NubiS

SYSTEM ADMINISTRATOR MANUAL

USC Dornsife Center for Economic and Social Research

April 2021

PREFACE

Supporting every successful survey is a strong data collection tool.

At the University of Southern California's Center for Economic and Social Research (CESR), we have developed NubiS, accessible and versatile software for administering a questionnaire. NubiS:

- Runs on any server, PC, laptop or netbook, as well as on Android tablets or smartphones.
- Works with surveys that are self-administered, face-to-face and via telephone (*i.e.*, all traditional modes of data collection).
- Can collect continuous information from smartphones, tablets or other external sources, such as accelerometers, GPS devices and blood-pressure meters.

Following NubiS' development in 2014, it has been the foundation of CESR's Understanding America Study, a panel of thousands of households representing the entire United States. NubiS also has been used in several large-scale longitudinal surveys and other projects around the world.

You, as a system administrator, want a survey that is easy to install, program and run – yet not so simple as to elicit only the most rudimentary information. We think you will find NubiS provides the underpinnings for data collection of breadth and depth while remaining user-friendly to you, surveyors and researchers.

For more information, feel free to contact us. CESR also can help with access to the Understanding America Study's panel, as well as providing assistance for survey hosting, design and management.

We're proud NubiS can support the quest for greater knowledge and wish you the best with your surveys.

Sincerely,

The NubiS team

TABLE OF CONTENTS

1. GETTING STARTED

Installing and setting up NubiS has been designed to be as easy as possible. Therefore, it should work on almost any computer, operating system and browser, as long as the requirements listed below are met. However, if you encounter problems during the installation process, help can be found by contacting nubissurveying@gmail.com.

1.1 System Requirements

NubiS has been designed as a web application and as such its system requirements are similar to those of any other web application:

- A web server (*e.g.*, Apache).
- A database. Currently, MySQL is natively supported; support for other databases is available on request.
- A scripting engine. NubiS is written in PHP and runs on PHP 5.3 or higher.

Required to access the NubiS interface is a modern browser (*e.g.*, Internet Explorer 9 or higher, Firefox 23 or higher, Safari, Opera and Chrome) with JavaScript supported and enabled. Recommended but not mandatory is a device with a relatively large screen size.

Meanwhile, respondents can access a survey using any modern browser with JavaScript enabled. There is no requirement in terms of device or operating system. Note that visual limitations may be imposed by the survey design itself, such as by the display of a large table on a smartphone screen, but this is not inherently imposed by NubiS.

1.2 Component Details

NubiS comes bundled with third party software. None of them impose any restrictions on usage EXCEPT FOR Highcharts, which is free to use for academic/non-profit usage only. The following third party software is bundled:

Name	Location	License	Version
Bootstrap	http://getbootstrap.com/	Apache License, Version 2.0	3.0.0
Bootstrap additional scripts (affix.js, alert.js, button.js, carousel.js, collapse.js, dropdown.js, modal.js, popover.js, scrollspy.js, tab.js, tooltip.js, transition.js)	http://getbootstrap.com/	Apache License, Version 2.0	3.0.0
Bootstrap-calendar, app.js, appsurvey.js	https://github.com/Serhioromano/bootstrap-calendar	MIT license	0.1
Bootstrap-colorpicker	http://mjolnic.github.io/bootstrap-colorpicker/	Apache License, Version 2.0	Not provided
Bootstrap-datetimepicker	https://github.com/Eonasdan/bootstrap-datetimepicker	MIT License	4.14.30
Bootstrap-formhelpers	http://bootstrapformhelpers.com	Apache License, Version 2.0	2.3.0
Bootstrap-select	http://silviomoreto.github.io/bootstrap-select	MIT License	1.6.3
Bootstrap-session-timeout	http://www.orangehilldev.com	MIT License	Not provided

Name	Location	License	Version
Bootstrap-slider	https://github.com/seiyria/bootstrap-slider	Apache License, Version 2.0	1.0.1
Bootstrap-switch	http://www.bootstrap-switch.org	Apache License, Version 2.0	3.2.2
Bootstrap-touchspin	https://github.com/istvan-ujjmeszaros/bootstrap-touchspin	Apache License, Version 2.0	3.0.0
Bootstrap-wysiwyg	http://github.com/mindmup/bootstrap-wysiwyg	MIT License	Not provided
BrowserDetect	https://github.com/sinergi/php-browser-detector	MIT License	Not provided
Calculator.js	http://codepen.io/GianNipitella/pen/vNjqyE	Not provided	Not provided
Codemirror	http://codemirror.net/	Custom (no restrictions on usage)	3.19
Datatables	http://www.datatables.net	MIT License http://www.datatables.net	1.10.2
Datatables bootstrap	http://www.datatables.net	MIT License http://www.datatables.net	1.10.2
Highcharts	http://www.highcharts.com	Free for **academic/non-profit usage** only. See www.highcharts.com/license for details.	4.0.3

Name	Location	License	Version
Highcharts Exporting	http://www.highcharts.com	Free for **academic/non-profit usage** only. See www.highcharts.com/license for details.	4.0.3
Highchart Export-csv	https://github.com/highcharts/export-csv	MIT License	4.0.3
Hover-dropdown	https://github.com/CWSpear/bootstrap-hover-dropdown	MIT License	Not provided
Javascript Only Barcode_Reader	https://github.com/EddieLa/BarcodeReader	MIT License	1.0
JShrink	https://github.com/tedious/JShrink	BSD License	Not applicable
Jquery.appendgrid	https://appendgrid.apphb.com/	Dual licensed under the LGPL and MIT licenses	1.5.0
Jquery.calc	https://github.com/zoul0813/jquery-calc	Not provided	2.0.0
Jquery.cookie	https://github.com/carhartl/jquery-cookie	MIT License	1.4.1
Jquery.dirtyform	https://github.com/snikch/jquery.dirtyforms	MIT License	1.2.0
Jquery.dirtyforms.helpers.tinymce (bundled with Jquery.dirtyform	https://github.com/snikch/jquery.dirtyforms	Custom (no restrictions on usage)	1.2.0

Name	Location	License	Version
for TinyMCE integration)			
JQuery fileupload	https://blueimp.github.io/jQuery-File-Upload/	MIT license	9.28.0
Jquery-hotkeys (standalone and bundled with Bootstrap-wysiwyg)	https://github.com/jeresig/jquery.hotkeys	Dual licensed under the MIT or GPL Version 2 licenses	Not provided
Jquery-inputmask	http://github.com/RobinHerbots/jquery.inputmask	MIT License	Not provided
Jquery.sidr	https://github.com/artberri/sidr	MIT License	Not provided
Jquery.textcomplete	https://github.com/yuku-t/jquery-textcomplete	MIT License	Not provided
JqueryUI	http://jqueryui.com	MIT License	1.10.4
Jquery.validate	http://bassistance.de/jquery-plugins/jquery-plugin-validation/	MIT License	1.11.1
jsTimezoneDetect	http://pellepim.bitbucket.org/jstz/	MIT License	1.0.5
LZ-string	http://pieroxy.net/blog/pages/lz-string/testing.html	WTFPL, Version 2	1.4.4

Name	Location	License	Version
MobileDetect	https://github.com/serbanghita/Mobile-Detect	MIT License (custom)	2.8.17
Modernizr	http://pellepim.bitbucket.org/jstz/	Dual licensed under the BSD and MIT licenses (www.modernizr.com/license/)	2.6.2
Moment.js	momentjs.com	MIT License	2.8.1
Mootools	http://mootools.net/	MIT license	1.4.5
PHPParser	https://github.com/nikic/PHP-Parser	Custom (no restrictions on usage)	Not provided
roundSlider	https://roundsliderui.com/	MIT license	1.3.3
TableSaw	https://github.com/filamentgroup/tablesaw	MIT License	2.0.2
TabWindowVisibilityManager	http://greensock.com/forums/topic/9059-cross-browser-to-detect-tab-or-window-is-active-so-animations-stay-in-sync-using-html5-visibility-api/	Custom (no restrictions on usage)	1.0
TinyMCE	https://www.tinymce.com/	GNU Lesser General Public License, Version 2.1	4.2.1
Underscore	https://github.com/jashkenas/underscore	Custom, no restrictions (https://github.com/jashkenas/underscore/blob/master/LICENSE)	Not provi

1.3 Installation

To install NubiS, follow the steps below:

1. Set up the MySQL database in which NubiS will store its data and meta-data.
2. Download the tar/zip file (depending on your platform).
3. Extract it into the web-accessible directory of your server (*e.g.*, on Linux for Apache, **/var/www/html/**). By default, NubiS will be extracted into a sub-directory called **nubis**. The name of the directory may be changed but keep in mind the resulting URL is where respondents will access the survey.
4. Locate the **conf.php** file in the NubiS root directory and ensure your web server has write rights to it.
5. Open your browser and point it to the install.html file in the root location of NubiS (*e.g.*, www.example.org/nubis/install.html). Click **Start installation** to start the wizard.
6. Follow the installation wizard's steps. Note that only the database specifics are required, so after entering those to complete the installation you can simply click the **Finish** button; NubiS then will use a default configuration. Alternatively, advanced users may want to explore the different tabs following the **Database** tab to look at configuration options (most of which also can be modified later within NubiS).
7. During the installation process a system administrator access key is automatically generated. NubiS will require the access key to be entered any time someone logs into a system administrator account. Failing to do so will result in NuibS denying access to the system administrator interface. If you want to disable this feature, then after set up has been completed locate **conf.php** in the root NubiS directory and change the line starting with CONFIGURATION_ENCRYPTION_SYSADMIN to:

   CONFIGURATION_ENCRYPTION_SYSADMIN => "".
8. Upon completion of the wizard, the NubiS login screen will appear. You can log on with the default account and password **sysadmin**/**sysadmin**. If you opted to set a system administrator key, NubiS will also ask for the configured key. To further set up NubiS, follow the steps in Sec. 4.
9. Make sure your web server no longer has permissions to write to **conf.php** in the root NubiS directory.
10. On Mac operating systems PHP does not have a save cookie path or tmp dir set by default. Make sure to set it manually. Refer to the PHP manual for instructions.

1.4 Interaction Between the Nubis Sample Management System and Survey System

NubiS is comprised of two main components in terms of functionality:

1. Sample management system (SMS): provides functionality required to manage a sample. This includes activities such as uploading a sample, updating respondent information and tracking the progress of data collection (*e.g.*, in terms of number of completed interviews). Because every data collection project takes place in its own unique context with unique requirements and constraints, the sample management system within NubiS should be thought of as a common-denominator sample management system. That is, the NubiS SMS contains commonly required functionality, but is not intended to be a one-size-fits-all SMS.
2. Survey system: provides functionality required to program and run a survey. This includes activities such as adding questions, defining skip patterns and downloading data files. It is designed to be for programming a survey while ensuring flexibility for any custom requirements (such as ways for respondents to answer questions).

The SMS and survey system can be used in conjunction, while the survey system can be employed in a stand-alone fashion. You can also use the survey system with your own SMS as long as the sample management system interacts with the NubiS survey system in a prescribed fashion.

2. CONCEPTS

NubiS uses a number of common concepts in its interface and documentation. These are, for the most part, in line with typical terms employed within data collection, but it is good to familiarize yourself with NubiS' lexicon.

2.1 General Concepts

NubiS is a data collection tool designed for a variety of interview modes and languages. An **interview mode** (or mode, for short) expresses the manner in which a survey is being administered. NubiS supports four modes:

1. Face-to-face interviewing
2. Telephone interviewing
3. Web-based (self) interviewing
4. Data entry

In each of these modes, any number of **languages** can be used. The default language is English but this is customizable. It is possible to use both character-based (such as Japanese or Chinese) and non-character based language (such as Spanish or French).

2.2 Sample Management System Concepts

On a high conceptual level, a sample will be either household-based or respondent-based. A **household-**based sample is one in which multiple respondents from the same unit of grouping (*e.g.*, a household) are part of the sample. Within NubiS, this means individual respondents are grouped within a household, their data marked so they can easily be associated with the data of other respondents within the same household, and tracking is initially done on the household level (with the option to drill down to an individual respondent's level). In contrast, in a **respondent-**based sample, respondents may belong to the same unit of grouping – but this is not considered during data collection. As a result, tracking of the sample is done on a respondent level and not on a household level.

During the data collection period, there will be interaction with respondents. To keep track of such interactions, NubiS utilizes the notion of contacts. Each **contact** captures an interaction in terms of, for example, its mode, the time at which the interaction occurred and the outcome. Typically, contacts occur in the context of face-to-face or telephone interviewing. In those contexts, contacts

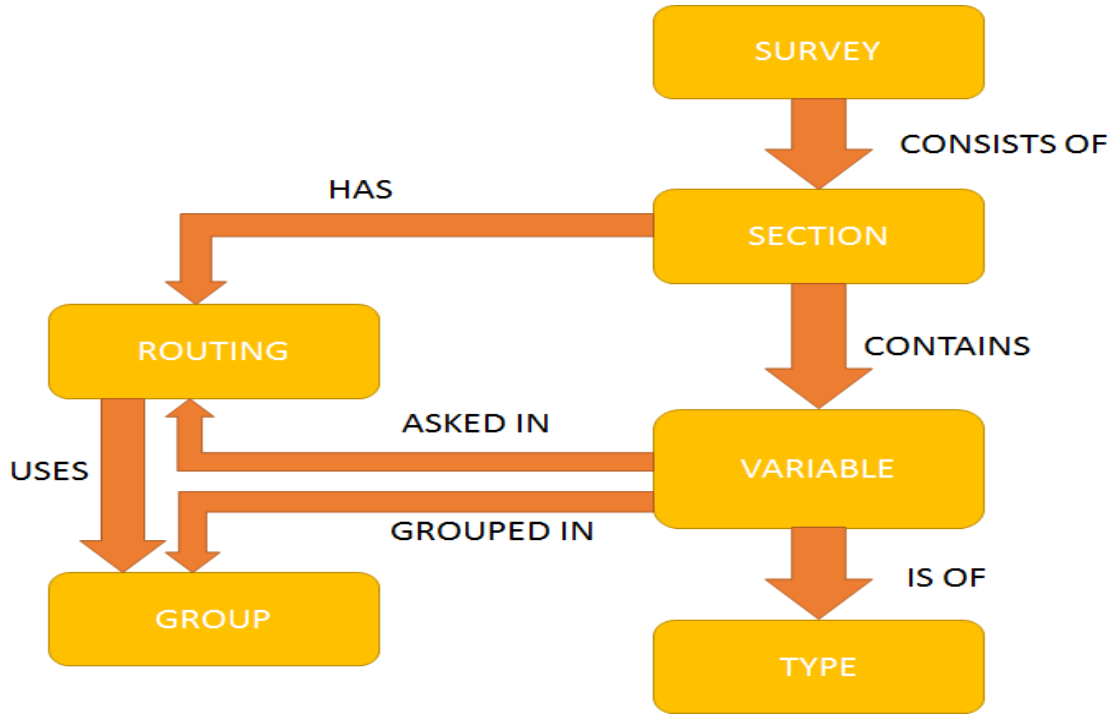
are either automatically generated by the sample management system or added by an interviewer or supervisor.

An **interviewer** represents a person responsible for conducting interviews with respondents, be it by talking to them on the phone or by meeting them in person. Interviewers are managed by a supervisor. A **supervisor** represents the person within the data collection project responsible for tracking progress, assigning interviews to interviewers, closing interview cases if a respondent refuses to participate, etc. If the sample size is sufficiently high, there may be a need for multiple supervisors, who themselves can be supervised by higher-level supervisor(s).

Other roles within the context of sample management represented in NubiS are nurses and researchers. A **nurse** represents a person responsible for entering data such as height and weight. A **researcher** represents a person who will be using the data collected.

2.3 Survey system concepts

The survey system in NubiS is designed to operate on and utilize the different objects displayed in the diagram below. As shown, a questionnaire in NubiS is represented by the concept of a **survey**. Multiple surveys can exist within NubiS (*e.g.*, a baseline and follow-up questionnaires). A survey comprises all the components required to program and run a survey: sections, variables, types, groups, routing and settings.



Typically, a survey is divided into sections. A **section** is a collection of logically-related variables. A **variable** can represent a question you want to ask, text to present on the screen, a placeholder for internal purposes (*e.g.*, performing a calculation) or for deriving so-called dynamic text (*i.e.*, text dependent on previous respondent answers).

A variable can be of a certain type. A **type** is a mechanism for sharing similar settings across multiple variables, such the answer options *Yes* and *No*.

Variables can be referenced in the routing of sections in order to ask questions to the respondent. The **routing** of a section contains the instructions the survey system must follow to administer the survey to a respondent. To combine multiple variables on the same screen, these instructions can include skip patterns, loops and groups. A **group** in this regard defines the collection of variables to be combined, the layout to be used and other characteristics to be applied.

Lastly, but not shown in the diagram above, each of the concepts discussed has an associated collection of settings. A **setting** represents a characteristic with a value. These values can be dependent on the interview mode and language in which the survey is conducted. This detail will be further explored in Sec. 16.

3. RUNNING EXAMPLE

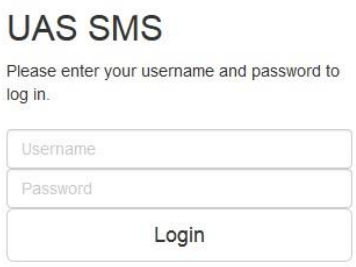
Over the remainder of this manual, to illustrate concepts and the manner in which they are realized and implemented in NubiS, used as an example will be the Health and Retirement Study (HRS). This is a longitudinal panel study that surveys, every two years, a representative sample of approximately 20,000 Americans older than 50. The study is comprised of a series of interrelated surveys that collect information about income, work, assets, pension plans, health insurance, disability, physical health and functioning, cognitive functioning and expenditures on health care. Detailed information about the HRS may be found here.

The examples to be used mostly draw upon the development of Sections A and A2 of these surveys, which focus on collecting general background information about respondents as well as their household composition. For further illustrative purposes, this manual also may reference other sections of the HRS.

4. SETTING UP YOUR FIRST SURVEY

After successfully completing the installation of NubiS, you can start working on your survey's development. To do so, use your browser to open the location in which NubiS was installed. For example, if NubiS was installed in your server's /nubis/ directory of your server www.example.org, the URL to browse to is www.example/org/nubis/index.php?se=2 (the se=2 part informs NubiS you wish to access its administrative backend).

The first screen to appear is the login screen:



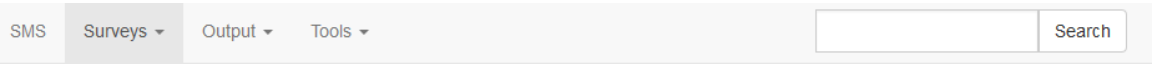
As mentioned in the installation procedure, by default NubiS comes with a single system administrator account:

Username: *sysadmin*

Password: *sysadmin*

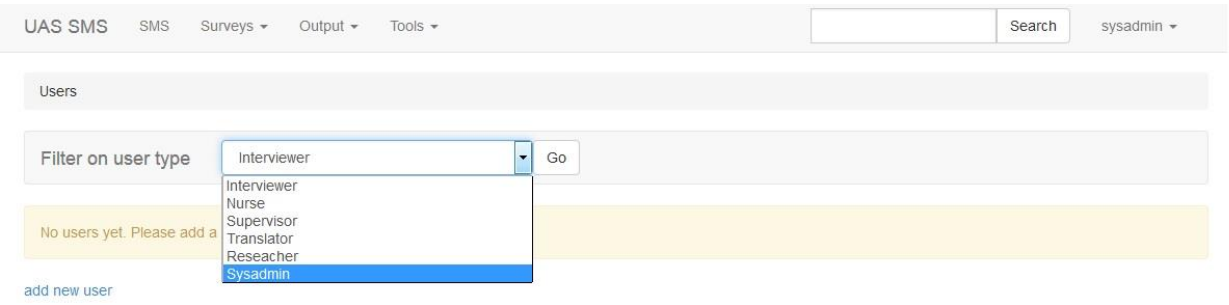
Log on with the above credentials. If you specified a system administrator key, in the next screen NubiS will prompt you to enter it. After that, the NubiS system administrator interface will open:

For later use, note the search function in the top right corner of the screen:

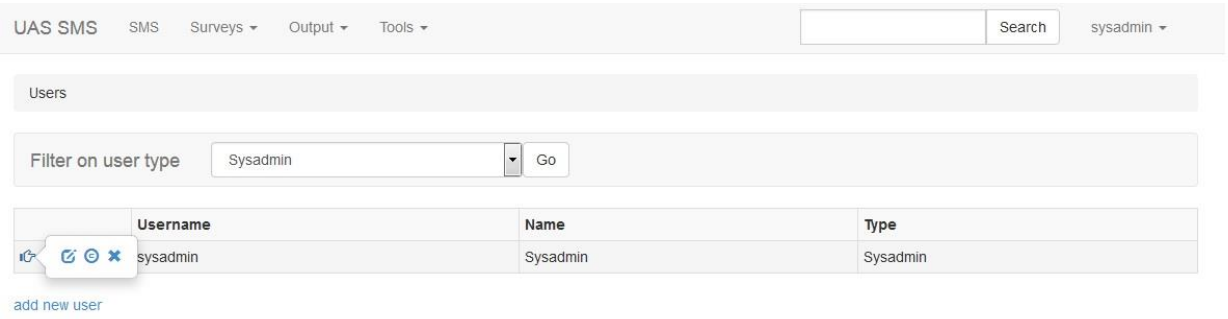


After clicking on the search button NubiS will present a list of results on the left-hand side in a results bar that slides into the screen from the left.

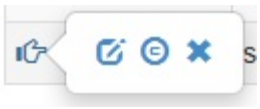
Now, change the password from the default. To do this, in the dropdown in the top right corner, click **Users**:



In the **Filter on user type** dropdown, select **Sysadmin** and click **Go**. This will bring up the overview of all known system administrators (in this case, the single default account):

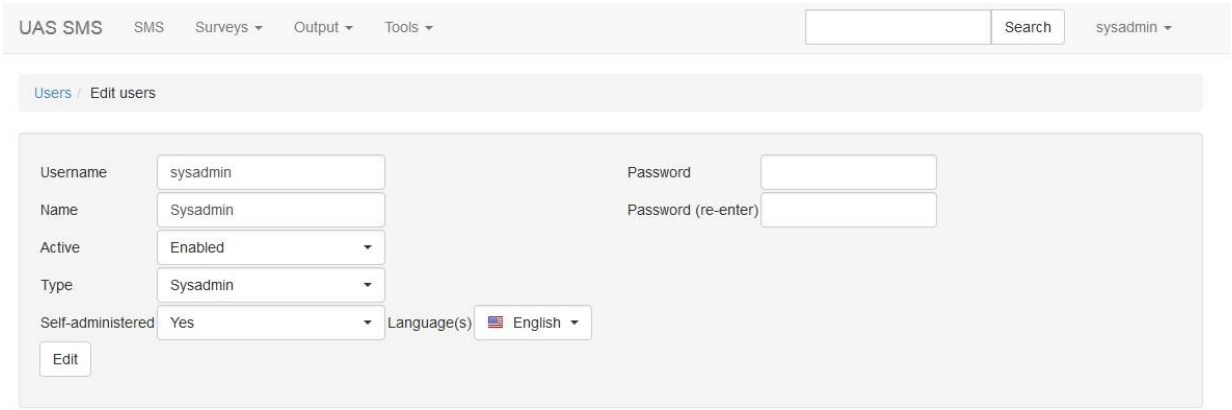


You will see that as your cursor hovers over the pointing finger icon…

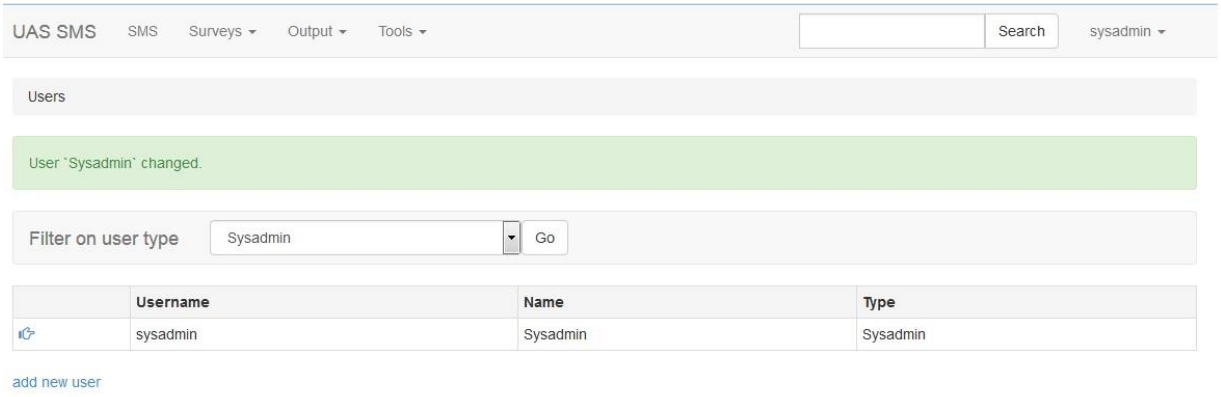


…several options appear. This feature consistently appears throughout the NubiS interface, such as in the variables list screen. Here, as elsewhere, clicking on the pointing finger itself equals editing (in this case the user account). The pencil symbol also represents editing, the copyright symbol represents copying, whereas the cross symbol conveys deleting. As you learn how to add sections and variables, several other symbols will be introduced.

