
- platform (string) Valid values are web and pro.

on
Subscribe listener to specified event.
Parameters
- type (string) Type of event to subscribe to
- listener (Function) Listener function to subscribe to event.

Events
- load

  Fired when the api is loaded and initialized.
  Returns Boolean

Viewer functions and events

Functions

searchImages
Search images that contains the selected asset.
Parameters
- point (JSON) Selected Point; format is similar to Point JSON object of JavaScript API for ArcGIS; example –

```
{"x": -122.65, "y": 45.53, "spatialReference": {"wkid": 4326}}
```

- url (string) Url of oriented imagery catalog
- properties (JSON) Optional. A JSON object with following parameters: -
  - token (JSON) Optional. Pass token object so the user does not have to log in at runtime; example –

```
{"token":"TrSeUNJ0_lnAzTF5Rl8JGoX9sNspfcUo81XovhSikQcyFB58h87AKkDW4rowWSLLVrUcqQN9
H_m5fUJ40doBPhkI3x0PKSKLxXaKXtxtBxai0a25U3llY7GB9YSxRciSjV5dynK2nqcdVpngpxr_Wvv1dqE
k4XEiuGXWFfwe2Rw9oZ9KSeUfYO4wd0jWtt0q2bkiZ6y4zm3FfrARnjY8PpfisPFQoiHpC0aAVNml6lti_
C2f7YicJuyRQhi2nUKb","expires":1547465355,"ssl":
false,"userid":"V_B","server":"www.arcgis.com"}
```

  - maxDistance (integer) Optional. Search distance in meters.
  - camera (JSON) Optional. Camera position and orientation.

```
{
  "heading": 55,
  "tilt": 35,
  "fov": 118,
  "elevation": 300, // ground elevation
  "position": {"x": -122.65, "y": 45.53,
                "z": 45, // height above ground
                "spatialReference": {"wkid": 4326}}
}
```

  - objectId (number) Optional. Open a specific image. Pass the OBJECTID of the feature in the feature class.
  - extent (JSON) Optional. Extent of the map.

```
{"xmin": -122.65, "ymin": 45.53, "xmax": -121.65, "ymax": 46.53, "spatialReference": {"wkid": 4326}}
```

  - mapSize (JSON) Optional. Container size of map or scene.

```
{
"token": {
            "token":"TrSeUNJ0_lnAzTF5Rl8JGoX9sNspfcUo81XovhSikQcyFB5",
        "expires":1547465355,
            "ssl": false,
            "userid":"V_B",
            "server":"www.arcgis.com"
        },
    "maxDistance": 1000,
```

```
"camera": {
            "heading": 55,
            "tilt": 35,
            "fov": 118,
            "elevation": 120,
            "position": {"x": -122.65, "y": 45.53, "z": 45,
                            "spatialReference": {"wkid": 4326}}
        },
"objectId": 2334,
"mapSize": {w: 1080, h: 760},
"extent": {
            "xmin": -122.65, "ymin": 45.53, "xmax": -121.65, "ymax": 46.53,
             "spatialReference": {"wkid": 4326}}
            }
```

displayGroundPointInImage

Converts ground point to image point and displays it in viewer.

Parameters

- point (JSON) Ground Point; format is similar to Point JSON object of JavaScript API for ArcGIS; example –

```
{"x": -122.65, "y": 45.53, "spatialReference": {"wkid": 4326}}
```

groundToImage

Returns image points for the corresponding ground points.

Parameters

- points (Array [JSON Point]) Ground Points; format is similar to Point JSON object of JavaScript API for ArcGIS; example –

```
[{"x": -122.65, "y": 45.53, "spatialReference": {"wkid": 4326}},
{"x": -132.58, "y": 12.54, "spatialReference": {"wkid": 4326}}
]
```

Returns (Array[JSON Point]) Image Points

Example

```
        groundToImage([{"x": -122.65, "y": 45.53, "spatialReference": {"wkid":
4326}}]).then(function(imagePoint) {
});
```

imageToGround

Returns ground points for the corresponding image points.

