# Using ODK XLSX Converter

ODK Survey offers a rich set of features that can be seamlessly integrated into a custom form. A lot of the functionality can be implemented solely within an Excel workbook. This guide is designed to help you take advantage of this functionality via a guided tour of example tasks.

* [Creating and Loading a Form into ODK Survey](https://docs.opendatakit.org/odk2/xlsx-converter-using/#creating-and-loading-a-form-into-odk-survey)
* [Creating a Simple Survey Form](https://docs.opendatakit.org/odk2/xlsx-converter-using/#creating-a-simple-survey-form)
* Skip Patterns Using Conditional Statements
* Conditions and Required Questions
* [Adding Multiple Choice Questions](https://docs.opendatakit.org/odk2/xlsx-converter-using/#adding-multiple-choice-questions)
* Instance Names
* [Using Custom Section Worksheets](https://docs.opendatakit.org/odk2/xlsx-converter-using/#using-custom-section-worksheets)
* [Using Calculations](https://docs.opendatakit.org/odk2/xlsx-converter-using/#using-calculations)
* [Using Queries](https://docs.opendatakit.org/odk2/xlsx-converter-using/#using-queries)
* Queries Using Csv Files
	+ [Linked Tables](https://docs.opendatakit.org/odk2/xlsx-converter-using/#linked-tables)
	+ Order By
* [Internationalization](https://docs.opendatakit.org/odk2/xlsx-converter-using/#internationalization)
* [More Advanced Branching](https://docs.opendatakit.org/odk2/xlsx-converter-using/#more-advanced-branching)
* [Creating a Custom Initial Worksheet](https://docs.opendatakit.org/odk2/xlsx-converter-using/#creating-a-custom-initial-worksheet)
* [Using Validate](https://docs.opendatakit.org/odk2/xlsx-converter-using/#using-validate)
* [Customizing Prompts](https://docs.opendatakit.org/odk2/xlsx-converter-using/#customizing-prompts)
* [Other Features](https://docs.opendatakit.org/odk2/xlsx-converter-using/#other-features)

**Tip**

For a full reference to all the functionality available, see the [ODK XLSX Converter Reference](https://docs.opendatakit.org/odk2/xlsx-converter-reference/).

## [Creating and Loading a Form into ODK Survey](https://docs.opendatakit.org/odk2/xlsx-converter-using/#id21)

Application Designer has a structure that is critical for the ODK XLSX Converter to “find” things correctly. Within app-designer there are folders such as forms and tables that correspond to a survey. Below are the steps to create your directory and a new form from the exampleForm:

1. Within the Application Designer's folder, create the following directory structure app/config/tables/your\_table\_id/forms/your\_table\_id/
2. Copy the exampleForm.xlsx from app/config/tables/exampleForm/forms/exampleForm/ into this new directory.
3. Rename the XLSX file to your\_table\_id.xlsx
4. Edit the XLSX file and on the **settings** worksheet, change the value for table\_id to your\_table\_id. Then update the display title for the survey and the form version. Save the changes.
5. If you have not already, run **grunt** to launch the **Chrome** browser and open the Application Designer home page.
6. Navigate to the **XLSX Converter** tab, choose this file to convert it. Once converted, choose **Save to File System** and click **OK** on the 3 pop-ups that alert you to the saving of 3 files to the file system. The three files that are saved are:
* app/config/tables/your\_table\_id/definition.csv – defines the user-defined columns in your table
* app/config/tables/your\_table\_id/properties.csv – defines the appearance and available detail and list view HTML files for the table
* app/config/tables/your\_table\_id/forms/your\_table\_id/formDef.json – defines the ODK Survey form defined by the XLSX file
1. The first two files are written only if the form id matches the table id. That form and the XLSX file define the data table.
2. Repeat the edit, conversion, and save steps to update the columns in your table and your survey form.
3. Connect your device to your computer with a USB cable.
4. In a separate **command** window, navigate to the Application Designer directory and type:

$ grunt adbpush

to push the contents of the app/config directory to your device.

1. Start ODK Survey. The form should now be available in ODK Survey.

## [Creating a Simple Survey Form](https://docs.opendatakit.org/odk2/xlsx-converter-using/#id22)

Typing the following in the **survey** worksheet of a workbook with an appropriate **settings** worksheet will result in a simple survey. This example form uses several optional columns (clause and condition) in addition to the required columns.