For now, return to editing the sysadmin account by clicking the pointed finger symbol or the pencil symbol:



In the password boxes on the right, twice enter your new password and click Edit.



A message will appear confirming the sysadmin account has been updated. Confirmation messages in NubiS always are in green, while yellow messages provide information and red messages represent errors.

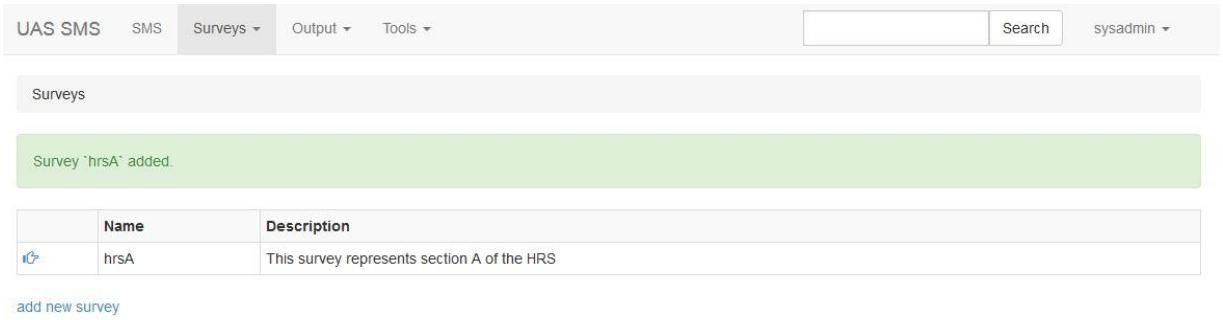
With the sysadmin account updated, now you can turn your attention to starting a survey. On installation, NubiS comes without any pre-defined surveys. To add one, in the top navigation bar, locate UAS SMS:

Click on **Surveys** to show the list of current surveys (currently empty):

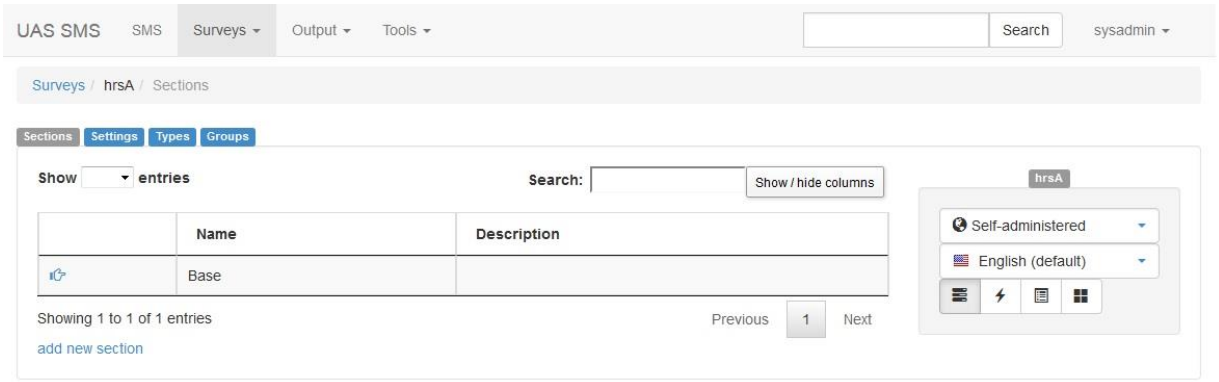
Next, click **add new survey**:

In this screen, enter a name for the survey (which should be unique across surveys), a title (which will appear as the title of the browser window when the survey is being programmed and, later, filled out) and a description. Note that upon the addition of more surveys, available is the option to indicate which the survey is the default. In this regard, the default survey is what NubiS launches if no other indicator is provided on which survey is to be started.

Name the survey **hrsA**, as it will represent section A of the Health and Retirement Study, and enter a description. On clicking **Add**, NubiS will create a new survey:



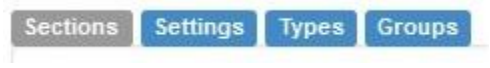
To start editing the new survey, click the pointed finger (options to copy or delete a survey also are available on hovering over the pointed finger). This brings up the following screen:



There are several things on this screen worth noting. First, the screen reflects the manner in which most NubiS screens are laid out when components are being added, edited or removed. Typically, the list of components (*e.g.*, the list of sections here) is shown on the left, whilst the right side gives access to the selections for interview mode and language, as well as shortcuts to other types of components. NubiS' defaults for interview mode and language are, as seen above, self-administered in English.

Below the dropdowns for interview mode and language are icons giving access to, respectively, a survey's sections, settings, types and groups:

Those also can be navigated to by using the blue headers above the sections list:



For now, focus on the sections list itself. Later in this manual, you will learn about other components as well as how to develop surveys for multiple interview modes and/or languages.

In the sections list, there already is one section: Base. It comes standard with NubiS and cannot be deleted (nor can its name be edited). This standard section contains a set of variables used by NubiS to administer surveys.

Click the pointed finger on *Base* to open the section's contents. (Please note this is the only instance in NubiS where clicking on the pointed finger does not give the option to edit the section.) To edit the section's properties, such as its name or description, the pencil symbol for that section must be clicked.

The *Base* section contents screen looks like this:

	prim_key	primary key	PRIMARY KEY
	begintime	timestamp start	TIMESTAMP START
	endtime	timestamp end	TIMESTAMP END
	version	version info	VERSION INFO
	mode	interview mode	INTERVIEW MODE
	language	interview language	INTERVIEW LANGUAGE
	platform	platform and browser information	PLATFORM AND BROWSER INFORMATION
	introduction	Welcome to this survey. 	INTRODUCTION SCREEN
	thanks	This is the end of the interview.	THANKS SCREEN
	completed	We already received your responses. Thank you!	ALREADY COMPLETED SCREEN
	locked	The system encountered an error and as a result your interview was locked. Please contact your survey administrator.	PROCESSING SCREEN
	direct	This survey is only accessible from an external page.	DIRECT ACCESS ONLY SCREEN
	execution_mode	execution mode	EXECUTION MODE
	login	Please enter your login code and click 'Next' to start the survey. 	LOGIN SCREEN
	closed	The survey is currently closed. Please try again later.	CLOSED SCREEN

As the screen shows, there are two main tabs: variables and routing (also accessible through the second right-side menu with the header *Base*). The list of standard variables is as follows:

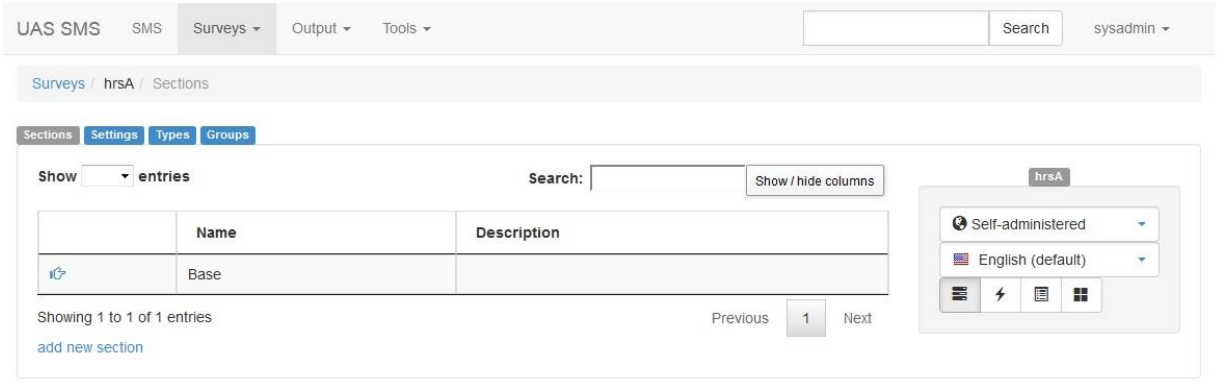
- prim_key: the identifier for an administered survey. Can consist of both alpha and numeric characters.
- begintime: the time at which a survey is started.
- endtime: the time at which a survey is completed.
- version: the version of the survey.
- mode: the interview mode in which the survey is conducted.
- language: the language in which the survey is conducted.
- platform: the browser user agent string capturing the operating system and browser used.
- template: the template in which the survey is conducted.
- introduction: the introduction screen displayed on survey start.
- thanks: the thank-you screen displayed on survey completion.
- completed: the already completed screen displayed on attempting to re-enter an already completed survey (if re-entry is not allowed).
- locked: the locked screen displayed if the interview was locked in response to a malfunction.
- direct: the direct access-only screen displayed, but only when entering the survey is allowed from a website external to NubiS.
- execution_mode: the mode of survey execution, which can be normal or test.
- login: the login screen shown when entering the survey is done by means of a login code.
- closed: the screen shown when a respondent attempts to access the survey while it is not open.
- inprogress: the screen shown when a respondent attempts to access the survey while a previous request is still being processed.

NOTE: if you are planning to use NubiS in a multi-mode and/or multi-language project, you should carefully consider which interview mode and language you choose as your survey's default interview mode and language, since changing these mid way can have unintended consequences. Please have a close look at Section 16 of this manual prior to starting development of your survey.

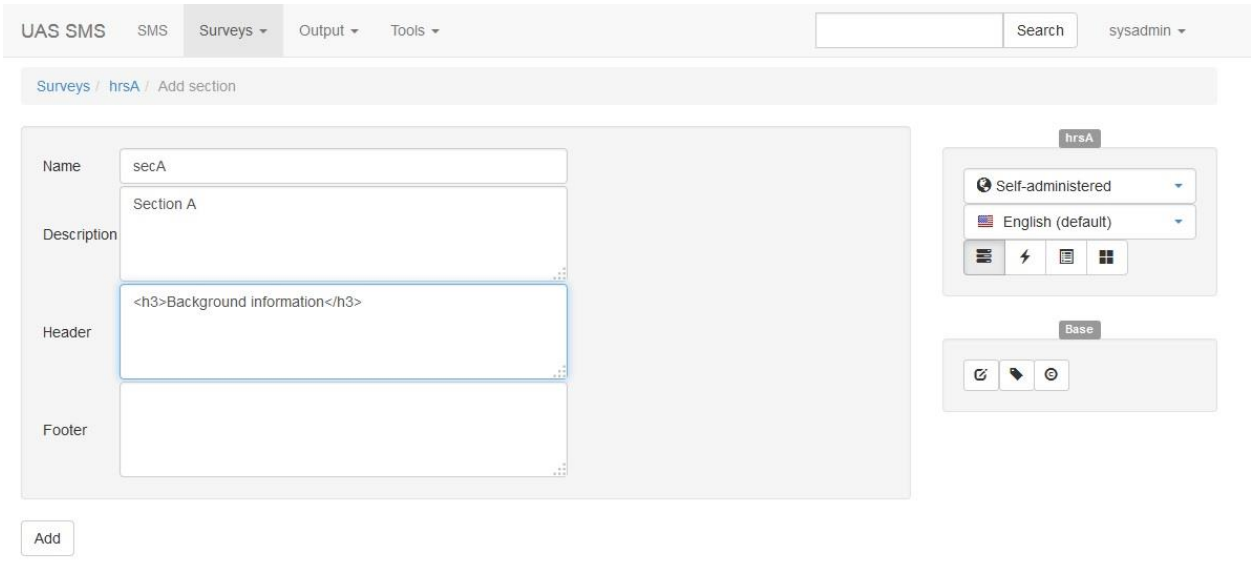
5. ADDING YOUR OWN CONTENT

Though the *Base* section is useful, it does not contain any content needed for creating your survey. To do that, you need to add sections and variables. Begin by adding a new section that will group together the variables to ask. Note: In principle, it is possible to simply add variables to the *Base* section. However, for programming convenience and the purpose of clarity, you are advised to add your own section(s); and if your survey is relatively long, break it up into multiple sections.

To add a new section, move back to the sections list by clicking on *hrsA* in the navigation links just below the top navigation bar:



On this screen, click **add new section**:

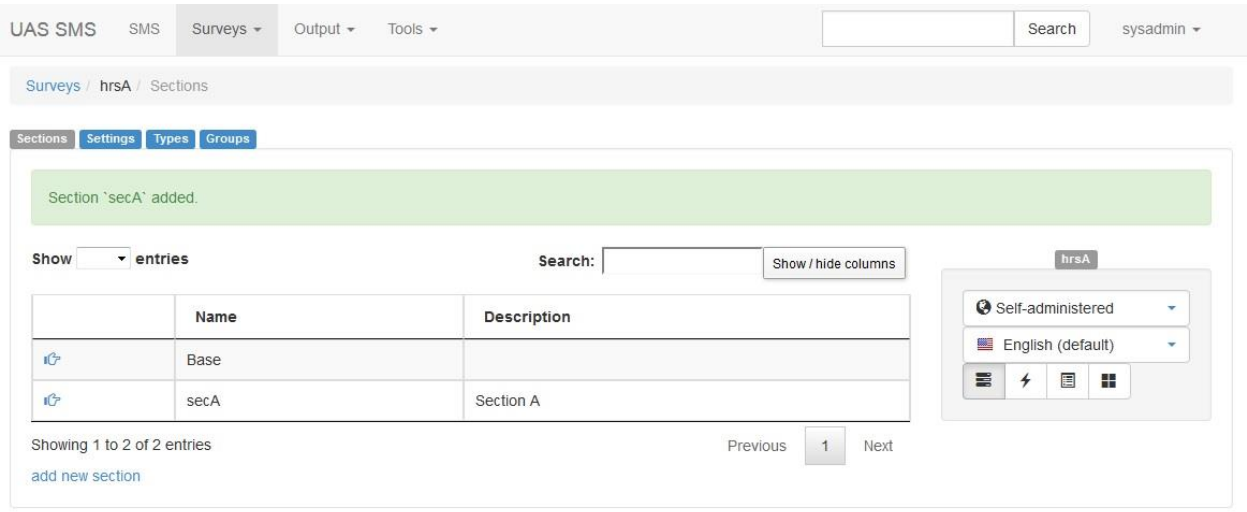


As you will be adding section A and section A2 of the HRS, start by creating a section called **secA**. Give a short description as well, which will make it easier to discern between sections in the list. Lastly, optionally specify a section header, which will be inserted by NubiS at the top of the survey

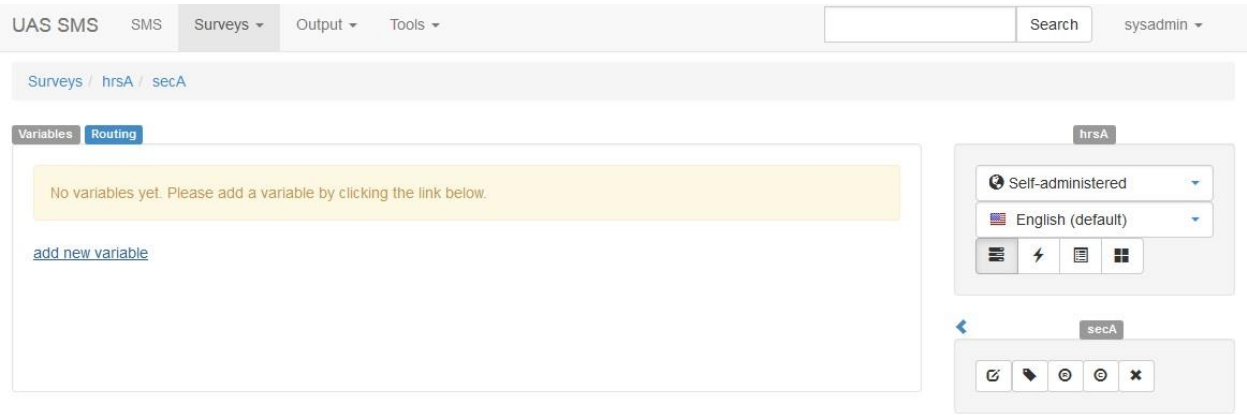
screen. This will help respondents keep track of where they are in the survey. In the example above, note that HTML tags (<h3></h3>) were used. For elements to be seen by a respondent (*e.g.*, section header, question/answer text), you may use HTML. NubiS treats any HTML tags in properties as regular text; consequently, it gets outputted as is. Now the section header appears as:

Background information

After the section is added, NubiS will confirm and show the updated sections list:

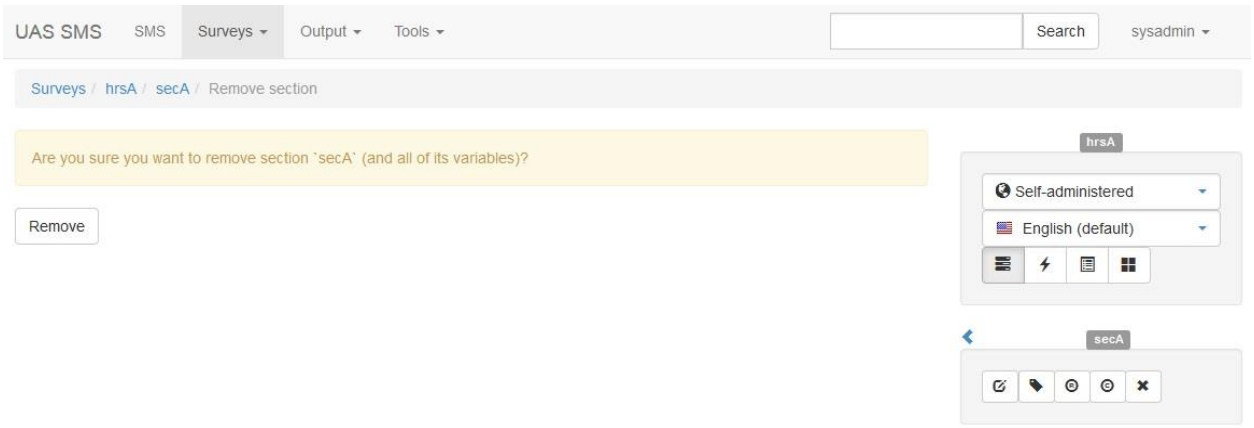


Go to the new section by clicking the pointed finger to the left of *secA*:



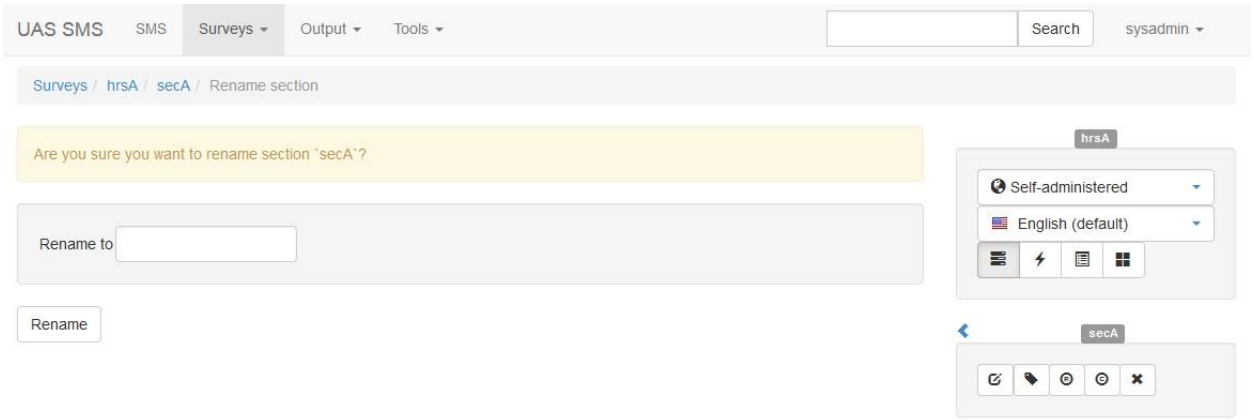
As expected, no variables are yet present.

Note that in the right-side menu for *secA*, there are two extra options compared to the *Base* section. One is the refactor option, represented by the circle with an R in it, and the other is the delete option. The delete option will prompt a confirmation:



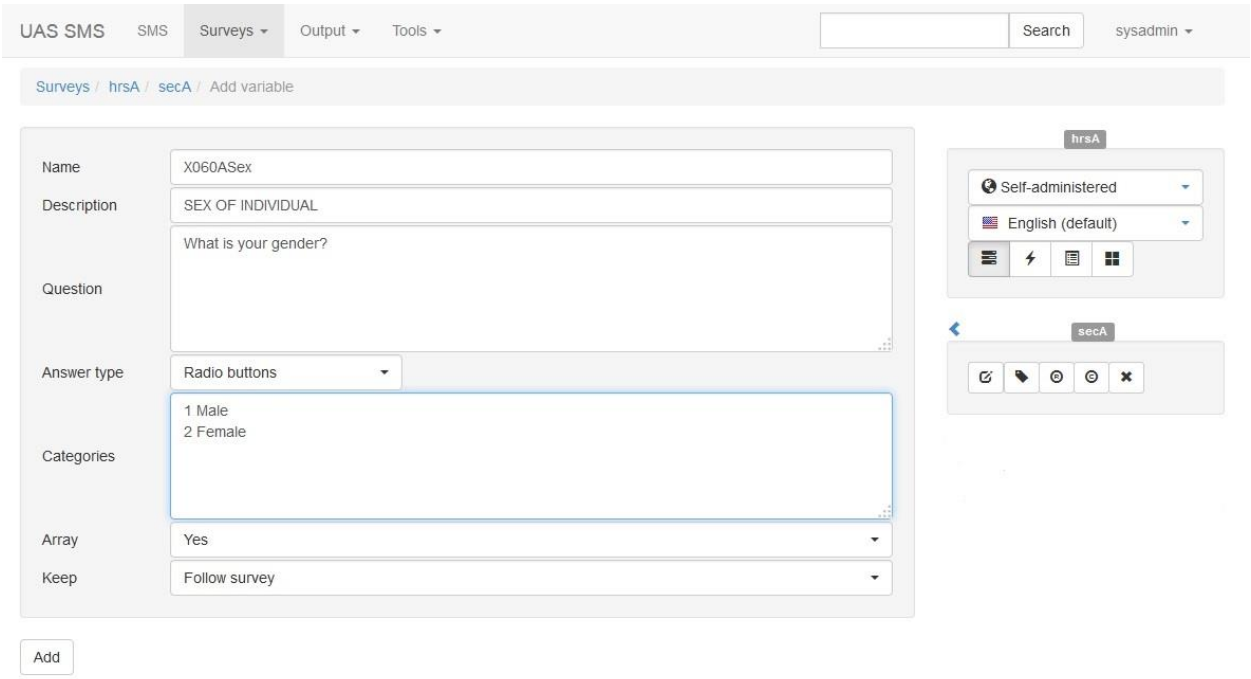
Only upon clicking **Remove** will the section (and its variables) be deleted. This series of steps applies to any delete option within NubiS; that is, all removal actions must be confirmed.

The refactor option allows you to modify the name of a component (in this case a section):



Refactoring, like deletion, also must be confirmed. The intent of refactoring is to modify the name of a component in such a way that NubiS automatically updates any other component within the survey that is referencing it (which is particularly useful for large surveys).

Click **add new variable**:



Specifically, add a question to ask the gender of the respondent. The screen above shows the specifics of the question: a name, a description, a question text, the answer type, answer categories (for specific answer types only), and whether the question is an array and whether it should be kept. (What "kept" implies will be explored in Sec. 12.).

Start with the name. Following the HRS, the question is named **X060ASex**. It may not always be the case, however, that the survey specification lists names for its questions. In these instances, you will need to devise a naming convention. As a general rule, when defining a convention the most important factor is consistency; that is, it should be adhered to throughout the survey. Question names can be thematic (such as **gender** for the question above), a more logical scheme (*e.g.*, naming questions in the form **secA_001**, **secA_002**, etc.) or a combination of both (as seen above).

