Advanced XLSForm Techniques in ArcGIS Survey123

Philip Wilson, Jim Moore, Brett Stokes

2021 ESRI
DEVELOPER SUMMIT

Advanced XLSForm Techniques

- Inbox workflows
- JavaScript functions
- Managing choice lists
- Image-map appearance

Inbox workflows

Modeling routine inspection workflows

Asset Inspections

- Nearly every organization has assets that need to be inventoried & inspected
  - Fire Hydrants
  - Well machinery
  - Boats
- Creating a successful inspection workflow requires
  - Data structure that supports inspections
  - Forms that make it easy to inspect
  - Information products that inform stakeholders of inspection status

Data Structure

- Asset and inspection information in the same table
  - Limited inspection history
  - Always editing the asset features
- Asset and inspection information in separate tables (preference)
  - Use relationship classes to link assets to inspections (encourage GUID-based keys)
  - Stores inspection history by adding new records to the inspection table
  - Can use feature layer views with a join to display latest inspection information

Form Design

- Asset + Inspection – same table
  - Use the *Inbox* to load records & update
- Asset + Inspection – different tables
  - Requires relationship class using GUID key in data
  - Use the *Inbox* to load the assets information
  - Add new inspection record within the form
  - Optionally load inspection history to view or update

Form Design

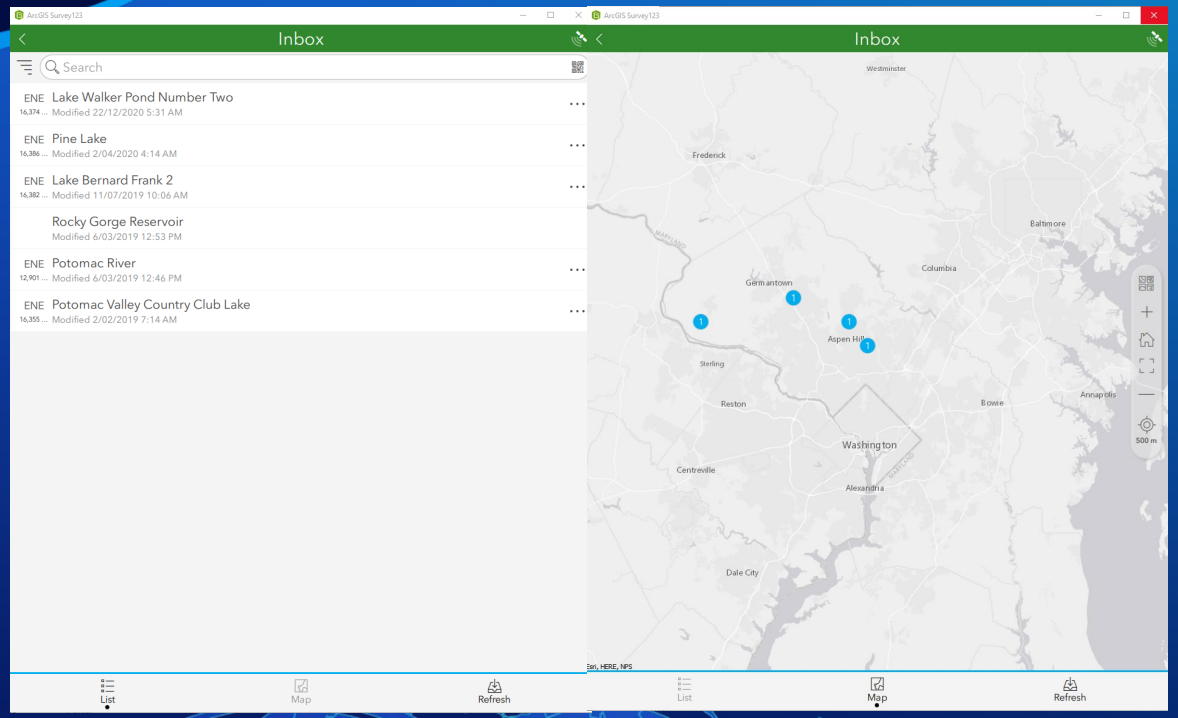
- **Inspection Only**
  - Assumes separate inspection table (not required)
  - Doesn't require *Inbox*
  - Doesn't require GUID-based relationship
  - Requires asset ID to be set in the form (barcode/text, dropdown, URL parameter)
  - Commonly opened via URL from other applications (e.g., Workforce, Explorer)

Form Design

- Use question properties to control data entry
  - Remove questions
  - Read Only (can't update)
  - Hidden question type & appearance (non-visible)
  - Repeat parameters
- Enable editing through the Inbox capability

Edit Existing Survey Data

- Enable the `Inbox` option for the survey
- Disable the `Sent` box to allow records to return to the `Inbox`
- Query on fields and profile information, ex:
  - status=‘pending’
  - assigned=${username}
- This capability must be enabled on the survey BEFORE publishing
- Editing Existing Data with Survey123 video



Inbox workflows

Philip Wilson

Advanced XLSForm Techniques

- Inbox workflows
- JavaScript functions
- Managing choice lists
- Image-map appearance

JavaScript functions

Run scripts in your Survey123 forms

JavaScript functions

- JavaScript and XLSForm
- Common uses – with examples!
- Demonstration
- Limitations
- Resources

JavaScript functions in Survey123

- Complement XLSForm expression syntax
- The pulldata() function is the link between XLSForm and JavaScript files
- Supported in the Survey123 field app and web app
- User must be signed in

- Use XLSForm functions if you can!

JavaScript functions in XLSForm

- Scripts tab in 3.12 for managing JS files
- JavaScript files need to be added to the scripts folder
- Use the pulldata("@javascript") function to execute JavaScript
- Often used in conjunction with pulldata("@json")