Parameters

- points (Array [JSON Point]) Image Points; format is similar to Point JSON object of JavaScript API for ArcGIS; example –

```
[{"x": 3000, "y": 100}, {"x": 212, "y": 320}]
```

Returns (Array[JSON Point]) Ground Points
Example

```
    imageToGround([{"x": 100, "y": 420}]).then(function(groundPoint) {
});
```

resize
resize image (Should be called after the resizing the container)

on
Subscribe listener to specified event.
Parameters
- type (string) Type of event to subscribe to
- listener (Function) Listener function to subscribe to event.

off
Remove an event listener (or listeners).
Parameters
- type (string) Type of event to remove listeners from. If not specified, all listeners are removed.
- listener (Function) Listener function to remove. If not specified, all listeners of specified type are removed.

destroy
destroy the viewer.

getCoverageMap
Get the coverage map for the input extent. Maximum camera source points selected are 100.
Parameters
- extent (JSON) Extent; format is similar to Extent JSON object of JavaScript API for ArcGIS; example –

```
{
   "xmin":-122.68,"ymin":45.53,"xmax":-122.45,"ymax":45.6,
   "spatialReference":{"wkid":4326}
}
```

- url (string) Url of oriented imagery catalog

Returns Object {
    coveragePolygon: Object {Polygon JSON Object}; example –

- {"rings":[[[-122.63,45.52],[-122.57,45.53],[-122.52,45.50],[-122.49,45.48],
    [-122.64,45.49],[-122.63,45.52],[-122.63,45.52]]],"spatialReference":{"wkid":4326 }}

addVectorLayer

Register a vector layer with the Oriented Imagery viewer. Vector Layer will appear in the Overlay window of the viewer.

Parameters

- layer (string) Name or title of the layer.
- URL/ FeatureClass (string / JSON Array) URL of the vector layer or the feature class. If feature class is passed, then it should be JSON array that is defined like this

```
[
{
        geometry: {"x": -122.65, "y": 45.53, "spatialReference": {"wkid": 4326}},
        attributes: {
                            OBJECTID: 100, // OBJECTID parameter is required
                            Asset: "Car"

        }
}
]
```

- renderer (JSON) Symbology of the layer. Graphic will be rendered using the symbol object in the renderer; Renderer are of two types - simple and uniqueValue. Symbol can be point (circle, square, cross, x, triangle, diamond), polyline and polygon. example –
  renderer - simple, symbol type - point

```
{
"type": "simple",
"symbol": {
 "type": "point",
 "style": "circle",
 "color": [255,244,244,0.6], //rgba
 "outline": {
                            "width": 1, // in pixels
                            "color": [255,140,0,0.5] //rgba
               },
 "size": 4 // in pixels
}
}
```

renderer - simple, symbol type – polyline

```
{
"type": "simple",
"symbol": {
 type": "polyline",
 "color": [255,244,244,0.6], //rgba
 "width": 2 // radius in pixels
}
}
```

renderer - simple, symbol type – polygon

```
{
“type”: “simple”,
“symbol”: {
 "type": “polygon”,
 "color": [255,244,244,0.6], //rgba
 "outline": {
                        "width": 1, // in pixels
                        “color”: [255,140,0,0.5] //rgba
              }
}
}
```

renderer - uniqueValue, symbol type – point

```
{
“type”: “uniqueValue”,
“field1”: “Asset”, //field in layer based on which features are rendered
“uniqueValueInfos”: [
{
    “label”: “Car”,
    “value”: “1”,
    “symbol”: {
                        "type": “point”,
                        "style": “circle”,
                        "color": [255,244,244,0.6],
                        "outline": {
                                        "width": 1,
                                        “color”: [255,140,0,0.5]
                              },
                        “size”: 4 // radius in pixels
              }
},{
    “label”: “Pool”,
    “value”: “2”,
    “symbol”: {
                        "type": “point”,
                        "style": “circle”,
                        "color": [237, 81, 81, 1],
                        "outline": {
                                        "width": 0.75,
                                “color”: [153, 153, 153, 1]
                              },
                        “size”: 3
              }

}
]
```

```
}
```

renderer - uniqueValue, symbol type – polyline

```
{
"type": "uniqueValue",
"field1": "Asset", //field in layer based on which features are rendered
 "uniqueValueInfos": [
{
    "label": "Car",
    "value": "1",
    "symbol": {
                        "type": "polyline",
                        "color": [255,244,244,0.6],
                        "width": 2
            }
},
{
    "label": "Pool",
    "value": "2",
    "symbol": {
                        "type": "polyline",
                        "color": [237, 81, 81, 1],
                        "width": 2
            }
}
]
}
```

