Intro/Read me

This document isn’t intended as a comprehensive review of how to code using ODK Survey 2.0. Its goal is to provide complementary information on the tutorials already provided on opendatatkit.org: things that haven’t yet been documented, that are unclear or special cases.

For example, it isn’t relevent to go over the installation of a new form on the phone. But if someone has useful Grunt commands that makes replacing the formDef.json faster/easier then this would be an interesting addition, as long as those commands aren’t already in the ODK documentation.

(If you bring information from other sources that's OK because our main source of information will be ODK .org)

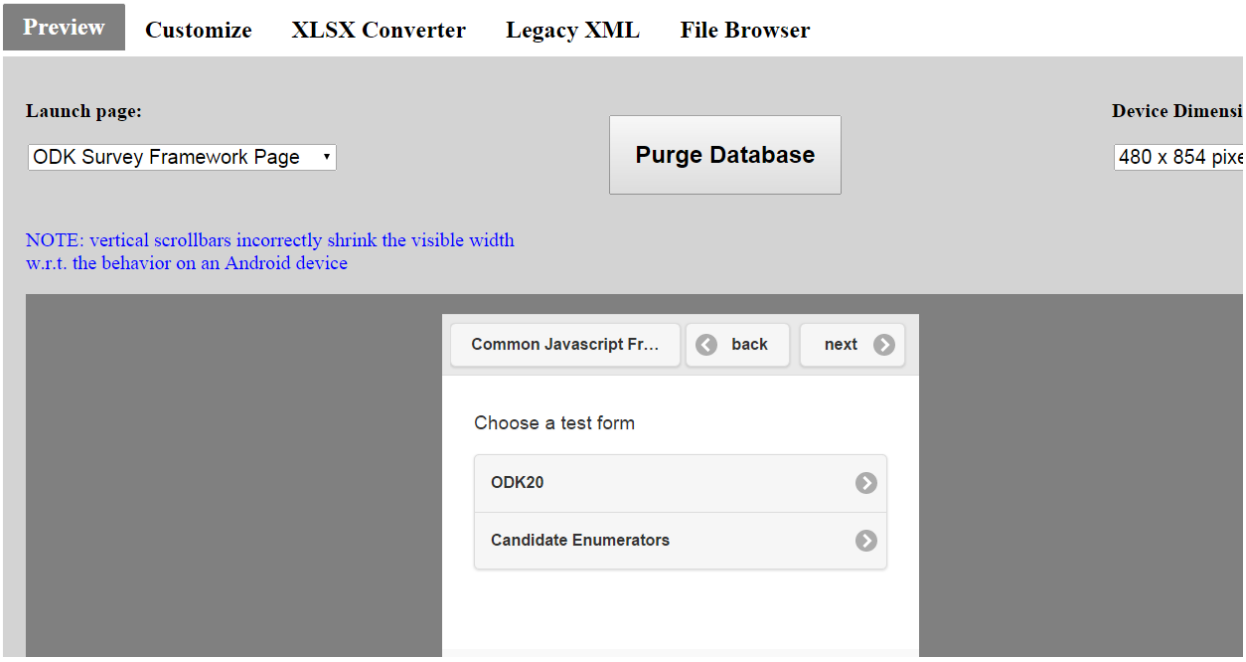
THE GOLDEN RULES

1. Always use Headers to organize your intervention. You can find them in *Format/ Paragraph style* or use the ctrl + alt + (number). This allows the Table of Content to automatically update when refreshed.
2. Update the ToC when you make additions (click in the general area and then click the circle/arrow that appears top right).
3. Before making an addition:
    a. Find the portion of the document where it will fit best
    b. If you create a new section for it, ensure it wouldn't be better off as a sub-topic.
    c. Ensure it isn't already covered. You can edit a section a section if it is incomplete but don't duplicate them needlessly.
    d. It might be relevent to quickly check in the opendatakit.org document if this is already covered or not.
4. Ensure that what you say is true. If unsure, add a comment/note. If it's only been partially tested, mention it.
5. Try to be precise and to the point.

