Form Datasets

ODK forms can use datasets in a variety of ways. These datasets can be either internal or external to the form.

Internal datasets are defined in the **choices** sheet of an XLSForm and are typically used as choices for selects. You can also define a dataset in the **choices** sheet to look up values based on user input. Learn how to define internal datasets in the section on selects.

External datasets are useful when:

- data comes from another system. Using data files attached to the form generally requires fewer steps than adding the data to a form definition.
- data changes frequently. One or more data file attached to the form can be updated without modifying the form definition.
- data is reused between forms. It may be easier to attach the same data file to multiple forms instead of copying the data into all the form definitions.
- the same forms are used in different contexts. For example, the exact same form definition could be used in multiple countries with different data files listing regions, local products, etc.

Note

Most mobile devices released in 2019 or later can handle lists of 50,000 or more without slowdowns. If you experience slowdowns, please share the size of the dataset, the device you are using, and any expressions that reference the dataset on the community forum or to support@getodk.org.

Building selects from CSV files

CSV files can be used as datasets for select questions using `select_one_from_file` or `select_multiple_from_file`. CSV files used this way must have `name` and `label` columns unless you use the `parameters` column to customize the columns used. For each row in the dataset, the text in the value column (defaults to `name`) will be used as the value saved when that choice is selected and the text in the label column (defaults to `label`) will be used to display the choice. For select multiples, `name` must not contain spaces.

These files may also have any number of additional columns used in choice filters or other expressions. The example below uses one select from internal choices followed by selects from two different external CSV files.

survey

type	name	label	choice_filter
select_one states	state	State	
select_one_from_file lgas.csv	local_gov_area	Local Government Area	state=${state}
select_multiple_from_file wards.csv	wards	Wards	lga=${local_gov_area}

choices

list_name	name	label	population
states	abia	Abia	4112230
states	ebonyi	Ebonyi	2176947

lgas.csv

name	label	state
aba_n	Aba North	abia
aba_s	Aba South	abia
afikpo_n	Afikpo North	ebonyi

wards.csv

name	label	lga
eziama	Eziama	aba_n
umuogor	Umuogor	aba_n
ezeke_amasiri	Ezeke amasiri	afikpo_n
poperi_amasiri	Poperi amasiri	afikpo_n

Building selects from GeoJSON files

New in ODK Collect v2022.2.0, ODK Central v1.4.0

> ⚠ Warning
>
> GeoJSON attachments are not yet available in web forms (Enketo).

GeoJSON files that follow the GeoJSON spec can be used as datasets that populate select questions using `select_one_from_file`. Selects from GeoJSON may be styled as maps using the map appearance but can also use any other select appearance. In order to be used by a form, a GeoJSON file:

- must have a `.geojson` extension (NOT `.json`)
- must define a single top-level FeatureCollection
- must include a unique identifier for each feature (by default, a top-level `id`, falling back to an `id` property, or can be configured)
- must only include features with Point geometry

survey

type	name	label
select_one_from_file museums.geojson	museum	Select the museum closest to you

GeoJSON files referenced in forms can have any number of `features` and any number of custom `properties`.

```
{
  "type": "FeatureCollection",
  "features": [
      {
          "type": "Feature",
          "geometry": {
              "type": "Point",
              "coordinates": [
                  7.0801379,
                  46.5841618
              ]
          },
          "properties": {
              "id": "fs87b",
              "title": "HR Giger Museum",
              "annual_visits": 40000
          }
      }
  ]
}
```

All properties are displayed by select from map questions and can be referenced by any part of the form.

> ✎ Note
>
> There are two properties with a special meaning that can be used for styling markers (specifying its color and the displayed symbol):
>
> - **marker-color** - it should hold a valid hex color in the long or the short version (e.g. `#aaccee` or `#ace`).
> - **marker-symbol** - it should hold a single character (e.g. `A` or `7`).

A feature's `geometry` can be accessed as `geometry` and is provided in the ODK format. Given the GeoJSON file and the

form definition above, if the user selected "HR Giger Museum", the value of `${museum}` would be `"fs87b"`.

The expression `instance('museums')/root/item[id=${museum}]/geometry` would evaluate to `46.5841618 7.0801379 0 0` which is a point in the ODK format.

Building selects from XML files

XML files can be used as datasets that populate select questions using `select_one_from_file` or `select_multiple_from_file`. This is typically less convenient than using CSV files. However, knowing about the XML representation is helpful for understanding how to reference values in both CSV and XML files.

XML files used for selects must have the following structure and can have any number of `item` blocks:

```
<root>
  <item>
    <name>...</name>
    <label>...</label>
    ...
  </item>
  ...
</root>
```

The `item` blocks are analogous to rows in the CSV representation. Each `item` must have at least `name` and `label` nested nodes and can have any number of additional nodes. These nodes correspond to columns in the CSV representation.

Referencing values in datasets

XPath paths can be used to reference values in internal or external datasets. These paths will start with the `instance(<instance name>)` function to identify which dataset is being accessed. The next part of the path is generally `/root/item` because of the XML structure used to represent datasets for selects. The only exception is when using custom XML files which may have arbitrary schemas if not used for selects.

For internal datasets, the instance name is the `list_name` specified on the **choices** sheet. For example, to reference the population of the selected state given the form above, the instance name to use is `states`. The expression would be `instance("states")/root/item[name = ${state}]/population`. To understand this expression better, read the section on XPath paths and especially the subsection about XPath paths for filtering. You could also do things like count the number of states with a population above a certain threshold using an expression like `count(instance("states")/root/item[population > ${pop_threshold}])`.

> **Note**
>
> Due to a pyxform limitation, it is necessary for there to be some value in the *choice_filter* column (for at least one question) when referencing internal datasets. If none of the questions in your form need filtering, put *true()* as the *choice_filter* value.

For external datasets, the instance name is the filename specified in the `select_one_from_file` or `select_multiple_from_file` declaration without the file extension. For example, to look up a ward's label given the form above, the instance name to use is `wards` because the filename referenced is `wards.csv`. The expression would be `instance("wards")/root/item[name = ${ward}]/label`.