

Working with custom JavaScript functions

in ArcGIS Survey123 Forms

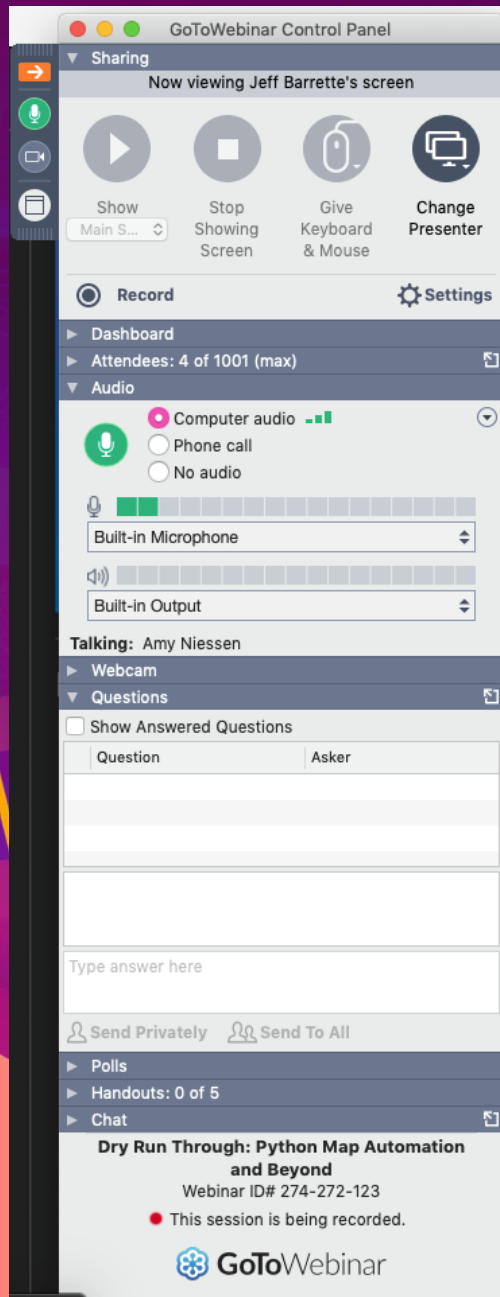
We will start shortly...



Ismael Chivite



Travis Butcher



Before we start...

Working with custom JavaScript functions

in ArcGIS Survey123 Forms



Ismael Chivite



Travis Butcher

Agenda

- **The Basics, by Ismael Chivite (20 minutes)**
 - Why should you care about custom JS functions in Survey123?
 - How to work with custom JS functions
 - Demonstration
 - Limitations
- **Advanced topics, by Travis Butcher (20 minutes)**
 - Getting serious with webpack
 - Demonstration
- **Where to learn more**
- **Live Q&A**

Why JavaScript functions in ArcGIS Survey123?

- **Complement XLSForm expression syntax**
- **Use in calculation, constraint and relevant expressions**
 - Working with data in feature services
 - Spatial analysis
 - Work with values across repeats
 - Access third-party APIs
 - Build complex calculations
 - Parse complex data structures
 - Data validation rules and constraints
- **Supported in the Survey123 **field app** and **web app****

Work with third-party APIs

```
// Query the Open Weather API: https://openweathermap.org/api
function runWeatherCalcs(lat, lon, key){
  // Check to make sure we have latitude, longitude and an API key
  if (lat == null || lon == null || key == null) {
    return "";
  }

  // Create the request object
  var xmlhttp = new XMLHttpRequest();
  // Format the URL with the input parameters
  let lat_param = `lat=${lat}`;
  let lon_param = `lon=${lon}`;
  let key_param = `APPID=${key}`;
  let format_param = 'format=json';
  let units_param = `units=imperial`;
  let parameters = [lat_param, lon_param, key_param, format_param, units_param].join("&");
  var url = `https://api.openweathermap.org/data/2.5/weather?${parameters}`;

  // Make the request
  xmlhttp.open("GET",url,false);
  xmlhttp.send();

  // Check the response. 200 indicates success from the API
  if (xmlhttp.status!=200){
    return null;
  } else {
    // Check the information in the response for an error
    var responseJSON=JSON.parse(xmlhttp.responseText)
    if (responseJSON.error){
      return responseJSON.error;
    } else {
      if (responseJSON){
        return JSON.stringify(responseJSON);
      }
      else {