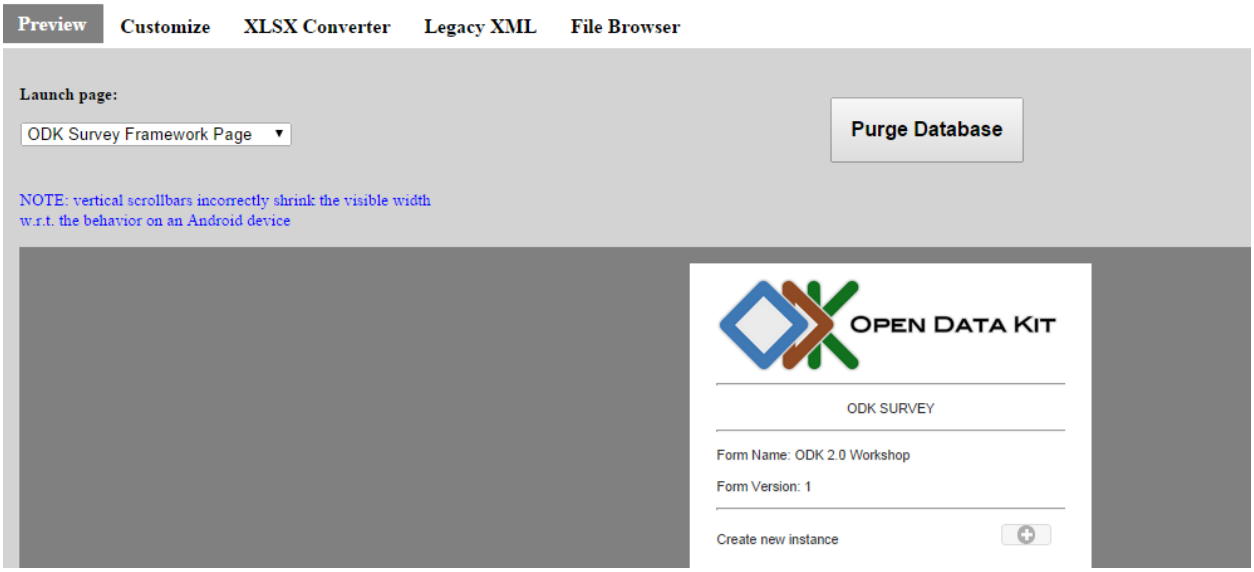
Important note on purging the database

When you "Purge Database", it is important to note that *the purge happens only in the form that is selected, not to all forms in your Application Designer.*

In other words, purging the database at this screen is unlikely to achieve much:



While doing it a this screen will do what you expect, i.e. allow you to open the form without getting a “Error while accessing to saving to the database”:



Doing this will save lots of time in the long run. You might think your form has mistakes while it might not be the case, the error being generated by the discrepencies between the database and the modification made in your form.

Special prompt types

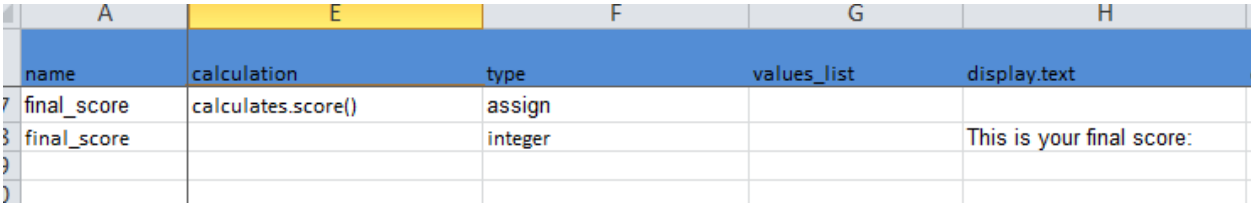
This section details particularities & tips about using certain prompt types.

Assign

Using assign to set the value of a variable

Let's say that you have a calculation to give the user of a form a certain score, depending on the answers given. You're using the ***calculates*** sheet. What you want to do is to call on this calculation at the end of your form, and assign that value to a variable that you will show to the user.

To use assign you actually need 2 rows, as follow:



	A	E	F	G	H
	name	calculation	type	values_list	display.text
7	final_score	calculates.score()	assign		
8	final_score		integer		This is your final score:
9					
10					

- Both rows need to have the same variable name (final_score in this case)
- The first row must be of prompt type ***assign***. It must use a calculation: you could write the calculation directly in the ***calculation*** column, or call a formula you've written in your ***calculates*** sheet.
- The second row must have the same variable name. The prompt type must be the data type that you will be using (most likely interger or decimal).

If you try to assign the value only with one row (prompt type assign), you will get the error "**Error while initializing database"**. See troubleshooting below.