*Creating a Simple Survey Example Form*

|  |
| --- |
| **clause** | **condition** | **type** | **name** | **display.prompt.text** |
|   |   | integer | person\_age | How old are you? |
| if | data('person\_age') >= 18 |   |   |   |
| begin screen |   |   |   |   |
|   |   | text | pizza\_type | What is your favorite kind of pizza? |
|   |   | integer | num\_slices | How many slices would you like? |
| end screen |   |   |   |   |
| else |   |   |   |   |
|   |   | note |   | You are too young to be eating pizza |
| end if |   |   |   |   |

## [Skip](https://docs.opendatakit.org/odk2/xlsx-converter-using/%22%20%5Cl%20%22id24) Patterns Using Conditional Statements

The clause and condition columns can create skip patterns in the survey. They can direct users to certain sections or questions in the survey, depending on preceding data. In the *Simple Survey Example Form* above, the first row contains an empty clause and an empty condition column. Therefore, the **display.prompt.text** will always be shown on the screen, and the resulting integer answer will be stored in the variable person\_age.

On the next line there is an if in the **clause** column and data('person\_age') >= 18 in the condition column. If the answer stored in the variable person\_age is greater than or equal to 18, the following commands should be done until either an else or an end if **clause** is reached. Notice the other columns are left blank.

In the next row, there is a begin screen in the **clause** column. The remaining four columns are left blank. Until an end screen is reached in the **clause** column, all the following questions will be displayed on one screen. In this case, the user will be asked to input their favorite type of pizza and how many slices they would like on the same page, assuming they are 18 or older.

In the next row, there is an else clause. Until end if is reached, anyone who did not satisfy the requirement for the if condition will be asked the following questions. In this case, a note to the user that they are too young to be eating pizza will be displayed.

**Note**

See the built in Formula Functions that can be used to write condition statements. Note that using JavaScript operators for conditions can allow for more advanced conditions that involve more than one variable or more than one response from a single variable.

It is also possible to have conditions within conditions (for example, if followed by if again). However, an important thing to remember when using the clause column is when to open and close new clauses. The general rule is that the most recently opened clause is the first to be closed.

**Tip**

Fixes to common error messages received when converting form in ODK XLSX Converter**:**

* Ensure that all parentheses and quotes are closed or matched.
* Ensure that syntax is in appropriate row. For example, make sure that if clauses are not on the same row as **type** and **values\_list.**
* Ensure that the end if clause is present and corresponds with the correct if clause. Incorrect end if statements. It is recommended to indent corresponding if and end if statements for organization.

Conditions and Required Questions

The **required** column is optional but very useful to ensure everyone who should answer a question does so. Users need to respond to all of their required questions before they can finalize the survey. However, it is important that if a **required** statement is used within a **condition** statement, the **required** statement needs to match the **condition**. Therefore, the question will only be required if the user meets the **condition** to be asked the question.

An example **survey** worksheet could look like this:

| *Skip Patterns and Required Questions Survey Worksheet* |
| --- |
| **clause** | **condition** | **type** | **name** | **display.prompt.text** | **required** |
|   |   | integer | person\_age | How old are you? | TRUE |
| if | data('person\_age') >= 18 |   |   |   |  |
|   |   | text | pizza\_type | What is your favorite kind of pizza? | data('person\_age') >= 18 |
| end if |   |   |   |   |  |

In this example, it is ideal for a required column to be TRUE for the age question because all users need to answer the question before they can finalize the survey. It would be problematic for a required column to be TRUE for the pizza\_type question, because users under the age of 18 do not get asked the question due to the condition. Users under the age of 18 would get an error when they are trying to finalize their surveys and then would be asked to answer the question about pizza. In this example, it is correct to only require the pizza\_type question if the user is 18 or older.

## [Adding Multiple Choice Questions](https://docs.opendatakit.org/odk2/xlsx-converter-using/#id23)

There are three types of multiple choice questions supported by ODK Survey:

* select\_one
* select\_one\_with\_other
* select\_multiple

Multiple choice questions use the **values\_list** column in the **survey** worksheet. The **values\_list** column is what links a multiple choice question to its answer set contained on the **choices** worksheet.