```



Work with a feature service

```
// Query a feature layer and returns the feature that intersects the location
function featureByLocation(layerURL, location, token) {
  // Output value. Initially set to an empty string (XLSForm null)
  let outValue = "";

  // Check to make sure both layerURL and location are provided
  if (layerURL == null || layerURL === "" || location == null || location === "") {
    // The function can't go forward; exit with the empty value
    return location;
  }

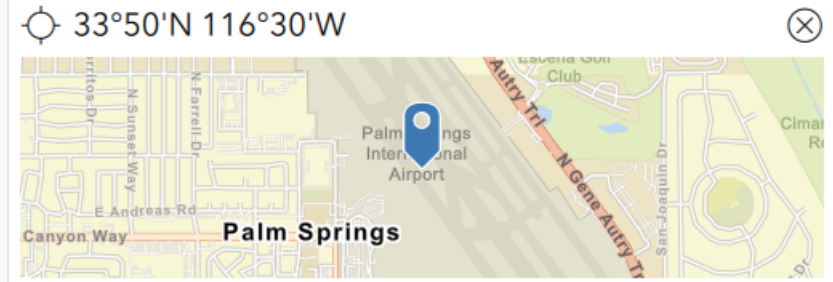
  // The coordinates will come in as ` <lon> <alt> <acc>`.
  // We need <lon>,<lat> for the query
  // Note that I'm using the relatively new `` string that lets me place variables ${var}
  let coordsArray = location.split(" ");
  let coords = `${coordsArray[1]},${coordsArray[0]}`;

  // Set up query parameters
  let f = "f=json";
  let geometry = `geometry=${coords}`;
  let geometryType = "geometryType=esriGeometryPoint";
  let inSR = "inSR=4326";
  let spatialRel = "spatialRel=esriSpatialRelIntersects";
  let outFields = "outFields=*";
  let returnGeometry = "returnGeometry=false";
  let returnCount = "returnCount=1";
  let parameters = [f,geometry,geometryType,inSR,spatialRel,outFields,returnGeometry,returnCount];
  if (token) {
    parameters = parameters + `&token=${token}`;
  }
  let url = `${layerURL}/query?${parameters}`;

  // Create the request object
  let xhr = new XMLHttpRequest();
```

Administrative Divisions

Please set a location



The 1st level admin area feature the location is in

```
{"attributes":{"FID":
933,"NAME":"California","COUNTRY":"United
States","ISO_CODE":"USCA","ISO_CC":"US","ISO_SUB"
"CA","ADMINTYPE":"State","DISPUTED":0,"NOTES":"
","AUTONOMOUS":0,"COUNTRYAFF":"United
States","CONTINENT":"North
America","LAND_TYPE":"Primary land","LAND_RANK":
5,"Shape__Area":647615925620.137,"Shape__Length":
7058203.1681138}}
```