1. Using queries

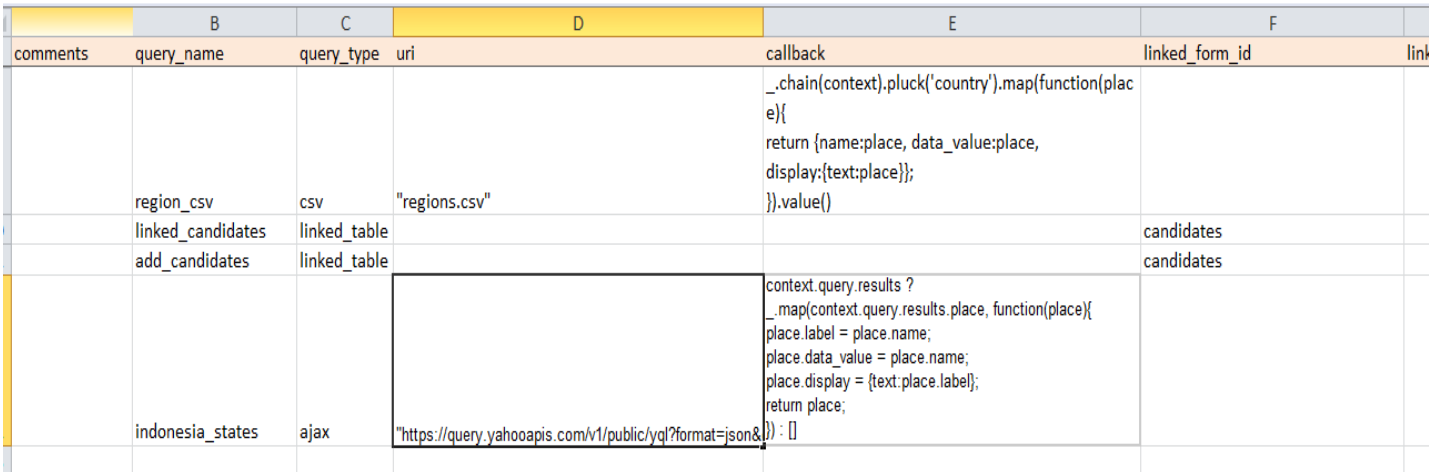
Query types

Ajax

An ajax query calls a server where the data are stored and returns to you an object. It can be a lot values, such as a list of all countries in the world.

There are many API to which you can address an Ajax query. In the example files an example is set up using Yahoo's API, which we will detail here. The general way to use Ajax queries is very similar to CSV queries (see below), if you can make a CSV query work in your form, you can make an Ajax query work.

The main difference between the 2 queries is the URI. Let's take the following example:

