SurveyCTO

Help - Designing forms - Advanced topics

Below, please find information about more advanced form-design techniques.

Randomizing survey elements

You can easily generate random numbers in your form and use those random numbers to influence your skip logic. In SurveyCTO, random draws are always between 0 and 1, drawn from the uniform distribution – so there is, for example, a 50% chance that a given draw will be between 0 and 0.50.

In the form designer, click to + *Add hidden field*, select *calculate* as the field type, and click *Configure*. Next, give your random-number field a name, click *Specify with a wizard* for the calculation, and select *Generate a random number*. You can then refer to that field, for example, in another field or group's *relevance* conditions.

In the spreadsheet form definition, add a *calculate* field with "once(random())" in its *calculation* column. For example, you might have a calculate field named *randnum1* with the calculation expression "once(random())". Once you have drawn a random number, you can then refer to it in your *relevance* expressions (e.g., "${randnum1} <= 0.50").

For a working example, see *Randomization: Randomizing form elements*.

Dynamically filtering lists of multiple-choice options

As discussed in the Core concepts help topics, you can use *select_one* and *select_multiple* fields to ask multiple-choice questions, and you can specify all of the valid choices in the *choices* sheet of your workbook. In the standard case, every user answering a multiple-choice question is always shown every available option.

However, it is also possible to filter the list of options shown to users, based on their prior selections. Such filtering is made possible by a feature called "cascading selects."

In the form designer, you can add a *filter* value for any multiple-choice option, and you can add a *choice_filter* expression for any multiple-choice field – but you'll need to choose those filter values and expressions as discussed below, in the much the same way you would if you were editing the spreadsheet form definition directly.

This feature is best explained in the context of a simple example form, which you can select as a starting-point when you click + and then *Start new form* to start a new form from the Design tab: enable the *Use a sample form as your starting point* option, then choose *Cascading selects: Filtering multiple-choice option*

lists from the list of samples available. In this example form, the user is asked to select the survey region, then the country, then the city. All of these are simple *select_one* fields, but they have been configured to filter the options so that the country list is appropriate to the selected region and the city list is appropriate to the selected country.

To filter the country and city lists, two special columns are used:

1. **The *filter* column on the *choices* sheet.** For countries, this column contains the region code for which each country should be listed; for the cities, it contains the country code for which each city should be listed. The values specified in the *filter* column correspond to the values specified earlier in the *value* column (i.e., you filter by the internal values stored in the form data, not by the choice labels).
2. **The *choice_filter* column on the *survey* sheet.** This specifies, for each field, which prior field should be used when filtering the list of options. For the survey_country field, the *choice_filter* is "filter=${survey_region}"; this means that, when showing the country list, SurveyCTO should compare the *filter* column of the *choices* sheet with the user's selection from the survey_region field – and only show those choices for which the filter matches (i.e., for which the given filter expression is true). The *choice_filter* for the survey_city field is "filter=${survey_country}", so that only those cities in the selected country are shown. For more details about this example, see *Cascading selects: Filtering multiple-choice option lists*.

More advanced usage is also possible. For example, rather than having just one *filter* column in the *choices* sheet, you could have more than one, each with a unique name (e.g., *filter_region* and *filter_country*). You could then specify more complex expressions in the *choice_filter* column of the *survey* sheet (e.g., "filter_region=${survey_region} and filter_country=${survey_country}"). However, a single filter column works for most cases, and the online form designer only supports a single one.

For an example where the filtered choices have dynamic labels, see *Rosters: Choosing among earlier entries*. (That particular sample also demonstrates that you can use different types of expressions for the *choice_filter* expression, as in its "${numfamily}>=filter" expression.)

Pre-loading data into a form

If you have pre-existing data from a baseline survey, a listing survey, or any other source, you may want to reference that data in your survey form. For example, you might already have data on a household's agricultural plots when you go to collect follow-up data about those plots.

Methods of attaching pre-loaded data

When you want to reference pre-existing data in a survey form, you can pre-load that data into your form in one of two ways:

1. By directly attaching .csv data to your survey form.
2. By creating a server dataset, then attaching that dataset to your survey form.

