Form Logic

ODK Collect supports a wide range of dynamic form behavior. This document covers how to specify this behavior in your XLSForm definition.

See also

XLSForm

Warning

Relevance, constraint and calculation evaluation within the same screen is supported in Collect v1.22 and later. In earlier versions of Collect, questions tied by logic must be displayed on different screens.

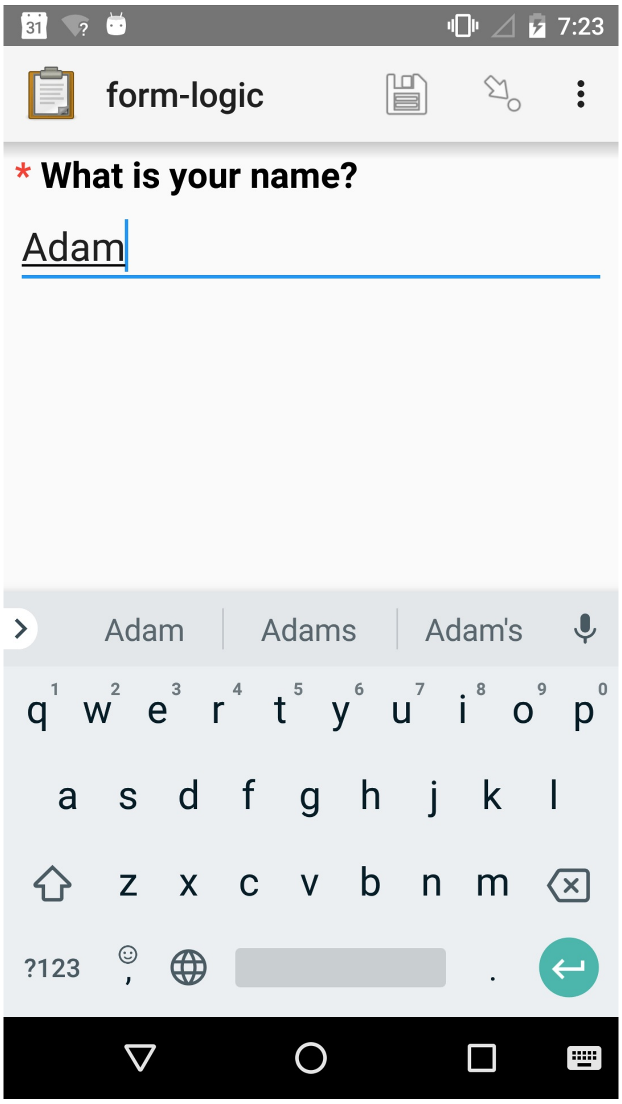
Form logic building blocks

Variables

Variables reference the value of previously answered questions. To use a variable in XLSForm, put the question's `name` in curly brackets preceded by a dollar sign:

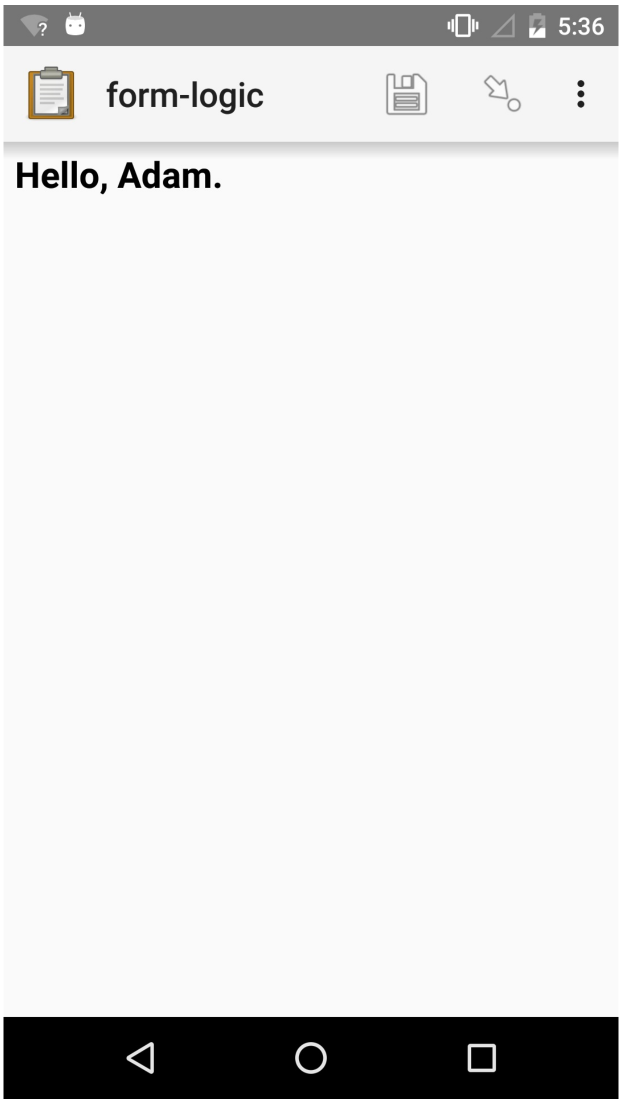
`${question-name}`

Variables can be used in `label`, `hint`, and `repeat_count` columns, as well as any column that accepts an expression.


form-logic

* What is your name?

Adam


form-logic

Hello, Adam.

XLSForm

survey

type	name	label
text	your_name	What is your name?
note	hello_name	Hello, ${your_name}.

You can also refer to the current question or to the current question's parent group or repeat:

	Explanation	Example	Notes
.	current question's value	. >= 18	Used in constraints.
..	current question's parent group	position(..)	Used with `position()` to get a parent repeat instance's index.

Advanced: XPath paths

The `${}` notation in XLSForm is a convenient shortcut to refer to a specific field. When an XLSForm is converted, `${}` references are expanded to XPath paths which describe where the field is located in the form.

Some tools like ODK Build do not support `${}` notation so XPath notation must be used. Even in XLSForm, it can be advantageous to use XPath notation, especially in the context of repeats or datasets. The `${}` and XPath notations can be mixed freely.

One way to think about XPath is that it sees a form or dataset like a series of folders and files on your computer. Questions are like files while groups and repeats are like folders because they can contain other elements. Path elements are separated by /. Imagine a form with a group with name `outer` which contains another group with name `inner` which contains a question with name `q1`. The absolute path to `q1` is `/data/outer/inner/q1`.

The `data` in the example above is the name of the form root. This root is named `data` by default but can be modified by adding a `name` column in the XLSForm **settings** sheet and specifying a value below it. This is rarely needed. The `/` at the start of the path indicates that the path is absolute.

XPath paths can also be relative. For example, let's say there's a `relevance` expression for `q1` in the example above and that this expression refers to a question with name `age` in the `outer` group. We could refer to it using an absolute expression: `/data/outer/age`. We could also write a relative expression: `../../age`.