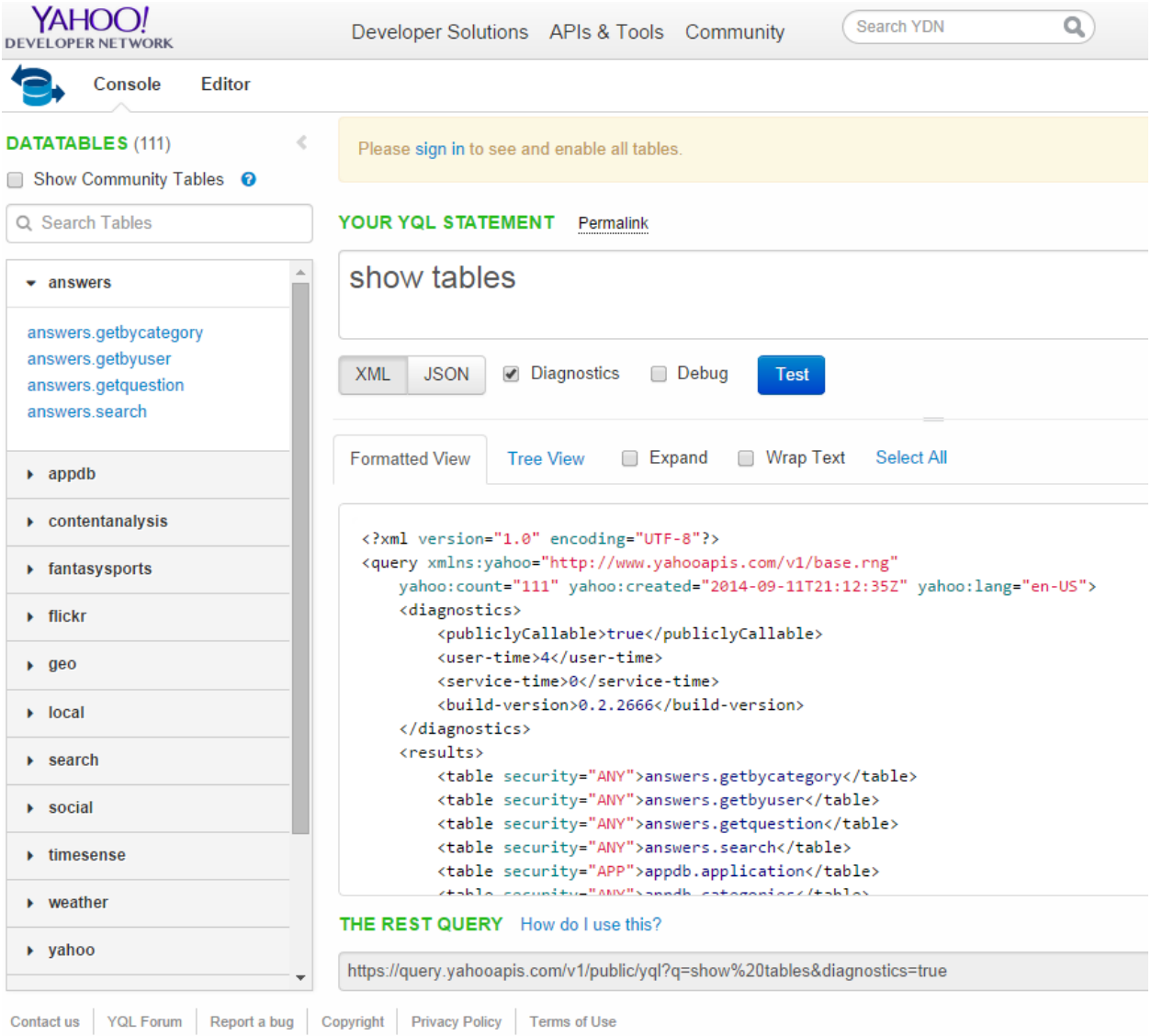


comments	query_name	query_type	uri	callback	linked_form_id	link
	region_csv	csv	"regions.csv"	_.chain(context).pluck('country').map(function(place){ return {name:place, data_value:place, display:{text:place}}; }).value()		
	linked_candidates	linked_table			candidates	
	add_candidates	linked_table			candidates	
	indonesia_states	ajax	"https://query.yahooapis.com/v1/public/yql?format=json&q=" + encodeURIComponent("select * from geo.states where place='Indonesia'")	context.query.results ? _.map(context.query.results.place, function(place){ place.label = place.name; place.data_value = place.name; place.display = {text:place.label}; return place; }) : []		

The URI is the “question” you ask to the server such as:

"https://query.yahooapis.com/v1/public/yql?format=json&q=" + encodeURIComponent("select * from geo.states where place='Indonesia'")

You can go to : https://developer.yahoo.com/yql/console/, which looks like this:



You can use the dropdown menues on the left to see what kind of information Yahoo's API contains (and thus what question you can ask him).

In "Your YQL statement", you can write the YQL "question" you can to ask the server (which is pretty much SQL). The YQL statement is actually part of the URI we send to Yahoo and is highlighted in RED below:

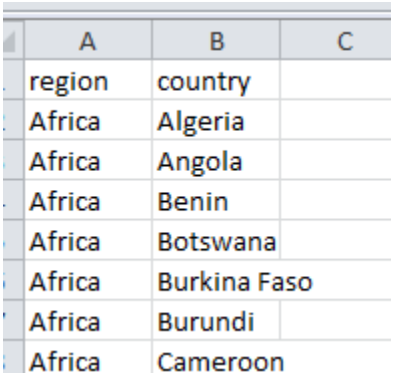
"https://query.yahooapis.com/v1/public/yql?format=json&q=" + encodeURIComponent("select * from geo.states where place='Indonesia'")
In the white box in the middle, this is the answer the server will give your. If you look at the "REST QUERY", you will see that the rest query is actually what we call in ODK the URI. Therefore, if you want to construct an AJAX query on Yahoo, you can play around with these elements, and once you get the REST query you can put it down in the URI cell of your form.

CSV

- The file must have a .csv format (not or other excel format).

What is callback?