JavaScript functions in XLSForm

Execute a JavaScript function

Function name

Function parameter

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

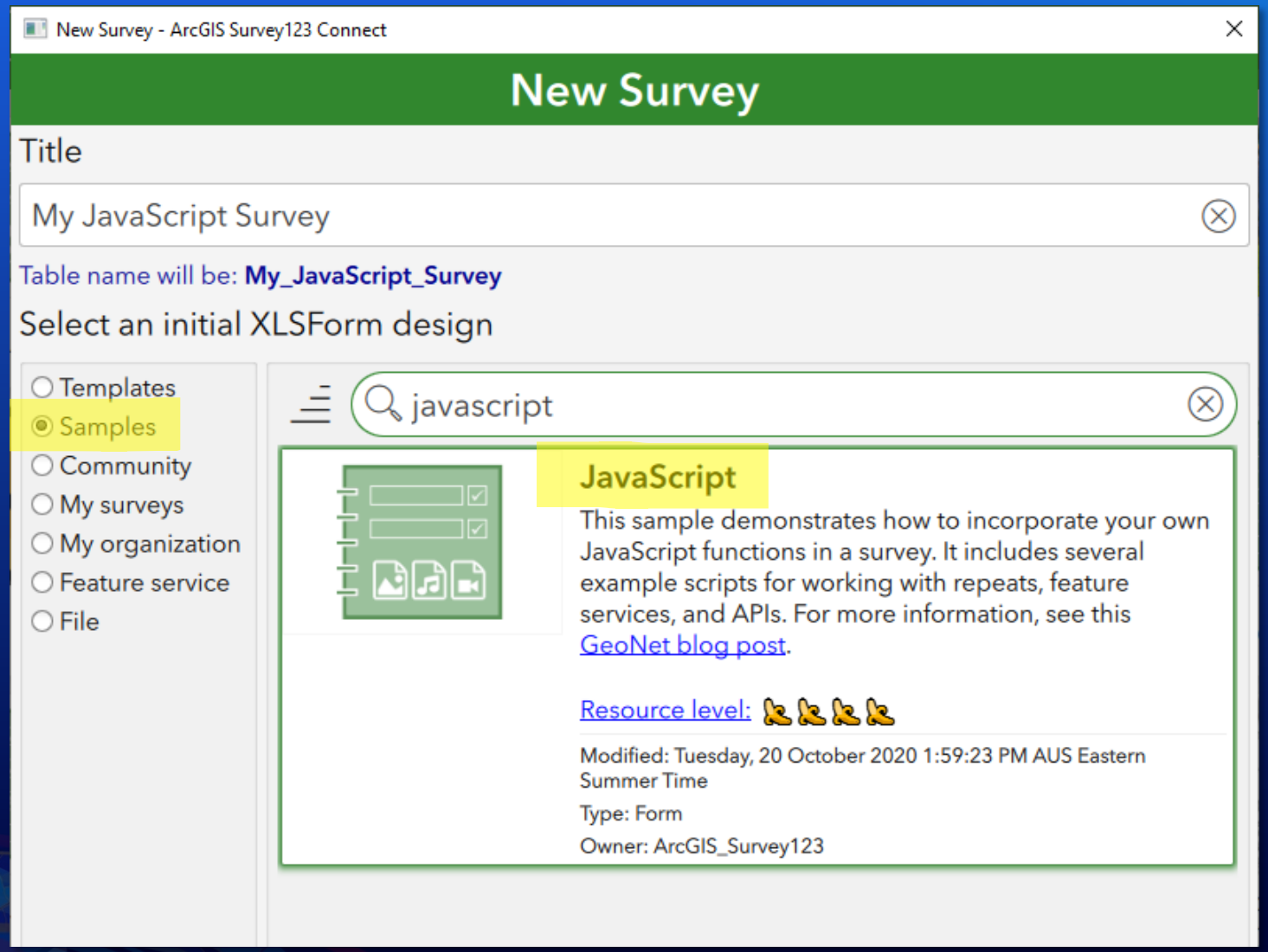
JavaScript file name

Function parameter

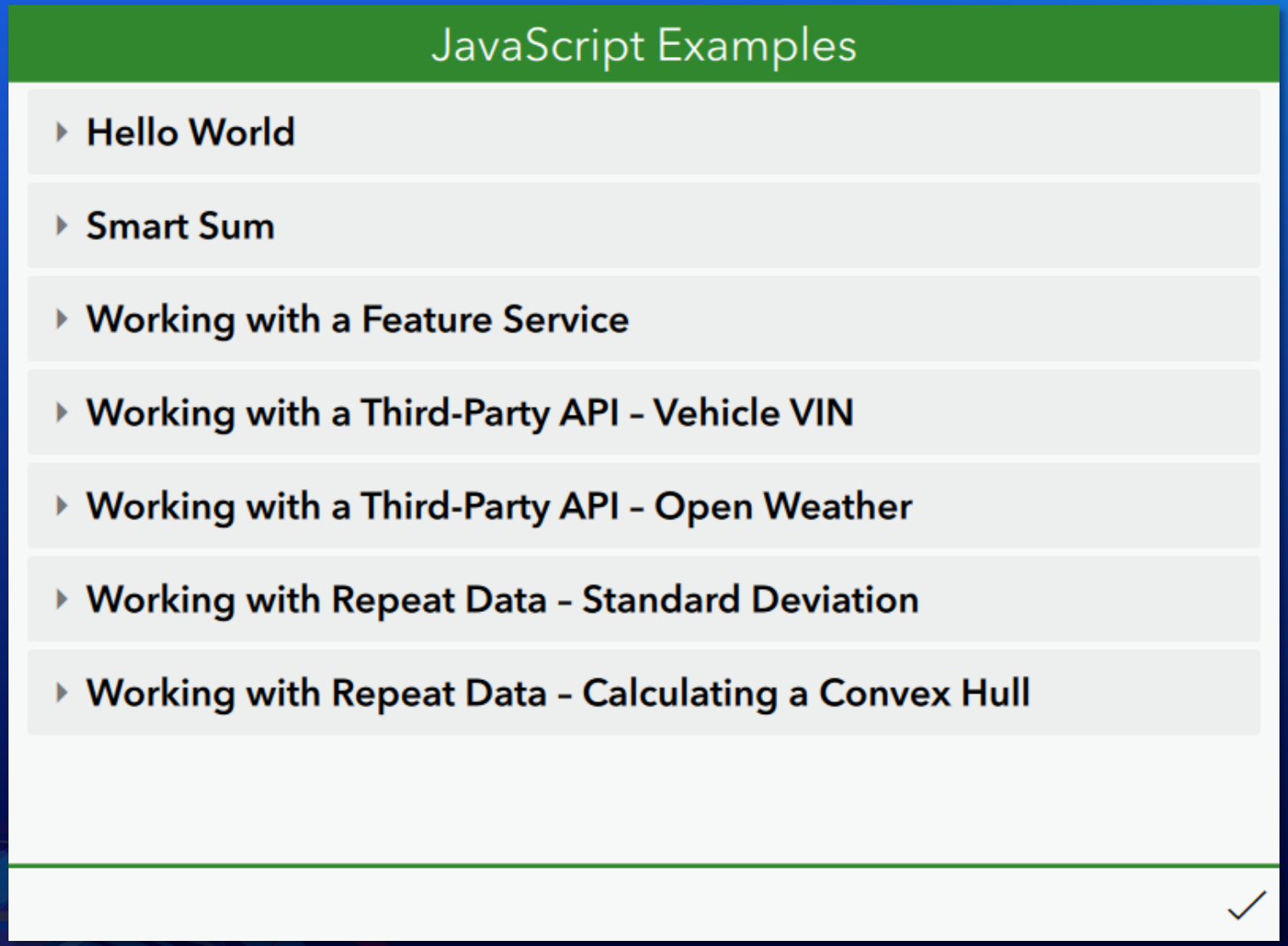
Common uses for JS functions in Survey123