The `../..` part of the relative expression says to go up two levels from the current position of `/data/outer/inner/q1`. The first `..` goes up one level to `/data/outer/inner` and then the second `..` goes up another level to `/data/outer/`. We want to access a question in the `outer` group so we add that question's name to get `../../age`.

ODK tools support a subset of XPath described in the ODK XForms specification.

XPath predicates for filtering

In repeats and datasets, an XPath path can refer to multiple nodes. This is called a nodeset. XPath predicates are True/False (boolean) expressions in square brackets that filter the nodeset they come after. When you define a choice filter for a select, that expression is used as an XPath predicate to filter the choice items.

You can also write your own expressions with predicates. For example, consider a form with a repeat with name `people` and a question inside with name `age` (see XPath paths for repeats for the form definition). The expression `/data/people[age < 18]` evaluates to a nodeset that includes all instances of the `people` instance for which the value of the `age` question is less than 18. `age` in the predicate is a relative expression evaluated in the context of each node in the nodeset. In this case, the relative expression `age` is evaluated in the context of `/data/people`, giving the path expression `/data/people/age`. This means that `/data/people/age` is compared to 18 for every `people` repeat instance.

You can add more path steps after a predicate. For example, `/data/people[age < 18]/pet_count` evaluates to a nodeset that includes all the pet counts for instances of the `people` repeat that have `age` values under 18. Nodesets can be passed in to functions like `sum()` or other functions that take nodeset arguments.

Sometimes forms may use groups to organize question sections within repeats. Those groups must be accounted for in predicates. If the `age` question were nested in a group called `inner`, the predicate expression would need to be `inner/age < 18`. Additionally, if the `pet_count` question were nested in a group called `details`, the full expression would be `/data/people[inner/age < 18]/details/pet_count`.

XPath predicates are also the way to reference specific values in a dataset. Learn more in the section on referencing values in datasets.

XPath paths for repeats

When a form definition includes a repeat, corresponding filled forms will have 0 or more instances of that repeat. Using the file and folder analogy described above, each repeat instance is like a folder and all of these folders have the name of the repeat. Repeat instances are differentiated by their index (first, second, ...).

When writing expressions within a repeat, it can be helpful to use the position of the repeat instance an enumerator is currently filling out. This can be done by using the `position()` function. One context in which this is useful is if you want to first collect a roster of people or things and then ask additional questions about each of those. As shown in the example in the `position()`, you can use a first repeat for the roster and then a second repeat that references items in the first repeat based on their position.

Another use of the `position` function is to access a preceding repeat instance. See an example of this in the section on dynamic defaults in repeats.

XPath paths can be useful to reference some or all repeat instances from outside the repeat. XPath notation is particularly helpful for filtering repeat instances, for example to provide a summary from data collected in repeats:

XLSForm

survey

type	name	label	calculation
begin repeat	people	Person	
int	age	Age	
int	pet_count	How many pets does this person have?	
end repeat	people		
int	total_pets		sum(${people}[age < 18]/pet_count)
note	total_note	Total pets owned by children: ${total_pets}	

In the path expression `${people}[age < 18]/pet_count`, `${people}` uses `${}` notation to refer to all of the instances of the repeat. You could also expand this to the XPath path of */data/people*. See the section on XPath predicate for more details. In this example, the `total_pets` value is displayed to the user. It could be used in many different contexts such as to define the relevance of a group if there's a section of questions that only need to be filled out if there are more than one child-owned pets in the community.

Expressions

An *expression* is evaluated dynamically as a form is filled out. It can include XPath functions, operators, values from previous responses, and (in some cases) the value of the current response.

Example expressions

`${bill_amount} * 0.18`
: Multiplies the previous value `bill_amount` by 18%, to calculate a suitable tip.

`concat(${first_name}, ' ', ${last_name})`
: Concatenates two previous responses with a space between them into a single string.

`${age} >= 18`
: Evaluates to `True` or `False`, depending on the value of `age`.

`round(${bill_amount} * ${tip_percent} * 0.01, 2)`
: Calculates a tip amount based on two previously entered values, and then rounds the result to two decimal places.

Expressions are used in:

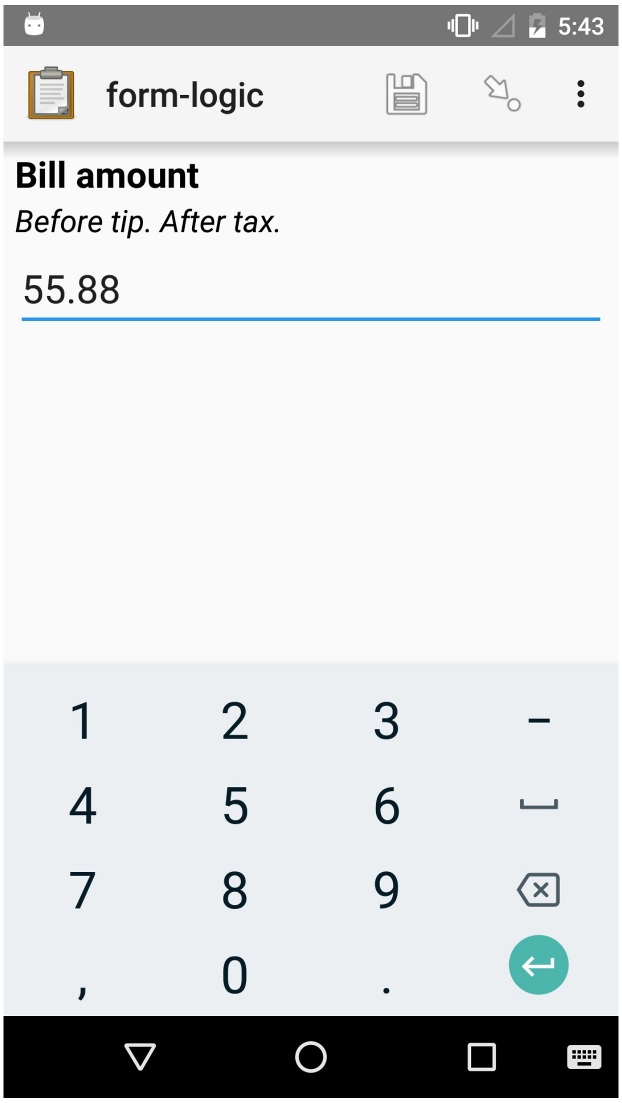
- Calculations
- Validating and restricting responses
- Conditionally showing questions

Calculations

To evaluate complex expressions, use a `calculate` row. Put the expression to be evaluated in the `calculation` column.

Then, you can refer to the calculated value using the calculate row's `name`.