NubiS imposes no restriction on what name is chosen except that it must start with a letter and otherwise consists only of letters, numbers and underscores (_). NubiS is case insensitive, (*i.e.*, *secA_001* is the same as *SECA_001*.) There is no maximum length, but for usability concise names are preferred. Also, the name must be unique within the survey. Attempting to add a variable, section, group or type with an existing name will result in an error:

The description entered in this example is SEX OF INDIVIDUAL. The capitalization here is by choice and not imposed by NubiS; the reason is to improve the text's readability when it is used as a variable label in a Stata data file to be created for this survey. When possible these variable descriptions should be short yet informative about what the variable represents.

Next comes the question text, here simply, What is your gender? HTML markup can be used in these entries.

Less straightforward is the next characteristic, which is the answer type. Here, radio buttons have been selected to show the question's possible answers: Male or Female.

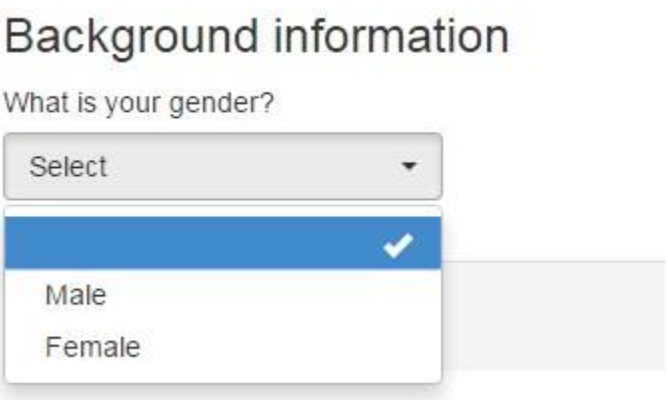
The following default answer types are available:

- String: allows a respondent to give a short textual answer.

  - Open: allows a respondent to give a longer textual answer.

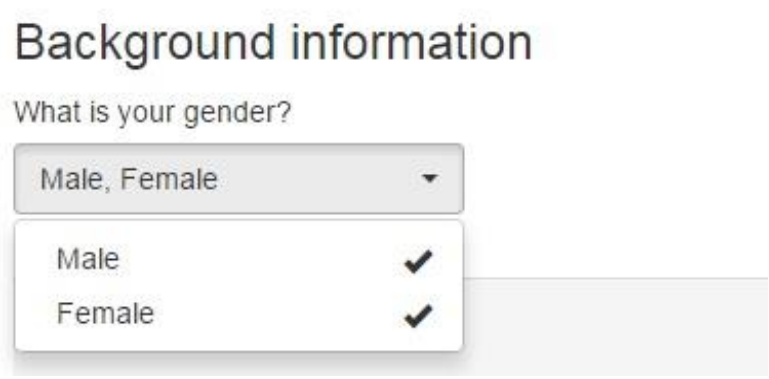
  - Radio buttons: allows a respondent to choose a pre-defined option.

- Drop down: allows a respondent to choose from pre-defined options in a dropdown menu.



- Checkboxes: like radio buttons, allows a respondent to choose a pre-defined option.

- Multi-select dropdown: allows a respondent to choose one or more pre-defined options from a dropdown menu.

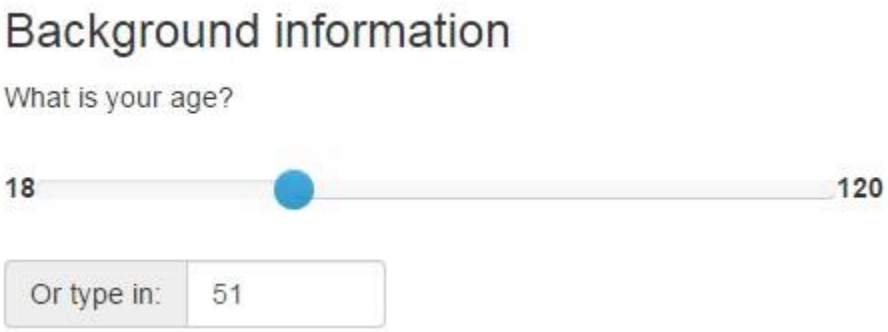


- Integer: allows a respondent to enter a whole number. Provides (if enabled) automatic prevention of entering anything other than a whole number and automated error checking.

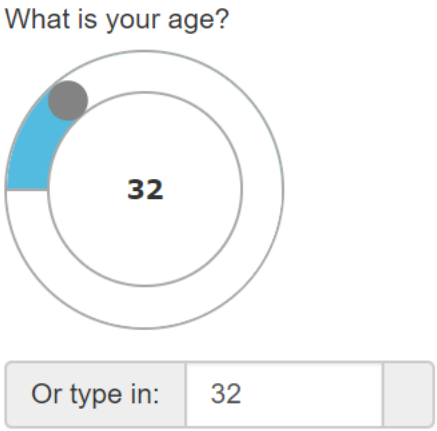
- Real: allows a respondent to enter a number. Provides (if enabled) automatic prevention of entering anything other than a real number and automated error checking.

- Range: allows a respondent to enter a (whole) number in a range. Provides (if enabled) automatic prevention of entering anything other than a whole number and automated error checking.

- Slider: allows a respondent to choose a number on a slider.

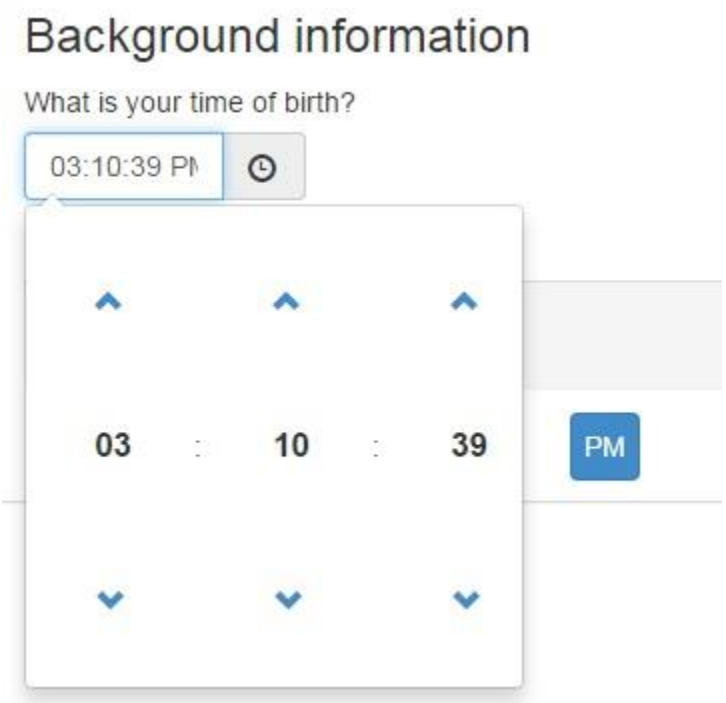


- Knob: allows a respondent to choose a number by moving a knob.

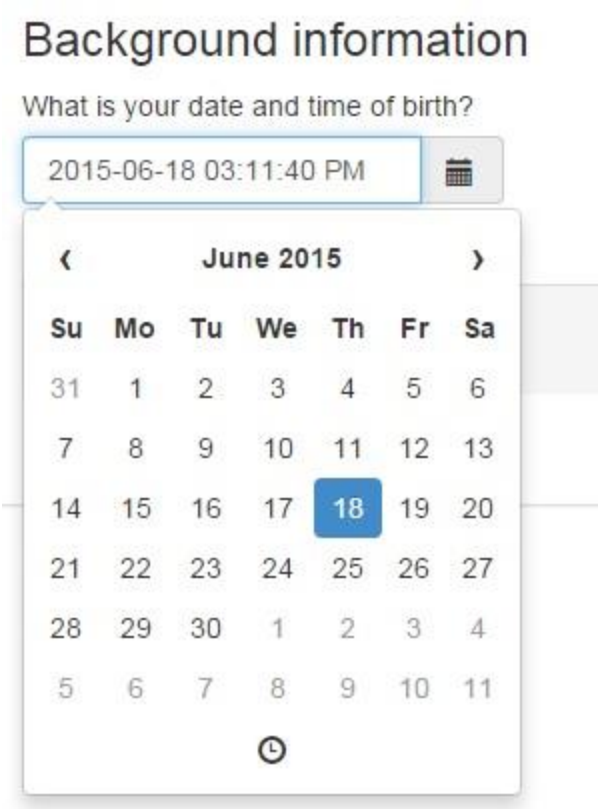


- Date picker: allows a respondent to choose a date.

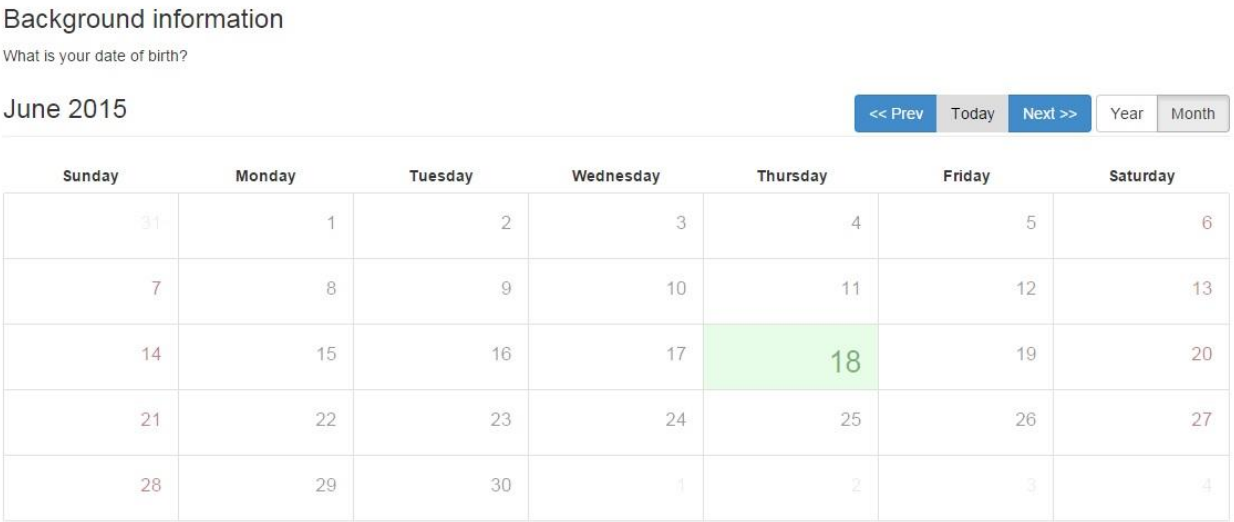
- Time picker: allows a respondent to choose a time.



- Date/time picker: allows a respondent to choose a date and time.

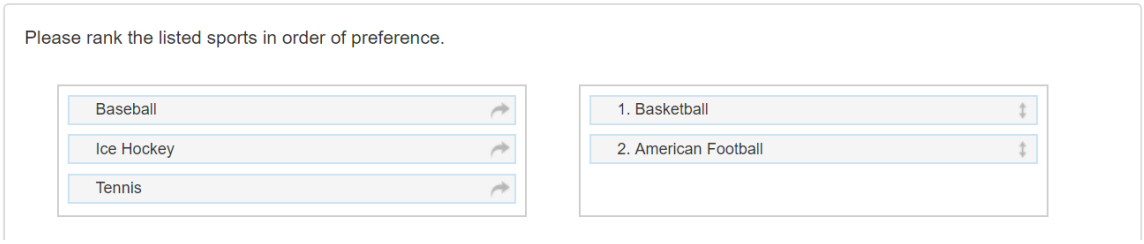


- Calendar: allows a respondent to choose a date on a calendar.

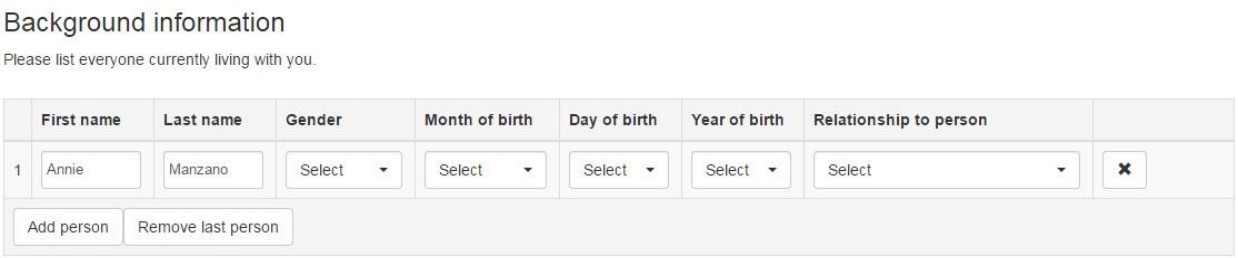


- None: presents information to a respondent where no input from the respondent is required.

- Section: use a variable as a section. This is a type used for advanced survey programming, discussed in Sec. 17.1. Variables of this type are not directly shown to a respondent.
- Rank: ask respondents to rank a series of options in a preferred order. An example would be:



- Custom: allows custom code to construct the manner in which a respondent can answer. This answer type is for advanced usage only and is discussed in Sec. 18. An example of a custom answer type is below.



Beyond the answer type, there are two other basic properties that can be set when adding a variable: whether the variable is an array and whether it should be kept. If the array property is set to *Yes*, then NubiS will treat this variable as if it can have multiple instances. For example, when asking gender you may want to ask the same question of the respondent and any other people in the household. Rather than constructing separate variables for each person, you can declare that *X060ASex* is an array, allowing you to capture the gender of multiple people through a single defined variable. Arrays will be discussed in greater detail later, after the usage of loops in defining skip logic.

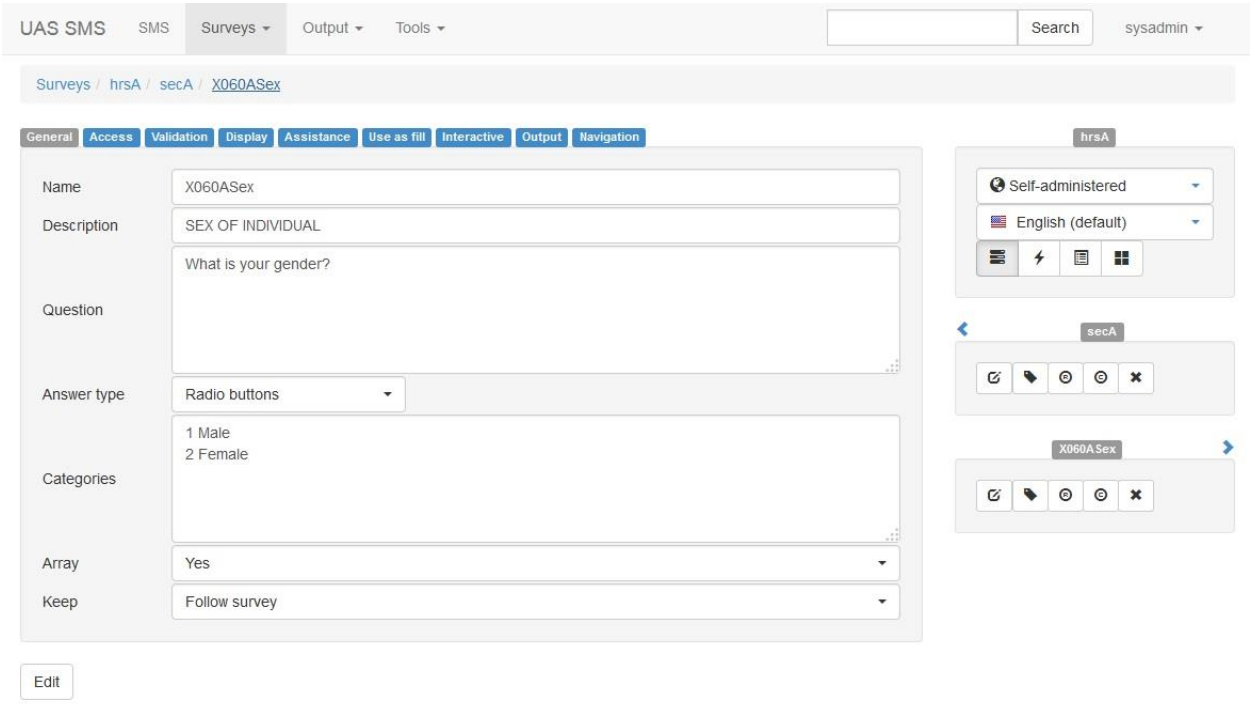
Finally, the keep property is a more advanced construct. If set to *Yes*, it instructs NubiS to keep the value the variable received through a programmatic instruction (as opposed to a direct answer

given by a respondent) if a respondent goes back in the survey. As you later will see, this is useful to preserve the values of randomizer variables.

After you fill out variable *X060ASex* in the manner seen above, click *Save*.

6. MORE SETTINGS FOR VARIABLES

In the variable *X060ASex* added just now, several characteristics were specified. There are many other settings that can be defined for a variable. You can look at these settings by opening up the variable again, clicking the pointer finger in the variable list overview of section specified. The following screen should appear:



The same characteristics are seen as when you added the question. NubiS labels these characteristics as General, which simply means they don't fit in any of the setting categories. The other categories of setting that NubiS offers can be seen next to the *General* tab, but it must be mentioned that not all settings will be available for all variables. Rather, it is dependent on the answer type of the variable.

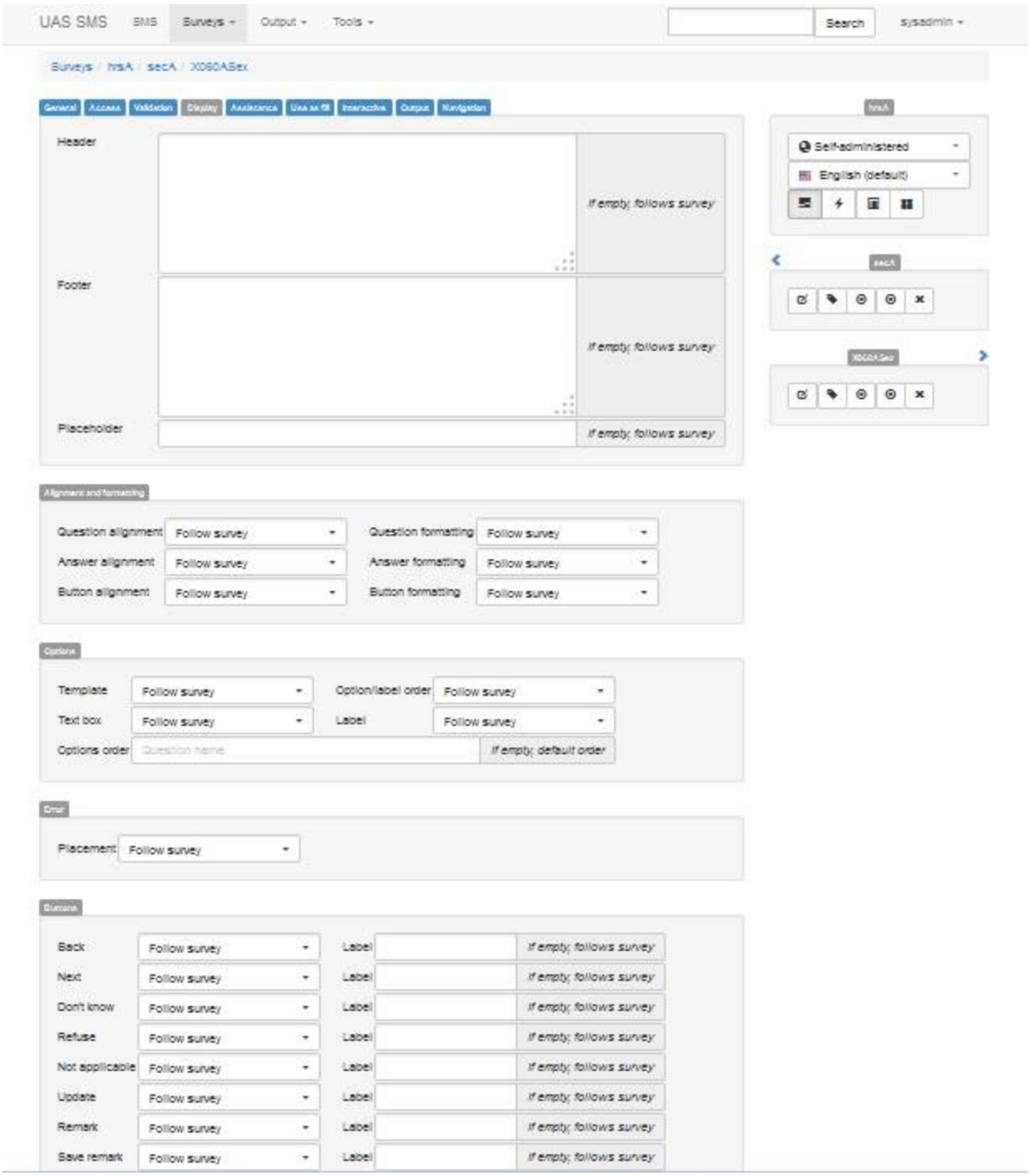
With that in mind, the following types of setting are available in NubiS:

- Access: governs the behavior in terms of if and how the survey may be accessed.
- Validation: allows the specification of the criteria for determining whether the answer to the question is valid.
- Display: controls how the question is displayed on the screen.
- Assistance: enables the specification of assistive messages shown to the respondent.

- **Interactive**: supports the usage of JavaScript to implement interactive behavior on the question screen.
- **Output**: provides the ability to manage if and which data is stored and in which manner.
- **Navigation**: gives the option to facilitate keyboard-based control.

The observant reader will notice that one tab is missing in the above list, which is the **Use as fill** tab. This tab does not contain settings, but instead provides functionality for using the variable as dynamic text. Use as fill will be discussed in Sec. 10.

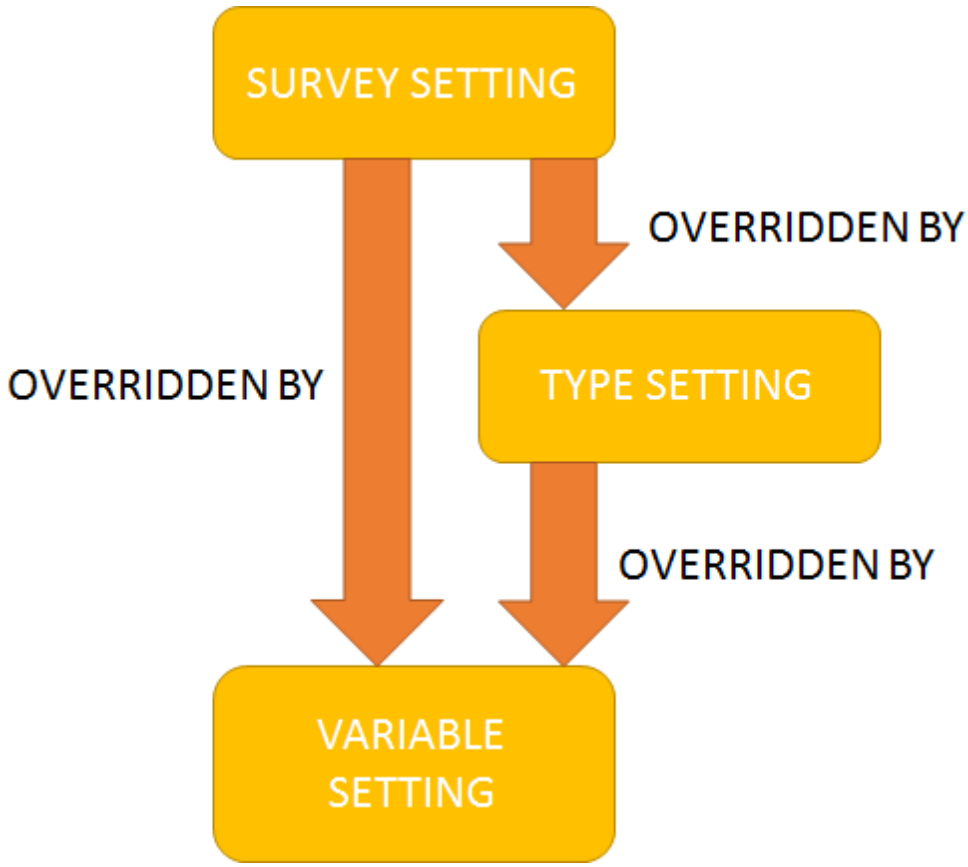
For now, to showcase some of the specific settings in the above list, click the **Display** tab:



While it is apparent here NubiS has a large number of settings, it is not the intent of this manual to discuss them all. Rather, the goal is to instruct how settings can be defined and how NubiS interprets what is specified. We encourage you to try out different settings and see what their effect is.

