

	B	C	E	F
	name	label::Français	relevant	constrain
one prity	NEEDPRIORW_1	I1.M What is the		
one prity	NEEDPRIORW_2	I1.M What is the second	not(selected({NEEDPRIORW_1}),'0')	not(={NEE
one prity	NEEDPRIORW_3	I1.M What is the third	not(selected({NEEDPRIORW_1}),'0') and not(selected({NEEDPRIORW_2}),'0')	not(={NEE ) not(={NEE

## ADVANCED XLS FORMS CODING

### Table of contents

<b>I. Sorry, what's your name again?</b>	<b>1</b>
<b>I.1. Same street name, different towns</b>	<b>2</b>
<b>I.2. Beauty in simplicity</b>	<b>2</b>
<b>I.3. Familiar faces</b>	<b>2</b>
<b>II. First things first</b>	<b>3</b>
<b>III. Patterns, everywhere</b>	<b>3</b>
<b>III.1. The wizardry explained</b>	<b>5</b>
<b>III.2. VIP Guest</b>	<b>5</b>
<b>III.3. More patterns</b>	<b>7</b>
<b>III.4. Alternative for "If other, specify"</b>	<b>8</b>
<b>III.5. Caveat</b>	<b>8</b>
<b>IV. Calculations: dealing with blanks</b>	<b>9</b>
<b>V. Testing, testing, testing</b>	<b>9</b>
<b>VI. Conclusion</b>	<b>10</b>

By now, many adepts of mobile data collection know the basics about how to code XLS forms, with good online resources for support.

However, past the basics of the syntax needed to make a form work, not that much is available and it's exactly that gap I'll try to address here. So let's assume you have coded a couple forms already and they work. But now you have to code a long one, a half-dozen (paper) pages, full of tiny multiple choices and tables with a complicated survey-flow pattern... how to make this faster? This will be the first part of two posts on this thematic.

### I. Sorry, what's your name again?

When coding a long form, a naming convention can go a long way in streamlining the process. There isn't one "best" answer to this, however it boils down to:

"Do what you want, but do it for a reason."

The fewer decisions you make without really having some reasoning behind it, the better your coding method will be. The following tips have all been set up with this in mind. Therefore, feel free to take them whole, or adapt to needs & preferences.

### I.1. Same street name, different towns

I started some time ago assigning similar names to the “list name” items (that make up the multiple choice items in the “choice” tab) and to the prompt’s name in the “survey” sheet. Some differentiators can be used, such as add “L” at the end. This way, you can easily edit any given list choice without having to constantly lookup the main survey sheet (and the other way around). You then know that to edit the *relationHosts* prompt, you need to edit the *relationHostsL* list in the choice tab. If you place the different list choice in alphabetical order, it also makes it easier to find it in the choice tab (which is why you would put the differentiators at the end).

The exception to this would be very common choice lists shared by multiple prompts, such as Yes/No, Increase/Decrease/Stable, etc.

### I.2. Beauty in simplicity

When making up a new variable name, I try to be both concise & descriptive. For example, let’s say we want to assign a name to a prompt about vulnerable groups of internally displaced people by shelter type. There are many ways to name this:

- vulnerable\_groups\_displaced\_people
- vulnerableGroupIDP
- VLNGRP\_IDP
- Question\_5\_2A

Of course, the first one is the most descriptive, but isn’t concise. What if you have that question nested inside a group of questions that is related to Protection and Security, which is itself nested inside a group of vulnerability factors? If you’re being consistent, you might have named the groups “protection\_security” and vulnerability\_factors, so in the database you have a column name such as “vulnerability\_factors/protection\_security/vulnerable\_groups\_displaced\_people”. There are limits to the length that can be assigned to those columns, and they are database-specific. Bumping into those can lead to problems further down the road.

The very last option of naming the prompt after the question number (probably issued in the paper form), though commonly used, has many shortcomings:

- Adding/removing questions means the numbering will have to be adjusted (useless work)
- It forces you to go back and forth between the XLS form, the paper form & the output to know what means what
- Finally, your XLS form isn’t self-contained anymore: to really know how the form is structured, you almost need to refer to this other document (word or printed form).