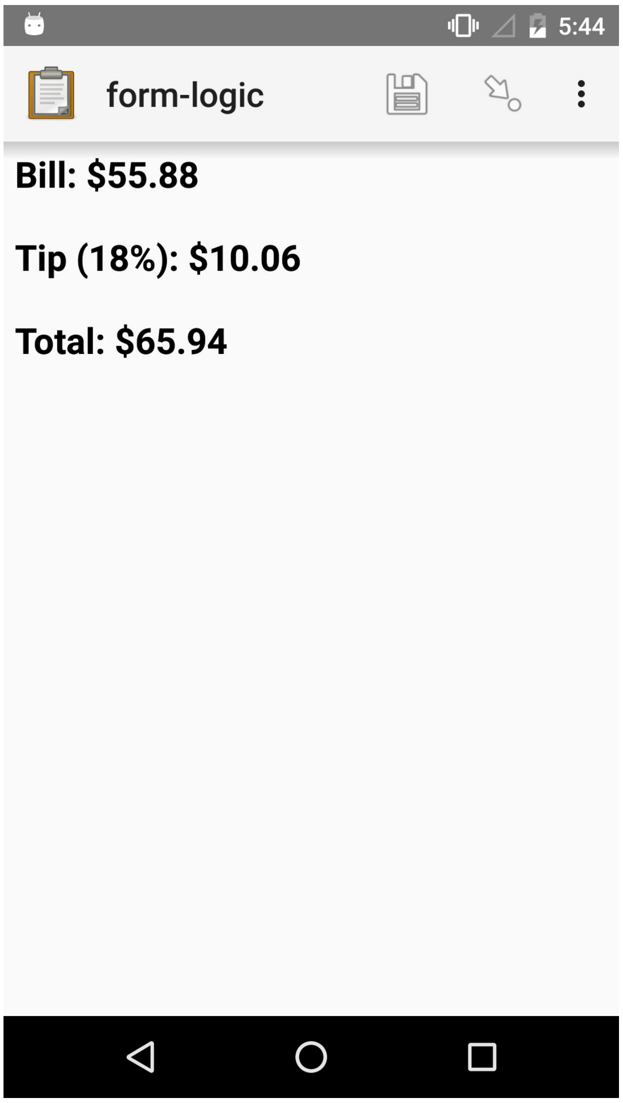
Expressions cannot be used in `label` and `hint` columns, so if you want to display calculated values to the user, you must first use a `calculate` row and then a variable.



Bill amount

Before tip. After tax.

55.88



Bill: $55.88

Tip (18%): $10.06

Total: $65.94

XLSForm

survey

type	name	label	calculation
decimal	bill_amount	Bill amount:	
calculate	tip_18		round((${bill_amount} * 0.18),2)
calculate	tip_18_total		${bill_amount} + ${tip_18}

note	tip_18_note	Bill: $${bill_amount}
		Tip (18%): $${tip_18}
		Total: $${tip_18_total}	

Values from the last saved record

⚠ Warning

We only recommend using last saved values as defaults. References to the last saved record could be used as part of any expression wherever expressions are allowed but this may lead to unexpected results on submission edit when the last saved record is likely to have changed.

The last-saved feature does not work with encrypted forms.

Support for last-saved was added in Collect v1.21.0 and Central v1.3.0. Using older versions or encrypted forms will have unpredictable results.

You can refer to values from the last saved record of this form definition:

`${last-saved#question-name}`

This can be very useful when an enumerator has to enter the same value for multiple consecutive records. An example of this would be entering in the same district for a series of households.

XLSForm that shows using a last-saved value as a dynamic default

survey

type	name	label	default
text	street	Street	${last-saved#street}

The value is pulled from the last saved record. This is often the most recently created record but it could also be a previously-existing record that was edited and saved. For the first record ever saved for a form definition, the last saved value for any field will be blank.

Questions of any type can have their defaults set based on the last saved record.

Warning

Last-saved copies over literal answer values and not binary attachments so it won't really work well with binary questions. The filename will be copied over but the actual file won't be available to Collect.

Form logic gotchas

When expressions are evaluated

Every expression is constantly re-evaluated as an enumerator progresses through a form. This is an important mental model to have and can explain sometimes unexpected behavior. More specifically, expressions are re-evaluated when:

- a form is opened
- the value of any question in the form changes
- a repeat group is added or deleted
- a form is saved or finalized

This is particularly important to remember when using functions that are not connected to fields in the form such as `random()` or `now()`. The value they represent may change as the conditions listed above take place.

To control when an expression is evaluated, use dynamic defaults or trigger calculations on value change. Dynamic defaults are evaluated exactly once on form load or repeat creation.

Empty values in math

Unanswered number questions are nil. That is, they have no value. When a variable referencing an empty value is used in a math operator or function, it is treated as Not a Number (`NaN`). The empty value **will not** be converted to zero. The result of a calculation including `NaN` will also be `NaN`, which may not be the behavior you want or expect.

To convert empty values to zero, use either the `coalesce()` function or the `if()` function.

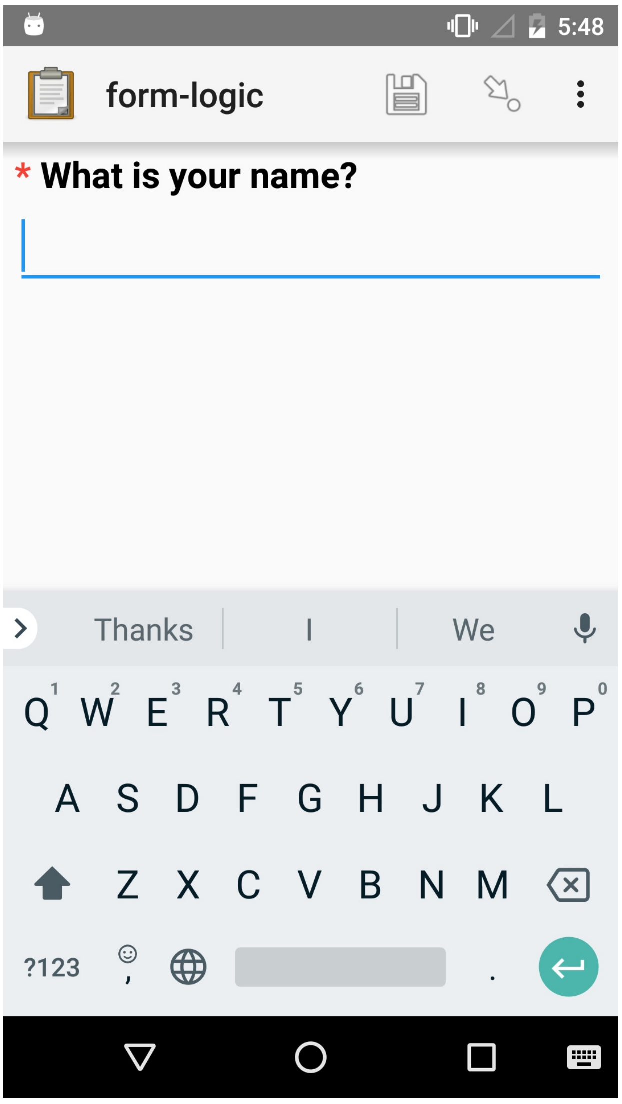
```
coalesce(${potentially_empty_value}, 0)
```