The settings for a variable can be set on a “higher” level for types and higher level yet for the survey as a whole. In this context, higher reflects the “inheritance” style structure that NubiS

employs. That is, a value for a setting typically can be set at the survey level. It is then inherited by a type or variable unless overridden by setting another value for that type or variable.



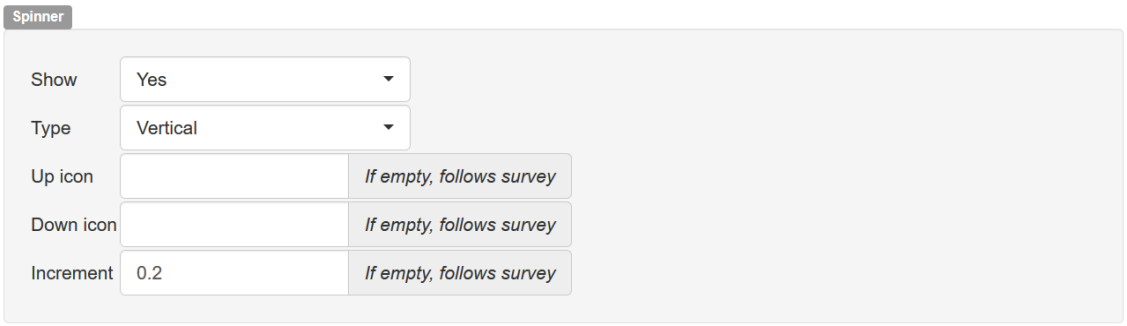
For settings with certain options, like Show or Hide, this is conveyed by the option Follow survey. This is, for example, the case for the << Back button setting towards the bottom:

(As an aside, if the variable had been of a certain type instead, then instead of the *Follow survey* option it would have been the Follow type option.)

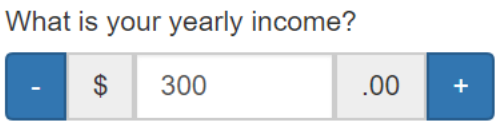
There also are settings not of a Drop down variant, instead allowing for the free text specification of some text. An example is the Back button label setting. This kind of setting also is inherited and used, unless a non-empty value is specified. To override the normal back button label, you could specify here << myback. The result would be that for all questions shown in the survey, the back button would appear with its default label – except for the question about the respondent's gender. In this regard, NubiS considers an empty string to be an empty value. So if, for example, every question screen came with a certain header (*e.g.*, some HTML code to display a logo defined

as a survey-level setting), then in order to have it not appear for the variable, you would have to specify a non-empty string in the header setting textbox. The best candidate for such a string is   (which is HTML's no-break character and is not displayed in a browser screen).

Other examples of this mechanism are in the settings governing the display of a spinner for numerical text boxes:



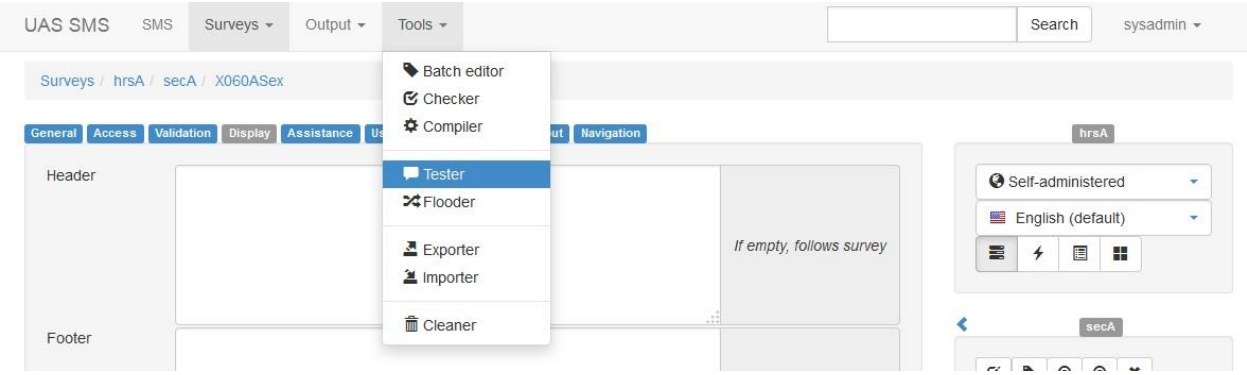
Here, as before, dropdowns allow to follow the higher type or survey settings explicitly while leaving text box settings empty. These control display such as:



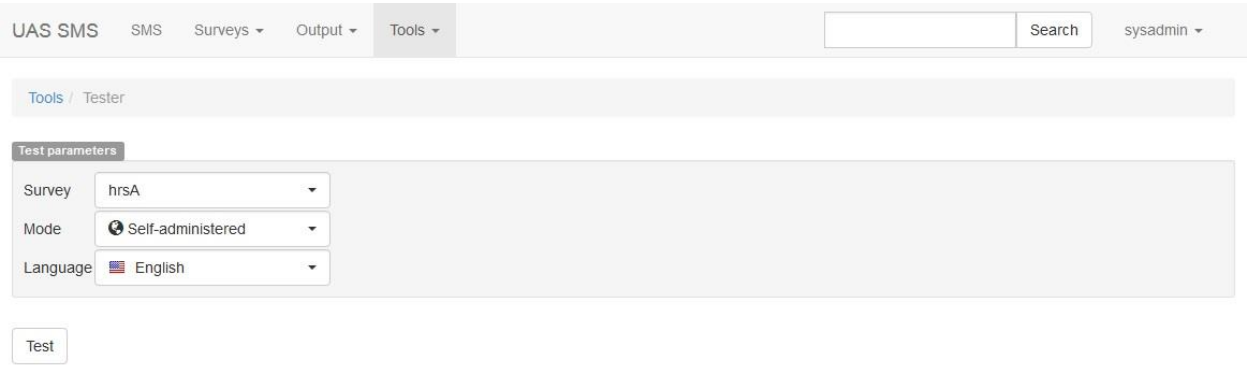
In the above the minus and plus button enable respondents to lower or increase the value of their income by a defined amount (e.g. $100)

7. GETTING NUBIS TO ASK A QUESTION

Now that you know how to add a variable and edit its settings, you may think the survey is ready to ask your question about the respondent's gender. But the survey is not ready, and it is instructive to demonstrate why via a test. Under **Tools** is the **Tester**:



Choosing *Tester* brings up the resulting screen, showing the test options:



When **Test** is clicked, the result is:

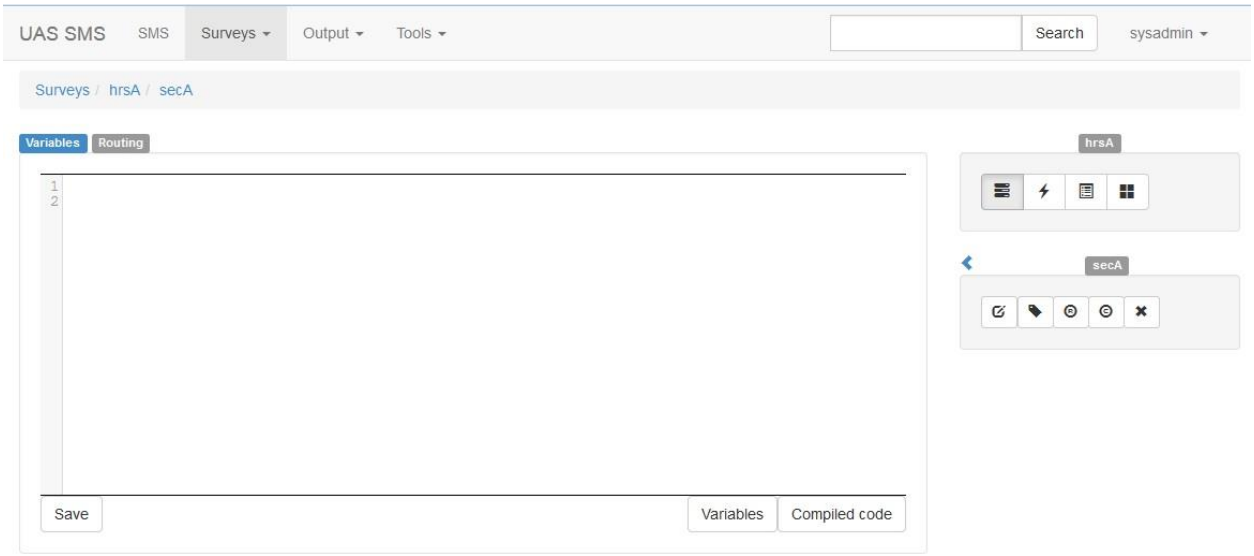
If this looks familiar, it should be: You're back inside NubiS' area for developing surveys. Simply adding a variable is not sufficient for a question to be asked; rather, NubiS must be explicitly instructed to do so. Because NubiS wasn't, in this case, when you hit *Test*, the survey began but found nothing to ask, and so returned to where you first were (because the survey was in test

mode). If you had tried to run the survey in normal mode as a respondent, the result would have been:

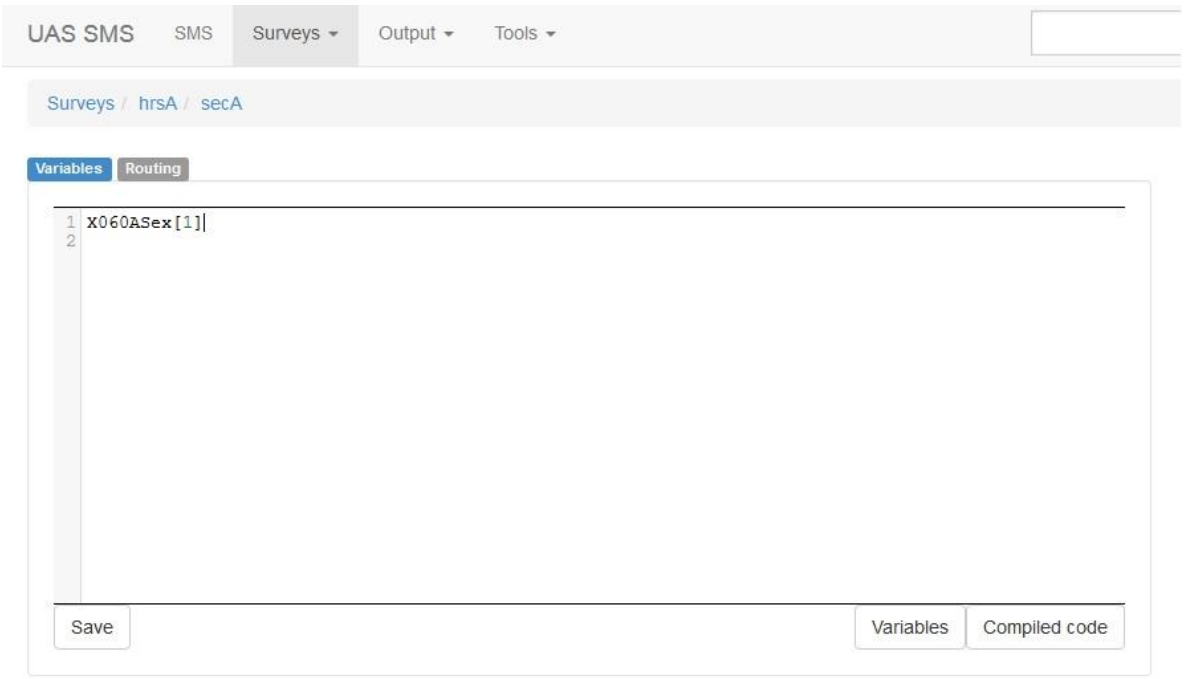
Here as well, NubiS went through the survey, found nothing to ask and so finished with the Thanks screen.

To get NubiS to ask the question about gender it must be included in the routing of the survey. Survey routing, or skip logic, is a series of statements telling NubiS which variables should be used during a survey and in what manner. Routing is defined per section; that is, each section has its own routing. Those then must be integrated into the survey as a whole by incorporating a section into the *Base* section routing.

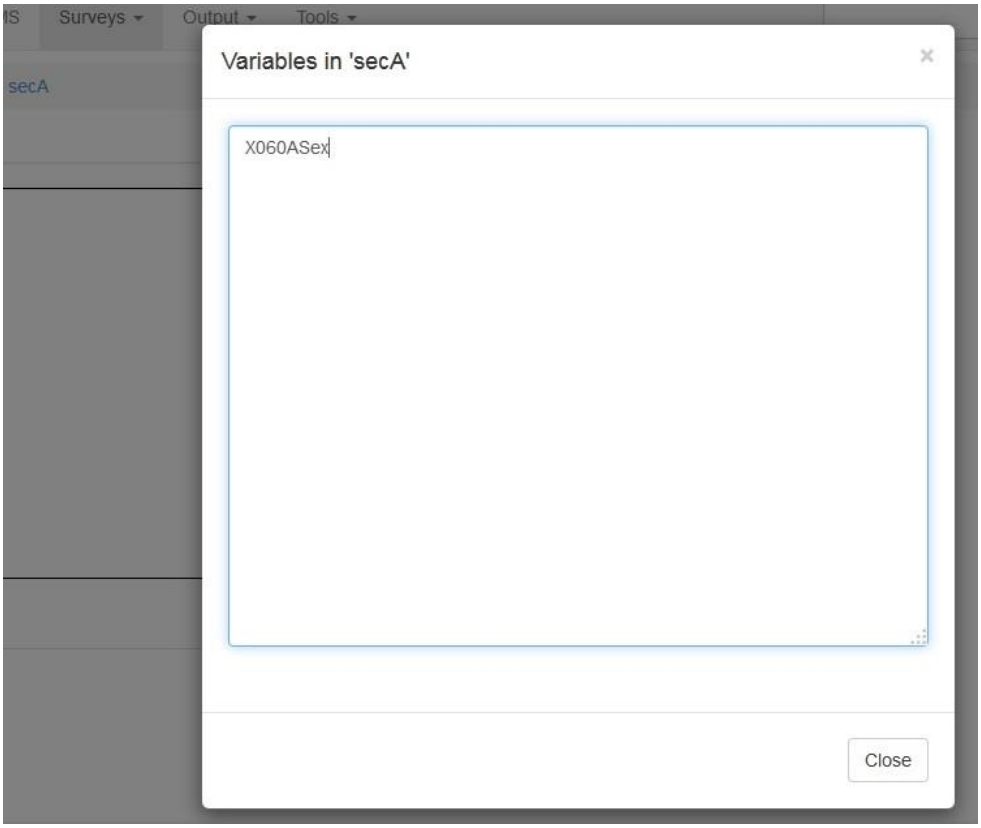
On the screen that came up as a result of the failed test, select Surveys then click the pointed finger in the overview to open up your survey. Next, if not selected yet, choose the sections tab and click the pointed finger in front of the *secA* section. Then, click on the *Routing* tab (next to the *Variables* tab), which should lead to the following screen:



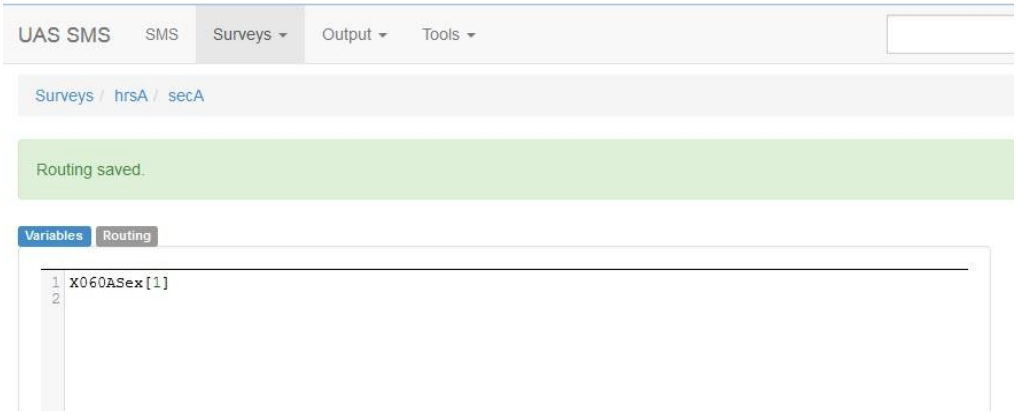
The routing screen consists of a single text box in which routing instructions can be entered. To instruct NubiS to ask a question, simply use the variable's name and add it on a line. In this specific case, enter X060ASex[1]:



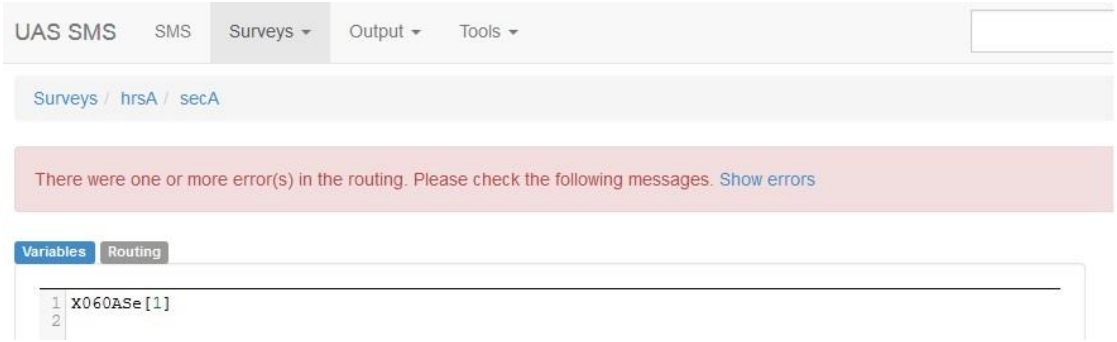
It will be explained later why the *[1]* was added. If you forgot the name(s) of specific variables in the section, clicking the *Variables* button below the routing text box on the right side will open a list:



To save the routing, click the Save button on the bottom left. NubiS will verify the instructions entered and confirm the changes:



If you make a mistake (such as referring to a variable that does not exist), NubiS will come back with an error message:

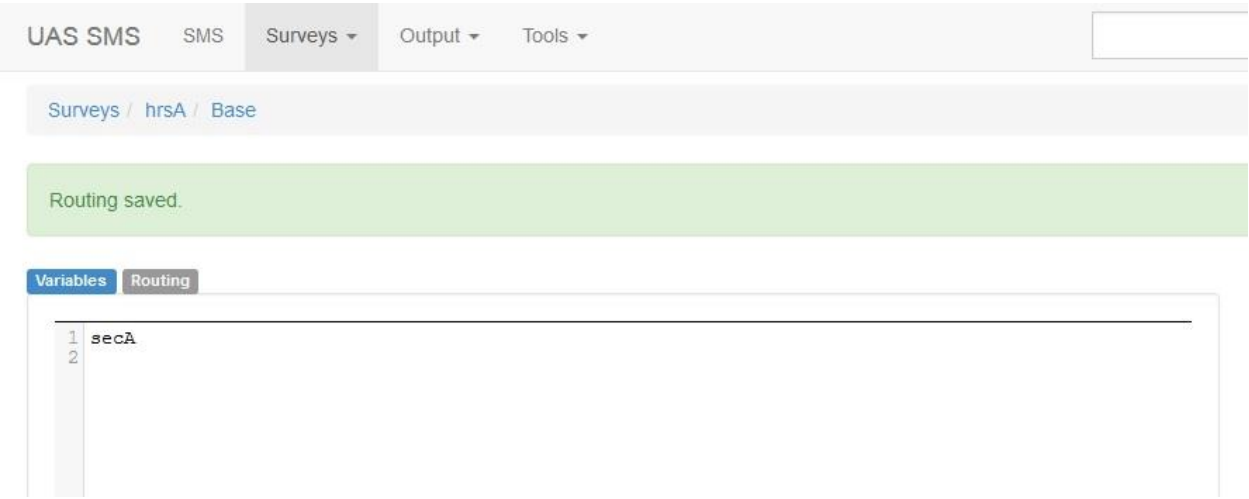


You can then click Show errors to see which line(s) contain error(s):

In this case, *X060ASex* was misspelled, so NubiS could not find this variable.

As said earlier, for the survey to ask questions, the section with the new routing must be brought into the survey via the Base section routing. It is not enough to simply add the variable to the routing of *secA*, because while NubiS now knows if *secA* is selected then *X060ASex[1]* should be

asked, NubiS does not yet know that *secA* itself should be asked. To accomplish this, open the routing screen belonging to the *Base* section. Type in *secA* on the first line and save the routing:



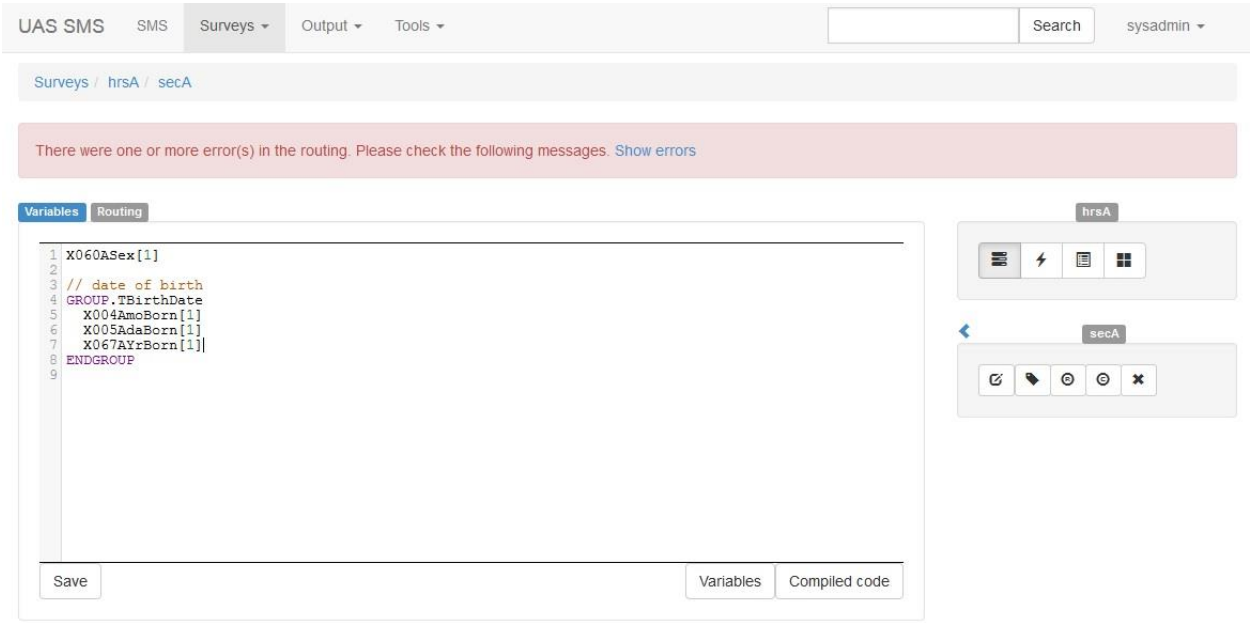
Then go to *Tools* | *Tester* and select *Test*. The result should be the question and available answer options as radio buttons:

Also seen is the header specified for section *secA*. (To recall, this was *<h2>Background information</h2>*, which the browser interprets into a heading.), In addition, there is a progress bar (which is completely full here because there is only one question) and a **Next>>** button. There is no back button because this is the first and only screen, meaning there is no previous screen.

Because this survey is running in test mode, there also is a bar in the top with a dropdown in the right corner. This bar does not usually appear but here can be used to exit test mode and return to the NubiS survey development interface. It also shows the identifier of the case (*i.e.*, the primary key) and the current question(s) shown.