renderer - uniqueValue, symbol type – polygon

```
{
"type": "uniqueValue",
 "field1": "Asset", //field in layer based on which features are rendered
"uniqueValueInfos": [
{
    "label": "Car",
    "value": "1",
    "symbol": {
                        "type": "polygon",
                        "color": [255,244,244,0.6], //rgba
                        "outline": {
                                        "width": 1,
                                        "color": [255,140,0,0.5]
                            }
            }
},
```

```
{
    "label": "Pool",
    "value": "2",
    "symbol": {
                    "type": "polygon",
                    "color": [237, 81, 81, 1],
                    "outline": {
                                    "width": 0.75,
                                    "color": [153, 153, 153, 1]
                    }
        }
}
]
}
```

- editable (boolean) Whether the layer is editable or not. If true, then edit toolbar is active for the layer.

    Returns JSON {

        success: boolean,

        error: string – error message why the add vector layer failed

    } example :- {success: true} , {success: false, error: "Invalid renderer."}

Example: - viewer.addVectorLayer("pointoi",url, renderer,true).then(function(response) {});

removeVectorLayer
Remove layer from the viewer.
Parameters
- layer (string) Name or title of the layer.

applyEdits
Update the OI API after a feature has been added or deleted in the vector layer.
Parameters
- layer (string) Name or title of the layer.
- success (boolean) whether the edit operation was successful or not
- mode (string) Mode can be "add" or "delete"
- objectId (number) If the mode is "add" and success is true, then pass the OBJECTID of the new feature.
- error (string) if success is false, then pass the appropriate error message.

refreshVectorLayer
Refresh vector layer in the viewer. Makes a query to the server.
Parameters
- layer (string) Name or title of the layer.

zoomIn
Zoom in in the image.

zoomOut
Zoom out in the image.

toggleToolbar
Toggle the visibility of the toolbar in the viewer.
Parameters
- visible (boolean)

Example: - viewer.toggleToolbar(true)

toggleTool
Toggle the visibility of various tools in the toolbar of the viewer.
Parameters
- toolProperties (JSON) toolProperties such as name, toggle state and type of the tool.
  - name (string) – Valid values are Measurement, Navigation, Enhancement, DepthImage, Overlay, Gallery, Autoswitch and About
  - state (boolean) – toggle the tool in the viewer
  - type (string) – Optional. To select a particular type of tool. Type is only valid if the name is Measurement or Autoswitch. If name is Measurement, then type can be Distance, Area, Height, or Location.
    If name is Autoswitch, then type can be Angle lock on, Angle lock off, or Rotate.

    Examples: -
    viewer.toggleTool({ "name": "Measurement", "state": true, "type": "Area" });

    Viewer.toggleTool({ "name": "Autoswitch", "state": true, "type": "Rotate" });

    Viewer.toggleTool({ "name": "Navigation", "state": false});

enableFocusPoint
Toggle the visibility of focus point (red cross) in the viewer.
Parameters
- visible (boolean)
  Examples: -
  viewer.enableFocusPoint(true);

toggleEditTool
Toggle the edit tools – display, add, and delete in the viewer.
Parameters
- toolProperties (JSON) toolProperties such as layer name, tool name and tool state.
  - layer (string) – Name or title of the layer.
  - tool (string) – Edit tools. Valid values – display, add, or delete
  - state (boolean) – True means turn on the edit tool and false means turn off.

    Examples: -
    viewer.toggleEditTool({"layer": "PointsOI", "tool": "add", "state": true });

selectFeaturesInImage
Select (highlight) the features in the image.
Parameters
- Object {JSON Object}; example –

// json object with keys as layer title and array of OBJECTID(s) for that layer.

```
{
"PointTestOI": [234,213],
"PolylineTestOI": [434]
}
```

deselectFeaturesInImage
Deselect (remove highlight) the features in the image.
Parameters
- Object {JSON Object}; example –

// json object with keys as layer title and array of OBJECTID(s) for that layer.

```
{
"PointTestOI": [234,213],
"PolylineTestOI": [434]
}
```

setMapView
Turn on/off the 3D view mode.
Parameters
- properties (JSON)
  - width (number) – Width of the sceneView container.
  - height (number) – Height of the sceneView container.
  - state (string) – Valid values are On and Off. On to set the view mode to 3D and Off to set the view mode to 2D.