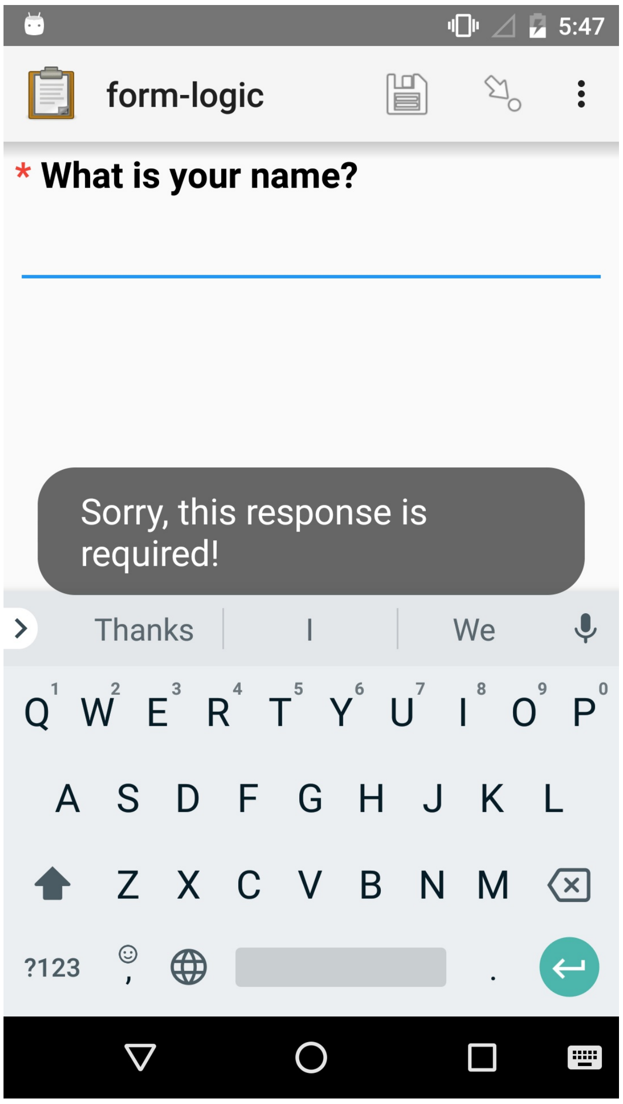
```
if(${potentially_empty_value}="", 0, ${potentially_empty_value})
```

Requiring responses

By default, users are able to skip questions in a form. To make a question required, put `yes` in the `required` column.

Required questions are marked with a small asterisk to the left of the question label. You can optionally include a `required_message` which will be displayed to the user who tries to advance the form without answering the question.





XLSForm

survey

type	name	label	required	required_message
text	name	What is your name?	yes	Please answer the question.

Setting default responses

To provide a default response to a question, put a value in the `default` column. Defaults are set when a record is first created from a form definition. Defaults can either be fixed values (static defaults) or the result of some expression (dynamic defaults).

> ⚠ Warning
>
> Defaults are not supported for media question types. The only exception is that images can have static defaults. This can be useful for annotations.

Static defaults

The text in the `default` column for a question is taken literally as the default value. Quotes should **not** be used to wrap values, unless you actually want those quote marks to appear in the default response value.

In the example below, the "Phone call" option with underlying value `phone_call` will be selected when the question is first displayed. The enumerator can either keep that selection or change it.

XLSForm to select "Phone call" as the default contact method

survey

type	name	label	default
select_one contacts	contact_method	How should we contact you?	phone_call

choices

list_name	name	label
contacts	phone_call	Phone call
contacts	text_message	Text message
contacts	email	Email

Dynamic defaults

Warning

Support for dynamic defaults was added in Collect v1.24.0 and Central v1.0.0. Using older versions will have unpredictable results.

If you put an expression in the `default` column for a question, that expression will be evaluated once when a record is first created from a form definition. This allows you to use values from outside the form like the current date or the server username. Dynamic defaults as described in this section are evaluated once on record creation. See below for using dynamic defaults in repeats or setting the default value of one field to the value of another field in the form.

XLSForm to set the current date as default

survey

type	name	label	default
date	fever_onset	When did the fever start?	now()

In the example below, if a username is set either in the server configuration or the metadata settings, that username will be used as the default for the question asked to the enumerator.

XLSForm to confirm metadata like username

survey

type	name	label	default
username	username		
text	confirmed_username	What is your username?	${username}

Tip

If enumerators will need to enter the same value for multiple consecutive records, dynamic defaults can be combined with last saved. For example, if enumerators are collecting data about trees and trees of the same kind grow together, you can use the last saved tree species as the default for new records.

Dynamic defaults in repeats

Dynamic defaults in repeats are evaluated when a new repeat instance is added.

One powerful technique is to use a value from a previous repeat instance as a default for the current repeat instance. For example, you could use the tree species specified for the last visited tree as the default species for the next tree.

Note

If you are collecting data about multiple entities such as trees, you can choose to use repeats or to use one form record per entity. See the repeats section for more information on making that decision. If you use one form record per entity, you can use last saved to get the same behavior as described in this section.

XLSForm to set a default value based on the last repeat instance

survey

type	name	label	default
begin_repeat	tree	Tree	
text	species	Species	${tree}[position() = position(current()/..) - 1]/species

In the default expression above, `${tree}` is a reference to the repeat. The expression `[position() =`

`position(current()/..) - 1]` in brackets says to filter the list of possible `tree` repeat instances to only include the one with a position that is one less than the current repeat's position. Finally, `/species` specifies that the `species` question from the repeat should be used. This is a mix of XLSForm's `${}` shortcut syntax for specifying question names and raw XPath syntax.

Dynamic defaults from form data

Warning

Support for dynamic defaults from form data was added in Collect v1.24.0 and Central v1.1.0. Using older versions will have unpredictable results.

It can be helpful to use a value filled out by the enumerator as a default for another question that the enumerator will later fill in. Dynamic defaults as described above can't be used for this because they are evaluated on form or repeat creation, before any data is filled in.

You also **can't use the calculation column on its own for this** because the expression in the `calculation` would be evaluated on form save and replace any changes the enumerator has made. Instead, use a combination of `calculation` and `trigger`. The question reference in the `trigger` column will ensure that your `calculation` is only evaluated when that reference changes.