From my perspective, the 3rd option will work best, especially if the expression “Vulnerable groups” comes up often: in no time VLNGRP will be synonymous to “vulnerable\_group”. It strikes a balance between being descriptive enough to be recognizable at first glance, while not being too cumbersome.

### I.3. Familiar faces

In the choice list, I tend to favor using numbers as values instead of letters. There are pro & cons to this, however I like the objectivity that comes with assigning numbers to each option: a 5 is a 5, whereas the choice “Religious Minorities” could have a number of “values” assigned to it, endless combinations of camel-case or underscore separators, capital letters, etc.

Another potential benefit of this (although it can certainly be possible using letters as values as well) is to use the same value for very common options that appear in many lists:

- -88 for "Others"
- -99 for "Do not know"
- 0 for "No", 1 for "Yes"

You can establish similar conventions for any common value in your survey

The upside being that when coding constraint, relevant conditions or "If other, please specify" prompts, the value to check for will always be the same. No need to go check the choice list for that, ever. Common relevant/constraint can also be copied & pasted, because the action to take if the answer is "Other" or "No" are often similar.

On the other hand, there is nothing wrong with using readable names for those values instead of numbers – as explained above, there are pros and cons to both. Depending who will be working with the output, and what your analysis set-up is, you might decide that it is better for you to have those value readable right in the output file –especially if your analysis system isn't programed and a real person will have to manually interpret the data directly from the server output.

## II. First things first

For a short form, it might make sense to just code everything prompt by prompt. However, for a long or complex form, the best structure to adopt or the kind of relevant / constraints / calculation that should be set might not always be obvious: this might come either through testing or through the training sessions of the end user (if that's possible in your situation). It might be more efficient to code those advanced elements progressively, as the form matures.

I would suggest keeping it to the labels, hints, prompt name & types, and adding already a little bit of structure (grouping questions either for the presentation on the phones if you have larger screen, or because they are thematically related, etc).

## III. Patterns, everywhere

When coding a more complex survey, some common patterns arise such as questions about rankings:

### ***What are your top 3 three priorities in order of importance?***

- Food
- Lodging
- WASH
- Clothing
- Health
- No other priorities

In a paper form, this would translate into small boxes with numbers 1-2-3 written next to a list of priorities. However, with ODK Collect, there isn't necessarily just one way to approach such a prompt. For ease of analysis, I tend to prefer avoiding multiple choices when possible, as they are generally more difficult to deal with when comes the time to setup the analysis. My inclination in this case would be to code 3 select\_one prompts, with the choice list being the list above for each.

Logically, if a respondent expresses not having any other needs at the 2nd prompt, you wouldn't want the next one to show up. They also shouldn't be able to select the same need twice

type	name	label::Français	relevant	constraint
select_one needPriority	NEEDPRIORM_1	I1.M What is the most important need for men?		
select_one needPriority	NEEDPRIORM_2	I1.M What is the second most important need for men?	not(selected({\$NEEDPRIORM_1}),'0')	not(.={\$NEEDPRIORM_1})
select_one needPriority	NEEDPRIORM_3	I1.M What is the third most important need for men?	not(selected({\$NEEDPRIORM_1}),'0') and not(selected({\$NEEDPRIORM_2}),'0')	not(.={\$NEEDPRIORM_1}) and not(.={\$NEEDPRIORM_2})

A few things to note:

To code any similar group of question, I can simply copy-past all 3 prompts and then use "Find & Replace":

Find and Replace

Find

Replace

Find what:

NEEDPRIORM\_

Replace with:

NEEDPRIORW\_

Options >>

Replace All

Replace

Find All

Find Next

Close

- Copy the full 3 rows of the men's priority questions and past them further down.
- Select the 3 questions you just pasted, then ctrl+F
- In that box, simply replace the common root for the men's prompt by the common root for the woman's prompt & hit "Replace all":

Which will the yield:

type	name	label::Français	relevant	constraint
select_one needPriority	NEEDPRIORW_1	I1.M What is the most important need for		
select_one needPriority	NEEDPRIORW_2	I1.M What is the second most important need for	not(selected({\$NEEDPRIORW_1}),'0')	not(.={\$NEEDPRIORW_1})
select_one needPriority	NEEDPRIORW_3	I1.M What is the third most important need for women?	not(selected({\$NEEDPRIORW_1}),'0') and not(selected({\$NEEDPRIORW_2}),'0')	not(.={\$NEEDPRIORW_1}) and not(.={\$NEEDPRIORW_2})

Pay attention to the number of replacement (Excel will tell you how many replacements have been made). You want to make sure you selected only the cells in which you want excel to find & replace: in our case, the 3 rows we just copied. If Excel tells you it made 110 replacements, you might want to ctrl + z this and retry!