zoomImageOnMap
In the 3D view mode, call this function whenever the user zooms in the sceneView. Fires updateImageOnMap event which returns the Camera JSON object.
Parameters
- properties (JSON)
  - width (number) – Width of the sceneView container.
  - height (number) – Height of the sceneView container.
  - x (number) – Screen point X.
  - y (number) – Screen point Y.
  - fov (number) – Current FOV of the camera
  - heading (number) – Current heading of the camera
  - tilt (number) – Current tilt of the camera
  - position (JSON Point Object) – Current Camera Position
  - elevation (number) – Ground height of the camera position

- delta (number) – 0.5 to zoom in and 2 to zoom out.

panImageOnMap

In the 3D view mode, call this function whenever the user pan in the sceneView. Fires updateImageOnMap event which returns the Camera JSON object.

Parameters

- properties (JSON)
    - width (number) – Width of the sceneView container.
    - height (number) – Height of the sceneView container.
    - x (number) – Screen point X.
    - y (number) – Screen point Y.
    - fov (number) – Current FOV of the camera
    - heading (number) – Current heading of the camera
    - tilt (number) – Current tilt of the camera
    - position (JSON Point Object) – Current Camera Position
    - elevation (number) – Ground height of the camera position
    - action (string) – valid values are start, update and end.

resizeImageOnMap

In the 3D view mode, call this function whenever the sceneView dimensions are changed.

Parameters

- properties (JSON)
    - width (number) – Width of the sceneView container.
    - height (number) – Height of the sceneView container.
    - camera (JSON Object) – Camera JSON Object.
        - fov (number) – Current FOV of the camera
        - heading (number) – Current heading of the camera
        - tilt (number) – Current tilt of the camera
        - position (JSON Point Object) – Current Camera Position

Events

- searchImages

    Fired when searchImages method is called

    Returns Object {

    point: Object {Point JSON Object}; example –

    {"x": -122.65, "y": 45.53, "spatialReference": {"wkid": 4326}}

    imageSourcePoints: Array [Point JSON Objects]; example –

    {"x": -122.65, "y": 45.53, "spatialReference": {"wkid": 4326}, "imageID": 2}

    coveragePolygons: Array [Polygons JSON Objects] Coverage Polygons for all the images; format similar to Polygon JSON object of JavaScript API for ArcGIS; example –

{"rings":[[[-122.63,45.52],[-122.57,45.53],[-122.52,45.50],[-122.49,45.48],
  [-122.64,45.49],[-122.63,45.52],[-122.63,45.52]]],"spatialReference":{"wkid":4326 }, "imageID": 2}

coverageFrustums: Array [Mesh JSON Objects] Coverage Frustums for all the   images; format
similar to Mesh JSON object of JavaScript API for ArcGIS

imageAttributes: JSON Image Attributes such as current imageID, vfov, hfov, pitch, yaw, roll, z, x,y etc.

- changeImage

    Fired when the new image is loaded in the viewer
    Returns Object {imageID: Number}
- updateImage

    Fired when the image view changes in the viewer
    Returns Object {imageID: Number,
                        coveragePolygon: Object {Mesh JSON Object};

- imageToGroundPoint

    Fired when the user does alt + click on the image. Corresponding ground Point is returned.
    Returns Object {Point JSON Object}; example –

    {"x": -122.65, "y": 45.53, "spatialReference": {"wkid": 4326}}

- addFeature

    Fired when add feature button is activated for a particular layer in the viewer and the user clicks on
    the image.
    Returns Object {
    layer: String Name or title of the layer.
    attributes: Object {JSON Object}; example –

    {
    "ImgUrn": "ASDASDASD", // global ID
    "ImgGeom": {"x": 3000, "y": 200}// image geometry
    }

    geometry: Object {JSON Object}; example –

    {"x": -122.65, "y": 45.53, "spatialReference": {"wkid": 4326}}

    }

- deleteFeature

    Fired when delete feature button is activated for a particular layer in the viewer and the user clicks
    on the graphic in the image.
    Returns Object {

imageID: String global ID
layer: String Name or title of the layer
featureId: Number ID of the feature to be deleted from feature class
}