XLSForm that uses current age as the default for diagnosis age

survey

type	name	label	calculation	trigger
text	name	Child's name		
integer	current_age	Child's age		
select_one gndr	gender	Gender		
integer	diagnosis_age	Age at malaria diagnosis	${current_age}	${current_age}

In the example above, `${current_age}` in the `trigger` column means that when the value of the `current_age` question is changed by the enumerator, the `calculation` for the `diagnosis_age` question will be evaluated. In this case, this means the new value for `current_age` will replace the current value for `diagnosis_age`. If the enumerator then changes the value for `diagnosis_age`, this value will be retained unless the value for `current_age` is changed again.

Another option for the scenario above is to clear out the value for `diagnosis_age` when `current_age` changes. Making `diagnosis_age` a required question will force the enumerator to update `diagnosis_age` if `current_age` is corrected.

XLSForm that clears diagnosis age if current age is updated

survey

type	name	label	required	calculation	trigger
text	name	Child's name			
integer	current_age	Child's age			
select_one gndr	gender	Gender			
integer	diagnosis_age	Age at malaria diagnosis	true()	''	${current_age}

In the example above, `diagnosis_age` is cleared any time the value of the `current_age` question is changed.

This kind of default is particularly useful if a form is being filled in about entities that there is already some knowledge about. For example, if you have a list of people to interview and you know their phone numbers, you may want to use the known phone number as a default and allow the enumerator to update it as needed.

XLSForm that looks up default values based on a selection

survey

type	name	label	calculation	
select_one participants	participant	Participant		
text	phone_number	Phone number	instance('participants')/root/item[name=${participant}]/phone_number	$

choices

list_name	name	label	phone_number
participants	kwame_onwuachi	Kwame Onwuachi	+1-850-555-0168
participants	sophia_roe	Sophia Roe	+36 55 562 079

In the example above, when a participant is selected, his or her phone number is populated as a default and can be updated as needed. If the selected participant changes, the phone number is replaced.

Note

The `true()` in the `choice_filter` column for the `select_one` in the example above is necessary to be able to look up participants' phone numbers. This is currently needed to overcome a `pyxform` bug.

Triggering calculations on value change

Warning

Support for triggering calculations on value change was added in Collect v1.24.0 and Central v1.1.0. Using older versions will have unpredictable results.

Calculations are recomputed any time one of the values in its expression changes. For example, if your form includes the calculation *${q1} + ${q2}*, it will be recomputed any time either of the values for *${q1}* or *${q2}* changes.

Calculations can also be triggered when a value not involved in the calculation changes. This uses the same `trigger` column as defaults from form data. It is particularly useful for triggering calculations that involve values not in the form like random numbers or time.

Lightweight timestamping

Knowing how long an enumerator spent answering a question can help with quality control and training. ODK Collect provides an audit log that contains rich information about how an enumerator navigated a form. This log is captured as a separate file and can be complex to analyze. A simpler alternative is to capture timestamps when specific questions' values change. This is similar to the `start` and `end` timestamp metadata types.

Capturing last update timestamps

survey

type	name	label	calculation	trigger
string	name	Name		
dateTime	name_last_update_time		now()	${name}
string	life_story	Life story		
dateTime	life_story_last_update_time		now()	${life_story}

In the example above, the `name_last_update_time` field will be populated with the current time whenever the enumerator changes the value of the `name` question.

You can also capture the first time a question's value is changed using an `if` in the *calculation*:

Capturing first update timestamps

survey

type	name	label	calculation	trigger
string	name	Name		
dateTime	name_first_update_time		if(${name_first_update_time}='', now(), ${name_first_update_time})	${name}
string	life_story	Life story		
dateTime	life_story_first_update_time		if(${life_story_first_update_time}='', now(), ${life_story_first_update_time})	${life_story}

Validating and restricting responses

To validate or restrict response values, use the `constraint` column. The `constraint` expression will be evaluated when the user advances to the next screen. If the expression evaluates to `True`, the form advances as usual. If `False`, the form does not advance and the `constraint_message` is displayed.

The entered value of the response is represented in the expression with a single dot (`.`).

Constraint expressions often use comparison operators and regular expressions. For example:

`. >= 18`
: True if response is greater than or equal to 18.

`. > 20 and . < 200`
: True if the response is between 20 and 200.

`regex(.,'\p{L}+')`
: True if the response only contains letters, without spaces, separators, or numbers.

`not(contains(., 'prohibited'))`
: True if the substring `prohibited` does not appear in the response.

`not(selected(., 'none') and count-selected(.) > 1)`
: False if the response includes `none` and any other choice.

Note

Constraints are not evaluated if the response is left blank. To restrict empty responses, make the question required.

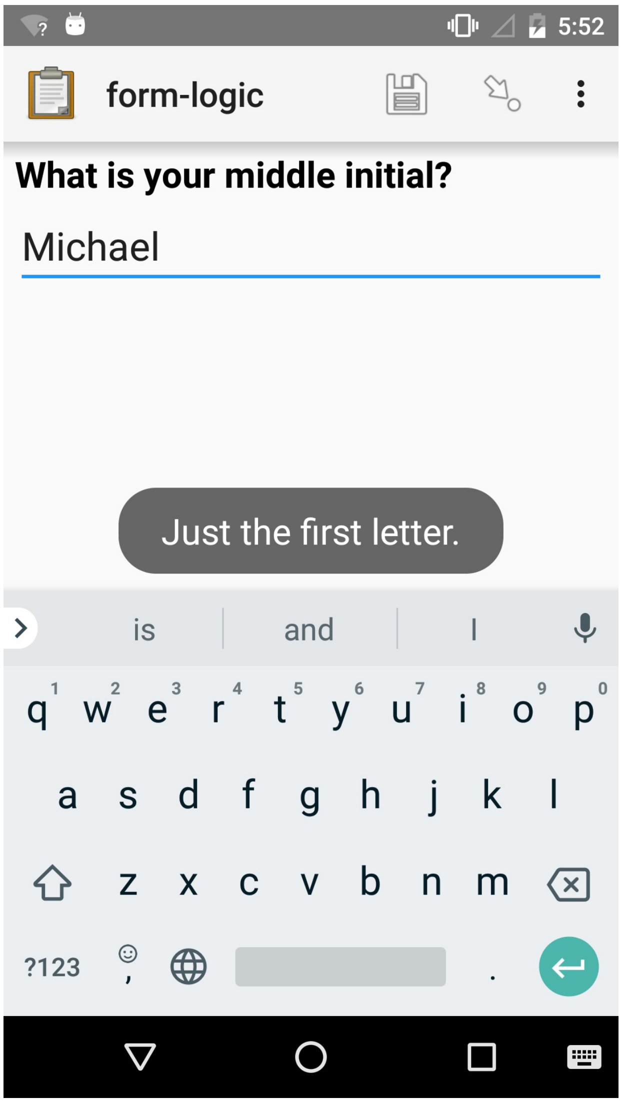
Tip

For "soft" constraints or warnings, use a note question that is relevant when the "soft" constraint is violated. For example, you can show a note that participant's age of 110 is allowed, but unlikely.