Both methods are conceptually the same: you are attaching data to your form.

In the first case, you directly attach the data as one or more .csv files when you upload your form (the same way you attach image, audio, or video support files when designing your form in the designer or uploading a form to the *Your forms and datasets* section of the Design tab). The data becomes effectively part of your

form, and when you want to update that data you have to upload a new version of the form with a new version of the data attached.

In the second case, you first create the dataset by scrolling down to the *Server datasets for pre-loading data* section of the Design tab, clicking *New dataset*, and uploading your .csv file under a unique title and ID of your choosing. You then attach that dataset to your form by clicking the *Attach* button under your newly-created dataset and then selecting the form(s) to which you would like to attach it. In this case, you are still uploading and attaching a .csv file that contains your data – but this second option offers several key advantages over attaching the data directly:

1. **You can later update the data without having to edit or upload a new version of your form.** Simply return to the *Server datasets for pre-loading data* section, find your dataset, and click *Upload*; you'll be given the option to use your new .csv upload to either fully replace the existing data or append to it. As soon as you update the dataset, any forms to which that dataset is attached will be automatically updated. You won't need to upload updates to the forms themselves.
2. **Second, your datasets can be updated automatically, with data streaming in from other forms**. See *Publishing form data into server datasets* for more on that possibility (but your subscription has to support streaming of data into server datasets for that to work).

Format of pre-loaded data

However you attach data to your forms, the format of your pre-loaded data should be the same:

1. The first row of each .csv file that you upload should consist of a header that assigns short, unique names for each column. Subsequent rows should then contain the data itself.
2. Columns in your .csv files must be separated by commas. Depending on your regional settings, Excel may save .csv files with semi-colons or other characters between columns, instead of commas; if that's the case, you will need to manually replace the column separators with commas, or save your data using a different program.
3. At least one column should uniquely identify each row of pre-loaded data. Such columns will be used at survey time to look up which row's data to pull into the survey. For the columns that will be used for looking up rows, you should add "_key" to the end of the column name (as in "hhid_key"). Any columns with names ending in "_key" will be indexed for faster look-ups on your survey devices.

Doing the actual pre-loading

The easiest way to use pre-loaded data in your survey form is to indicate that you'll be using pre-loaded data when you start a new form from the Design tab. In the *Your forms and datasets* section, click + then *Start new form*, enable *Advanced options*, and mark the checkbox next to *Auto-generate fields necessary for pre-loading data*; when prompted to choose your pre-loaded data, select a .csv file or server dataset to give an example of how your pre-loaded data will look (it doesn't have to be the final data, but it should have the correct columns); confirm that the data looks correct; identify which column uniquely identifies each row; and answer a few other questions regarding how the survey form should identify which row to load. Your new form will start out with all of the fields necessary to pre-load your data.

You can also add pre-loading fields to your form manually, of course.

In the form designer, click to + *Add hidden field*, select *calculate* as the field type, and click *Configure* to add a field that you want to pull into your survey. Next, give your field a name, click *Specify with a wizard* for the calculation, and select *Pull pre-loaded data from attached dataset or .csv file*. Finally, attach the appropriate .csv or dataset, specify the column name for the column you want to pull into your form, specify the column name for the column you want to use for looking up the appropriate row, and indicate which form field has

the value to use in looking up the row. The wizard will save the appropriate calculation for you, and you can then refer to this new calculate field whenever you want to refer to the pre-loaded data elsewhere in your form.

In the spreadsheet form definition, add one *calculate* field to the *type* column of the *survey* sheet for each data field that you want to pull into your survey. Give that field a name, and use the pulldata() function in its *calculation* column to indicate which field to pull from which row of which server dataset or .csv file. Here is an example:

pulldata('hhplotdata', 'plot1size', 'hhid_key', ${hhid})

This example expression will pull the value from the *plot1size* column of the attached dataset with ID *hhplotdata* – or from the directly-attached .csv file named *hhplotdata.csv*, if there is one – using the form's *hhid* field to identify the appropriate row in the pre-loaded data's *hhid_key* column.