- 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

XLSForm sample in Connect

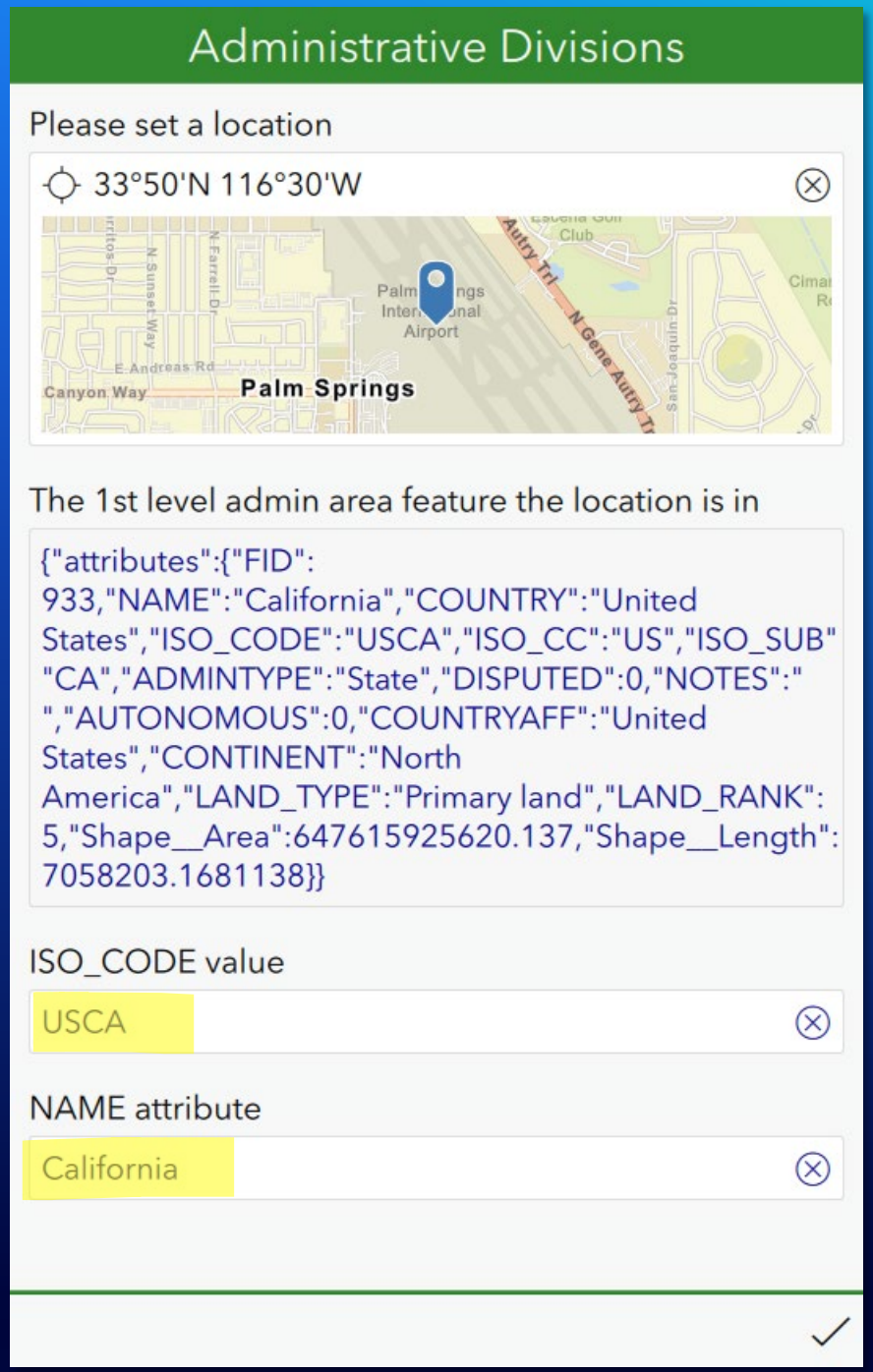


XLSForm sample in Connect



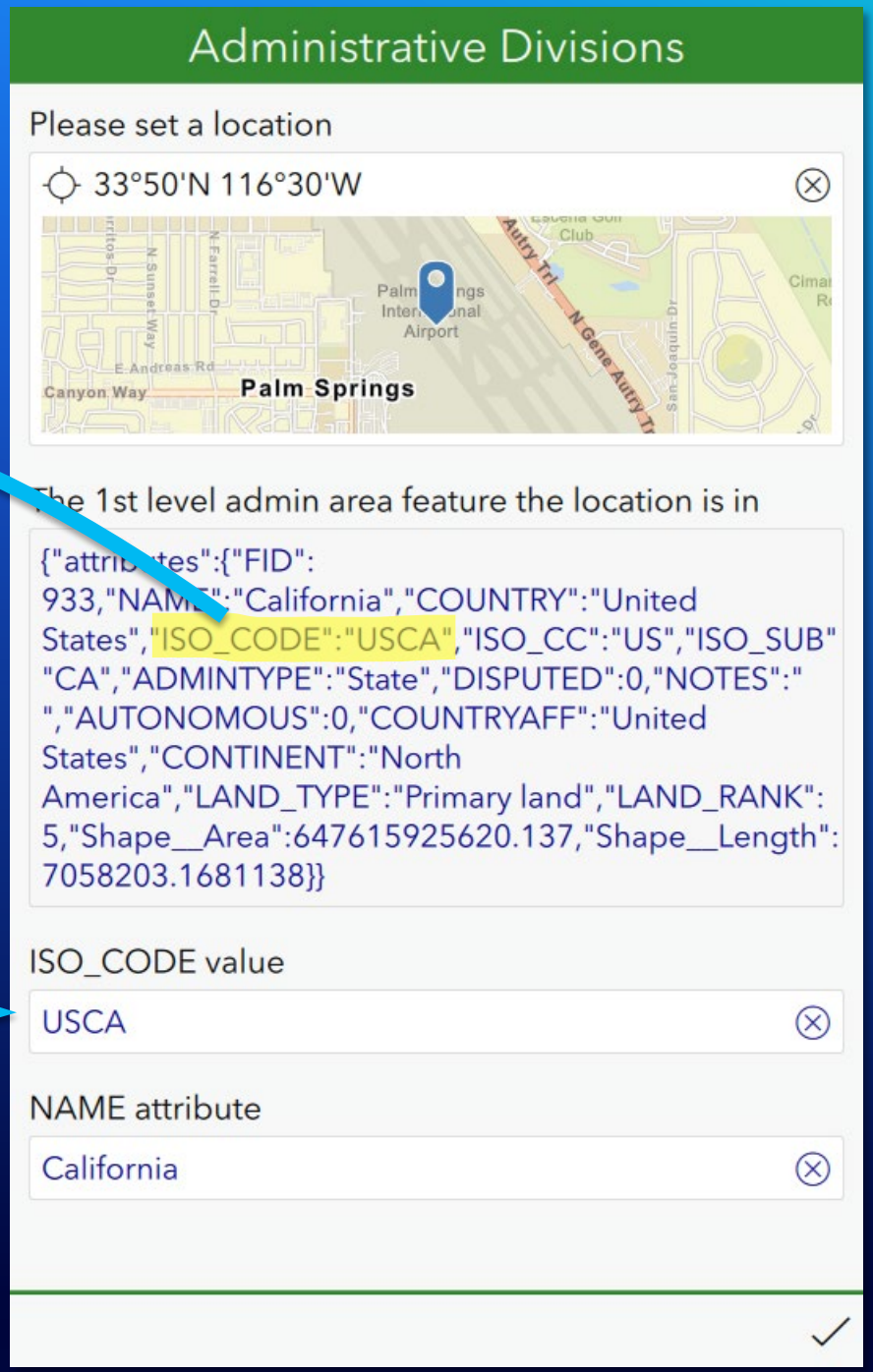
Working with a feature service

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


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