Notes can also be used for "hard" constraints that should be permanently displayed until they are resolved by using the technique above and setting required to `true()`. For example, you can show a note if a percentage total is not 100 and ask the enumerator to correct the input values.

See also

Using regular expressions



What is your middle initial?

Michael

Just the first letter.

XLSForm

survey

type	name	label	constraint	constraint_message
text	middle_initial	What is your middle initial?	regex(., 'p{L}')	Just the first letter.

Read-only questions

To completely restrict user-entry, use the `read_only` column with a value of `yes`. This is usually combined with a default response, which is often calculated based on previous responses.

XLSForm

survey

type	name	label	read_only	default	calculation
decimal	salary_income	Income from salary			
decimal	self_income	Income from self-employment			
decimal	other_income	Other income			
calculate	income_sum				${salary_income} + ${self_income} + ${other_income}
decimal	total_income	Total income	yes	${income_sum}	

Conditionally showing questions

The `relevant` column can be used to show or hide questions and groups of questions based on previous responses.

If the expression in the `relevant` column evaluates to `True`, the question or group is shown. If `False`, the question is skipped.

Often, comparison operators are used in relevance expressions. For example:

`${age} <= 5`

True if `age` is five or less.

`${has_children} = 'yes'`

True if the answer to `has_children` was `yes`.

Relevance expressions can also use functions. For example:

`selected(${allergies}, 'peanut')`

True if `peanut` was selected in the Multi select widget named `allergies`.

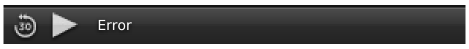
`contains(${haystack}, 'needle')`

True if the exact string `needle` is contained anywhere inside the response to `haystack`.

`count-selected(${toppings}) > 5`

True if more than five options were selected in the Multi select widget named `toppings`.

Simple example



XLSForm

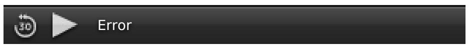
survey

type	name	label	relevant
select_one yes_no	watch_sports	Do you watch sports?	
text	favorite_team	What is your favorite team?	${watch_sports} = 'yes'

choices

list_name	name	label
yes_no	yes	Yes
yes_no	no	No

Complex example



XLSForm

survey

type	name	label	hint	relevant	constraint
select_multiple medical_issues	what_issues	Have you experienced any of the following?	Select all that apply.		
select_multiple cancer_types	what_cancer	What type of cancer have you experienced?	Select all that apply.	selected(${what_issues}, 'cancer')	
select_multiple diabetes_types	what_diabetes	What type of diabetes do you have?	Select all that apply.	selected(${what_issues}, 'diabetes')	
begin_group	blood_pressure	Blood pressure reading	selected(${what_issues}, 'hypertension')		
integer	systolic_bp	Systolic			. > 40 and . < 400
integer	diastolic_bp	Diastolic			. >= 20 and . <= 200
end_group					
text	other_health	List other issues.		selected(${what_issues}, 'other')	
note	after_health_note	This note is after all health questions.			

choices

list_name	name	label
medical_issues	cancer	Cancer
medical_issues	diabetes	Diabetes
medical_issues	hypertension	Hypertension
medical_issues	other	Other
cancer_types	lung	Lung cancer
cancer_types	skin	Skin cancer
cancer_types	prostate	Prostate cancer
cancer_types	breast	Breast cancer
cancer_types	other	Other
diabetes_types	type_1	Type 1 (Insulin dependent)
diabetes_types	type_2	Type 2 (Insulin resistant)

Groups of questions

To group questions, use the `begin_group...end_group` syntax.

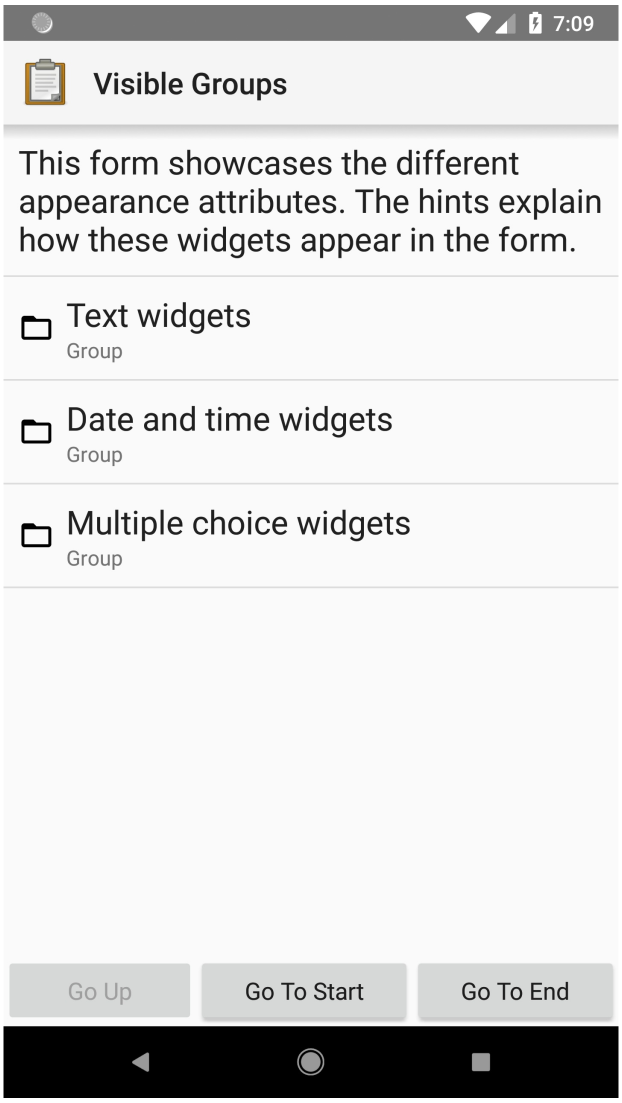
XLSForm — Question group

survey

type	name	label
begin_group	my_group	My text widgets
text	question_1	Text widget 1
text	question_2	These questions will both be grouped together
end_group		

You can use the `field-list` appearance on a group to display multiple questions on the same screen.

If given a `label`, groups will be visible in the form path to help orient the user (e.g. My text widgets > Text widget 1). Labeled groups will also be visible as clickable items in the jump menu:


7:09

Visible Groups

This form showcases the different appearance attributes. The hints explain how these widgets appear in the form.

Text widgets
Group

Date and time widgets
Group

Multiple choice widgets
Group

Warning

If you use ODK Build v0.3.4 or earlier, your groups will not be visible in the jump menu. The items inside the groups will display as if they weren't grouped at all.

Groups without labels can be helpful for organizing questions in a way that's invisible to the user. This technique can be helpful for internal organization of the form. These groups can also be a convenient way to conditionally show certain questions.