The **callback** column is where you tell the CSV Query what information to get. This is a little more complicated so let's detail it. First, let's look at the **callback** to extract data from a list of countries in a .csv file. (it actually contains all countries, organized by continent)



	A	B	C
	region	country	
	Africa	Algeria	
	Africa	Angola	
	Africa	Benin	
	Africa	Botswana	
	Africa	Burkina Faso	
	Africa	Burundi	
	Africa	Cameroon	

To get a list of all the countries the callback would be:

```
_.chain(context).pluck('country').uniq().map(
        function(place)
                {
                return
                        {
                         name:place,
                        label:place,
                        data_value:place,
                        display:{text:place}
                        };
                }                               ).value()
```

Explanations:

- **pluck('country'):** It tells Survey which column you want to use. If we had used 'region' instead we would get a list of continents (Africa, Europe, etc).
- **function(place)**: We could use any correct variable name instead of "place". This is the name of the object that will store your countries.
- **.uniq()**: If you only want distinct values. In this case we wouldn't need it (each country is listed only once in this list)

- **return {name:place, label:place, data_value:place, display{text:place}}**: A collection of ***attributes:value*** pairs that you want each country to have:
    - I removed **label** and it didn't seem to matter.
    - **data_value**: this the value you would use in another prompt, for example if you wanted to add a condition (only present that question if the answer is "Canada" for example).
    - **display:{text:place}**: This is what the prompt actually shows (if you use a drop-down menu for example). **text** specify what type of data it is.
- These are ***Underscore*** library commands, so you can refer to their documentation for creater details.

Callback for cascading selects

We would first need to run the previous callback on region, not country. This is easy to change. Then the second callback will be on country:

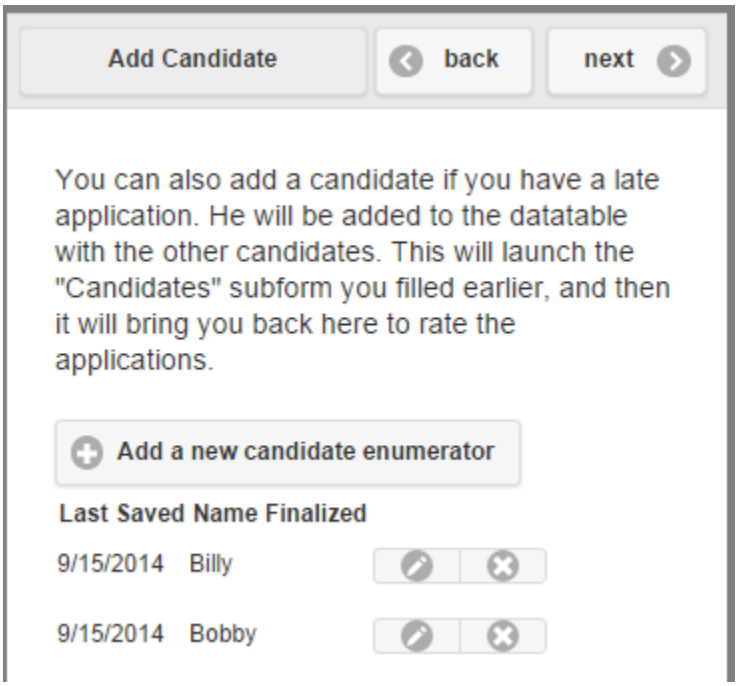
Linked_table

[All the steps will be added soon I thought this section was actually completed already]

Specify or not a linked_table_id

If you do NOT specify a linked_table_id for a linked_table query where you allow addition/ removal of instances (for example, you a a Household form and you have a linked_table where you add info on each household members to a different data table), then you will not have a list of instances to edit/remove.

If you DO specify the table, then you will have something along theses lines:



As you can see, instances Billy & Bobboy are listed (along with any other candidate that might have been entered in the data table). This is because in the query, linked_table_id was specify. This is true even if the table_id is the same as form_id: you need to explicitely refer to it if you want to have a list of instances in that table.

User_branch for form flow control

This can be use to let the user decide which subform they want to fill and in which order, as opposed to a linear flow where everything is decided by whoever coded the form.