    // Create the request object
    let xhr = new XMLHttpRequest();
    // Make the request.  Note the 3rd parameter, which makes this a synchronous reques
    xhr.open("GET", url, false);
```



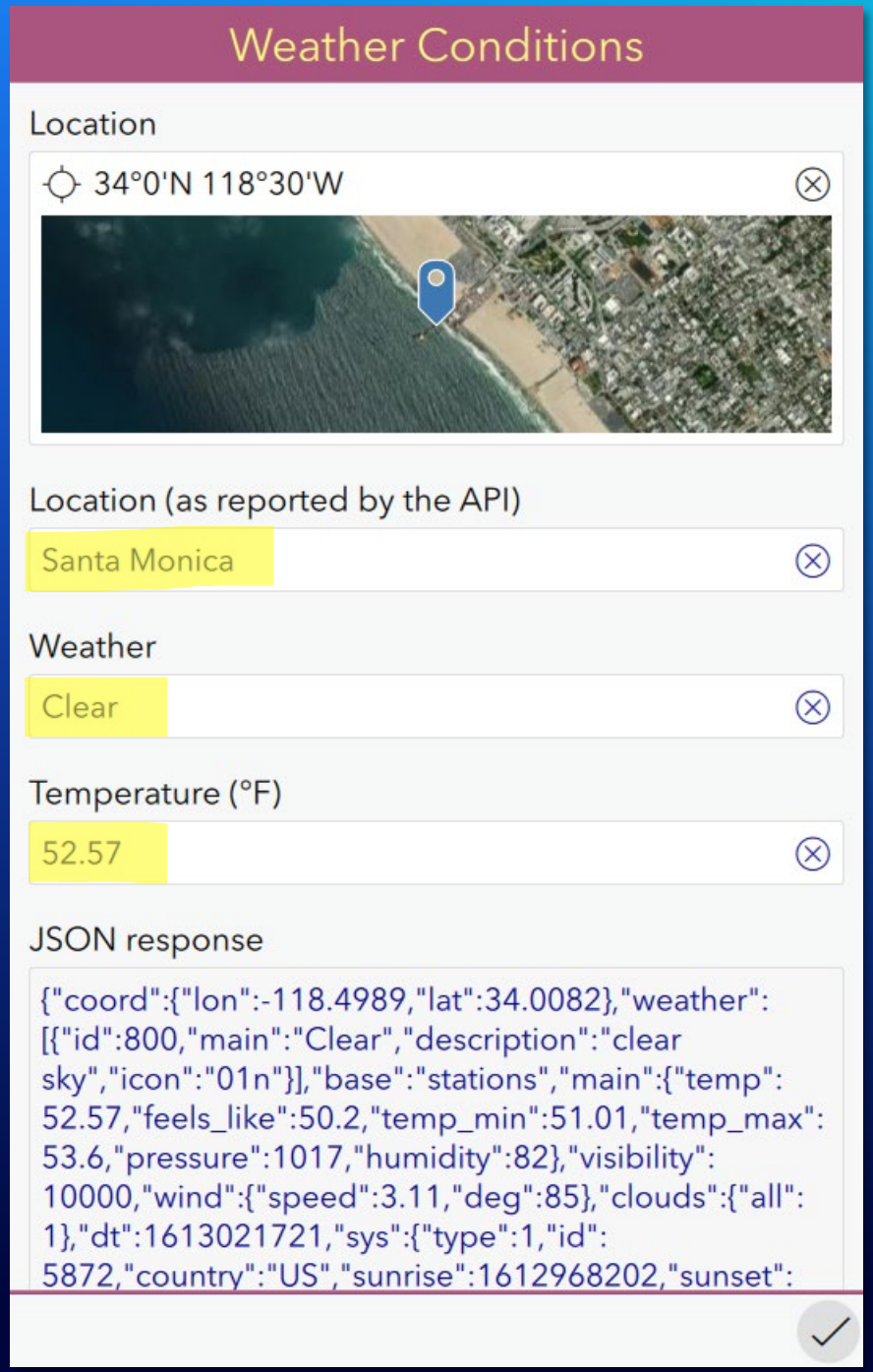
Working with a feature service

```
pulldata("@json", ${adminFeature}, "attributes.ISO_CODE")
```



Working with third-party APIs