Repeating questions

You can ask the same question or questions multiple times by wrapping them in `begin_repeat...end_repeat`. By default, enumerators are asked before each repetition whether they would like to add another repeat. It is also possible to determine the number of repetitions ahead of time which can make the user interaction more intuitive. You can also add repeats as long as a condition is met.

XLSForm — Repeating one or more questions

survey

type	name	label
begin_repeat	my_repeat	repeat group label
note	repeated_note	All of these questions will be repeated.
text	name	What is your name?
text	quest	What is your quest?
text	fave_color	What is your favorite color?
end_repeat		

Warning

You can apply the `field-list` appearance to a repeat to make all of the repeated questions go on a single screen. However, you can't have a `repeat` inside of another group with the `field-list` appearance.

See also

Repeat Recipes and Tips describes strategies to address common repetition scenarios.

Tip

Using repetition in a form is very powerful but can also make training and data analysis more time-consuming. Repeats exported from Central or Briefcase will be in their own files and will need to be joined with their parent records for analysis.

Before adding repeats to your form, consider other options:

- if the number of repetitions is small and known ahead of time, consider "unrolling" the repeat by copying the same questions several times.
- if the number of repetitions is large and includes many questions, consider building a separate form that enumerators fill out multiple times and link the forms with some parent key (e.g., a household ID).

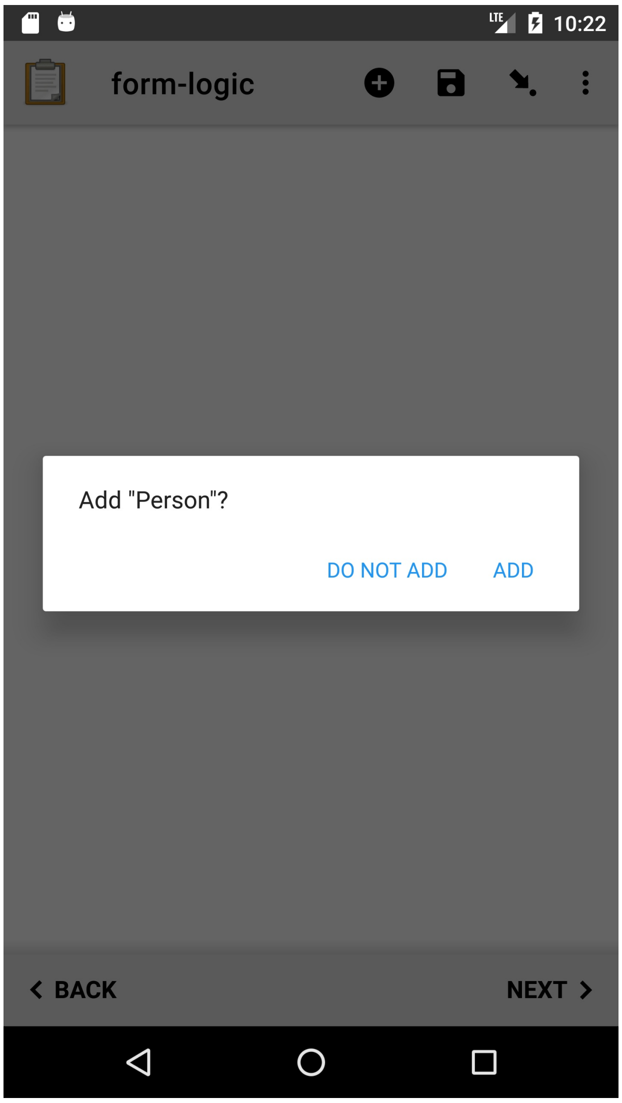
If repeats are needed, consider adding some summary calculations at the end so that analysis will not require joining the repeats with their parent records. For example, if you are gathering household information and would like to compute the total number of households visited across all enumerators, add a calculation after the repeats that counts the repetitions in each submission.

Controlling the number of repetitions

User-controlled repeats

By default, the enumerator controls how many times the questions are repeated.

Before each repetition, the user is asked if they want to add another. The user is given the option to add each repetition.



Add "Person"?

DO NOT ADD   ADD

Note

The `label` in the `begin_repeat` row is shown in quotes in the **Add ...?** message.

A meaningful label will help enumerators and participants navigate the form as intended. We generally recommend using a singular noun or noun phrase such as "observation" or "household member".

This interaction may be confusing to users the first time they see it. If enumerators know the number of repetitions ahead of time, consider using a dynamically defined repeat count.

Tip

The jump menu also provides shortcuts to add or remove repeat instances.

Fixed repeat count

Use the `repeat_count` column to define the number of times that questions will repeat.

XLSForm

survey

type	name	label	repeat_count
begin_repeat	my_repeat	Repeat group label	3
note	repeated_note	These questions will be repeated exactly three times.	
text	name	What is your name?	
text	quest	What is your quest?	
text	fave_color	What is your favorite color?	
end_repeat			

Dynamically defined repeat count

The `repeat_count` column can reference previous responses and calculations.

XLSForm

survey

type	name	label	repeat_count
integer	number_of_children	How many children do you have?	
begin_repeat	child_questions	Questions about child	${number_of_children}
text	child_name	Child's name	
integer	child_age	Child's age	
end_repeat			

Repeating as long as a condition is met

If the enumerator won't know how many repetitions are needed ahead of time, you can still avoid the "Add ...?" dialog by using the answer to a question to decide whether another repeat instance should be added. In the example below, repeated questions about plants will be asked as long as the user answers "yes" to the last question.

survey

type	name	label	calculation	repeat_count
calculate	count		count(${plant})	
begin_repeat	plant	Plant		if(${count} = 0 or ${plant}[position()=${count}]/more_plants = 'yes', ${count} + 1, ${count})
text	species	Species		
integer	estimated_size	Estimated size		
select_one yes_no	more_plants	Are there more plants in this area?		
end_repeat				

choices

list_name	name	label
yes_no	yes	Yes
yes_no	no	No

This works by maintaining a `count()` of the existing repetitions and either making `repeat_count` one more than that if the continuing condition is met or keeping the `repeat_count` the same if the ending condition is met.

In the *repeat_count* expression, *${count} = 0* ensures that there is always at least one repeat instance created. The continuing condition is *${plant}[position()=${count}]/more_plants = 'yes'* which means "the answer to *more_plants* was *yes* the last time it was asked." The expression *position()=${count}* uses the `position()` function to select the last plant that was added. Adding */more_plants* to the end of that selects the *more_plants* question.