The “name” column have been automatically taken care of, because they all share a common root, **NEEDPRIORW**, that replaced the **NEEDPRIORM** for men. Because we selected the relevant & constraints columns as well when we hit ctrl+F, Excel replaced those as well.

Setting up the same pattern for various different groups would take a few minutes at most, whereas coding each of those questions one by one can quickly become tedious and error-prone. Even if those ranking questions do not refer to the **needPriority** list of the original prompt, all that is needed then is to replace the list name as well and you can even use the find & replace method for that, too. The relevant and constraints conditions are still correctly coded, assuming you followed the previous advice of consistently using “0” as a value for “No” (or in that case, “No other needs”)

### III.1. The wizardry explained

As an aside, it might be worthwhile explaining how the previous “constraint” works (the “relevant” pattern is well-documented on the net already, so we’ll skip that part). It is meant to prevent the user from selecting the same option twice, as you can’t have “Food” as both your first and second priority. The notation:

***not(.= \${NEEDPRIORW\_1}) and not(.= \${NEEDPRIORW\_2})***

- The “.” before the equal signs, in all XLS forms, means “the current prompt”
- A CONSTRAINT expression must always evaluate to TRUE if the user is to be allowed to go to the next prompt
- In case the current prompt (“.”) is equal to NEEDPRIORW\_1, then that will evaluate to “TRUE”. Since we don’t want the user to be able to proceed to the next prompt in that case, we inverse its value by wrapping it into a not() statement. If the current prompt is different from the NEEDPRIORW\_1, then the expression would yield “FALSE” and after negation it will evaluate to “TRUE”, and the user will be able to proceed.
- For NEEDPRIORW\_3, we want the value to be different from both the 1<sup>st</sup> and the 2<sup>nd</sup> prompt in the series. We therefore added the “and” statement between the 2 conditions: if any of the condition is not met (so the selection is equal to either the 1<sup>st</sup> or 2<sup>nd</sup> prompt) then the whole expression will evaluate to “FALSE” and the user will have to change his selection.

As those constructs get more complex, it might be worthwhile to test them more thoroughly, to ensure they work as intended. For example, making a mistake here might prevent the user from selecting different answers between the 3 prompts – definitely not what we want!

### III.2. VIP Guest

Those who coded forms directly in XML can definitely recognize the value of the “choice filter” column in XLS forms, which allows to easily filter the choice being offered in a prompt, based on a previous selection. For example, you wouldn’t want to show the full list of 196 countries if the user already told you he comes from North America.

However, for some types of lists such as villages in a remote region, it can be hard to ensure you have an exhaustive list, therefore it would be preferable to always allow the user to select “Other” and then offer a text input for the missing value.

In this example, we have a list of social workers, based in different region ("**prefectures**"). We want to use the choice filter to show only the workers in the selected region of operation, however we need to offer some flexibility in case there's a new hire or a replacement. For this to work, we must first add one entry in the social worker list in the "**choice**" tab. We add "Other" ("Autre") option and assign it the value **-88**. In the column "**prefecture**", where the choice filter operates, we also put "**-88**" as a value (or any value you usually assign to your "Other" statements).

list name	name	label::Français	prefecture
ts	18	TOLNO	conakry
ts	19	SANGARE	conakry
ts	20	KEITA	kindia
ts	21	BARRY	kindia
ts	22	CAMARA	kindia
ts	23	BAH	telimile
ts	24	KOUYATE	telimile
ts	25	CAMARA	telimile
ts	-88	Autre	-88