```
// Query the Open Weather API
// https://openweathermap.org/api
// How ot get started with the API: https://openweathermap.org/appid
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 {
```



Working with third-party APIs

Weather - latitude	pulldata("@geopoint",${weather_location},"y")
Weather - longitude	pulldata("@geopoint",${weather_location},"x")
JSON response	pulldata("@javascript", "weather.js", "runWeatherCalcs", ${weather_lat}, ${weather_lon}, ${weather_api_key})
Location (as reported by the API)	pulldata("@json", ${weather_json}, "name")
Weather	pulldata("@json", ${weather_json}, "weather[0].main")
Temperature (°F)	pulldata("@json", ${weather_json}, "main.temp")

Location (as reported by the API)

Santa Monica

Weather

Clear

Temperature (°F)

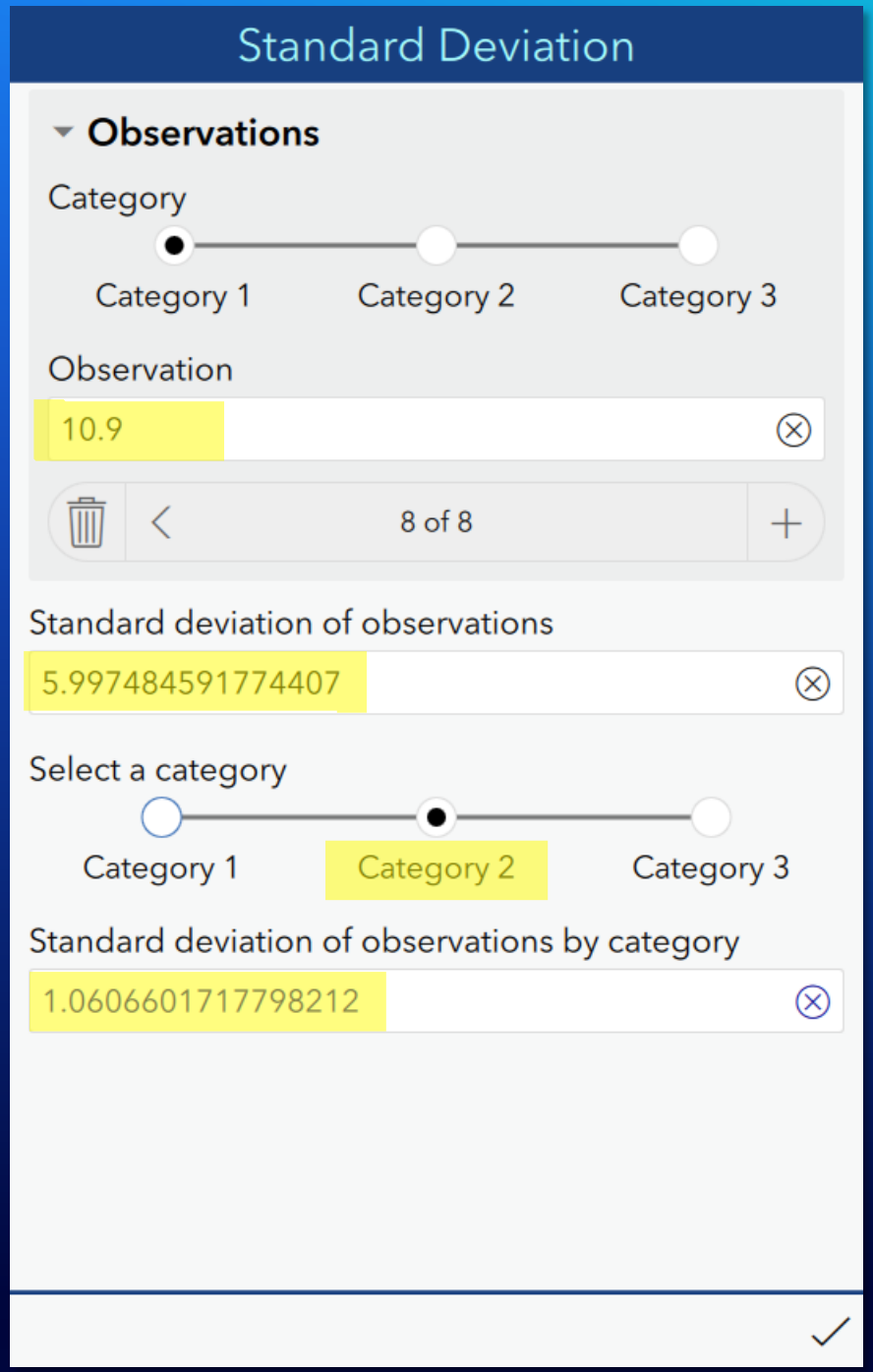
52.7

JSON response

{"coord":{"lon":-118.4989,"lat":34.0082},"weather":[{"id":800,"main":"Clear","description":"clear sky","icon":"01n"}],"base":"stations","main":{"temp":52.7,"feels_like":50.34,"temp_min":51.01,"temp_max":53.6,"pressure":1018,"humidity":82},"visibility":10000,"wind":{"speed":3.11,"deg":85},"clouds":{"all":1},"dt":1613023901,"sys":{"type":1,"id":5872,"country":"US","sunrise":1612968202,"sunset":1613007195},"timezone":-28800,"id":5393212,"name":"Santa Monica","cod":200}

Working with repeats

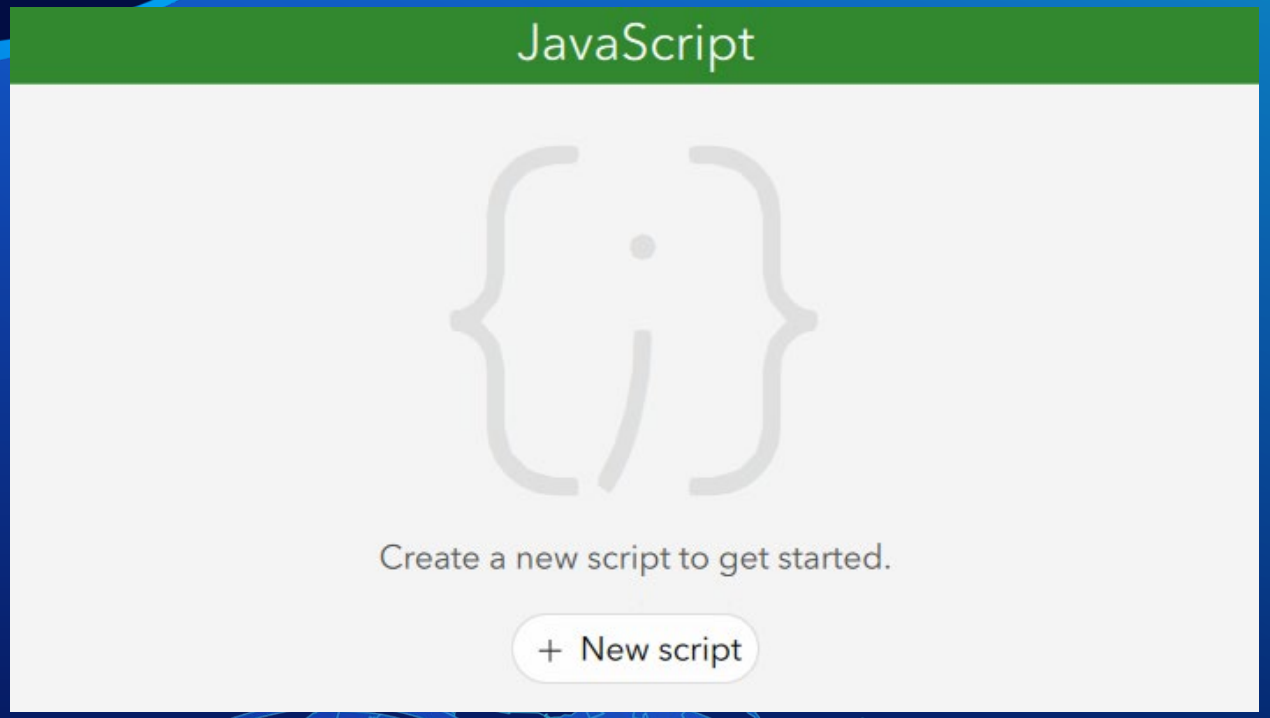
```
function stddev(invals) {
    if (Array.isArray(invals) && invals.length > 1) {
        var mean = invals.reduce(function (total, num) {return total + num;}, 0
        var sqdiffs = invals.map(function (value) {return (value - mean)*(value
        var meansqdiffs = sqdiffs.reduce(function (total, num) {return total +
        var stdev = Math.sqrt(meansqdiffs);
        return stdev;
    }
    return null;
}

function stddev_by_cat(repeat, cat_field, obs_field, cat_value) {
    let invals = [];
    // filter out the values based on matching the category
    if (cat_value !== '') {
        repeat = repeat.filter(record => record[cat_field] === cat_value)
    }
    repeat = repeat.filter(record => record[obs_field] != undefined);
    // grab the observation values from each record
    invals = repeat.map(record => record[obs_field]);

    // Call the existing stddev function
    return stddev(invals);
}
```



Parsing complex data structures

- **Refer to the GeoNet Esri Community blog post:**
  - [Extending Survey123 smart forms with custom JS functions]