- selectFeature

  Fired when the user clicks on the graphic in the image. Returns the OBJECTID of the selected
  Feature(s) for a vector layer.
  Returns Object {JSON Object}; example –
  // json object with keys as layer title and array of OBJECTID(s) for that layer.

  ```
  {
  "PointTestOI": [234,213],
  "PolylineTestOI": [434]
  }
  ```

- measurement

  Fired when the measurement is taken in the viewer.
  Returns Object {
                  Distance: {
                          value: 20,
                           accuracy: 0.1
                      },
                  Area: {
                          value: 20,
                           accuracy: 0.1
                      },
                  Height: {
                          value: 20,
                           accuracy: 0.1
                      },
                  Point: {
                          value: {
                                      x: 23.45,
                                      y:12.34,
                                      z:23
                              },
                          accuracy: {
                                      XY: 0.12
                                      Z: 0.1
                              },
                      }

- addImageOnMap

Fired when the searchImages function is called and the view mode is set to 3D.
Returns Object {

        image: Mesh object which contains the image,
        error: error string if failed,
        properties: {

                pitch: pitch of the camera,
                yaw: heading of the camera,
                fov: fov of the camera,
                location: position of the camera

        },
        coveragePolygons: same as searchImages event,
        coverageFrustums: same as searchImages event,
        imageSourcePoints: same as searchImages event,
        imageAttributes: {

                imageID: 233
        }
        }

- updateImageOnMap

Fired when the zoomImageOnMap or panImageOnMap function is called and the view mode is set to 3D.
Returns Object {

        image: Mesh object which contains the image,
    }

- updateCamera

Fired when the zoomImageOnMap or panImageOnMap function is called and the view mode is set to 3D.
Returns Camera JSON Object {

                pitch: pitch of the camera,
                yaw: heading of the camera,
                fov: fov of the camera,
                location: position of the camera

        }

Oriented Imagery API source code structure

Learn more about the structure of the Oriented Imagery API source code.

File structure

Following are descriptions of the primary API folders and files.

css	05/11/2017 3:16 PM	File folder	
images	28/12/2017 9:58 AM	File folder	
js	05/11/2017 2:55 PM	File folder	
main	16/01/2018 4:02 PM	JS File	8 KB
viewer	27/12/2017 4:45 PM	HTML File	4 KB

css

This folder contains all the CSS required by the API. Style attributes for all HTML elements should be define in these files.

fonts	05/11/2017 3:16 PM	File folder	
orientedViewer	29/01/2018 11:16 …	CSS File	14 KB
pannellum	07/11/2017 9:59 AM	CSS File	9 KB

If styling of the panorama viewer needs to be updated, add/modify/delete classes in pannellum file. To update styling properties of the viewer and menu toolbar, buttons, sliders etc., add/modify/delete classes in orientedViewer file. This file is the main CSS file. Most of the styling classes are defined here.

images

This folder contains all the images that is required by the viewer such as icons for tools, pointer etc.

js

This folder contains all JavaScript files used by the API.

panoramaViewer	05/11/2017 12:56 …	File folder	
mrfDataset	07/01/2018 3:08 PM	JS File	33 KB
viewer	17/01/2018 12:55 …	JS File	118 KB

viewer.js file contains the main logic. Most of the logic is defined in this file such as ground to image transformation, querying the catalogs, finding best image, computing LUT table etc.

mrfDataset.js file contains the logic for reading the MRF files.

panoramaViewer folder contains the logic for displaying the panorama images. WebGL is defined in these files.

libpannellum	28/12/2017 3:03 PM	JS File	61 KB
pannellum	11/12/2017 2:55 PM	JS File	97 KB
RequestAnimationFrame	05/10/2017 5:21 PM	JS File	1 KB

main.js

main.js is the main javascript file. All the API functions/ methods are defined in this file. It loads rest of the files in the API like viewer.js etc.