The pizza survey example used earlier can be improved upon with multiple choice options. The resulting **survey** worksheet would look like this:

| *Adding Multiple Choice Questions Example Survey Worksheet* |
| --- |
| **clause** | **Condition** | **type** | **values\_list** | **name** | **display.prompt.text** |
|   |   | select\_one | yes\_no | person\_age | Are you 18 or older? |
| if | selected(data('person\_age'), 'yes') |   |   |   |   |
| begin screen |   |   |   |   |   |
|   |   | select\_multiple | topping\_list | pizza\_type | What are your favorite kind of pizza toppings (select up to 3)? |
|   |   | integer |  | num\_slice |  How many slices would you like? |
| end screen |   |   |   |   |   |
| else |   |   |   |   |   |
|   |   | note |  |   | You are too young to be eating pizza |
| end if |   |   |   |   |   |

The **choices** sheet allows you to specify the set of choices for multiple choice prompts. The **data\_value** column in the choice worksheet contains the value that will be assigned if the choice is selected. The **display.title.text** column is what the user sees as the choices.

So the corresponding **choices** worksheet would look like this:

| *Adding Multiple Choice Questions Example Choices Worksheet* |
| --- |
| **choice\_list\_name** | **data\_value** | **display.title.text** |
| yes\_no | yes | Yes |
| yes\_no | no | No |
| topping\_list | pepperoni | Pepperoni |
| topping\_list | olives | Black Olives |
| topping\_list | onions | Onions |
| topping\_list | mushroom | Mushrooms |
| topping\_list | pepper | Green Peppers |
| topping\_list | bacon | Canadian Bacon |
| topping\_list | pineapple | Pineapple |

Now, instead of typing their age, the user simply selects whether they are older than 18 or not. Furthermore, instead of entering the type of pizza they like, they can select from a list of toppings.

**Tip**

Because you determine whether a question is select\_one or select\_multiple from the **survey** worksheet, the same choice set on the **choices** worksheet can be used for both select\_one and select\_multiple questions.

## [Instance](https://docs.opendatakit.org/odk2/xlsx-converter-using/%22%20%5Cl%20%22id24) Names

By default, instances (observations or rows in a form’s table) are named things like “2017-07-02T19:46:53.975Z” (date and time). We can assign an instance name in the **settings** worksheet based on a variable from the **survey** worksheet.

In the example below, we assign the instance name of the members form to be the member’s name, where name is a variable in the survey sheet:

*Instance names settings Worksheet Example*

| **setting\_name** | **value** |
| --- | --- |
| form\_id | member |
| form\_version | 2018.08.01 |
| table\_id | member |
| default |  |
| instance\_name | name |

## [Using Custom Section Worksheets](https://docs.opendatakit.org/odk2/xlsx-converter-using/#id24)

Custom section worksheets can be added to a workbook to make the control flow of a survey more readable. We could move all the previous questions about pizza to a new worksheet and name it **Pizza**. Our **survey** worksheet would then look like this:

| *Custom Section Worksheets Example* |
| --- |
| **clause** | **condition** | **type** | **values\_list** | **name** | **display.prompt.text** |
| do | section Pizza |   |   |   |   |

**Tip**

When splitting a survey into different sections, it is wise to put a note before each section call with **display.prompt.text** set to read Section <name\_of\_section>. This is because a do section <name\_of\_section> call is not visible to the user. Unless the form designer explicitly adds a note, the user will not realize that they entered a section.

Also, after leaving a section, if the user swipes back, the survey will go to the row before the do section call. If the user then swipes forward at this point, the survey will go to the beginning of the section they just completed. It is often beneficial to the user to put a note before entering a section and before leaving a section.

## [Using Calculations](https://docs.opendatakit.org/odk2/xlsx-converter-using/#id25)

The **calculates** worksheet is an optional worksheet. It consists of two columns:

* **calculation\_name**: Each row of the **calculates** page represents a function that can be used elsewhere in the workbook by referencing the individual **calculation\_name**.
* **calculation**: The calculation to be performed.

**Note**

The **calculation** column can store any valid JavaScript expression.

**Tip**

There are also some built in functions for ODK Survey that can be used anywhere in the workbook. See the [Formula Functions](https://docs.opendatakit.org/odk2/xlsx-converter-reference/%22%20%5Cl%20%22xlsx-ref-formula) for more details.