```
@\n\u001e\rANSI 636004080002DL00410266ZN03070017DLDAQ123456789123\nDCSDOE\nDDEN\nDACJANE\nDDFN\nD
AD\nDDGN\nDCAC\nDCBNONE\nDCDNONE\nDBD08152015\nDBB08151987\nDBA08152020\nDBC2\nDAU070 in\nDAYBRO
\nDAG1100 NEW BERN AVENUE\nDAIRALEIGH\nDAJNC\nDAK276970001\nDCF0123456789\nDCGUSA\nDAZBRO\nDCLU
\nDCK000012345678NCSVTL01\nDDB10242014\nDDK1\nDDL1\nZNZNADUP\nZNB\nZNC0\n
```



JavaScript functions

Jim Moore

Limitations

- JavaScript functions are not supported for public surveys
- JavaScript functions are only supported in forms completed by 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

Resources

- GeoNet Esri Community blog post:
  - Extending Survey123 smart forms with custom JS functions
- Documentation:
  - JavaScript functions in survey forms
- JavaScript XLSForm sample in Connect

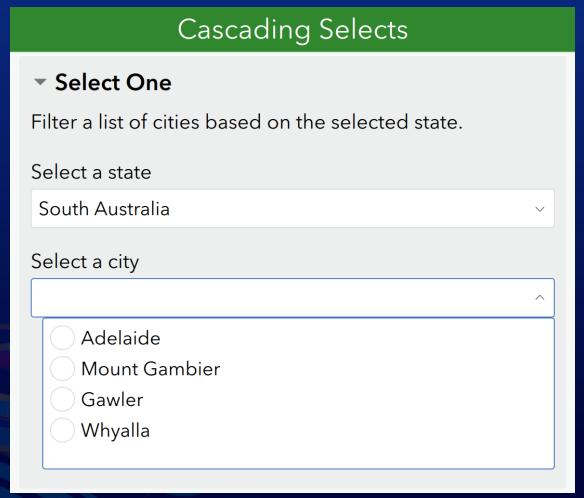
Advanced XLSForm Techniques

- Inbox workflows
- JavaScript functions
- Managing choice lists
- Image-map appearance

Managing Choice Lists

Background

- Choice lists provide the options for select_one and select_multiple questions.
- Lists can be managed in the XLSForm ('choices' or 'external_choices' worksheet), or as an external CSV file in the survey's media folder (eg 'select_one_from_file' question).
- Updating choice lists 'automatically' (using Python – see this blog post) has been cumbersome and requires users to continually update the survey on the device.
- Choice filters allow us to incorporate 'cascading selects' into our surveys.



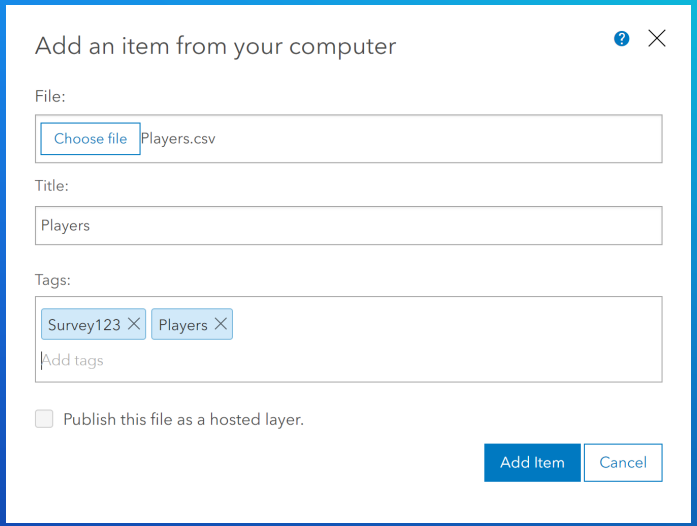
list_name	name	label	state
cities	melbourne	Melbourne	VIC
cities	ballarat	Ballarat	VIC
cities	geelong	Geelong	VIC
cities	bendigo	Bendigo	VIC
cities	kambah	Kambah	ACT
cities	kaleen	Kaleen	ACT