Rather than writing it out by hand, you can use the calculation-builder to construct your own pulldata() expression. The builder is available as both an option on the server (from the Design tab, click *Tools* under *Your forms and datasets*) and as an option in *SurveyCTO Sync* (on the *Tools...Form tools* menu). Once you run the builder, choose *Pull pre-loaded data from attached dataset or .csv file* as the calculation type, then enter your details and click *Build*. You can just copy and paste the resulting expression into your form's *calculation* column.

Once you have inserted pre-loaded data into a survey field using the pulldata() function, you can reference that field in later relevance or constraint expressions, or in other fields' labels. You can refer to your pre-loaded field as *${fieldname}*, just as you would any other field that was calculated or filled in by the user.

For a working example, see *Pre-loading: Referencing pre-loaded .csv data*.

Pre-loading into editable fields

The technique described above pre-loads data into hidden *calculate* fields, which you can then reference in labels, constraints, or relevance expressions. Instead, you may want to pre-load data directly into a visible, user-editable field. That is also possible.

In the form designer, change the *default* setting to *Calculated default* when adding or editing any visible field. Then choose *Add with a wizard* and follow the same process as described above.

In the spreadsheet form definition, just include the proper pulldata() expression in the *calculation* column of the editable field. That's basically it: you format the pulldata() expression in the same way described above, and you put it in the *calculation* column, also in the same way as above. The only difference is the field type of the row in which you place the expression: here, it's a regular, editable field type rather than a *calculate* field.

There are a few other things to consider when dynamically loading an editable field's default value, however, so see *Providing default entries or selections for fields* for more details.

Additional options and considerations

Five additional notes on usage of pre-loaded data:

1. If your .csv data contains non-English fonts or special characters, you will need to save your .csv file in UTF-8 format in order for Android devices or web browsers to render the text correctly. If you cannot directly save or export your .csv file in UTF-8 format, you can use *SurveyCTO Sync* to re-encode it: choose *Re-encode .csv file* from the *Tools* menu, select your file and the encoding for which its text appears correctly in the preview window, and then save the re-encoded .csv file.

UTF-8-encoded .csv text should appear properly, provided that your Android device or web browser supports the font or language that you're using.

2. Even when they are numbers, data fields pulled from a dataset or .csv file are considered to be text strings. Thus, you may sometimes need to use the int() or number() functions to convert a pre-loaded field into numeric form (for example, before performing some mathematical calculation with it, like "number(${preloadedinterestrate})*100").
3. SurveyCTO will ignore any .csv columns that appear after a fully-blank column, so be sure not to include blank columns in your pre-loaded data (i.e., the first row should have names for every column present in your .csv file).
4. If your .csv file is very large and you're attaching it directly to your form, you can compress it into a .zip archive before uploading it. Just compress one or more of your form's support files into a .zip file and upload that .zip as you would normally upload a .csv or other support file. When your form is loaded, the support files will be automatically unzipped.
5. If your .csv file contains very sensitive data, you may not want to upload that data to your SurveyCTO server. Instead, you can upload a blank .csv file as part of your form, then replace it with the real .csv file by copying that file onto each of your devices: once you've downloaded your form onto each device, locate the /SurveyCTO/forms directory on that device's storage; within that directory, locate the media subdirectory associated with your form; into that media subdirectory, you can copy your .csv file. That way, sensitive pre-loaded data never passes through the SurveyCTO server.

Finally, you can also use pre-loaded data to dynamically populate the choice lists for *select_one* and *select_multiple* fields. See the following help topic on *Loading multiple-choice options from pre-loaded data* for details.

Loading multiple-choice options from pre-loaded data

See the help topic just above for details on how to attach pre-loaded data to your survey form. Once your form has pre-loaded data attached, you can dynamically pull the choice lists for *select_one* and *select_multiple* fields from that data.

If your goal is to allow the user of your survey form to select the appropriate row for which you would like to pre-load data, your best bet is to start a new form with + and then *Start new form*, and to have the pre-loading code automatically added to your new form (as described in the topic above). You will be able to specify that the user should choose the data to pre-load in one, two, or even three steps (e.g., choosing district, then village, then household), and you will be able to select which pre-loaded data columns to use as the choice labels for each step.