In general, calculations are referenced in the **condition** column of **survey** worksheets. For example, suppose that on the **survey** page under the variable name birthday the user entered their birthday for a question of type date. The **calculates** worksheet might look like this:

| *Calculates Worksheet Example* |
| --- |
| **calculation\_name** | **calculation** |
| daysOld | (now().getTime()-new Date(data('birthday')).getTime())/1000/60/60/24 |
| isBirthdayToday | calculates.daysOld()%365 == (now().getTime()/1000/60/60/24)%365 |

and one of the **survey** worksheets may look like this:

|  | *Calculation Survey Worksheet Example* |
| --- | --- |
| **clause**  | **condition** | **type** | **name** | **display.prompt.text** |  |
| if | calculates.isBirthdayToday() |  |   |   |   |
|   |  | note | happyBirthday | Happy Birthday! |  |
| end if |  |   |   |   |   |

Notice that the <**calculation\_name**>s do not contain parentheses () at the end of them. However, when referencing them it is always in the format of **calculates.<calculation\_name>()**.

**Tip**

Variable names have scope for the entire workbook.

The **calculates** worksheet is handy because it adds readability to a workbook. Instead of having long, complicated JavaScript calculations in the **survey** worksheets, they can be consolidated in one, easy to reference location that allows for reusability. Also notice the consistent use of camelCase for variable naming across the different worksheets.

## [Using Queries](https://docs.opendatakit.org/odk2/xlsx-converter-using/#id26)

The **queries** worksheet is an optional worksheet that allows you to request data from external sources and linked tables (subforms).

For queries that get their data from external sources (for example, csv files), the following columns should be used:

* **query\_name**
* **query\_type**
* **uri**
* **callback**

For linked\_table queries, these columns should be used:

* **query\_name**
* **query\_type**
* **linked\_table\_id**
* **linked\_form\_id**
* **selection**
* **selectionArgs**
* **order\_by**
* **newRowInitialElementKeyToValueMap**
* **openRowInitialElementKeyToValueMap**

Each row of the queries page represents a choice set that can be used by select prompt types in the workbook. In general, **query\_name** is referenced in the **values\_list** column of **survey** worksheets.

Queries Using Csv Files

csv files are an ideal external source if your select prompt question calls on an expansive list. Additionally, csv files allow for you to use a **choice\_filter** to limit choices based on responses to previous questions**.** For example, suppose that on the **survey** page under the variable name region the user is asked to select the region they are from. Then the user is asked to select which country they are from. The choices for the list of countries can be filtered based on the region the user selected. The **queries** worksheet might look like this:

| *Queries Worksheet Example Number 1* |
| --- |
| **query\_name** | **query\_type** | **uri** | **callback** |
| regions\_csv | csv | "regions.csv" | \_.chain(context).pluck('region').uniq().map(function(region){return {data\_value:region, display:{title: {text: region} } };}).value() |
| countries.csv | csv | "regions.csv" | \_.map(context, function(place){place.data\_value = place.country;place.display = {title: {text:place.country} };return place;}) |

The data for the queries is coming from the regions.csv file that is located in the same directory as the formDef.json and specified in the **uri** column. Thus, the **query\_type** for both queries is csv. A snippet of the regions.csv file looks like the following:

| *regions.csv* |
| --- |
| **region** | **country** |
| Africa | Algeria |
| Africa | Angola |
| Africa | Benin |

Knowing the structure of the regions.csv helps in understanding the callback function provided in the **callback** column. The callback function maps the results from the regions.csv file to the **data\_value** and the **display.prompt.text** fields using JavaScript. The **survey** worksheet may look like this:

| *Queries Survey Worksheet Example Number 1* |
| --- |
| **clause** | **condition** | **type** | **values\_list** | **name** | **display.prompt.text** | **choice\_filter** |
| begin screen |   |   |   |   |   |   |
|   |   | select\_one\_dropdown | regions\_csv | birth\_region | Please select your birth region: |   |
|   |   | select\_one\_dropdown | countries\_csv | birth\_country | Please select your birth country: | choice\_item.region === data('birth\_region') |
| end screen |   |   |   |   |   |   |

The **choice\_filter** in this example ensures that the options that get displayed for the birth\_country question will only be the countries from the previously selected birth\_region. Notice that choice\_item.region is referring to the column titled “region” from the csv and specifies that any country with a corresponding region equal to the answer stored by the birth\_region question (data('birth\_region')) will be displayed. The countries were mapped to regions in the **callback** column of the **queries** worksheet.