cities	ngunnawal	Ngunnawal	ACT
cities	waniassa	Waniassa	ACT
cities	perth	Perth	WA
cities	rockingham	Rockingham	WA
cities	mandurah	Mandurah	WA
cities	bunbury	Bunbury	WA
cities	adelaide	Adelaide	SA
cities	mount_gambier	Mount Gambier	SA
cities	gawler	Gawler	SA
cities	whyalla	Whyalla	SA
cities	hobart	Hobart	TAS
cities	launceston	Launceston	TAS

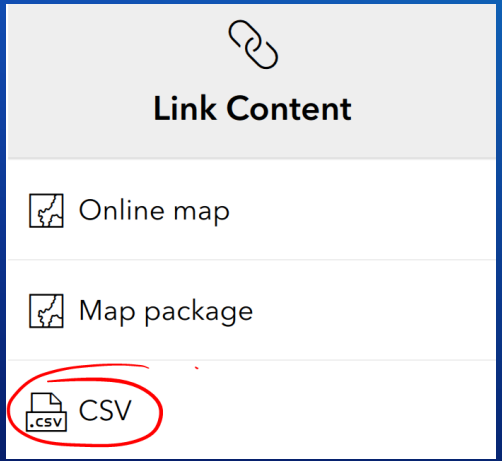
Linked Content (CSV)

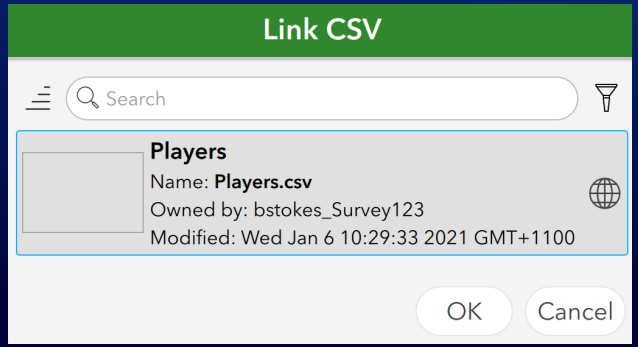
- No need to republish survey.
  - Used with *select_one_from_file* or *select_multiple_from_file*

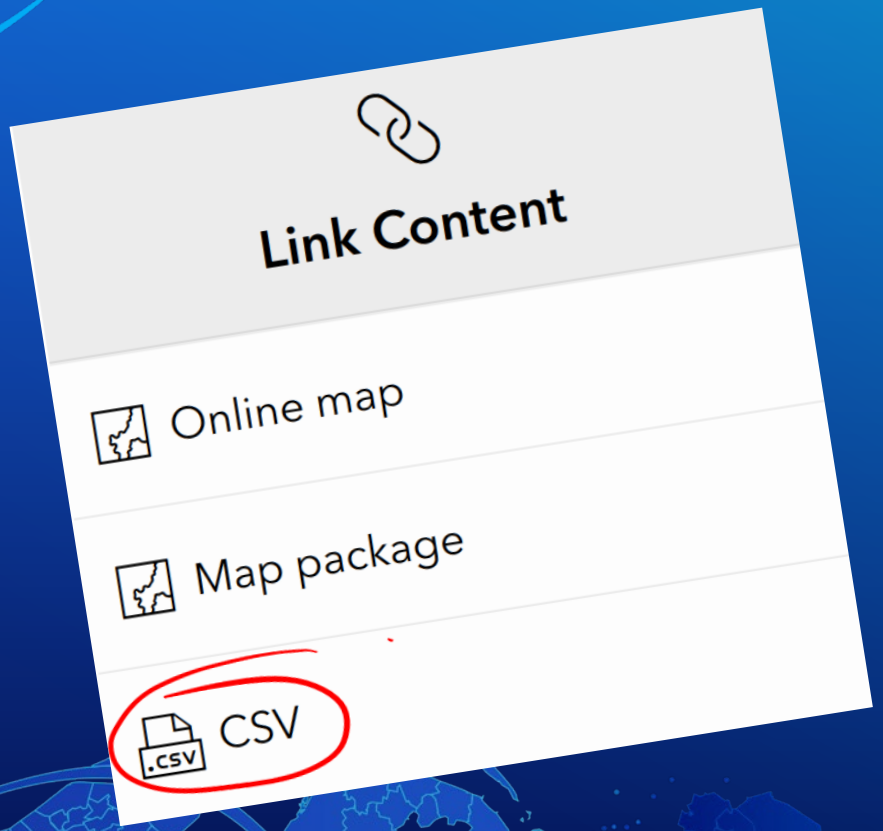
type	name	label
select_multiple_from_file players.csv	player	Choose a player

  - CSV item in your ArcGIS Organization
- The CSV is automatically downloaded / updated (if newer) from your ArcGIS Org when the survey is opened.
- Changes to the choice list contained in the CSV file can happen completely outside of Survey123 and will be used by your survey automatically.









Linked Content (CSV)

Demonstration

Linked Content (Map)

- Also used to link map content to a survey.
- More Information:

  GeoNet Blog – "Survey123 Tricks of the Trade: Configuring survey maps"

  (https://community.esri.com/t5/arcgis-survey123-blog/survey123-tricks-of-the-trade-configuring-survey-maps/ba-p/897815)

  Documentation – "Include a map in your survey"

  (https://doc.arcgis.com/en/survey123/desktop/create-surveys/includemap.htm)

  Documentation – "Prepare basemaps for offline use"

  (https://doc.arcgis.com/en/survey123/desktop/create-surveys/preparebasemaps.htm)

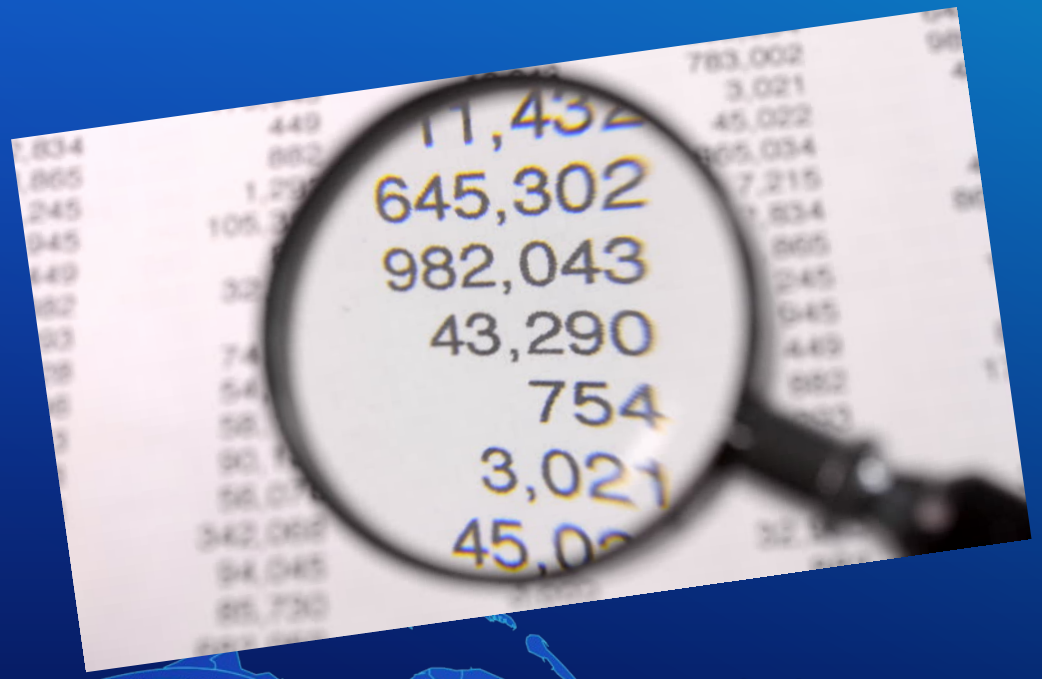
Dynamic Lists using the *search()* appearance

- Create choice lists dynamically from existing data.
- The full syntax has 6 elements (only tableName is required):