Flowcharts

The following chart describes how a client app interacts with the Oriented Imagery API.

```
                    ┌─────────────────┐
                    │   Client Apps   │
                    └─────────────────┘
                            ↕
               [Communication both ways]
                            ↕
                    ┌─────────────────┐
                    │     main.js     │
                    └─────────────────┘
                       ↓           ↘
          [Loads / calls function]  [Loads]
                       ↓              ↘
              ┌─────────────┐    ┌──────────────────┐
              │  viewer.js  │    │  ArcGIS API for  │
              └─────────────┘    │    JavaScript    │
                                 └──────────────────┘
              ↙       ↓       ↘
         [Loads]   [Loads]   [Loads]
          ↙          ↓          ↘
┌──────────────┐ ┌────────────┐ ┌────────────┐
│ mrfDataset.js│ │ viewer.html│ │  pannellum │
└──────────────┘ └────────────┘ └────────────┘
```
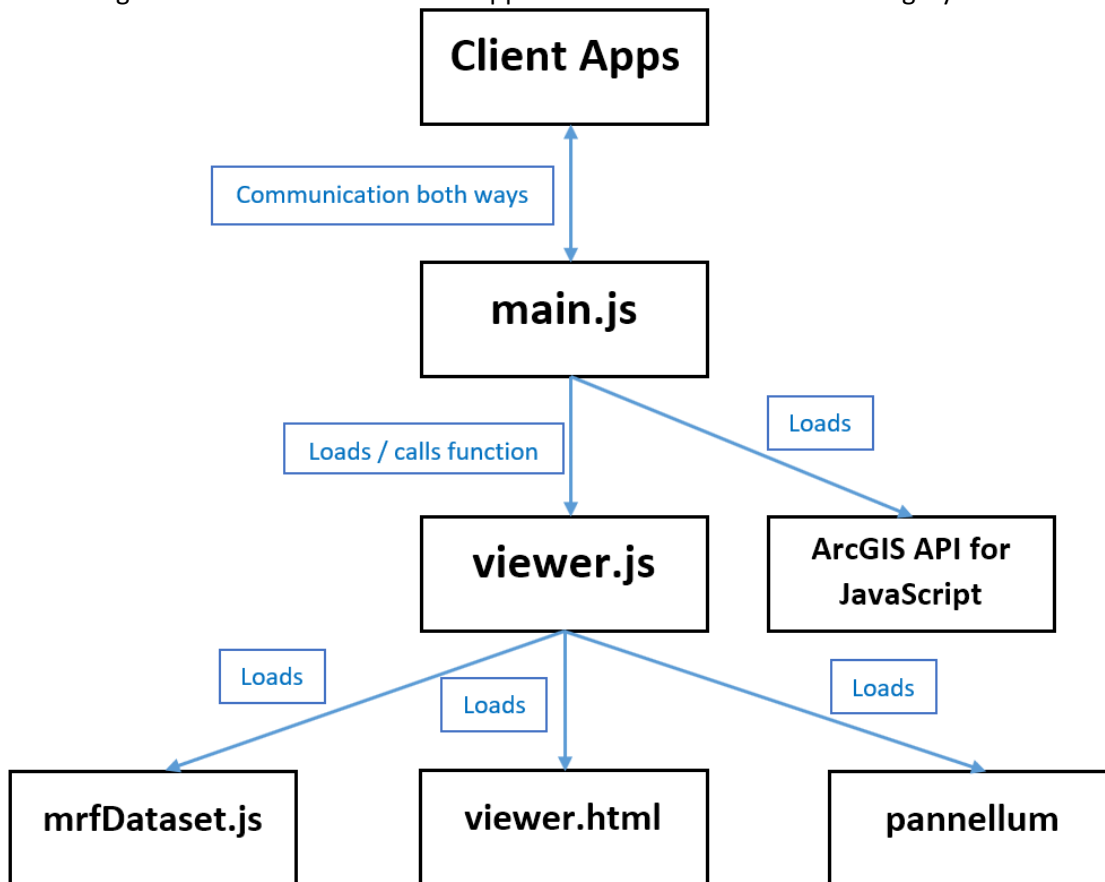
The following chart describes how ArcGIS Pro (via the Oriented Imagery add-in for ArcGIS Pro) and web clients interact with the Oriented Imagery API.

```
              ┌─────────────────┐
              │   Oriented      │
              │  Imagery API    │
              └─────────────────┘
               ╱               ╲
    ┌──────────────┐      ┌──────────────┐
    │  JS/HTML     │      │  HTML file   │
    │  files       │      │  (add-in)    │
    └──────────────┘      └──────────────┘
           │                     │
           │              ┌──────────────┐
           │              │ C# functions │
           │              └──────────────┘
           │                     │
    ┌──────────────┐      ┌──────────────┐
    │  Web map /   │      │  ArcGIS Pro  │
    │  web scene   │      │    map       │
    └──────────────┘      └──────────────┘
```