In the main "**Survey**" tab, we have 2 prompts of relevance to this: the **INT\_Prefecture** prompt, which is the region of operation that the user first selects. This is stored in the variable **INT\_Prefecture**. The prompt **INT\_NameWS** is the prompt where the social worker is selected. As can be seen in the **choice\_filter** column, the selection is based on the choice made in the **INT\_Prefecture** prompt. The form then takes that value, and looks into the "**ts**" **list\_name**, and shows all those for which the "**prefecture**" column value matches the **INT\_Prefecture** value. But because we also added "**or prefecture = -88**", it will also show any value in the "**ts**" **list\_name** that has value **-88**. This is our "Other" option.

type	name	choice_filter
select_one prefecture	INT_Prefecture	region = \${INT_Region}
select_one sw	INT_NameWS	prefecture= \${INT_Prefecture} or prefecture = -88

And the result is that we can both filter the list of workers according to the prefecture of activity AND always offer the option to enter another name, regardless of the prefecture selected:

**Name of social worker: \***

☐ BAH  
☐ KOUYATE  
☐ CAMARA  
☒ Autre

**If other, specify: \***

The selection will filter answer according to our filter value, as well always allowing "Other" value.

### III.3. More patterns

Another way that using those patterns can lead to faster coding is when setting up the rather common "If other, please specify". When coding a long form, it can be advantageous to setup a "generic" prompt of that type, without setting reference to any specific variable. It can then be copy-pasted as needed, while filling in the prompt name to which it refers only at the end:

type	name	label::Français	relevant
text	_OTHER	If other, please specify:	selected(\${'-88'}) or selected(\${'-88'}) or selected(\${'-88'})

I try to always append the suffix \_OTHER to this type of question, as that allows to quickly identify them. I will generally use as a core part the same name as the question it refers to. For example, if the men's question about need priority did allow for "If other", I would name it NEEDPRIORM\_OTHER.

I also have only set the generic envelop for the relevant condition for this question. In that case, if any of the need priority is answered by "Other priority" (which value, as always, is -88), then I want the prompt to appear.

The above can then be copy-pasted any number of times and to be functional it only requires adding a complete name and the references to the prompts in the relevant conditions. Proceeding this way is much faster than manually typing each and every such prompts in the form and the benefits add up the longer and more complex the form gets.



### III.4. Alternative for “If other, specify”

There is an alternative as well for every question which needs to offer a text input option for “Other” types of answers: one must simply append “or\_other” in the “type” column, as shown below:

type	name	label::Français
select_one vulnGroup or_other	VULNGRPPROT_1	What is the MAIN vulnerable group to those protection issues?
select_one vulnGroup or_other	VULNGRPPROT_2	What is the second most vulnerable group to those protection issues?

In the list choice, there is no need to add the “Other” option as this will be done automatically by the form:

A	B	C	D
list name	name	label::Français	admin2
vulnGroup	1	Déplacés vivant des familles hôtes (sans payer	
vulnGroup	2	Déplacés vivant dans des logements à louer	
vulnGroup	3	Déplacés vivant dans des logements non-finis, endon	
vulnGroup	4	Déplacés vivant dans les abris collectifs	
vulnGroup	5	Déplacés vivant dans des camps improvisés	
vulnGroup	6	Population locale qui héberge les déplacés	
vulnGroup	7	Population locale qui n'a pas été déplacée	
vulnGroup	8	Retournés	
vulnGroup	0	Aucun groupe n'est vulnérable	

In the end, the output file in your platform of choice will automatically add a column for the “Other” text field, just like that would be the case with the manual coding option described previously.

### III.5. Caveat

This is an interesting alternative because it is really quick: it allows you to basically forget everything about the “If other” option, and let the form handle it. However, there will be some defaults choices that you’ll have to live with, should you choose that option:

- You cannot decide what the label in the list of choices for the user will be: it will say “Other”. If you have a form in multiple languages, you will have to instruct your user that this means “Autre”, “Otro” or “Ostatni”.
- Same for the coded value of the option: the value will be “other” in small case. If you have (wisely) opted for some sort of system in the assignment of your value, you might have to take this into account. In that case, I would opt to make it uniform and always assign “other” instead of “-88” to all choices that mean “Other”. Remember, “Do what you want, but do it for a reason”.
- The name of the additional columns will always be “nameOfTheQuestion\_other”, with “\_other” being automatically added – this would be convenient with our system which is good.