In the form designer, you can check *Dynamically load choices from pre-loaded data* in the *appearance* of any *select_one* or *select_multiple* field. You can then use a wizard to choose which items (rows) to include from which attached dataset or .csv file. Finally, in the actual choice list, put just one choice that has the appropriate column names in the *label*, *value*, and (optionally) *image* fields; all matching rows will be loaded as choices, and the choice labels, values, and (optionally) image filenames will come from the columns you specify there in the choice list. (For example, I might put "name" as the *label* for my one choice and "id_key" as the *value*, if I wanted to list the names in the "name" column and record user selections using the ID numbers in the "id_key" column.)

In the spreadsheet form definition, multiple-choice fields with dynamic choice lists follow the same overall syntax as regular, static *select_one* and *select_multiple* fields: specify "select_one listname" or "select_multiple listname" in the *type* column of your *survey* sheet (where "listname" is the name of your choice list; e.g., select_one country), specify any special appearance styles in the *appearance* column, and include one or more rows for your listname on the *choices* worksheet. When the choice list should be pulled from pre-loaded data, there are just three important differences in the syntax to be aware of:

1. In the *appearance* column of the *survey* worksheet, you should include a search() expression that specifies which rows of pre-loaded data to include in the choice list. If you want to specify another appearance style, that should be entered into the *appearance* column first, followed by a space, followed by the search() expression (as in "quick search()"). Syntax for the search() expression itself is included below.

2. On the *choices* worksheet, a row should indicate which pre-loaded data columns to use for the label and selected value. In the *list_name* column, specify the name of your choice list as you normally would. In the *value* column, however, include the name of the column to use for uniquely identifying selected choices (this is normally an ID column of some sort). In the *label* column, include the name of the column to use for labeling the choices; if you wish to include multiple columns in the labels, just include a comma-separated list of all columns to include. At survey time, the *name* column will be dynamically populated based on the column name you put there, and the *label* column will be dynamically populated based on the column name(s) you put there.

3. Also on the *choices* worksheet, you can include a column name in the *image* column. If you do, the image filename to use will be pulled from the specified column. If you refer to image files in this way, you must always upload those image files as media file attachments when you upload your form to the SurveyCTO server (as you normally would).

For the search() expression itself, there are a series of options to indicate which rows of pre-loaded data to include in the choice list:

- "**search(source)**": This single-parameter search expression includes all distinct rows as choices; you just have to specify the data source, which is either the name of the attached .csv file or the unique ID of the attached dataset (e.g., "search('hhplotdata')" if either the *hhplotdata.csv* file was attached or a dataset with the ID *hhplotdata* was attached). All rows in the specified dataset or .csv file will be considered as choices, but only *distinct* rows – those with unique selection values – will be listed for the user. In other words, duplicates will be automatically filtered from the list shown to users.

- "**search(source, 'contains', columnsToSearch, searchText)**": This search expression includes all distinct rows that contain the specified text in the specified column(s) (e.g., "search('hhplotdata', 'contains', 'respondentname', ${nametofind})"). The third parameter specifies either a single column name to search, or a comma-separated list of column names to search. Rows with matches in any specified column will be included.

- "**search(source, 'startswith', columnsToSearch, searchText)**": This search expression includes all distinct rows that start with the specified text in the specified column(s) (e.g., "search('hhplotdata', 'startswith', 'respondentname', ${nameprefix})"). The third parameter specifies either a single column name to search, or a comma-separated list of column names to search. Rows with matches in any specified column will be included.

- "**search(source, 'endswith', columnsToSearch, searchText)**": This search expression includes all distinct rows that end with the specified text in the specified column(s) (e.g., "search('hhplotdata', 'endswith', 'respondentname', ${namesuffix})"). The third parameter specifies either a single column

name to search, or a comma-separated list of column names to search. Rows with matches in any specified column will be included.