8. ASKING MULTIPLE QUESTIONS AT THE SAME TIME

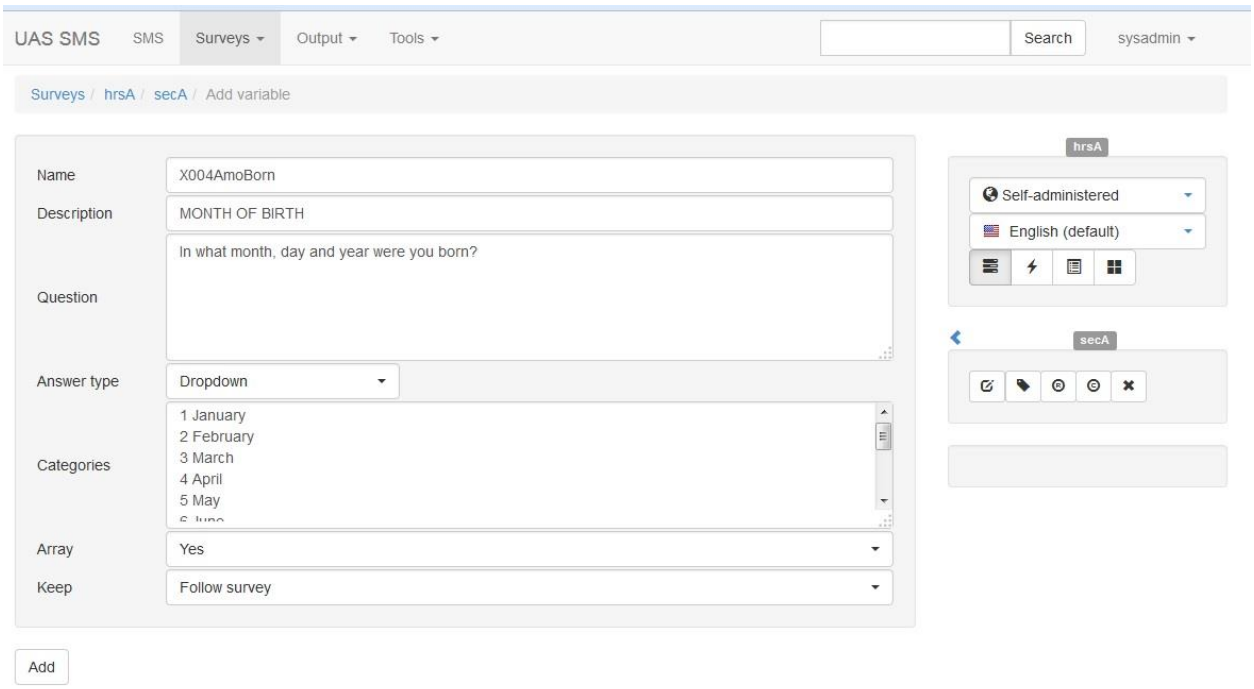
For this survey, an obvious next step might be to ask the respondent's birth date in order to calculate their age. That could be done by first asking the year of birth, going to another question screen to ask the month of birth and then jumping to a third question screen for the day of birth. More intuitive to the respondent, however, would be to have a single question screen in which they can enter their date of birth. In NubiS, this can be accomplished by using a group statement:



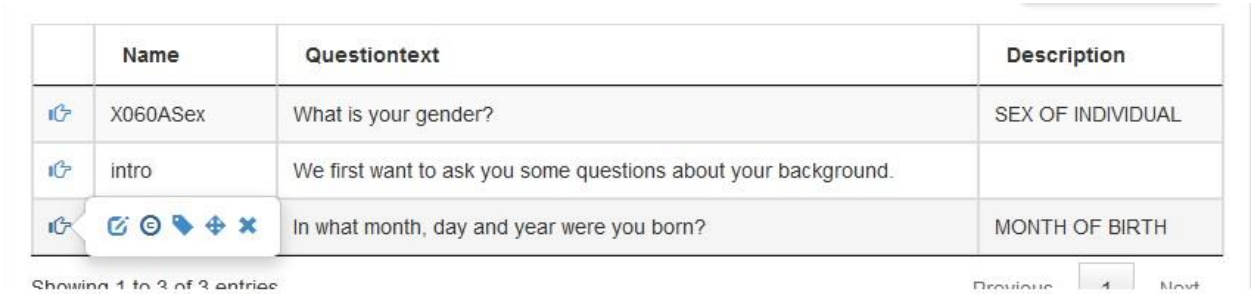
The proper syntax to use can be seen in the above code snippet, starting a group statement with GROUP and closing it with ENDGROUP. (Ignore, for this moment, the errors prompted by NubiS for not adding the questions referred to in the group statement.). One can think of this syntax as being similar as to how you would open and close HTML tags.

The opening *GROUP* statement mandates the presence of a group name, so in this case specify TBirthDate. This group does not yet exist, but when the routing is saved NubiS will create it automatically. (How to edit groups will come later.) Next, list the questions to be grouped together (here, three variables representing month, day and year of birth) and finish with *ENDGROUP*.

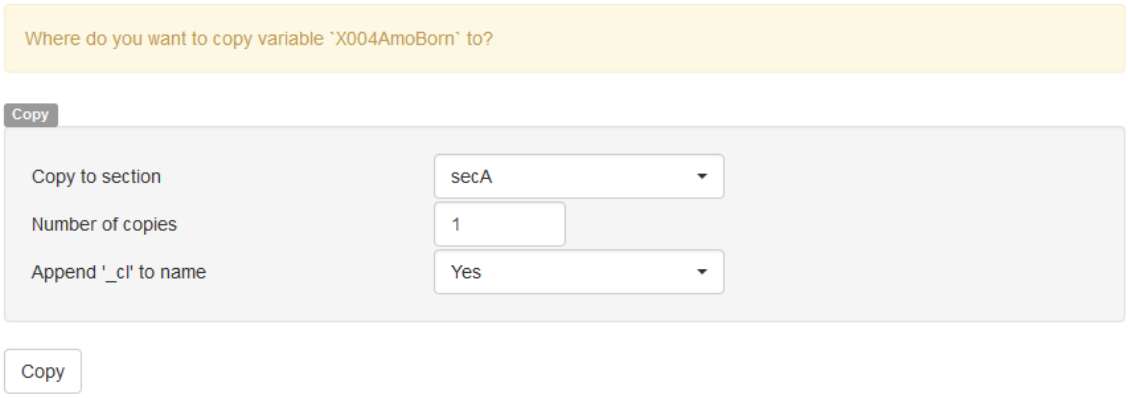
To see the effect of this group statement, now add these three questions. First, switch back to the *Variables* tab, click Add new variable, calling it X004AmoBorn, and give it the description MONTH OF BIRTH:



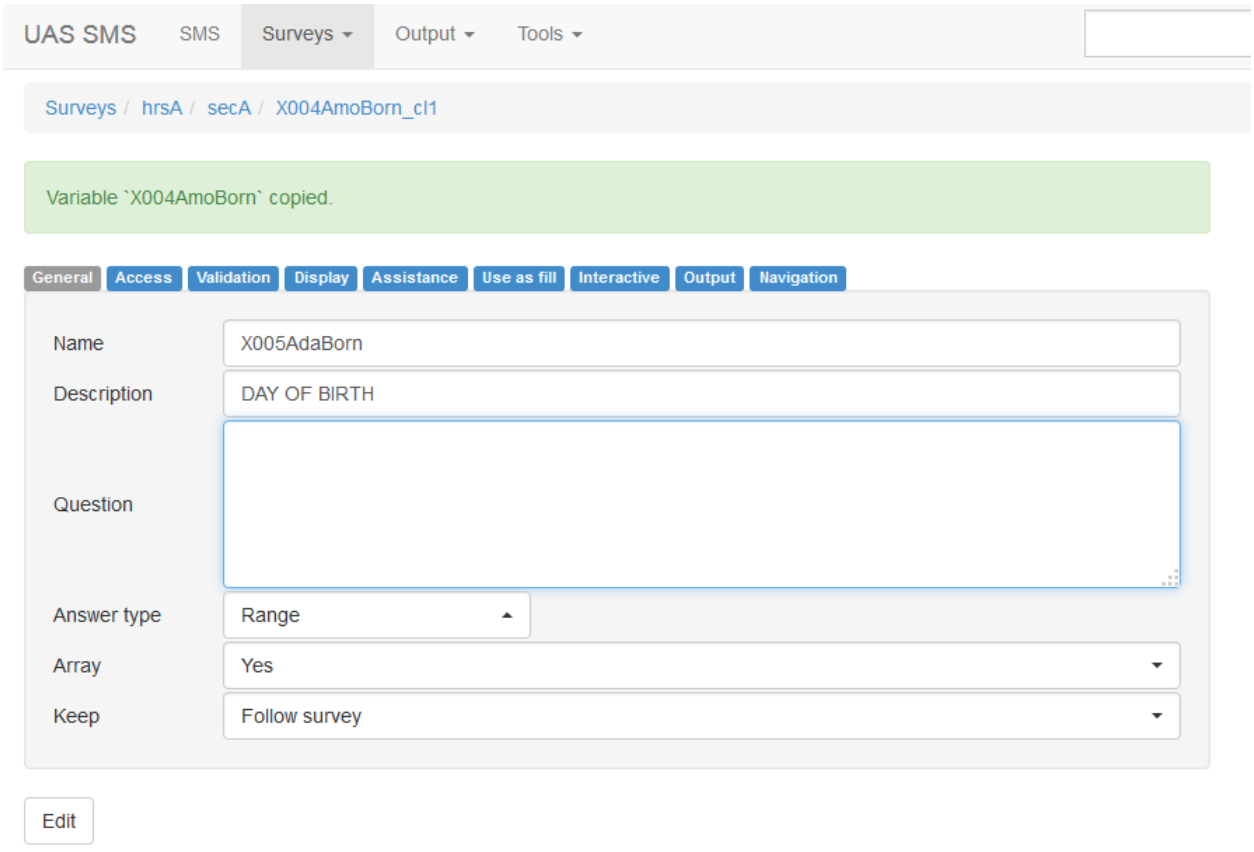
This variable is very similar to the one for gender; the only difference being the answer type is *Dropdown*, done to reduce the space taken up by the answer options. Next, add day of birth as a range variable. For ease, make a copy of *X004AmoBorn* by clicking the copy icon in front of it (note: to quickly locate the variable in the list you can start typing its new in the search box above the list):



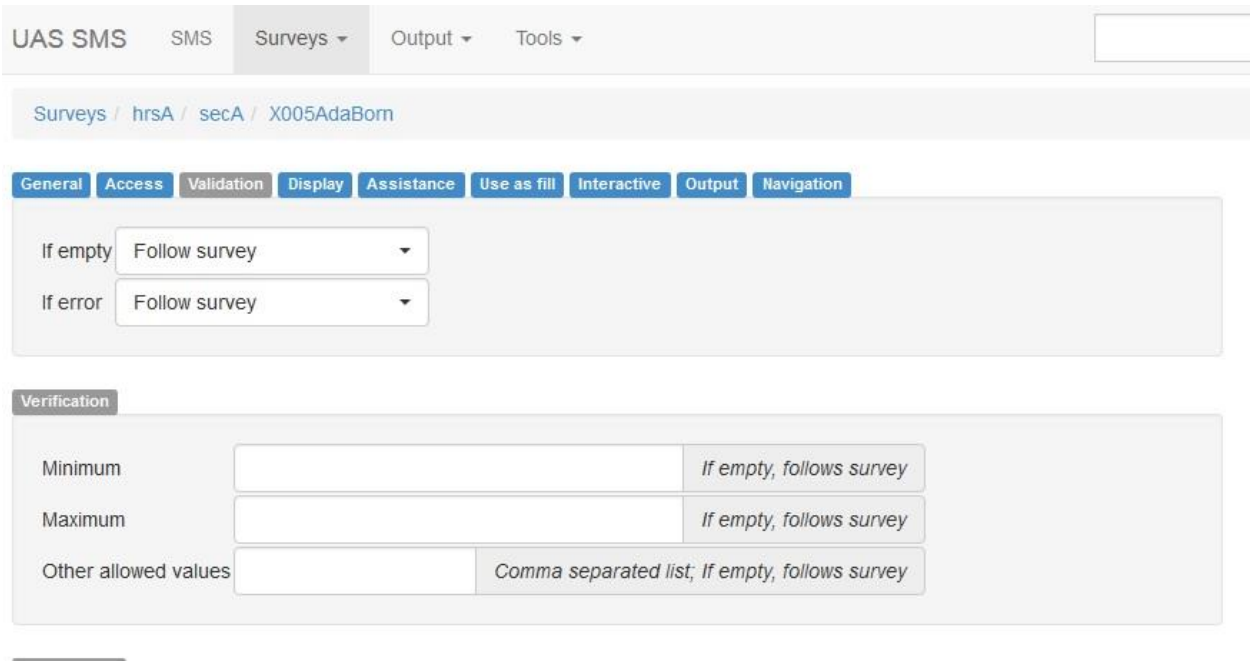
The following screen asks us where to make a copy and how many copies are wanted; for this example, the copy should be in *secA* and only one copy is needed:



In the resulting screen, change the name of the variable to X005AdaBorn and description to DAY OF BIRTH, remove the question text, change the answer type to *Range* and hit the *Edit* button to save the changes.

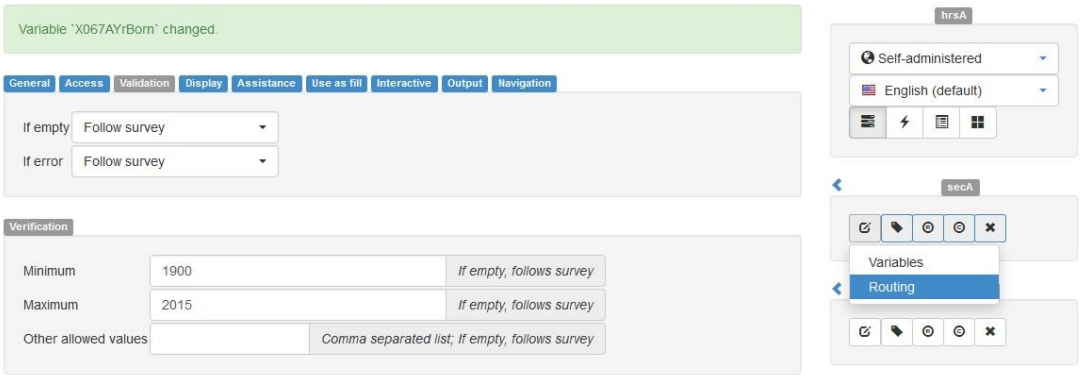


Given that the question asks the day of birth, the respondent's answer must be between 1 and 31. To make this the range, click *Validation*:



There are many settings that can specifed here, but focus on Minimum and Maximum. Change these to, respectively, 1 and 31. As the screen says, if either is left empty it will follow the survey level setting. This setting itself defaults to 0 and 9,223,372,036,854,775,807 (the maximum integer value in PHP).

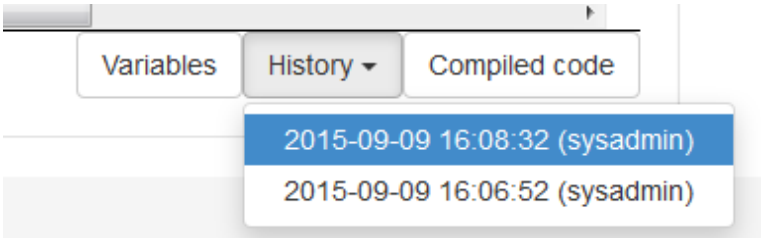
The third question to add is for the year of birth. A range also is appropriate here, so make a copy of *X005AdaBorn*, call it X067AYrBorn, adjust the variable description and save the changes. Then go to the *Validation* tab and specify 1900 as a minimum and 2016 as a maximum:



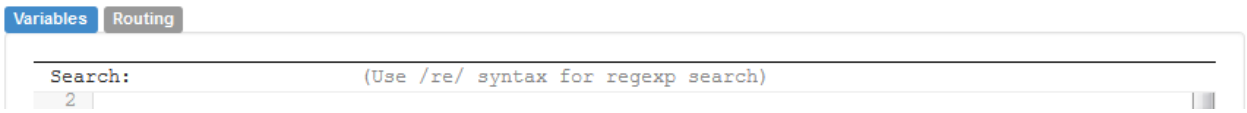
(An observant reader will remark that the maximum for the year of birth should change once the next year starts, obviously. Details will follow later on how NubiS facilitates this through its support for dynamic text and capability to leverage PHP functions.)

To update the routing, hit *Save*. It is important to note that re-saving gives NubiS the opportunity to re-assess code as to correctly interpret it when running a survey. Failing to re-save so may lead to NubiS attempting to use an older and flawed set of routing instructions.

Before testing our survey again, take a moment to notice an additional option appearing underneath the routing text box:

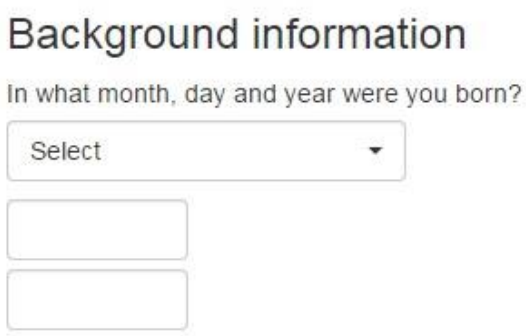


Whenever the routing is saved, NubiS logs an entry of the routing at that point in time. These snapshots are available through the **History** dropdown. This feature allows for the quick return to a previous version of the routing. Another use is to search for occurrences of certain text in the routing (*e.g.*, to quickly locate faulty code). One way to do this is by using the routing editor's built-in search, which can be activated by pressing Ctrl+F (or Cmd+F) while the editor box is active (that is, has focus). The result will be a bar above the box:

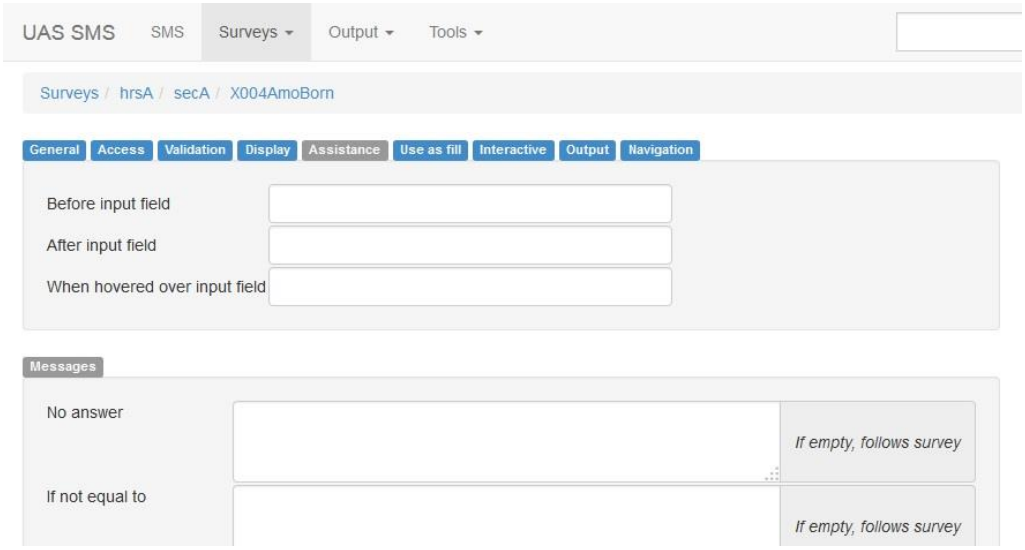


Here, you can enter the term to search (the search is case insensitive). On pressing Enter, each occurrence will be highlighted. You then can move from one occurrence to the next by using Ctrl+G (or Cmd+G). Similarly, you can use Shift+Ctrl+F (or Shift+Cmd+F) to replace certain occurrences. Note that the latter should not be used to rename a variable. If this is what is intended, then use NubiS' built-in refactoring option. Alternatively, you can search for text in the routing of sections by utilizing NubiS' generic search functionality.

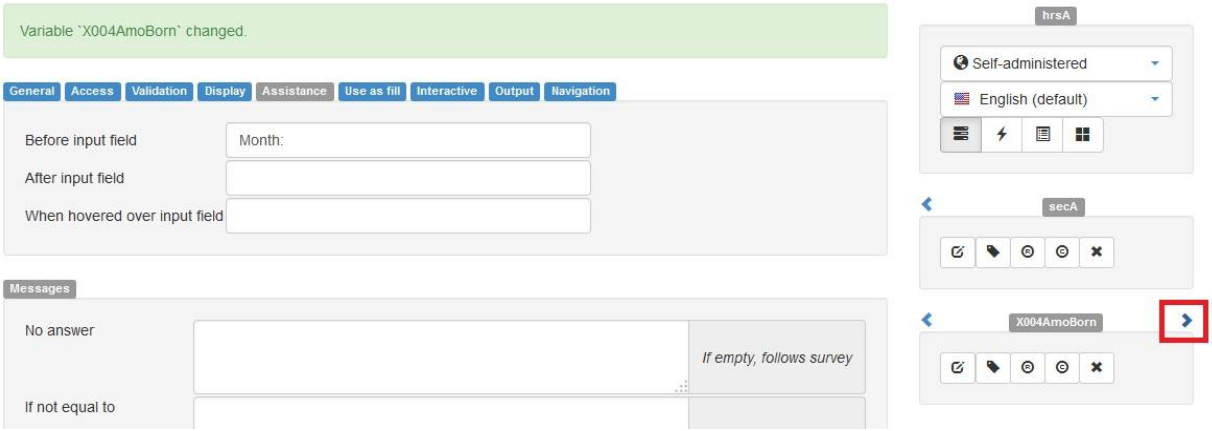
With all that in mind, it's a good opportunity to test your survey again: *Tools* | *Tester* and select *Test*. Give an answer to the first question about gender, then click Next:



Above are the results of your work: the question and three answer boxes. But which one is which? To make this clearer for the respondent, exit test mode and go back to *X004AmoBborn*. There, click the *Assistance* tab:

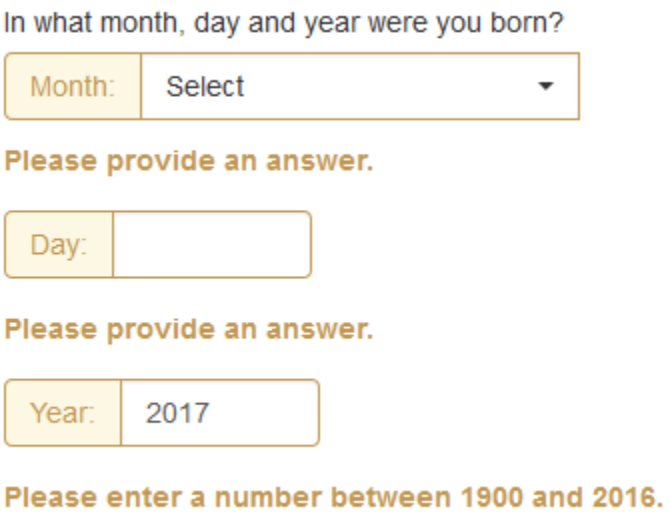


Ignoring the other settings available here, focus on **Before input field**. This setting allows text to be inserted before the answer box (in this case, the dropdown). Enter **Month:** and save the changes. Then make similar additions to the day and year of birth questions. To go to *X005AdaBorn*, you can use the navigational arrows on the right side (highlighted here in a red box):



These navigation links allow you to quickly move from one variable to another; a similar mechanism is in place to go from one section to another. Using these links, in the *Before input field* settings of *X005AdaBorn* and *X067AYrBorn*, add, respectively, **Day:** and **Year:**.

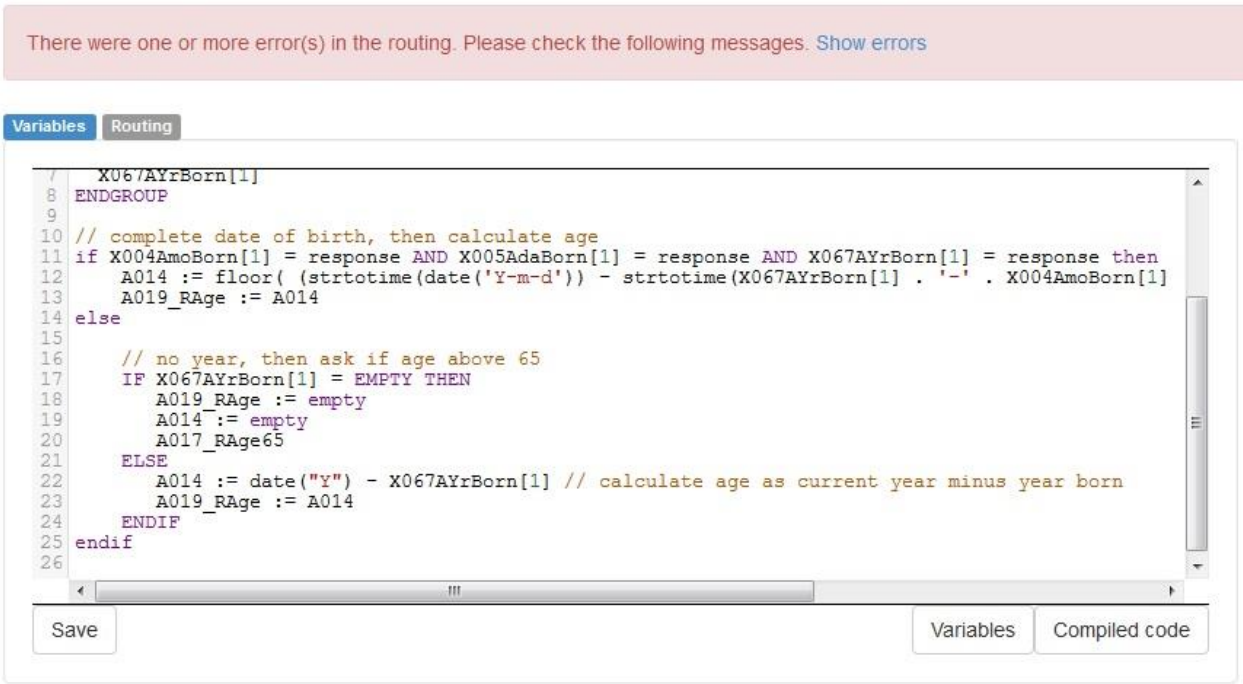
Testing the survey again gives you:



Notice this screenshot has three error messages, produced after *Next* was clicked: the fields for month and day were left empty, while 2016 is outside the range for year.