Finally, we are forced to have one specific “other” column for each question. In the previous example (by coding manually the constraints), we had a system that would show only one “Other” column if ANY of the 3 questions have “Other” selected. This is because while we want to know if some choices are missing in the list, we might not be interested in knowing specifically if it applies to the 1<sup>st</sup>, 2<sup>nd</sup> or 3<sup>rd</sup> question: perhaps it is sufficient for us to know that the user wanted to specify something we haven’t listed.

#### IV. Calculations: dealing with blanks

Performing in-form calculation can be very handy. This can help the enumerator double-check that the value do indeed add-up, or even code constraints in the range that is acceptable for a series of related questions. However, one problem is that if ANY of the prompt referred to in the calculation is left empty, then the result will be blank.

You may get around this by forcing an answer (“required” column), however it is wiser at times not to make all prompts mandatory – when answer cannot always be expected to be available.

There is a workaround for this, and it involved using IF statement to assign the value “0” to any unanswered question. Consider the following:

A	B	C	D
type	name	label::Français	calculation
integer	A1A_totAvant	Number before the conflict:	
integer	A1A_fled	Number having fled the community:	
integer	A1A_IDP	Number of IDP having joined the community:	
integer	A1A_return	Number of IDP having returned:	
end group			
calculate	A1_tot		if({A1A_totAvant}>=0,{A1A_totAvant},0) - if({A1A_fled}>=0,{A1A_fled},0) + if({A1A_IDP}>=0,{A1A_IDP},0) - if({A1A_return}>=0,{A1A_return},0)

- For every variable involved in the calculation, the calculation tests to see if the value is a number equal or greater than 0:
  - If(**{A1A\_totAvant}>=0**, {A1A\_totAvant},0)
  - If it is, then it will use that value in the calculation
    - If({A1A\_totAvant}>=0, **{A1A\_totAvant}**,0)
    - If not, then it will use 0 as a value for the calculation
      - If({A1A\_totAvant}>=0, {A1A\_totAvant},**0**)

This works, because for the XLS form if an answer is skipped (let’s say that the number A1A\_fled isn’t available), then the “value” assigned to that skipped question is blank, and blank will always be considered “less” than any other numeric value. Because a calculation can’t be performed on a blank value (a blank value isn’t 0, it is simply nothing), we must assign a value 0 if we want the rest of the calculation to be performed anyways.

Note that this does NOT assign a value in the original prompt – this is only to allow the calculation to be performed, so that a meaningful note can be showed to the user.

#### V. Testing, testing, testing....

Regardless of the platform on which your survey will ultimately be hosted, you will most likely be better off testing your form more often than less. Depending on your level of comfort with XLS form, things can get complicated. Catching problems early on, while they are fewer in

numbers and easier to track, will always be easier than to code a whole form and then have to debug problems after problems.

One option for quick testing of forms in development is to use the offline converter (available at <https://opendatakit.org/use/xlsform/> where all converter options are listed). When you submit an XLS form to the converter, it will tell you if any syntax is wrong, if there's a typo in one of the list\_choice or value you used in your constraints/relevant conditions, etc. The messages are generally very informative.

Another option is using the Kobo platform:

- It takes directly the XLS form used for coding, instead of requiring a prior conversion to XML before being uploaded to the server
- It can be tested directly in the browser, in a page-format that allows to have a global view of the form (instead of the page-by-page display on the phones)

Opening an account is free, therefore it makes sense to have one, even if it isn't meant as a development platform. It is worth noting, however, that the behaviour of the form in webform can be slightly different than on the phones. The webform also doesn't allow to have a feel of how the form will behave with the smaller display of the phones actually used for the survey. It should therefore be used while the form is in development, but it doesn't replace testing the form with the actual devices that it will be used on.

## VI. Conclusion

Before the XLS form, everything had to be coded in XML, which was more complicated and slower. Now that XLS forms have greatly reduced the complexity and time needed to code a survey, it is time to develop coding practices that maximize this benefit, and hopefully the ones above can serve as a good starting point to the development of a personal coding system that is accurate and effective.