```
search(tableName, searchType, searchColumn, searchText, filterColumn, filterText)
```

Eg: *search(tableName)* will simply return the entire contents of the CSV file as a choice list.

- CSV in media folder
  - Valid searchType for non-spatial data include: ‘contains’, ‘startswith’, ‘endswith’, & ‘matches’.
- Feature Services containing spatial data:
  - tableName = <tablename>?url=<Layer REST URL> *don’t forget to include the layer index, for example:
    - *Parcels?url=https://services5.arcgis.com/jMCHJcLe13FaKCFB/ArcGIS/rest/services/Parcel_Boundaries/FeatureServer/0*
  - searchType – additional values include: ‘intersects’, ‘contains’, ‘overlaps’, ‘within’, ‘touches’, ‘crosses’, ‘disjoint’, ‘envelope_intersects’, and ‘index_intersects’.
  - searchColumn – ‘@geopoint’, ‘@geotrace’ and ‘@geotrace’ depending on the geometry you are querying.
  - searchText – can reference a previous geo* question if conducting a spatial query.



Search() Appearance

Demonstration

Dynamic Lists using the *search()* appearance

- Create choice lists dynamically from existing data.
- The full syntax has 6 elements (only tableName is required):

```
search(tableName, searchType, searchColumn, searchText, filterColumn, filterText)
```

Eg: *search(tableName)* will simply return the entire contents of the CSV file / feature service as a choice list.

- CSV in media folder
  - Valid searchType for non-spatial data include: ‘contains’, ‘startswith’, ‘endswith’, & ‘matches’.
- Feature Services containing spatial data:
  - tableName = <tablename>?url=<Layer REST URL> *don’t forget to include the layer index, for example:
    - *Parcels?url=https://services5.arcgis.com/jMCHJcLe13FaKCFB/ArcGIS/rest/services/Parcel_Boundaries/FeatureServer/0*
  - searchType – additional values include: ‘intersects’, ‘contains’, ‘overlaps’, ‘within’, ‘touches’, ‘crosses’, ‘disjoint’, ‘envelope_intersects’, and ‘index_intersects’.
  - searchColumn – ‘@geopoint’, ‘@geotrace’ and ‘@geotrace’ depending on the geometry you are querying.
  - searchText – can reference a previous geo* question if conducting a spatial query.

References

- **Documentation:**
  - https://links.esri.com/survey123/LinkedContent
- **GeoNet Blog:**
  - Improvement on performance with loading large choice lists
  - Dynamic choice lists

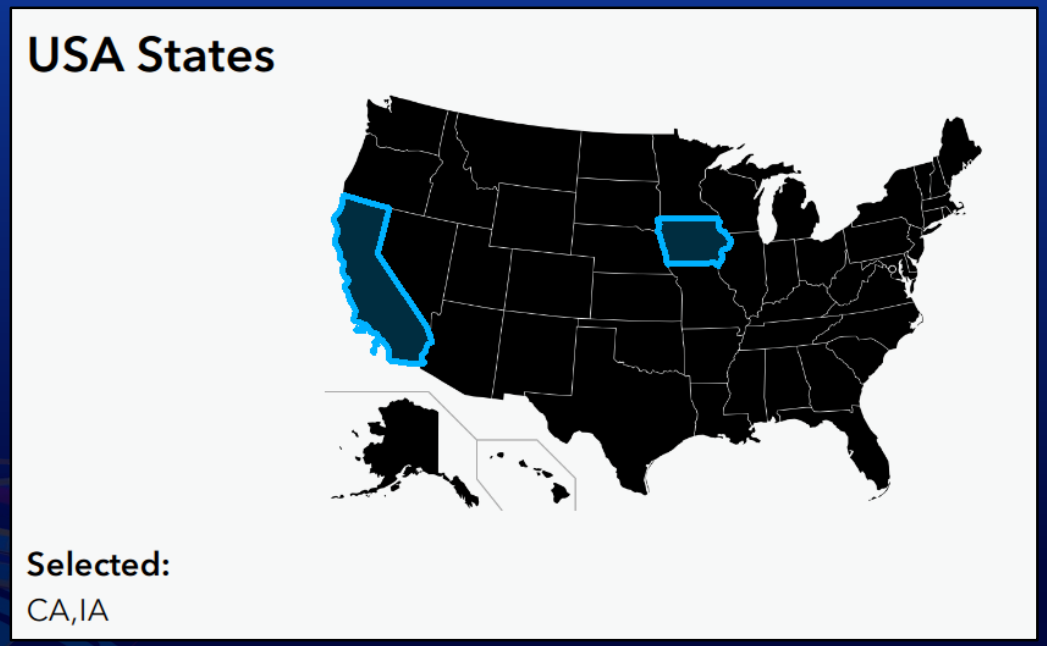
Advanced XLSForm Techniques

- Inbox workflows
- JavaScript functions
- Managing choice lists
- Image-map appearance

Image-map Appearance

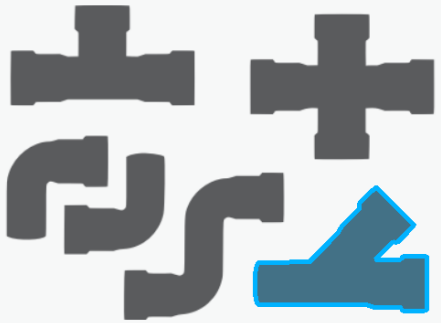
How is the image-map appearance used?

- When the image-map appearance is applied to a *select_one* or *select_multiple* question, it is displayed as an image instead of a list of text values.
- Areas in the image are selectable and correspond to a choice value.
- Even though the question is shown as an image, the responses are stored in the survey results in the same format as any other select_one or select_multiple question.



A few examples…

Pipe Fittings

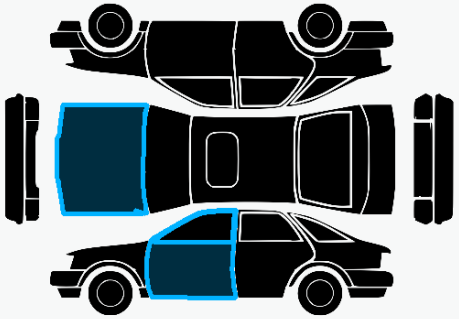


Selected:
Y-Junction

Lawn Variety

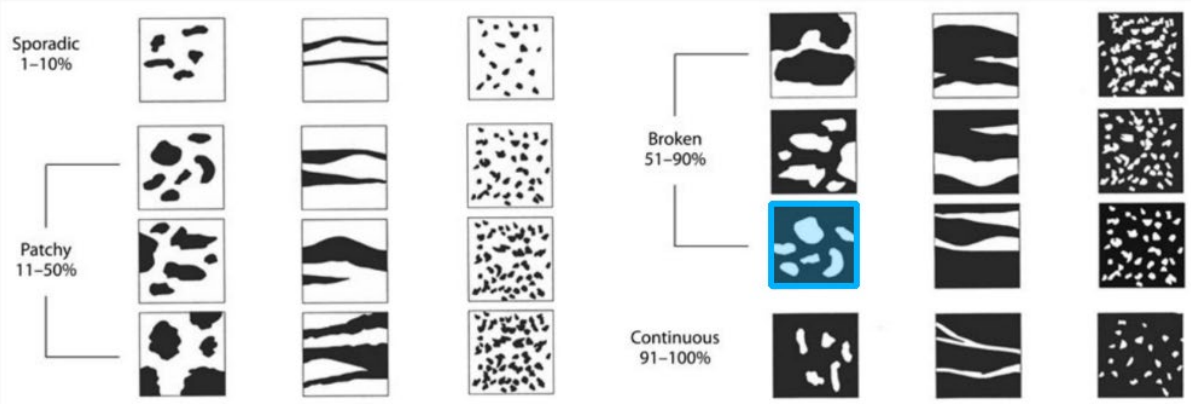
Selected:
Buffalo

Vehicle Damage



Selected: hood,front_left_door

Stranded Oil Distribution



ArcGIS Survey123 Connect

My Survey Designs | Tutorials | Community

+ New survey | Search

New Survey - ArcGIS Survey123 Connect

New Survey

Title

Form 47

Table name will be: **Form_47**

Select an initial XLSForm design

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

image-map

Image-map

This sample demonstrates the image-map appearance, which allows the use of SVG images for select_one and select_multiple questions.

Resource level:

Modified: Friday, October 16, 2020 2:06:06 AM Pacific Daylight Time

Type: Form

Owner: ArcGIS_Survey123

Sign in | Create survey | Cancel

Form 44 | Form 43 | testrun | Form 40

Form 39 | Report Streetlight Ma… | Herps | geonet_question_1006292

geocode.appearance (1) | Dynamic_CSV_itemset_test | geocode | dec_choice_data

How to configure the image-map appearance

1. Add **image-map** to the **appearance** column of your select_one / select_multiple.
2. Add the SVG filename (including '.svg') to the **media::image** column.
3. Ensure your choice list is set up and correctly referenced by the select_one or select_multiple question.
4. Copy the SVG file to the survey's **media** folder.

A17

	A	B	C	D	E	F
1	type	name	label	hint	appearance	media::image
2	select_multiple state	state_1	USA States		image-map	usa.svg
3						
4						

A22 state

	A	B	C
1	list_name	name	label
2	state	AK	Alaska
3	state	HI	Hawaii
4	state	AL	Alabama
5	state	AR	Arkansas
6	state	AZ	Arizona
7	state	CA	California
8	state	CO	Colorado
9	state	CT	Connecticut
10	state	DE	Delaware
11	state	FL	Florida
12	state	GA	Georgia
13	state	IA	Iowa

survey | choices | settings | Version | Question typ

SVG basics

- SVG: scalable vector graphic.
- XML-based.
- Can be edited in text editors, websites or vector graphics editing software.
- SVGs files are comprised of elements, which can be modified with attributes to define various aspects of the image.
  - For example The <svg> element is the outermost element in an SVG, and its “height” and “width” attributes define the ViewPort.

File Edit Format View Help

```
<svg viewBox="0 0 100 100"
height:"200"
width:"200"
 xmlns="http://www.w3.org/2000/svg">
  <path d="M 10,30
           A 20,20 0,0,1 50,30
           A 20,20 0,0,1 90,30
           Q 90,60 50,90
           Q 10,60 10,30 z"/>
</svg>
```

- SVGs can also contain raster images in the <image> element. This allows photos etc to be used as background images.

SVGs and Survey123 compatibility requirements

- **Requirements:**
  - Only one group per nest level (including the parent level).
  - No empty groups.
  - Must contain at least one path element with an id value matching a choice list value.
  - Must have a defined height and width (viewPort).
  - If you include a viewBox parameter, it must match the viewPort.

- **Recommendations:**
  - Avoid transform functions if possible.
  - Embed rasters that are used as background images.
  - Avoid having too many / too small selectable areas.

SVG files & Image-map

Demonstration

References

Documentation: **“Work with SVG files”**

(https://doc.arcgis.com/en/survey123/desktop/create-surveys/imagemap.htm)

GeoNet Blog: **“Using the image-map appearance in Survey123”**

(https://community.esri.com/t5/arcgis-survey123-blog/using-the-image-map-appearance-in-survey123/ba-p/897470)

Story Map: **“Spatial data to image-map in Survey123”**

(https://storymaps.arcgis.com/stories/9f27459bcac3408eab8472e212524b45)

esri®

THE SCIENCE OF WHERE®

Please provide your feedback for this session by clicking on the session survey link directly below the video.