9. ASKING QUESTIONS UNDER CERTAIN CONDITIONS

Besides the ability in NubiS to group questions, another useful routing construct is skip patterns. Skip patterns are realized through an **IF THEN** statement. These can be positive in nature ("If the answer to Question X is 'True,' then ask Question Y.") or negative ("If the answer to Question X is 'Not True,' then ask Question Z."). This example will show you how employ skip patterns to calculate the respondent's age based on the date of birth details provided. For this purpose, add the following instructions to the routing of section secA:



Note that the full code on line 12 is:

A014 := floor((strtotime(date('Y-m-d')) - strtotime(X067AYrBorn[1] . '-' . X004AmoBorn[1] . '-' . X005AdaBorn[1])) / 31556926);

Although this appears, at first glance, quite overwhelming, the intuition of each line should be easily understood. On line 11 is an *IF* statement specifying a condition: If that statement is true, then NubiS is instructed to perform the actions specified on lines 12-13; if the statement is not true, then NubiS processes the *ELSE* statement. An *ELSE* statement instructs NubiS what it should do when the *IF* statement (or *IF* statements, if multiple) before the *ELSE* is not true. In this regard, the actions listed in the *ELSE* can be thought of as the desired default behavior if the conditional behavior is not applicable.

The *IF* statement on line 11 lists the names of the variables, in a series of logical expressions combined by AND operators, just asked of the respondent. Informally, what the instructions call for is that if the respondent gives a month (X004AmoBorn[1] = response), a day (X005AdaBorn[1] = response) and a year (X067AYrBorn[1] = response), that is all the information needed to calculate the respondent's age. But if one of these answers is missing, then the instructions address the need to do more (specified in the *ELSE* statement starting at line 14).

Assume the respondent gave a full date of birth. This means the conditions on line 11 are satisfied, so NubiS attempts to perform the actions on line 12:

> *A014 := floor((strtotime(date('Y-m-d')) - strtotime(X067AYrBorn[1] . '-' . X004AmoBorn[1] . '-' . X005AdaBorn[1])) / 31556926);*
>
> *A019_RAge := A014*

These two statements are so-called assignments. An **assignment** is an expression which assigns a value as specified to the right of the **:=** to the preceding variable. Assignments can be as simple as line 13, which assigns the value of A014 to A019_Rage (two variables not yet added, but can be thought of as two integer variables that will hold the age of the respondent). However, an assignment can also be more complex. For example, *A014* gets assigned the result of a more elaborate expression:

> *floor((strtotime(date('Y-m-d')) - strtotime(X067AYrBorn[1] . '-' . X004AmoBorn[1] . '-' . X005AdaBorn[1])) / 31556926)*

Without getting into too many details, what is happening here is that NubiS is leveraging existing functions in PHP to determine how much time, in milliseconds, has passed between now and the respondent's given birth date. The result is divided by by 31,556,926 to get the equivalent period in years. The result then is floored to obtain the age as a whole number. For example, on June 24, 2016, if the respondent had entered June 23, 1978 as their date of birth, then the calculated age would be 38. This usage of existing PHP functions is very natural in NubiS and, in fact, extends to the usage of user-defined functions (to be discussed in Sec. 18).

Of course, all these calculations to deduce the respondent's age presupposed a complete date of birth was given. If that information was not provided, then the instructions have NubiS finding another way to determine the age. This is where the *ELSE* comes into play. Looking inside the *ELSE*, there is another *IF/ELSE* combination (illustrating that *IF* statements can be nested into one another, which can be done up to arbitrarily nested depths). Line 17 tells NubiS that if a year of birth was not provided, there is nothing on which to base a calculation. (EMPTY is a reserved word here indicating no answer was given.) So NubiS then is asked to perform the following actions:

A019_RAge := empty

A014 := empty

A017_RAge65

These three lines instruct NubiS to set the values for *A019_RAge* and *A014* to empty, then ask A017_RAge65, which is used in HRS section A to ask if the respondent is older than 65. Note: To resolve NubiS' error messages for these questions just add them to the survey. *A019_RAge* and *A014* are both integer questions, whereas A017_RAge65 is a radio button question with answer options 1 Yes, 2 No.

However, if a year of birth was provided, then NubiS calculates the respondent's age based on that information:

A014 := date("Y") - X067AYrBorn[1] // calculate age as current year minus year born

A019_RAge := A014

As an aside, you probably noticed in the above two lines and the snippet of code there were expressions such as // calculate age as current year minus year born. In routing, any text preceded by double slashes (//) is considered by NubiS to be a comment. Multi-line comments also are supported; these start with /* and end with */.

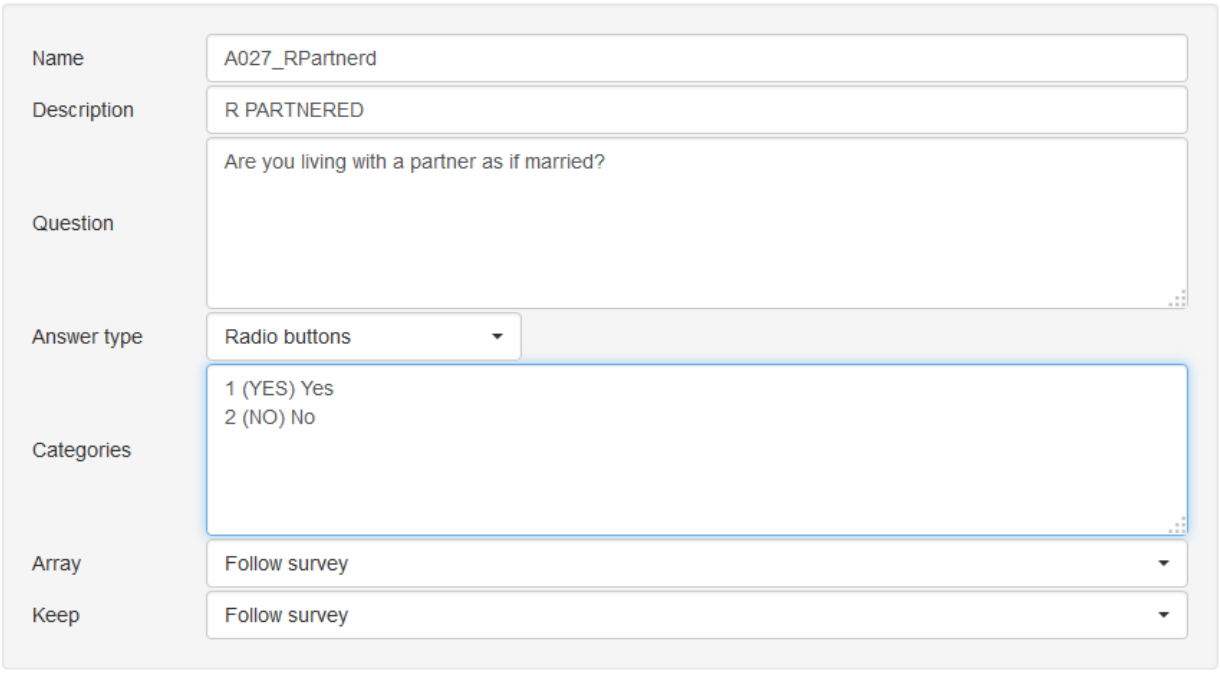
So far, you have created a survey that will ask respondents for their gender and date of birth. The next step is to ask about the respondent's marital status. To that end, you will introduce variables that ask if the respondent is married and, if not, if they have a partner as if they are married. First, define the necessary routing:

```
22          A014 := date("Y") - X067AYrBorn[1] // calculate age as current year minus year born
23          A019_RAge := A014
24      ENDIF
25  endif
26
27  // married or with partner
28  A026_Rmarried
29  if A026_Rmarried != YES then // not married
30      A027_RPartnerd
31      if A027_RPartnerd = YES then // with partner
32          A166_A020TSameSpP_A := YES
33          X065ACouplenss[1] := 3
34      else
35          A166_A020TSameSpP_A := NO
36          X065ACouplenss[1] := 6
37      endif
38  else
39      X065ACouplenss[1] := 1
40      A166_A020TSameSpP_A := YES
41  endif
```

Here NubiS is instructed to ask A026_Rmarried (is the respondent married). If the answer is not *Yes* (line 29), NubiS asks if the respondent has a partner (A027_RPartnerd). If the answer to

A027_RPartnerd (line 31) is *Yes*, variable **A166_A020TSameSpP_A** is set to *YES* (line 32) and the value **3** is assigned to **X065ACouplenss[1]** on line 33. (You will see shortly what 3 means when *X065ACouplenss* is added as a variable.) If the respondent did not answer *Yes* to A027_RPartnerd, then NubiS will perform the actions in the *ELSE* on line 34 (assigning different values to *A166_A020TSameSpP_A* and *X065ACouplenss*). If the respondent said they are married, then the *ELSE* actions following line 38 would be performed.

To a large extent, this is much the same in terms of routing constructs as seen when calculating the respondent's age. But there are a few new things. First, line 29 introduces a new symbol **!=**, which means *not equal to*. (NubiS also supports other types of comparisons, such as greater than, greater than or equal to, less than, and less than or equal to.; respectively, **>**, **>=**, **<** and **<=**.) Secondly, in some conditions and assignments (*e.g.*, line 31 or 32), the word *YES* is used. How does NubiS know what this word means when deciding, for example, whether the condition on line 31 is met? The answer lies in the definition of the variable(s) preceding the *YES*. On line 31, there is *A027_RPartnerd*. When NubiS encounters such a condition, it will attempt to find a definition for *YES* in the condition's variable. To learn how, first, save the new routing and ignore NubiS' complaints about variables it can't find. Next, go to the *Variables* tab and add a new variable. In the resulting screen, specify the following:



Mostly, this is a typical variable specification – except for the fact that in the categories are added so-called value labels:

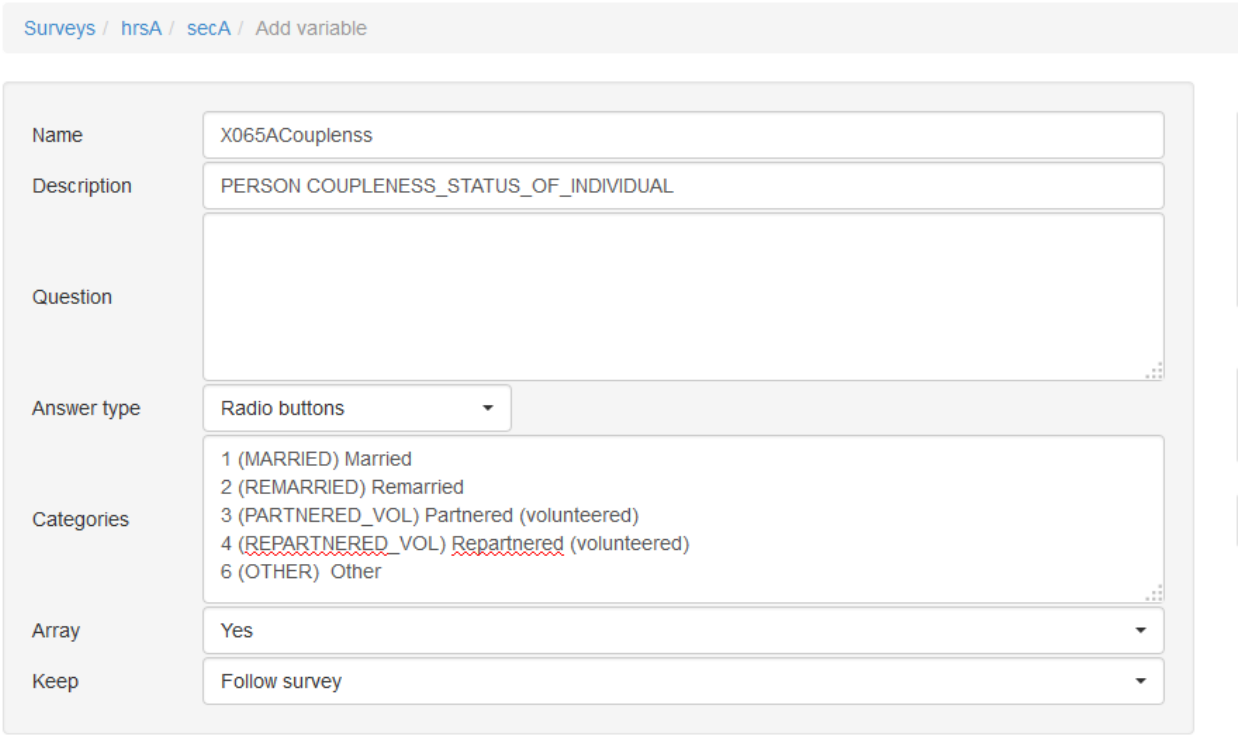
1 (YES) Yes

2 (NO) No

The proper syntax for a value label is to start with an answer code (which must always be the case for an answer option), then a value label enclosed by parentheses and, finally, the answer option text to be shown to the respondent. Here, the value labels are specified in capital letters; this is not mandated but typically helps to make them stand out among a set of answer categories.

In a similar vein, you can add *A026_Rmarried* with question text **Are you married?** (*e.g.*, by making a copy of *A027_RPartnerd*) and the variable *A166_A020TSameSpP_A*. This latter variable doesn't need a question text, as it is not explicitly asked in the HRS. But it does not answer categories *1 (YES) Yes/2 (NO) No*. Rather, the HRS surveys use it as a so-called flag variable to quickly determine if someone is married or living with someone as if married. For example, to ask a question based on this fact, you can simply say **IF A166_A020TSameSpP_A = YES THEN** instead of the longer **IF A026_Rmarried = YES OR A027_RPartnerd = YES**.

After adding these three variables, return to the *Routing* tab and save again. If all is well, there will only be a single source of errors: the non-existence of *X065ACouplenss*. Now add that variable as well:



As you can see, this variable has six answer options (each with its own value label). This also means that instead of:

X065ACouplenss[1] := 3

the code could have said on line 33:

X065ACouplenss[1] := PARTNERED_VOL

which is more readily understood than just 3. (The main reason for using value labels is, of course, to make the routing easier to understand.)

By and large, the constructs discussed so far for specifying conditions in NubiS cover most cases. However, there are a few additional items worth discussing.

For one, suppose you want to ask a question to everyone who is either married or partnered with someone else. One option would be to state:

```
if X065ACouplenss[1] = 1 OR X065ACouplenss[1] = 3 then
    // our question here
endif
```

Inherently there is nothing wrong with this. But suppose there are five eligible answers instead of two. Then you would have to specify five conditions separated by OR to cover them all. Fortunately, there is a more concise way, which takes the following form:

```
if X065ACouplenss[1] in [1,3] then
    // our question here
endif
```

This accomplishes the same as the code above, but in a much shorter fashion. For readability, it could be changed further to:

```
if X065ACouplenss[1] in [MARRIED,PARTNERED_VOL] then
    // our question here
endif
```

Here, the value labels in *X065ACouplenss* are incorporated to make it easier to understand the condition.

Another similar and frequently used construct deals with sets of enumerated (or multi-drop down) variables. Suppose you wish to ask, in question Q1, respondents about any childhood diseases they may have had. A list of them is provided:

1. *Asthma*

2. *Diabetes*
3. *Respiratory disease*
4. *Cancer*

Also assume you want to ask follow-up questions for each selected disease. Doing so could look like this:

if Q1 = 1 then

// ask follow up questions

endif

if Q1 = 2 then

// ask follow up questions

endif

And so on for all four categories. This would work – but only sometimes. Specifically, it will work if the respondent chooses one of the options. However, as soon as s/he selects more than one, NubiS will say the conditions are no longer satisfied. The reason is because NubiS internally handles answers to a set of enumerated/multi-dropdown variables by storing them all in one long string. For example, if the respondent checks 1 and 2, then it would be 1-2; if they checked 3 and 1, it would be 3-1. But neither is equal to *1* or *2*. To avoid this, you need to specify instead:

if 1 in Q1 then

// ask follow up questions

endif

if 2 in Q1 then

// ask follow up questions

endif

Another peculiarity about a set of enumerated/multi-dropdown variables is their behavior when assigned a value. Suppose you want to pre-select Asthma and Diabetes. As just explained, NubiS internally stores this as *1-2*. So to accomplish the pre-selection, you must state:

Q1 := '1-2'

Alternatively, you could say:

Q1_1_ := response

Q1_2_ := response

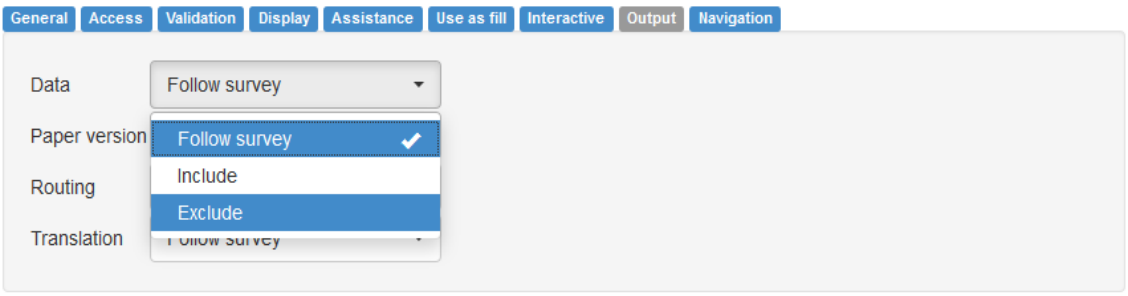
This reflects that both the first and second option of *Q1* have a response, which in the context of set of enumerated/multi-dropdown variables means that these options will appear selected when *Q1* is displayed on the screen. You will learn in the next section that references of the form Q1_1 can also be used when using previous answers as dynamic text.

10. USING DYNAMIC TEXT IN QUESTIONS

Now you'll add some questions about the respondent's spouse or partner (if any). Phrasing, of course, is of paramount importance. Say, for example, you want to ask the gender of the respondent's spouse/partner. One option is to write the question as, "What is the gender of your spouse/partner?" Although this is a valid question text, the respondent may be puzzled why the question would not just state "spouse" or "partner." After all, the respondent already has answered whether they are married or living with a partner as if married. In NubiS, previously-entered information can be used to construct so-called **dynamic text**, often referred to as a **fill**. Depending on known information, the text contained in a fill can be varied, allowing the question text (as well as other kinds of text) to be flexible.

The steps to add a fill are the following:

1. Add a new variable of type *String* and, following the lead of HRS, name it **FLHWP**. But you can leave empty the description and question text, as this question will not be used by itself to present a question and collect data.
2. Next, select the *Output* tab. In it, set **Data** to **Exclude**:



This has no effect on the survey workings, but instructs NubiS to not include this variable in any outputted data set.

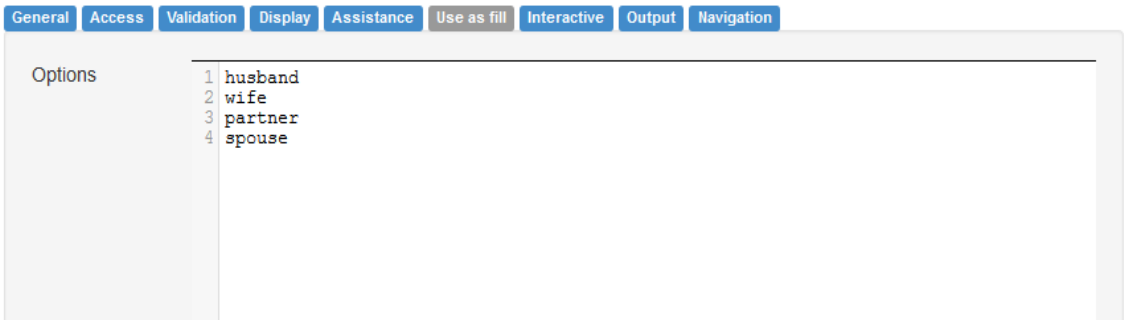
3. Open the *Use as fill* tab. This tab consists of two parts: options and fill code. In the options part, you can state the possible values the fill variable can take. Here (as can be seen in the image below), enter the following options:

husband

wife

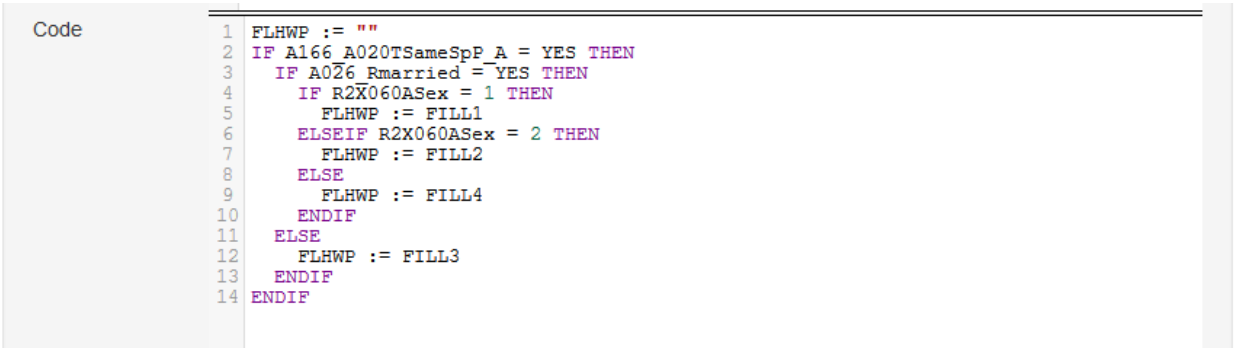
partner

spouse



Each option should appear on its own line be separated by a carriage return (Enter).

Then, in the fill code part, you can specify under what conditions the fill variable will take which value:

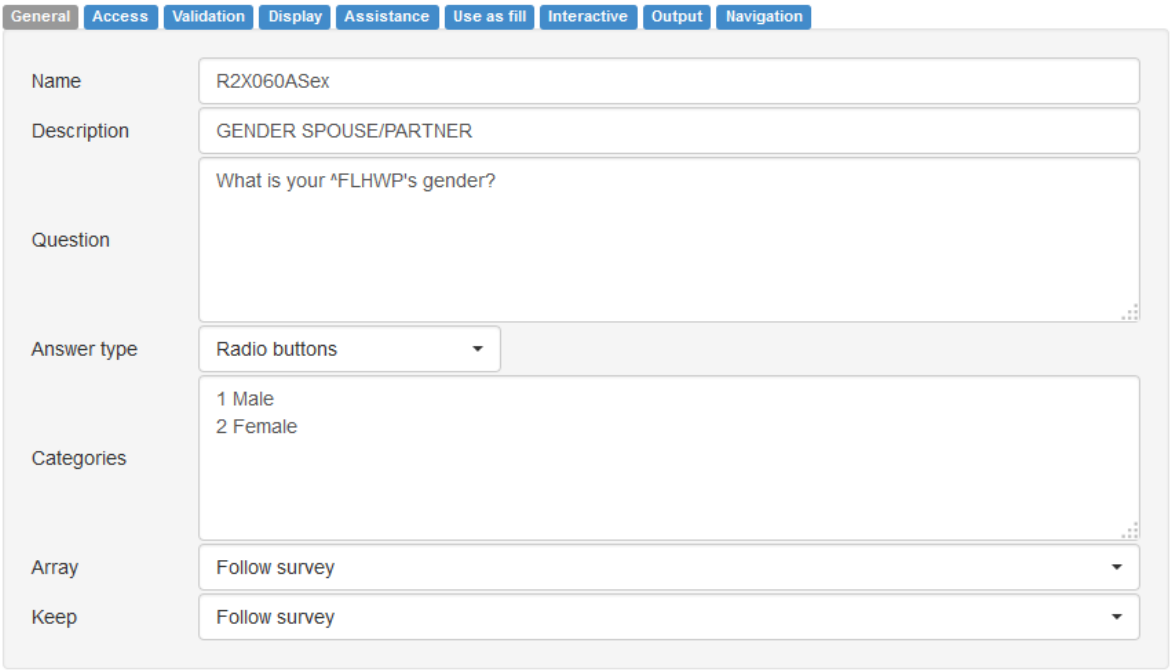


Most of the above syntax looks familiar. This is because fill code uses the same syntax as normal routing – with the exception that, in fill code, NubiS cannot be instructed to ask a variable. Instead, NubiS only can be told to perform assignments.