A brief summary of steps, followed by greater details:

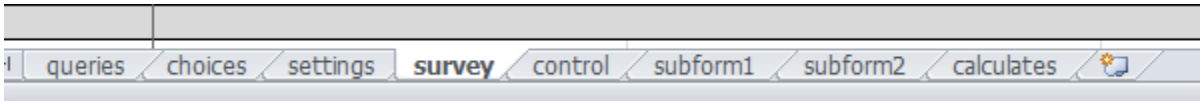
- Insert a “do section *name_of_my_control_sheet*” at some point in your survey sheet
- Create a worksheet for your ***control* subform & one sheet for each subforms**
- Create a choice_list in your ***choices*** worksheet for each new sheet
- Add an entry in your ***settings*** sheet for each new sheet
- Structure you branching logic in the ***control*** sheet

Editing your choice_list & worksheets

It will be clearer if you create a separate sheet for your control of branches. Let’s say that you want the users to fill a certain par of the survey (general informations perhaps). Then at the point of the survey where you want users to choose which branch to fill, you will have to insert a “do section name_of_my_worksheet”. Here it’s called “control”:

C	D	E	F	G
branch_label	clause	condition	calculation	type
				note
				select_one
	do section control			
			calculates.score()	assign
				integer

And below the list of all worksheet in this form:



You will need to add entries to ***choices***, ***survey, settings*** and then create your ***control*** sheet to code the branching options as well as one sheet per subform (here, subform1 & subform2).

You then need to create the choice_list for this. You can basically see the branching mechanism as a select_one prompt, thus you need to create the list:

choice_list_name	data_value	display.text
yesno	yes	Yes
yesno	no	No
sexes	male	male
sexes	female	female
subformlist	subform1	[SUBFORM1]
subformlist	subform2	[SUBFORM2]
subformlist	exitpath	Exit the subform selection menu & continue

The list is name subformlist. The data_values are the same as the subform names but you could choose diffently. *exitpath* is the choice that will let the user leave the subform selection menu and continue in the normal form flow.

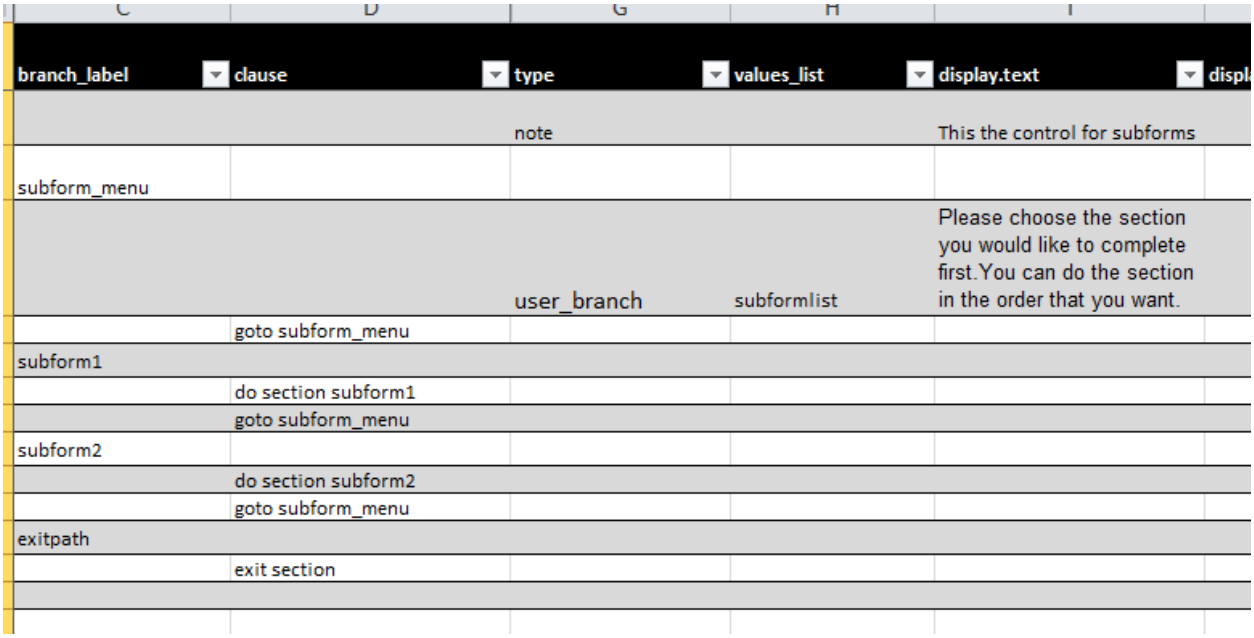
Editing settings sheet

setting_name	value	display.title
form_id	odk20	
form_version	1	
table_id	odk20	
survey		ODK 2.0 Workshop
control		Control menu for subform
subform1		[SUBFORM1]
subform2		[SUBFORM2]
surveyend		Last part of the survey

Just like for the main ***survey*** sheet, you need entries for other sheets you have just created.