ISO_CODE value

USCA

NAME attribute

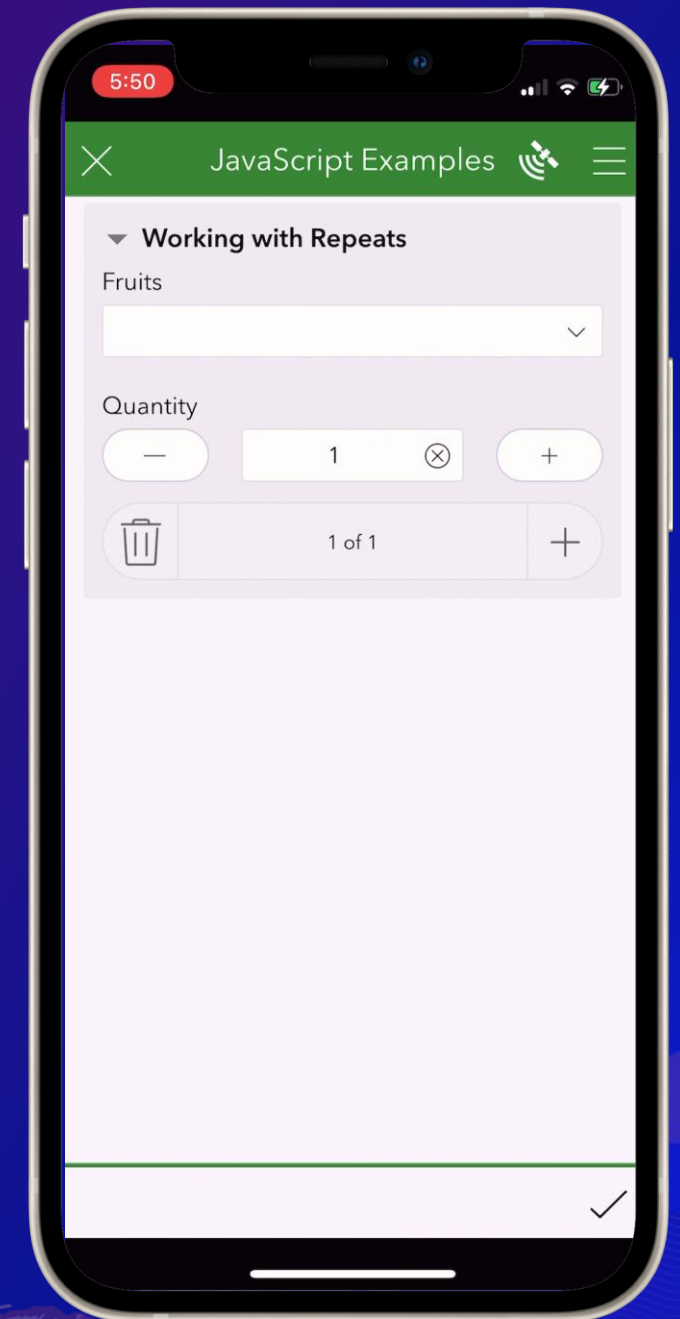
California




Work with repeats

```
function HasDups (myArray) {  
    return new Set(myArray).size !== myArray.length;  
}
```

type	name	label	relevant
begin repeat	fruits	Working with Repeats	
select_one fruits	fruit	Fruits	
decimal	qty	Quantity	
end repeat			
note		Fruits repeated	pulldata("@javascript", "JS.js", "HasDups", \${fruit})



JavaScript functions in ArcGIS Survey123

- Use the **pulldata("@javascript")** function to execute JavaScript
 - Often used in conjunction with **pulldata("@json")**
 - **Scripts** tab in Connect for managing JS files
 - JavaScript files live in the **scripts** survey folder
- 

JavaScript pulldata() function in XLSForm

Execute a
JavaScript function

Function name

Function
parameter

```
pulldata("@javascript", "yourJSFile.js", "yourFunction", "${parameter1}", "parameter2")
```

JavaScript
file name

Function
parameter

XLSForm sample in Connect

New Survey - ArcGIS Survey123 Connect

New Survey

Title


My JavaScript Survey

Table name will be: **My_JavaScript_Survey**

Select an initial XLSForm design

- Templates
- Samples
- Community
- My surveys
- My organization
- Feature service
- File

Search: javascript



JavaScript

This sample demonstrates how to incorporate your own JavaScript functions in a survey. It includes several example scripts for working with repeats, feature services, and APIs. For more information, see this [GeoNet blog post](#).

Resource level: **★★★★**

Modified: Tuesday, 20 October 2020 1:59:23 PM AUS Eastern Summer Time

Type: Form

Owner: ArcGIS_Survey123

XLSForm sample in Connect

JavaScript Examples

- ▶ Hello World
- ▶ Smart Sum
- ▶ Working with a Feature Service
- ▶ Working with a Third-Party API - Vehicle VIN
- ▶ Working with a Third-Party API - Open Weather
- ▶ Working with Repeat Data - Standard Deviation
- ▶ Working with Repeat Data - Calculating a Convex Hull



JavaScript




Create a new script to get started.

+ New script

JavaScript functions

Limitations

- **JavaScript functions are not supported in public surveys**
 - **Signed in users must be members of the same organization as the survey's author**
 - **You cannot access local files**
 - **Asynchronous calls are not supported**
 - **Document Object Model (DOM) is not supported**
 - **Frameworks such as JQuery, Ember, and Angular are not supported**
 - **A `pulldata("@javascript")` function cannot be called inside a `pulldata("@json")` function in the Survey123 web app**
- 

Learn more (The Basics)