Start by setting *FLHWP* to an empty string (“”). Then, if the flag variable *A166_A020TSameSpP_A* is equal to *YES*, NubiS determines whether the respondent is married (A026_Rmarried). If *Yes*, then if R2X060ASex equals 1, set *FLHWP* to FILL1. Obviously, *R2X060ASex* hasn’t yet been added, but this question will ask the gender of the respondent’s spouse/partner. *FILL1* corresponds to the first option specified in the options part (here, husband, as *1* in *R2X060ASex* is the code for Male).

Following the different conditions tells NubiS if the respondent is married and the spouse/partner gender is *2* (Female), then FILL2 is assigned (line 7). If the gender is not known, assign FILL4 (line 9). Lastly, if the respondent has a partner instead of being married, assign FILL3 (line 12).

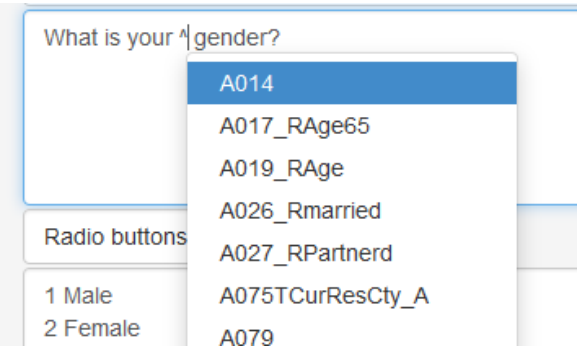
Now, add *R2X060ASex*:



This is another radio button variable with answer options *Male* and *Female*. What is different, though, is the question text:

What is your ^FLHWP's gender?

Introduced here, by using the caret (**^**), is a reference to the fill variable *FLHWP*. The ^ symbol is reserved in NubiS to specify so-called **variable value references**. These references instruct NubiS to look up the value of *FLHWP* and insert it into the question text at the place of the reference. So here, if *FLHWP* were "wife," then the outputted question text would be "What is your wife's gender?" As an aside, note that NubiS assists with entering variable references by auto-completion:



You can continue typing while the popup is there to locate *FLHWP*. For example, the moment you hit "F," the list will update to show all variables starting with the letter F. This auto-completion feature is available for any setting for which variable references are supported (including other

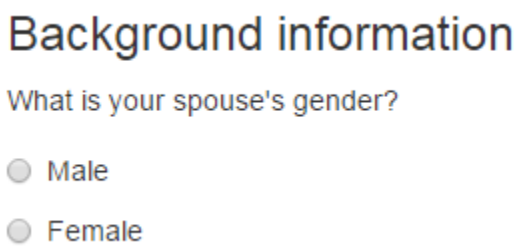
types of reference to be discussed later). Note that if you are referencing an array question, you still will need to type in the instance indicator (*e.g.*, *[1]*).

At this stage, there are only two things left to do to see the fill variable in action. One is to return to the *Use as fill* tab of *FLHWP* and re-save the routing code. This gives NubiS the chance to update its interpretation of the fill code now that *R2X060ASex* is present. The second is to instruct NubiS to ask *R2X060ASex* while using the *FLHWP* fill. This is accomplished on the *Routing* tab:

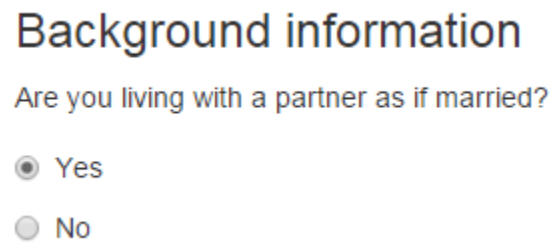
```
44 // set fills
45 FLHWP.FILL
46
47 // ask spouse/partner details
48 IF A166_A020TSameSpP_A = YES THEN
49  R2X060ASex
50 ENDIF
51
```

The new routing code starts on line 44 by adding a comment **//set fills**. Next follows a *.FILL* statement, which tells NubiS to look inside *FLHWP* to see if it has any fill code and, if so, execute it. Following that, *R2X060ASex* is asked on line 49 (conditional upon the respondent being married or with a partner as specified on line 48).

If you now run your survey again through the tester, answering the question that you are married, the following screen should come up:



If you answer you are not married but partnered, the screen is:

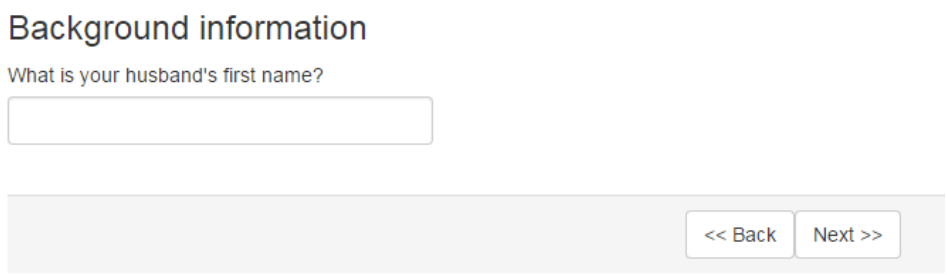


You may wonder why neither “husband” nor “wife” comes up. The reason is simple: Those fill options only will be triggered once the gender is know – and the gender is being asked right now. To showcase this, add another question asking about the first name. Make a copy of *R2X060ASex*,

rename it to R2X058AFName, and change the label to FIRST NAME SPOUSE/PARTNER, the question text to **"What is your ^FLHWP's first name?"** and, finally, the answer type to *String*. After adding this variable, return to the *Routing* tab and add two lines:

```
47 // ask spouse/partner details
48 IF A166_A020TSameSpP_A = YES THEN
49  R2X060ASex
50  FLHWP.FILL // reset
51  R2X058AFName
52 ENDIF
53
```

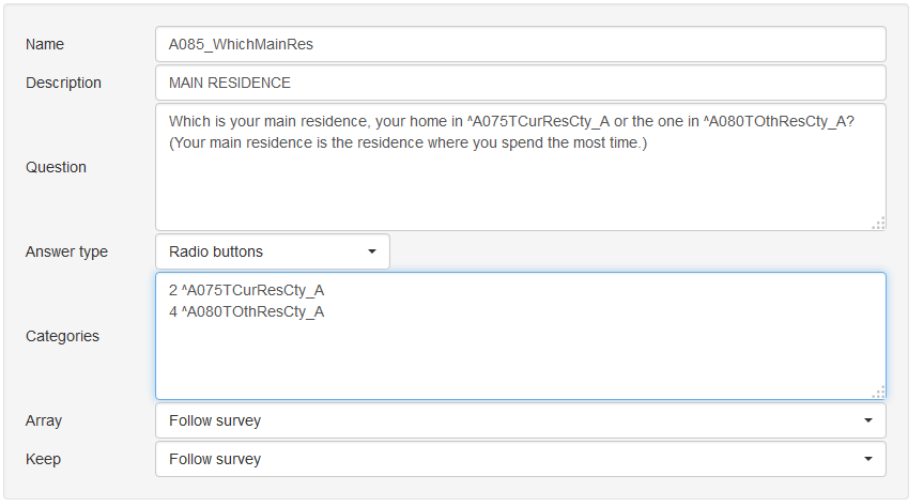
Here, NubiS first is instructed to execute the *FLHWP* fill again after *R2X060ASex* is answered. Then it asks *R2X058AFName*. The reset on line 50 is to ensure that the value for *FLHWP* is refreshed with the new information in mind. If the survey is run once more:



The usage of a fill like this is an extremely powerful way to incorporate dynamic text inside your survey. For further proof, add more variables to ask in which city the respondent is living. Specifically, add:

- A075TCurResCty_A: **"In what city is your current residence located?"** (answer type *String*)
- A079_: **"Do you have any other home?"** (answer type *Radio buttons* with answer options 1. *Yes* and 2. *No*)
- A080TOthResCty_A: **"In what city is your other residence located?"** (answer type *String*)

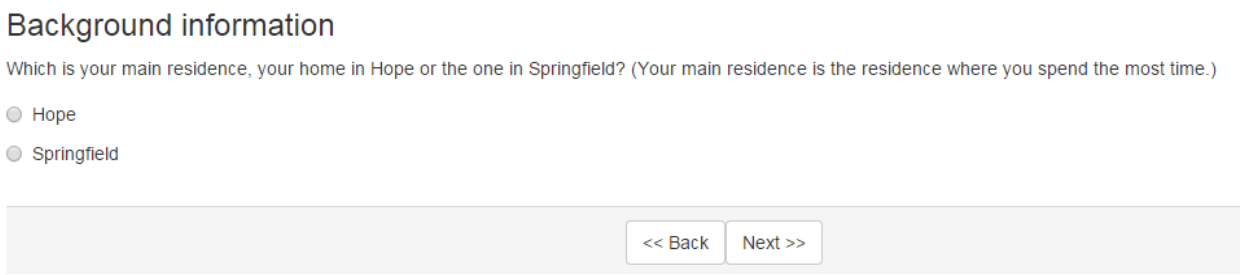
Then add this question to determine which one is the main residence:



Add in just a little bit of routing:

```
// current residence
A075TCurResCty_A
A079
IF A079_ = 1 THEN  //other home
  A080TOthResCty_A
  A085_WhichMainRes
ENDIF
```

And now when the survey is run and reaches this point, (supposing for a moment a current city was entered, as well as answering affirmatively to having another home in another city), the following should come up:



Two other often-used places for variable value references are the minimum or maximum of a range (*e.g.*, to restrict the minimum age for when a respondent wants to retire to their current age). If desired, they can be used in a wide variety of settings.

NubiS' support for variable value references encompasses almost all types of variables. That is, you can reference any type of variable – except for those of answer type *none* or *section*, because those variables never get a value. Also, variable value references can be of an exact or interpretative nature. The latter seen so far are indicated by a ^. Typically, the end is demarcated by a non-alphanumeric character, such as a plus sign (**+**) comma (**,**) or period (**.**). These

interpretative references instruct NubiS to retrieve the value of a variable, interpret it, and insert it into the reference location.

The interpretative part reflects the fact NubiS will inspect the answer type of the variable being referenced and base the text to be inserted off that. For most answer types, this means the exact value of the variable – but not for radio buttons, dropdowns, checkboxes and multi-select dropdowns. Rather, for those, NuBiS will look up the answer label(s) corresponding to the answer and return that text instead. To illustrate, if A085_WhichMainRes was referenced and the answer was *1*, you would not see *1* being inserted but *Hope*.

The exact form of a variable value reference does the same, but without the interpretative part. These take the form *B105_ and the exact value always is inserted. In the case of *A085_WhichMainRes*, this would be *1* if the first answer option were chosen and not *Hope*. This can be useful if an exact value is needed (*e.g.*, if the value is being used in a JavaScript calculation).

A noteworthy detail in this regard is the behavior of set of enumerated and multi-dropdown variables when used as fill. Recall the variable *Q1* from the previous section asking about childhood diseases: Suppose you want to use it as a fill in a follow-up confirmation question. You can do so by using a question text like this:

> *You said you had the following childhood diseases: ^Q1.*

If the respondent had selected *Asthma* and *Diabetes*, this would look like:

> *You said you had the following childhood diseases: Asthma,Diabetes.*

This is the default behavior of NubiS, that is, in a ^ reference to a set of enumerated/multi-dropdown variable, it will return a list of selected options separated by a comma. If you had in mind a list, this can be accomplished with:

> *You said you had the following childhood diseases:*
>
> *^Q1_1_*
>
> *^Q1_2_*

Referenced here are the individual answer options of the *Q1* variable. If checked, NubiS will insert the corresponding text; if not checked, it will replace the fill reference with the empty text ''.

11. ASKING THE SAME QUESTIONS MULTIPLE TIMES

This is a good place to stop and summarize the different routing constructs introduced. You now know how to instruct NubiS to ask questions, use skip patterns via *IF THEN* statements and leverage assignments to assign values to variables. However, you have yet to learn why NubiS makes some variables an array and others not. For example, *X060ASex* is specified as an array and subsequently included in the routing as *X060ASex[1]* instead of just *X060ASex*.

The basic explanation is because of the manner in which the HRS study captures information such as gender and name of the survey respondent plus any children or other members of the respondent's household. That is, they are all captured in the same conceptual variable (*i.e.*, any gender is stored in a variable whose name starts with *X060ASex*). Now, one possible implementation of this survey design choice could have been to simply introduce new variables with the stem *X060ASex* and append, for example, a *1*, a *2*, a *3* and so on. This is undesirable for two reasons. First, it would require the introduction of enough variables for the gender of any people mentioned by the respondent. This is complicated by the fact that it is not known *a priori* during survey programming how many variables should be defined. The second reason is that it would force the inclusion of each variable individually on the routing, leading to this:

X060ASex1

X004AmoBorn1

X060ASex2

X004AmoBorn2

X060ASex3

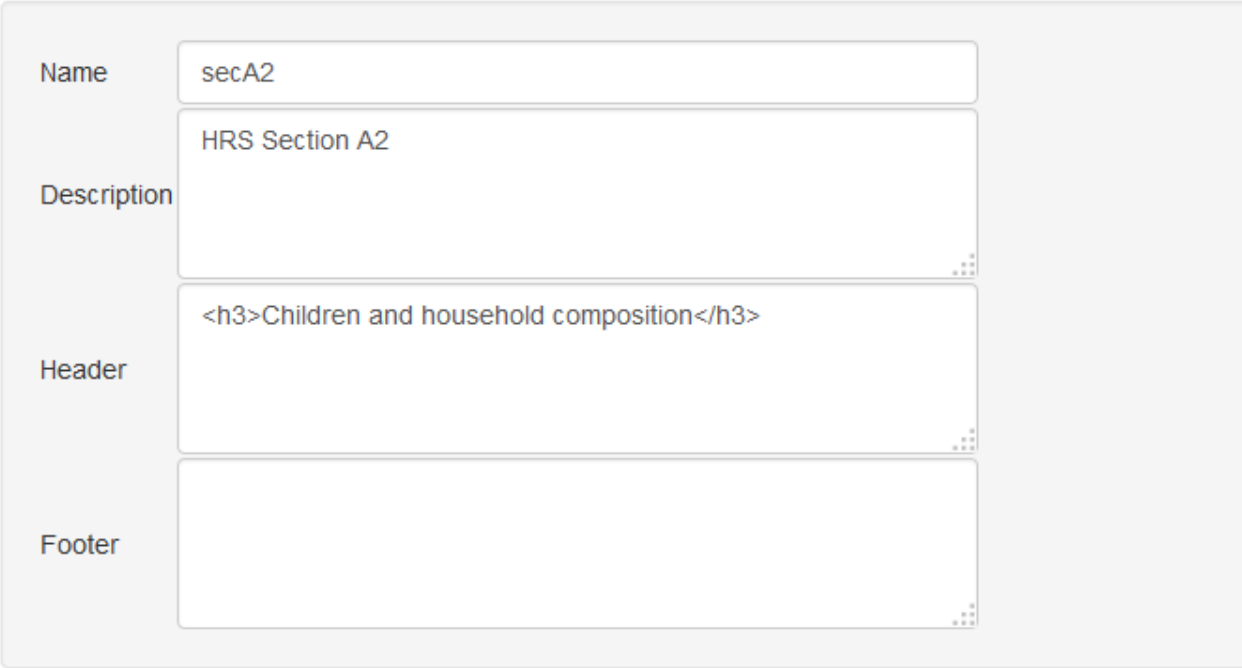
X004AmoBorn3

X060ASex4

X004AmoBorn4

Needless to say, this can quickly get out of hand. A better alternative is to employ the support of NubiS for array questions and combine this with the for loop routing construct. A for loop is an instruction to NubiS to carry out the actions inside the loop for a certain number of times. To demonstrate, it's time to add a snippet of section A2 of the HRS to your survey, which will allow you to ask if a respondent has any children – and if the answer is affirmative, the respondent can specify their gender and date of birth.

Start by adding a new section to the survey in the sections list:



Next, add a variable containing the question on whether the respondent has any children. There are several ways to go about this. For example, you could ask the respondent how many children they have. Or, you could ask if a respondent has a child and, if so, ask for gender and date of birth. Both are equally feasible and illustrate different parts of using a *for* loop – so it is useful learn to do both.

To ask "How many children do you have?," add an integer variable to your new section **secA2**. Staying with HRS's thematic naming convention, call this variable **numberofchildren**. As it also is necessary to track the number of cycles performed by the loop, you must introduce a **counter**. This variable is typically an integer question as well, so make a copy of *numberofchildren* and rename it to **cnt** with no other changes needed.

Now specify the *for* loop on the *Routing* tab of *secA2*:

```
numberofchildren
if numberofchildren > 0 then
    for cnt := 1 to numberofchildren do
        X060ASex[cnt]

        GROUP.TBirthDate
          X004AmoBorn[cnt]
          X005AdaBorn[cnt]
          X067AYrBorn[cnt]
        ENDGROUP
    enddo
endif
```

The first two lines are straightforward: Asked first is *"How many children do you have?"* – and if the answer is more than zero, the loop is entered. But take a closer look at the loop's different components. The first line (line 3) states:

for cnt := 1 to numberofchildren do

This reveals the general syntax of a *for* loop: Start with *for*, then list the counter field, *cnt,* setting its value (as indicated by the **:=** symbol) to the number at which you wish to start the loop (in this case, *1*). Follow this with **to** and the maximum number of times you wish to perform the loop. Here, the answer to the question is used as the maximum. (Note that you also could have set a fixed number, of course.) Finish the loop statement by adding **do**.

Next, on lines 4 and 6-10, NubiS is told to ask gender and date of birth. To ensure that the respondent's answers are captured separately for each child, add an array identifier to every variable reference. These array identifiers take the form **[variable/number]**, such as *[cnt]* here and *[1]* in section *secA*. Lastly, close the loop with **enddo,** and close the original **IF** statement with an **ENDIF**.

To see your new snippet in action, just add *secA2* to the base section routing:

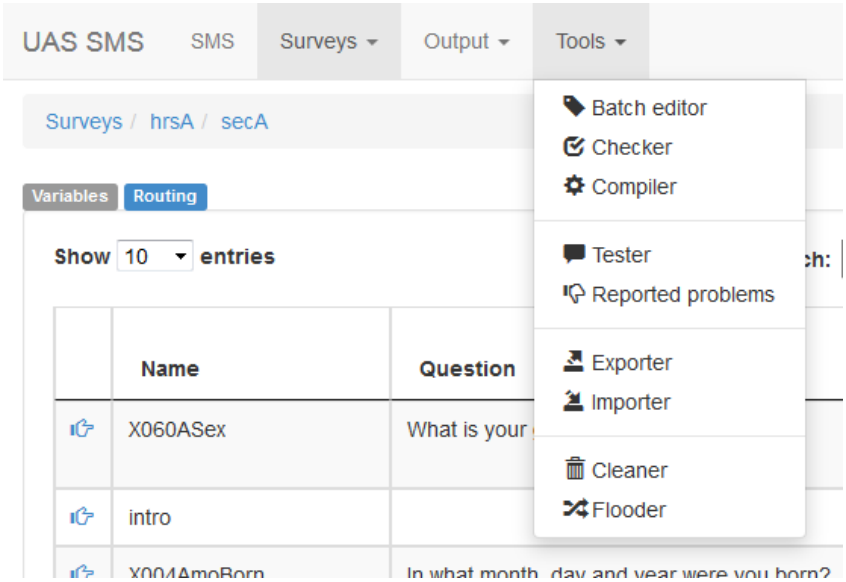
```
1 //secA
2 secA2
3
```

(For testing purposes, *secA* is commented out, so the test immediately starts with the new part.)

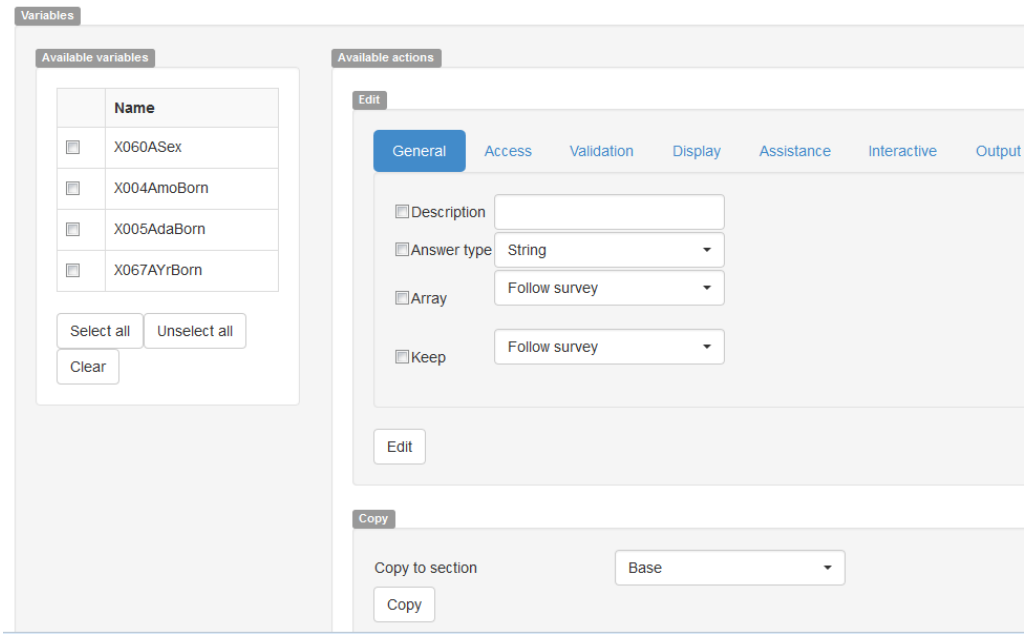
Now, if the question asking the number of children is answered with *2*, you will see the *"What is your gender?"* and *"What is your date of birth?"* questions twice before the survey concludes. Obviously, the question texts are wrong, because the same questions are asked two times and, moreover, they appear to be asked about the respondent himself or herself. There are two ways to fix this. One solution is to introduce fills dependent on the *cnt* variable to change the question text. The second fix is to define new variables with different question text and then store the values of these variables in those already existing. NubiS has no preference either way, so the choice between them is ultimately a programming decision: typically, what is easiest and most intuitive to accomplish.

Here let's make new variables by copying the relevant questions. To do this quickly, go to the *Variables* of *secA*. Hover over the pointed finger in front of *X060ASex* and click the *tag* symbol:

Do the same for the three questions pertaining to date of birth. Then, find *Tools* | **Batch editor** in the top navigation bar:



The screen that opens will look as follows:



Batch editor facilitates batch-style operations on multiple variables, types, groups, and sections. For now, click **Select all**, select *secA2* in the dropdown next to **Copy to section** and click **Copy**. The batch editor will confirm your action. When you return to *secA2*, you should find copies of these variables there. Rename all variables to remove the *_cl* and start them with **Child**.

	Name	Questiontext	Description
	numberofchildren	How many children do you have?	
	cnt		
	ChildX060ASex	What is the gender?	SEX OF INDIVIDUAL
	ChildX004AmoBorn	In what month, day and year was the child born?	MONTH OF BIRTH
	ChildX005AdaBorn		DAY OF BIRTH
	ChildX067AYrBorn		YEAR OF BIRTH

Note that the question texts were modified to make them more neutral.