In a separate example, if we needed a **choice\_filter** for a three (or more)-level list, the **callback** column in the **queries** worksheet would look like this:

| *Queries Worksheet Example Number 2* |
| --- |
| **query\_name** | **query\_type** | **uri** | **callback** |
| region\_csv | csv | " regions.csv" | \_.chain(context).pluck(region).uniq().map(function(region){return {name: region, label: region, data\_value: region, display: {title: {text: region } } };}).value() |
| country\_csv | csv | " regions.csv" | (function() { var seen = { };  return \_.chain(context).filter(function(place) { var keep = (seen[place. country] !== true); seen[place. country] = true; return keep; }) .map(function(place) {place.name = place. country;place.label = place. country;place.data\_value = place.name;place.display = {title: {text: place.label} };return place;  }).value();})() |
| city\_csv | csv | " regions.csv" | \_.map(context, function(place){place.name = place. city;place.label = place. city;place.data\_value = place.name;place.display = {title: {text: place.label} };return place;}) |

And the choice filter column would look like:

| *Queries Survey Worksheet Example Number 2* |
| --- |
| **clause** | **condition** | **type** | **values\_list** | **name** | **display.prompt.text** | **choice\_filter** |
| begin screen |   |   |   |   |   |   |
|   |   | select\_one\_dropdown | regions\_csv | birth\_region | Please select your birth region: |   |
|   |   | select\_one\_dropdown | countries\_csv | birth\_country | Please select your birth country: | choice\_item.region === data('birth\_region') |
|  |  | select\_one\_dropdown | city\_csv | birth\_city | Please select your birth country: | choice\_item.country === data('birth\_country) |
| end screen |   |   |   |   |   |   |

### [Linked Tables](https://docs.opendatakit.org/odk2/xlsx-converter-using/#id27)

**linked\_table** is the other use for the **queries** worksheet. **linked\_table** allows you to launch a subform that can edit a different data table. For example, if a survey is dealing with information about households, the user may want to ask questions about the general household but also questions about specific members. **linked\_table** can be used to launch subforms that ask questions about the specific household members.

 The **survey** worksheet may look like this:

| *Linked Table Survey Worksheet Example* |
| --- |
| **clause** | **condition** | **type** | **values\_list** | **name** | **display.prompt.text** | **choice\_filter** |
|   |   | text |   | house\_id | Input the unique household id: |   |
|   |   | integer |   | num\_members | How many people live in this house? |   |
|   |   | linked\_table | members |   | Add and enter information for the different household members |   |
|   |   | select\_one | members | household\_head | Who is the household head? |   |

The **queries** worksheet would look like this:

| *Linked Table Query Worksheet Example* |
| --- |
| **query\_name** | **query\_type** | **linked\_form\_id** | **linked\_table\_id** | **selection** | **selectionArgs** | **newRowInitialElementKeyToValueMap** |
| members | linked\_table | members\_info | house\_members | house\_id = ? | [ opendatakit.getCurrentInstanceId() ] | { house\_id: opendatakit.getCurrentInstanceId() } |

First the user enters a house\_id for the household and answers an arbitrary question about its residents. This information is stored in the data table for general household information (specified on the **settings** worksheet under **table\_id**). Then the user reaches a linked\_table prompt that uses the **values\_list** members. This is connected to the members query on the **queries** worksheet. It links to a different survey (subform) called members\_info that edits a different data table. The selection criteria is that the house\_id in the house\_members data table matches the instanceID of this current household.

Initially this list will be empty since no members have been added. The user can click on the **Create Instance** button to add new members for this household. The house\_id will be set automatically for this new member via the **newRowInitialElementKeyToValueMap** content, which specifies that the house\_id field in the linked table should be initialized with the instanceID of the current household.

**Note**

The selection criteria and its type (in this case, house\_id and text) must be added to the **model** sheet of the subform (members\_info) in order for selection criteria to be persisted to the database and for the subform to be found by its parent form; the selection criteria cannot filter on session variables since those values are never persisted.