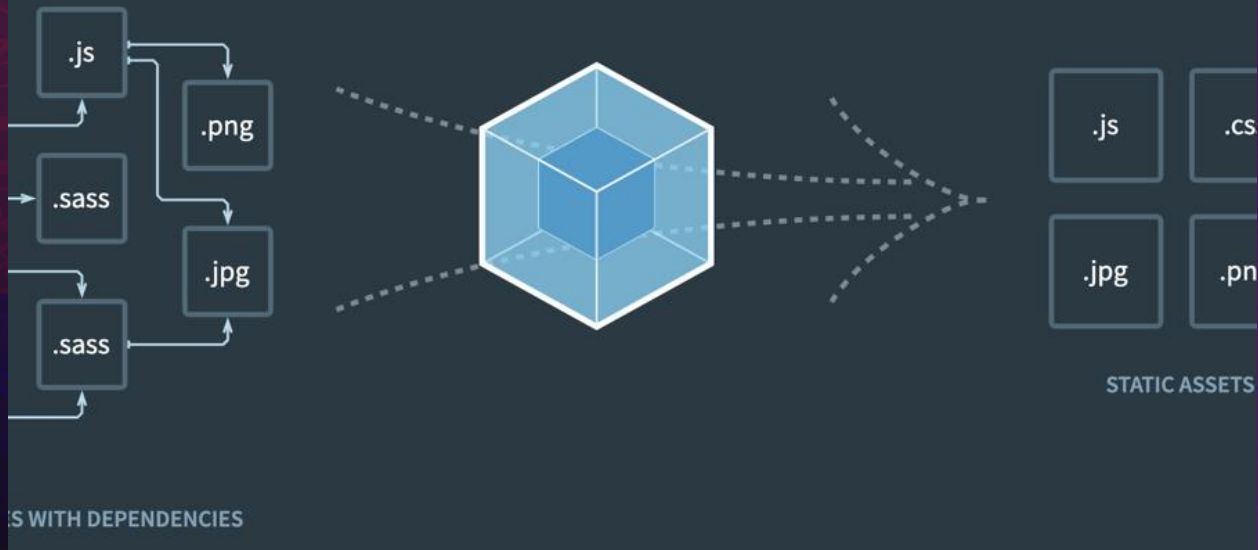
- **Esri Community blog post:**
 - [Extending Survey123 smart forms with custom JS functions](#)
- **Documentation:**
 - [JavaScript functions in survey forms](#)
- **JavaScript XLSForm sample in Survey123 Connect**



Agenda

- **The Basics, by Ismael Chivite (20 minutes)**
 - Why should you care about custom JS functions in Survey123?
 - Common uses – with examples!
 - Demonstration
 - Limitations
- **Advanced topics, by Travis Butcher (20 minutes)**
 - Getting serious with webpack
- **Where to learn more**
- **Live Q&A**

bundle your scripts



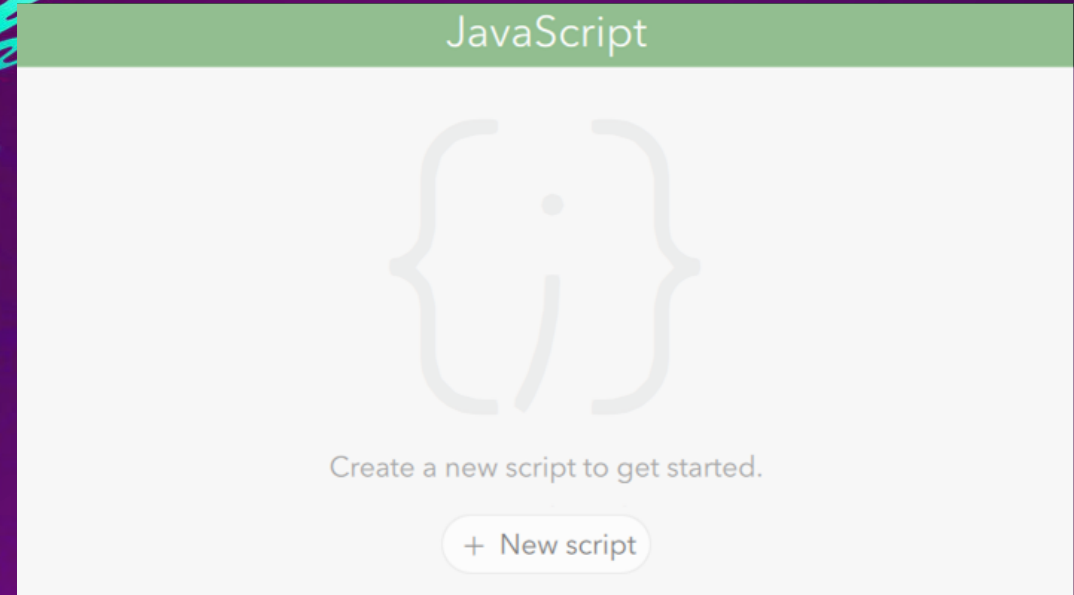
JavaScript functions with Webpack

Getting serious with webpack

JavaScript functions in ArcGIS Survey123 with Webpack

- Use third party libraries not included JavaScript
- Bundles function to single library to include with form
- Allows to store functions separately and reuse across multiple form easily





Demo: Survey123 with Webpack

Learn more (Advanced)

- **GitHub Repository**
 - <https://github.com/EsriPS/survey123-webpack>
- **Developer Tech Session**
 - <https://www.youtube.com/watch?v=iahSB3P4q1A>
- **WebPack**
 - <https://webpack.js.org/>



Live Q&A



Ismael Chivite



Travis Butcher