The control worksheet

This is the main part of the process, where you control the flow.



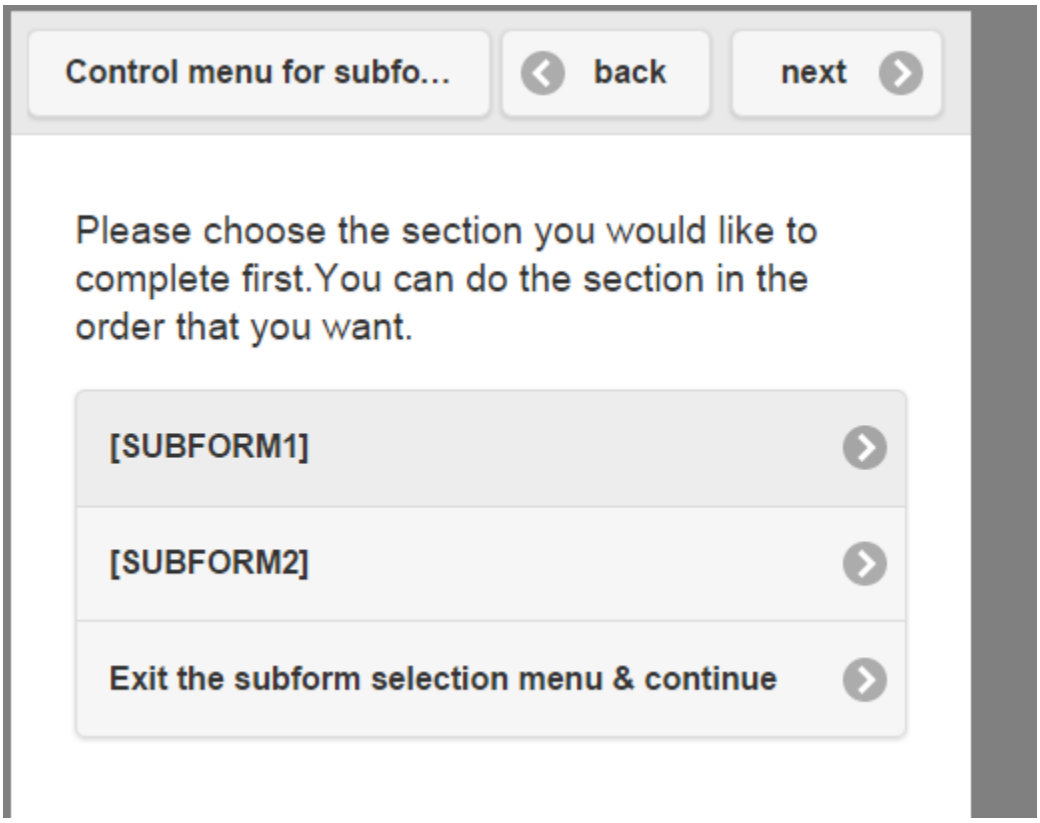
branch_label	clause	type	values_list	display.text	displ
		note		This the control for subforms	
subform_menu					
		user_branch	subformlist	Please choose the section you would like to complete first.You can do the section in the order that you want.	
	goto subform_menu				
subform1					
	do section subform1				
	goto subform_menu				
subform2					
	do section subform2				
	goto subform_menu				
exitpath					
	exit section				

Here we have named our control section ***subform_menu*** (arbitrary). Then, we have a user_branch prompt type that refers to the choices_list we have created earlier (subformlist). This will create a list of all subformed indicated in choice_list.

For each subform, we give a branch_label (the name of the choice in choice_list), so subform1 & subform2 here. When the user selects, say, subform1, there is a “do section” command that guides survey to that subform. Once the section is completed, it comes back in this section and reads the “goto subform_menu” command. This is so that the user is offered the choice of filling another form, say, subform2.

If he wants to exit the subform menu, he choices “exitpath”. There is a exit section command, that will bring him back to the rest of the survey sheet he was filling before entering this menu.