Repeating zero or more times

Sometimes it only makes sense to collect information represented by the questions in a repeat under certain conditions. If the number of total repetitions is known ahead of time, use Dynamically defined repeat count and allow a count of 0. If the count is not known ahead of time, Conditionally showing questions can be used to represent 0 or more repetitions. In the example below, questions about trees will only be asked if the user indicates that there are trees to survey.

survey

type	name	label	relevant
select_one yes_no	trees_present	Are there any trees in this area?	
begin_repeat	tree	Tree	${trees_present} = 'yes'
text	species	Species	
integer	estimated_age	Estimated age	
end_repeat			

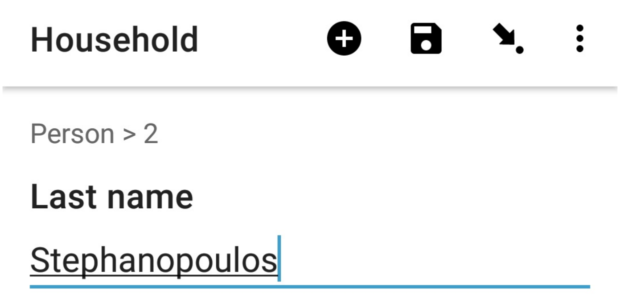
choices

list_name	name	label
yes_no	yes	Yes
yes_no	no	No

Naming repeats to help with navigation

The form location summary at the top of a form question shows the labels of all groups and repeats that the question is in and the index of any repeats.

For example, in a repeat collecting data about different people, the location summary for the second Person is `Person > 2`.

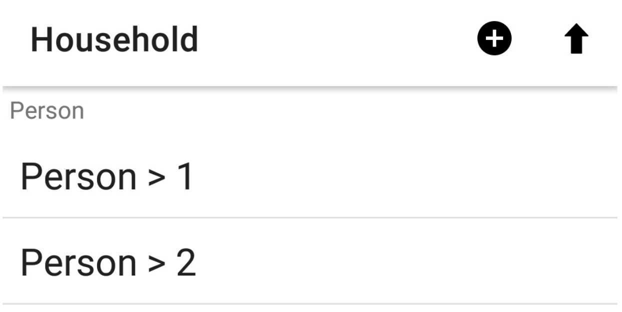


Person > 2

Last name

Stephanopoulos

The location summary is also shown in the jump menu.



Person

Person > 1

Person > 2

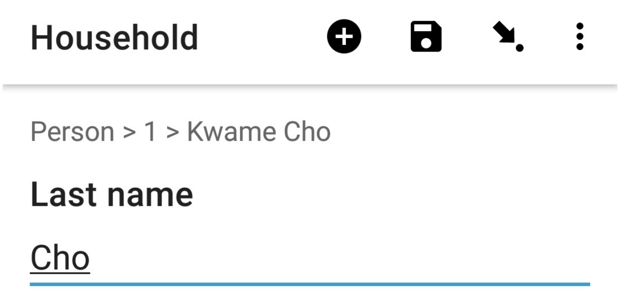
In many cases, the repeat index (2 in the above example) is enough information to give context to data collectors. However, in workflows where data collectors will need to jump between repeat instances, it can help to name those repeat instances based on the entity that they represent.

For example, using each person's name in the example above will be more helpful than just the index. To do this, add a group directly in the repeat and use one or more identifying questions in the label.

survey

type	name	label
begin_repeat	person	Person
begin_group	person_group	${first_name}&nbsp;${last_name}
text	first_name	First name
text	last_name	Last name
end_group		
end_repeat		

The repeat labels will be shown along with the index in the location summary.

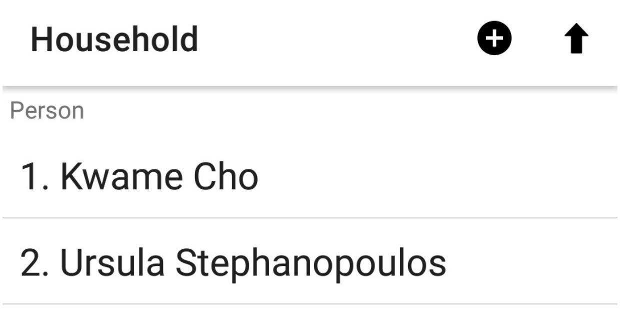

Household

Person > 1 > Kwame Cho

Last name

Cho

The labels are also shown in the jump menu.


Household

Person

1. Kwame Cho

2. Ursula Stephanopoulos

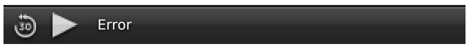
Filtering options in select questions

To limit the options in a select question based on the answer to a previous question, specify an expression in the `choice_filter` column of the **survey** sheet. This expression will refer to one or more column in the **choices** sheet that the dataset should be filtered by.

For example, you might ask your enumerators to select a state first, and then only display cities within that state. This is referred to as a "cascading select" and can be extended to any depth. The example below has two levels: job category and job title.

The `choice_filter` expression for the second select in the example is `category=${job_category}`. `category` is the name of a column in the **choices** sheet and `${job_category}` refers to the first select question in the form. The filter expression says to only include rows whose `category` column value exactly matches the value selected by the enumerator as `${job_category}`.

Any expression that evaluates to `True` or `False` can be used as a `choice_filter`. For example, you could add a `location` column to the **choices** sheet and also ask the user to enter a location they want to consider jobs in. If the new location question on the **survey** sheet is named `${job_location}`, the choice filter would be `category=${job_category} and location=${job_location}`. Another example of a complex choice filter is one that uses text comparison functions to match labels that start with a certain value. Consider, for example, `starts-with(label, ${search_value})` where `search_value` is the name of a text question defined on the **survey** sheet.



XLSForm

survey

type	name	label	choice_filter
select_one job_categories	job_category	Job category	
select_one job_titles	job_title	Job title	category=${job_category}

choices

list_name	name	label	category
job_categories	finance	Finance	
job_categories	hr	Human Resources	
job_categories	admin	Administration/Office	
job_categories	marketing	Marketing	
job_titles	ar	Accounts Receivable	finance
job_titles	pay	Payroll	finance
job_titles	recruiting	Recruiting	hr
job_titles	training	Training	hr
job_titles	retention	Retention	hr
job_titles	asst	Office Assistant	admin
job_titles	mngr	Office Manager	admin
job_titles	reception	Receptionist	admin
job_titles	creative_dir	Creative Director	marketing
job_titles	copywriter	Copywriter	marketing

Generating select ones from repeats

If you use a repeat, you can generate a follow-up `select_one` question using values from the repeat. For example, if you collect information about several household members in a repeat, you can then show a select one with all household members' names. To do this, add a question of type `select_one` followed by the name of the question in the repeat that you want to use for the select options.

survey

type	name	label	required	choice_filter
begin_repeat	person	Person		
text	person_name	Person's name?	true()	
integer	person_age	${person_name}'s age?	true()	
end_repeat	person			
select_one ${person_name}	tallest	Select the tallest person		
select_one ${person_name}	tallest_child	Select the tallest person under 18.		${person_age} < 18

As shown in the example above, you can combine this with other select features such as filtering. Note that in the example above, the question used as select option text is `required`. If a question used to generate a `select_one` is not required and it is left blank for some repeat instances, those repeat instances will not be included in the select.