When the user finishes the subform, the screen will return to the same linked\_table prompt in the parent form. At this point, the user can continue adding more members to the household, edit an existing member's info, or go to a different screen.

The **values\_list** for the select\_one question prompt in the example above also uses the members query. Instead of being able to launch subforms to edit information about different members, the selection criteria is used to populate a multiple choice question. The answer to the multiple choice question is saved to the general household data table, not the members data table.

## [Order](https://docs.opendatakit.org/odk2/xlsx-converter-using/%22%20%5Cl%20%22id26) By

**order\_by** is an optional column used in the **queries** worksheet. It is particularly useful for select prompts that query instances that were created through subforms. To use **order\_by** first put the name of the variable (in the subform) you want to order followed by the direction. ASC is for ascending and DES is for descending. The default direction is ascending if nothing is specified.

The **queries** worksheet would now look like this:

*order\_by queries Worksheet Example*

| **query\_name** | **query\_type** | **linked\_form\_id** | **linked\_table\_id** | **selection** | **selectionArgs** | **newRowInitialElementKeyToValueMap** | **order\_by** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| members | linked\_table | members\_info | house\_members | house\_id = ? | [ opendatakit.getCurrentInstanceId() ] | { house\_id: opendatakit.getCurrentInstanceId() } | age ASC |

The **survey** worksheet would remain the same as the linked table example:

| *order\_by Survey Worksheet Example* |
| --- |
| **clause** | **condition** | **type** | **values\_list** | **name** | **display.prompt.text** | **choice\_filter** |
|   |   | text |   | house\_id | Input the unique household id: |   |
|   |   | integer |   | num\_members | How many people live in this house? |   |
|   |   | linked\_table | members |   | Add and enter information for the different household members |   |
|   |   | select\_one | members | household\_head | Who is the household head? |   |

In this case, after we have completed all of our household member subforms, we are again asked the select\_one question prompt about the household head, but this time our **order\_by** will have the members from the household in ascending order by their age from our members query (in this example, age was asked in the subform).

## [Internationalization](https://docs.opendatakit.org/odk2/xlsx-converter-using/#id28)

Survey offers the ability to display text in different languages. This requires usage of the **settings** worksheet to establish an additional language to use. Extra display columns need to be added for any language other than the default language. For example, if one of the non-default language options was Spanish (2-letter language code "es"), every worksheet with a **display.prompt.text** or **display.title.text** column would also need a **display.prompt.text.es** or **display.title.text.es** column. This is true for all columns that need an alternate language option.

The **setting** worksheet now will look like this:

*Internationalization settings Worksheet Example*

| **setting\_name** | **value** | **display.title.text** | **display.title.text.es** | **display.locale.text** | **display.locale.text.es** |
| --- | --- | --- | --- | --- | --- |
| form\_id | household |  |  |  |  |
| form\_version | 2018.08.01 |  |  |  |  |
| table\_id | household |  |  |  |  |
| default |  |  |  | English | Inglés |
| es |  |  |  | English | Español |
| survey |  | Household form | Forma de hogar |  |  |

Thecolumns **display.locale.text** and **display.locale.text.es** show what the language options are in each locale.

The **survey** worksheets now may look like this to give the users the option answer questions in either language:

| *Internationalization framework\_translations Worksheet Example* |
| --- |
| **type** | **name** | **display.prompt.text** | **display.prompt.text.es** |
| text | user\_name | What is your name? | ¿Cuál es su nombre? |
| integer | user\_age | How old are you? | ¿Cuántos años tienes? |

The labels used in the buttons and prompts supplied by ODK Survey are defined in the **framework\_translations** sheet of the framework.xlsx file under config/assets/framework/forms/framework.xlsx Simply add your language code and translations to this sheet of this XLSX file and run **XLSXConverter** on it to enable support of your language across all of the built-in buttons and prompts within ODK Survey.

**Tip**

* If you receive message about an error in the template after you have added an additional language, double check that all display columns in the framework and survey workbooks are translated for both the default language and the non-default language. The error often occurs if the column headers are wrong or not yet added, or if any of the questions are available in one language option and not the other. For example, in the *Internationalization framework\_translations Worksheet Example if the* user\_name asked “¿Cuál es su nombre?” in Spanish, but had an empty cell for the default language, we would get an error.