This what the menu will look like in the Application Designer simulation:



Basic Javascript notions

This shouldn't turn into a full-on JS class. But a few basic JS notion will get you a long way in understanding calculations in Survey (as they ARE JS calculations).

Libraries used by Survey

It can be useful to know which libraries are used by Survey, because you can call on their functions in your forms (just as was done with the callback in CSV query). From potentially most useful for you coding form to the least usefull:

- Underscore: http://underscorejs.org/
- jQuery:http://api.jquery.com/
- Backbone: http://backbonejs.org/

I would NOT detail each & every one of theses here, BUT for some commonly used functions it might be worthwhile to detail their syntaxe & uses.

Compact Formulation: (condition) IF (formula)

You might have come accross this kind of formulation in the Survey examples:

```
selected(data('birthdayKnown'), 'yes') ? (now().getTime() - new Date(data('birthday')
).getTime()) / (1000 * 60 * 60 * 24) : data('monthsOld')*30
```

This is a compact statement in JS. Let's break it down. This statement can be broken into 4 main parts: before the ?, the question mark & after:

- *selected(data('birthdayKnown'), 'yes')*: This is a condition. It can be any condition that can be evaluated in JS. In that case, if the prompt "birthdayKnown' evaluates to "yes"....
- ? : The question mark basically means "IF" statement.
- *(now().getTime() - new Date(data('birthday')).getTime()) / (1000 * 60 * 60 * 24)*: This is the calculation performed, assuming the condition before the ? evaluates to TRUE.
- If it evaluates to FALSE, then it does the calculation after the ":", in other words: ***: data('monthsOld')*30***
- In English: *"If they answered yes to birhdayKnown, then use (a bunch of other answers to prompts) and get me the resulting value. If they answered something else, then do the calculation after the " : " and get me that value instead.*

Refer to a variable value in display.text or display.hint

In that case, you can't just use something like data('name_of_variable'). You must use the curly braces, as such:

" My name is Franck and I am glad to meet you {{name_of_interlocutor}} ".

In this case name_of_interlocutor will be replace by the variable value, Bob, John, Tracy…

Testing for existence of a value

Let's say you can to test if a value has been provided to your prompt "What is your name?". The name of the prompt is "your_name"

The following expression:

data('your_name')

will evalute to "TRUE" if the user entered any data, and "FALSE" if not. So for example:

data('your_name') ? "Thank you" : "Please enter your name"

Will return "Thank you" if an answer was provided, and "Please enter your name" if not. (See Compact Formulation section for details on this notation).

The difference between selected(data(my_variable), "my_variable_value") & data(my_variable)

In short: data(my_variable) returns the value of "my_variable".
Selected(data(my_variable), "my_variable_value") returns true if *my_variable* has value *"my_variable_value"*. You should use selected(....) for select_one, select_multiple kind of prompts & data(....) for the rest.

Quick tricks & tips

- If you put 2 forward dash (“ // “) in the first column to comment it out (but not the “comments” column, if you put yours first). It won’t be used in the JSON file, so you can try to spot errors this way while debugging or trying out different version of a prompt.

Troubleshooting/common errors

This section is geared primarly at helping debug non-specific error messages as they can be hard to fix. If you add something here:

- Try to indicate when the error occurs… Is it while opening the form? Saving it? Selecting some anser? Which one? In which context?

Error while initializing database tables

- **When/where it happens:** in AppDesign when you are just about to select which form you want to use
- **Possible cause**: by a misuse of “assign” prompt type: when using “assign”, you must have 2 rows (see section on “assign prompt”

Error while accessing or saving values to the database.

- **When/where it happens:** When you try to create a new instance of a form
    - **Possible cause:**
        - You might have changed variables in your form, uploaded the formDef.json file but forgot to purge the database. So when it tries to create a new form, the variables indicated in the formDef.json don’t fit the database tables. If you did purge, try it again.

Unexpected token ; (on form conversion)

- It should also tell you where the error happen (sheet xxxxx, column yyyy, row XX)
- **When/where it happens:** When trying to convert a XLSX form into JSON format
    - **Possible cause:**
        - You very likely forgot a parenthesis and the converter is confused about when your instruction ends.

Database.convertSelectionstring:unrecognized elementPath (some variable)

- Usually caused by a linked_query that doesn’t have a model sheet, incomplete model sheet. More specifically, you need to specify to the child table (i.e. the table you adress the query to) what auxilliaryHash you are using in your request. See the section about model sheet in the documentation for details.