- "**search(source, 'matches', columnsToSearch, searchText)**": This search expression includes all distinct rows that exactly contain the specified text in the specified column(s) (e.g., "search('hhplotdata', 'matches', 'respondentname', ${nametofind})"). The third parameter specifies either a single column name to search, or a comma-separated list of column names to search. Rows with exact matches in any specified column will be included.
- "**search(source, searchType, columnsToSearch, searchText, columnToFilter, filterText)**": Finally, any of the four search types above can be further filtered to only include a subset of pre-loaded data. Simply add two extra parameters to any of the search types above, with the first extra parameter being the column name to filter and the second extra parameter being the exact value to filter. For whichever search is specified in the first four parameters, only rows exactly containing the sixth parameter value in the column named by the fifth parameter will be included (e.g., "search('hhplotdata', 'contains', 'respondentname', ${nametofind}, 'villageid', ${villageid})" to list all matching names within a particular village).

If you're editing your spreadsheet form definition directly, you can use the calculation-builder to construct your own search() expression rather than write it out by hand. The builder is available as both an option on the server (from the Design tab, click *Tools* under *Your forms and datasets*) and as an option in *SurveyCTO Sync* (on the *Tools...Form tools* menu). Once you run the builder, choose *Pull pre-loaded data into a multiple-choice prompt* as the calculation type, then enter your details and click *Build*. You can just copy and paste the resulting expression into your form's *appearance* column.

For a working example, see *Pre-loading: Searching and selecting from pre-loaded data*.

Two additional notes on usage:

1. Choices will be ordered, by default, in the order that they appear in your dataset or .csv file. If you want to specify a different ordering, include a numeric column in your dataset or .csv file named *sortby*; if present, choices will be ordered numerically, according to this *sortby* column.
2. You can include one or more static choice options, in addition to the dynamic ones loaded from your pre-loaded data. Simply include static choices as you normally would, on the *choices* worksheet. These can appear before and/or after the row that indicates the columns to use for your dynamic choices. The only restriction is that the values you specify for your static choices in the *value* column must be numeric.

Dynamically naming filled-in forms

By default, whenever a new form is filled out, it is named the same thing: the title of the form. Unless users override this default and enter a different name when they save each form, the *Edit Saved Form*, *Send Finalized Form*, and *Delete Saved Form* lists in *SurveyCTO Collect* will show lots of forms all with the same name; the only way to distinguish them will be based on their "last saved" time and date.

Alternatively, you can automatically name each form in a way that utilizes the data entered into the form. For example, you might include a unique household ID and/or a respondent name in the name of the form. To do this, you need to specify an *instance_name* setting in your form's *settings* worksheet.

There is currently no way to dynamically name filled-in forms using the online form designer. However, you can choose *Export* to save the form definition from the designer, follow the instructions below to add the necessary code to the spreadsheet form definition, and then choose *Upload* on the Design tab to re-upload the form to your server. You can then re-open it and continue editing it in the designer. (While the designer won't show the option for dynamically naming forms, it also won't remove or overwrite anything that has been placed in the spreadsheet form definition.)

In the spreadsheet form definition, start by adding a new column to your form's *settings* worksheet and entering "instance_name" into the first row of that new column. Then, in the second row of that new column, add an expression like the following:

concat('HH SURVEY - ', ${hhid})

Replace "HH SURVEY" with whatever prefix you want to include in form names, and replace "hhid" with the name of a field contains a unique or meaningful identifier for your form. Alternatively, if you want to include multiple fields, you could enter an expression like this:

concat('HH SURVEY - ID ', ${hhid}, ' - Name ', ${respondent_name})

In this example, both a household ID and a respondent name are included in the form's name. That way, when you list filled-out forms on a device, you will be able to more easily find specific instances.

Using variations of the concat() function, you can concatenate together an arbitrary number of literal strings (enclosed by single-quotes) and an arbitrary number of form fields (enclosed by ${...}). Note that whatever you put in the instance_name column should evaluate to a string; so, for example, "today()" alone won't work to name forms using the current date, but "string(today())" will work. See *Using expressions in your forms* for more on the kinds of expressions you can use.