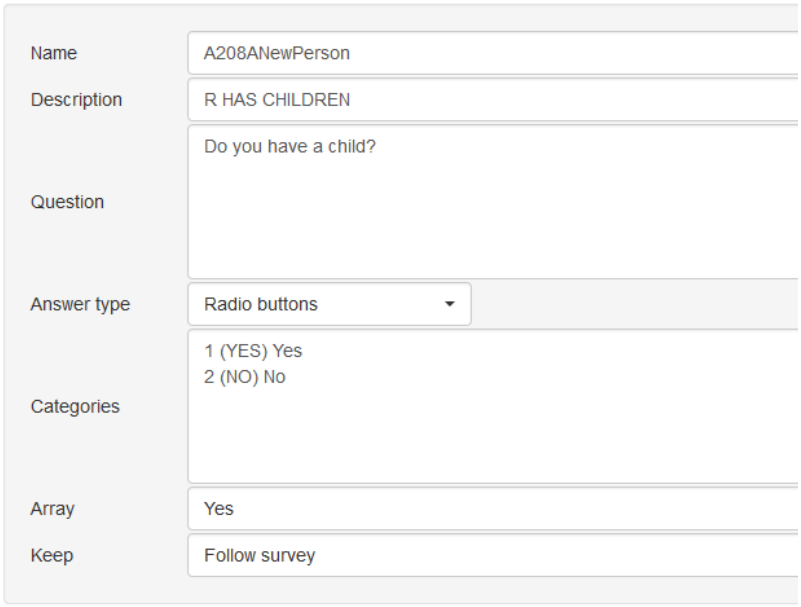
Switching to the *Routing* tab, adjust the code there to:

Variables | Routing

```
numberofchildren
if numberofchildren > 0 then
    for cnt := 1 to numberofchildren do
        ChildX060ASex[cnt]

        GROUP.TBirthDate
          ChildX004AmoBorn[cnt]
          ChildX005AdaBorn[cnt]
          ChildX067AYrBorn[cnt]
        ENDGROUP

        X060ASex[cnt] := ChildX060ASex[cnt]
        X004AmoBorn[cnt] := ChildX004AmoBorn[cnt]
        X005AdaBorn[cnt] := ChildX005AdaBorn[cnt]
        X067AYrBorn[cnt] := ChildX067AYrBorn[cnt]
    enddo
endif

```

This does the same thing as before, except that the newly introduced variables use the more neutral question texts and then assign their values to the variables from before.

At this point, to restructure the above to follow the HRS's way of asking these questions, it is necessary to introduce a variable containing the question of whether the respondent has any children (as opposed to asking how many children the respondent has). For this, go to the *Variables* tab and add a new variable A208ANewPerson as follows:



Observe that the variable is an array. This is because you want to set up your routing so that it will enter the loop, ask if the respondent has any children, upon a positive response ask the questions about gender and date of birth, and then ask the new question again. As such, if *A208ANewPerson* were not an array, then once NubiS entered the loop a second time the first answer still would be there (because NubiS would not know to distinguish between the answer for the first loop and the answer for the second loop). The accompanying routing now becomes:

```
for cnt := 1 to 50 do
   A208ANewPerson[cnt]
   if A208ANewPerson[cnt] = YES then
      ChildX060ASex[cnt]

      GROUP.TBirthDate
        ChildX004AmoBorn[cnt]
        ChildX005AdaBorn[cnt]
        ChildX067AYrBorn[cnt]
      ENDGROUP

      X060ASex[cnt] := ChildX060ASex[cnt]
      X004AmoBorn[cnt] := ChildX004AmoBorn[cnt]
      X005AdaBorn[cnt] := ChildX005AdaBorn[cnt]
      X067AYrBorn[cnt] := ChildX067AYrBorn[cnt]
   endif
enddo
```

A few things are different here. Gone is the question asking how many children the respondent has. Instead, the loop simply starts and there is a maximum of 50 (no person is expected to list

more than 50 children in the HRS). Immediately upon entering the loop, NubiS is instructed to ask only the added variable (line 2). If the answer is *Yes*, then NubiS asks the follow-up questions.

Clearly, the above will accomplish the same thing as the previous example of routing, just in a slightly different manner. Of course, some refinements are necessary to make things look more natural (*e.g.*, modifying the question text of *A208ANewPerson* so that after the first loop NubiS asks, **"Do you have another child?"** instead of *"Do you have a child?"* Another tweak to be made is more structural in nature and has to do with what should happen if the respondent answers *No* to *A208ANewPerson*. Currently, NubiS simply will ask the question again until it has completed the loop 50 times. Obviously, this should be avoided. Instead, if the respondent says *No*, the loop should end and NubiS should continue with whatever follows the loop.

And NubiS has a routing construct to do just that:

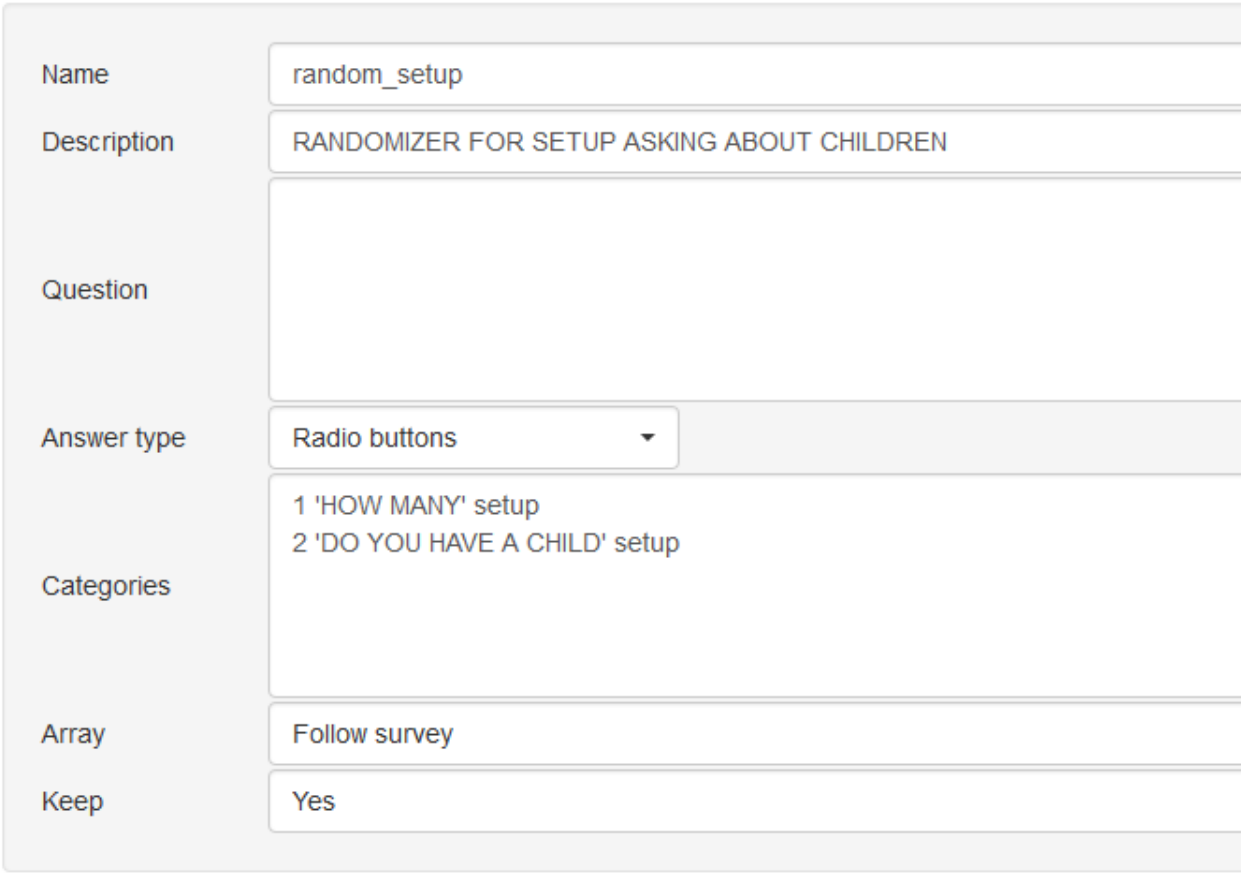
```
1  for cnt := 1 to 50 do
2     A208ANewPerson[cnt]
3     if A208ANewPerson[cnt] = YES then
4        ChildX060ASex[cnt]
5
6        GROUP.TBirthDate
7          ChildX004AmoBorn[cnt]
8          ChildX005AdaBorn[cnt]
9          ChildX067AYrBorn[cnt]
10       ENDGROUP
11
12       X060ASex[cnt] := ChildX060ASex[cnt]
13       X004AmoBorn[cnt] := ChildX004AmoBorn[cnt]
14       X005AdaBorn[cnt] := ChildX005AdaBorn[cnt]
15       X067AYrBorn[cnt] := ChildX067AYrBorn[cnt]
16    else
17       exitfor
18    endif
19 enddo
20
```

As you can see, added lines 16-18 comprise an *ELSE* statement with a single-action *EXITFOR*. This routing construct instructs NubiS to exit the *for* loop as soon as it is encountered. In other words, anything following the *EXITFOR* will be ignored by NubiS.

12. RANDOMLY ASKING QUESTIONS

Sometimes, a survey administrator may want to ask questions randomly. Perhaps there is a desire to ask a particular question only to a subset of respondents; or for different people to receive different question phrasing; or even that multiple questions should appear on the same screen in a random order. These requests in NubiS fall under the term randomization.

Randomization, to a large extent, consists of topics already discussed with some added features. To demonstrate, start with asking half the respondents how many children they have while the other half is asked whether they have any children. (The goal is to see if there is any difference in the quality and completeness of the collected data.) An imaginary example of code would be something like **if setup_one_selected then setup_one, else setup_two**. To implement this, needed is a variable that will be the randomizer:



A randomizer's values usually are numeric but do not have to be. In the above example, the randomizer is a radio button answer type rather than just an integer. Why? The reason has not so much to do with the programming, but rather with how any collected data gets outputted. By

making random_setup a radio button variable, labels can be specified for the values the randomizer will take – which, in turn, can be incorporated into the outputted Stata file, making data analysis easier.

Also, note that *Keep* has been set to *Yes*, which makes sure a variable does not lose its assigned value if a respondent goes back in the survey. To get a better sense of what that means, put the randomizer into action:

```
1  intro
2  if random_setup = empty then
3     random_setup := mt_rand(1,2)
4  endif
5
6  if random_setup = 2 then
7    for cnt := 1 to 50 do
8       A208ANewPerson[cnt]
9       if A208ANewPerson[cnt] = YES then
10         ChildX060ASex[cnt]
11
12         GROUP.TBirthDate
13           ChildX004AmoBorn[cnt]
14           ChildX005AdaBorn[cnt]
15           ChildX067AYrBorn[cnt]
16         ENDGROUP
```

Please note this screenshot shows only half of the necessary routing; the other half, with *numberofchildren*, was truncated for space. Here, several lines have been inserted before the *for* loop. The new lines 1-2 state if *random_setup* does not have a value, it will be given a value of either 1 or 2; mt_rand is a standard PHP function employed to perform the randomization. Then, if *random_setup = 1*, the respondent is asked whether they have any children. Otherwise, when *random_setup = 2*, they are asked how many children they have.

Note that included here is asking intro, a question of no input. It was added so there is a question screen prior to the randomizer being set. This is to illustrate its value being selected and the role of the *Keep* setting. Now if the survey is run through the tester, the introduction screen comes up. Then, upon clicking *Next*, in the background NubiS will assign a value to *random_setup* and, depending on its value, ask either *A208ANewPerson* or *numberofchildren*.

But suppose that at this point you go back to the first screen by clicking *Back* and then click *Next* again. What would this do to the randomizer? The short answer is that without the keep, the randomizer's result(s) would be forgotten – which is not the wanted outcome here.

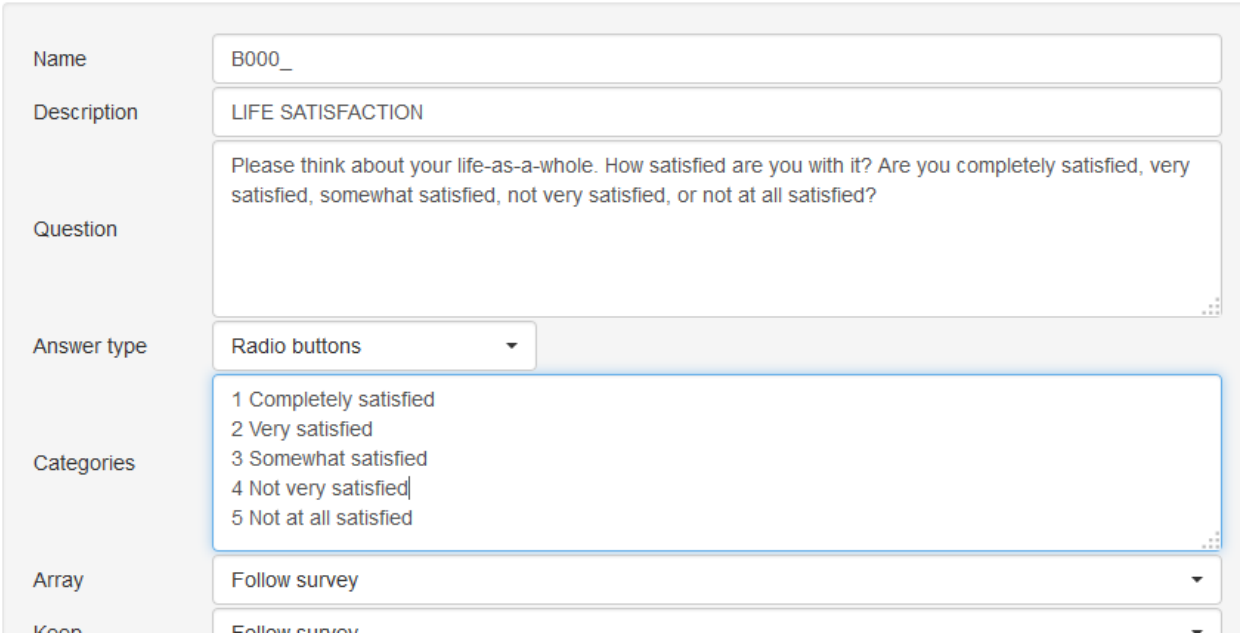
Under normal circumstances, NubiS, while going from one question screen to another, undoes any assignments it previously made, restoring the state of the survey to as it was. That is, NubiS pretends assignments never happened and resets any variables involved in an assignment back to

their prior value(s). This behavior ensures, for example, that a *for* loop counter is properly decremented when going back in a loop. It also prevents residue data to be left (*e.g.*, the result of an age calculation).

Here, though, the result would be undesirable. Trace the steps: The respondent clicked *Back* and, consequently, NubiS undid assigning a random value to *random_setup*. When the respondent clicked *Next* again, the *IF* condition on line 2 was met because NubiS gave *random_setup* its old value, which was an empty value. So NubiS then assigned a random value to *random_setup* – and this new random value may or may not be the same as before. As a result, the behavior that follows might vary.

Because this is an unwanted outcome, NubiS needs instructions to prevent this chain of events from unfolding. This is where the *Keep* setting proves its worth. By setting it to *Yes*, NubiS is told that upon going back, any assignments made for *random_setup* should not be undone. In a sense, NubiS treats *random_setup* as if it were a direct answer given by a respondent – and those answers always are kept by NubiS. (If you wish to experiment for yourself, just set *Keep* back to *Follow survey,* or *No* for *random_setup*, then test the survey a few times. Soon you will notice NubiS using both the first and second setups for asking about children.)

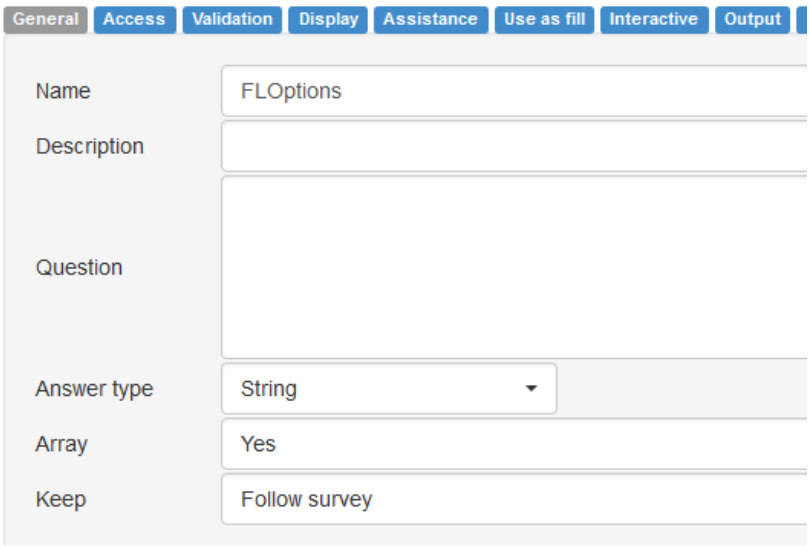
Another way in which a randomizer can be used is to switch between different texts, be they question text or answer options. The components are similar with the slight difference being that the randomizer now is used in the fill code of a fill variable to direct which text gets used when. To illustrate, use the opening question of HRS section B, first adding it a new section, secB.:



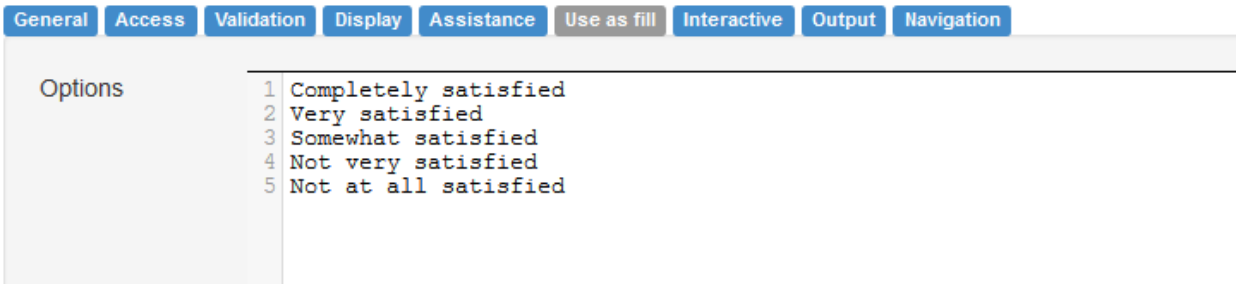
Then add B000_ to the routing of *secB* and put *secB* on the routing of the *Base* section, replacing *secA2*:

```
//secA
//secA2
secB

```

The answer options range from Completely satisfied to Not at all satisfied. But say you want to flip the order for half of the respondents. Not surprisingly, what's needed is a randomizer (call it random_order). Also needed is a fill variable for the answer options (because now they are going to be dynamic):



Make the variable an array, because it must contain all the answer options. Next, on the *Use as fill* tab, enter:

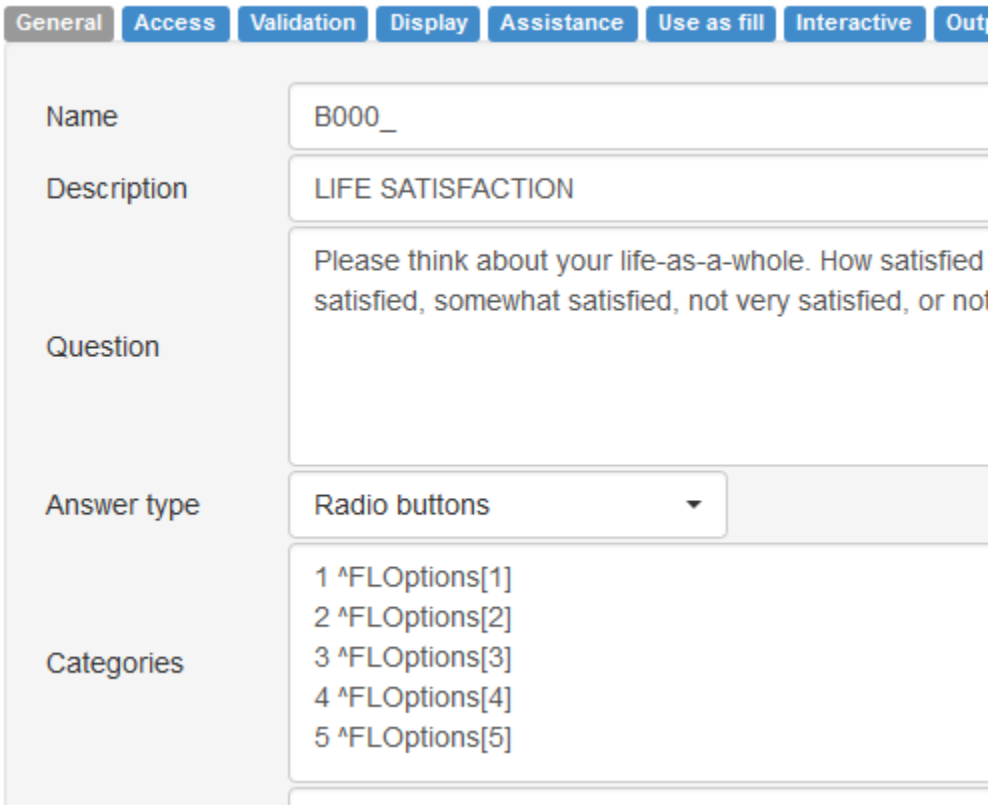


And:

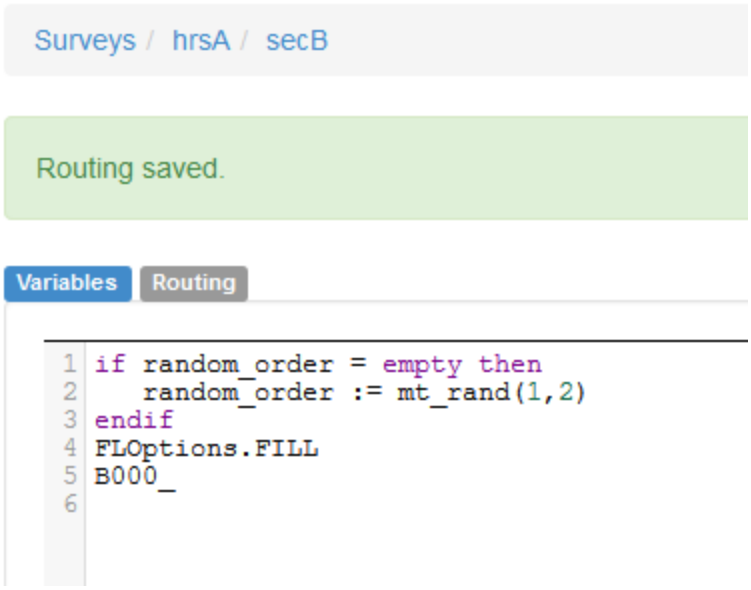
Code

```
if random_order = 1 then
  FLOptions[1] := FILL1
  FLOptions[2] := FILL2
  FLOptions[3] := FILL3
  FLOptions[4] := FILL4
  FLOptions[5] := FILL5
else
  FLOptions[1] := FILL5
  FLOptions[2] := FILL4
  FLOptions[3] := FILL3
  FLOptions[4] := FILL2
  FLOptions[5] := FILL1
endif
```

Next, update *B000_* to refer to FLOptions:



Finally, modify the routing of *secB*:

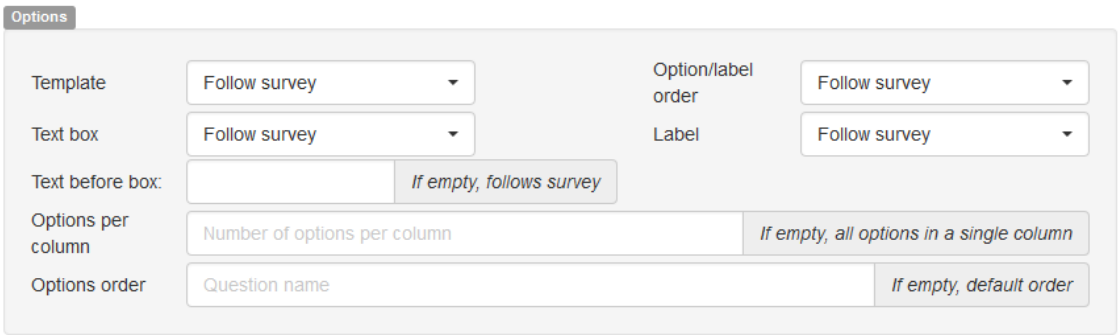


Based on that FLOptions.FILL, you now have made sure *random_order* gets a value (if not set before,), which then is used in *B000_* for the answer options: *completely satisfied* to *not at all satisfied* or *not at all satisfied* to *completely satisfied* depending upon the value assigned to *random_order*.

The downside of this approach crops up in data analysis, when it is not directly known which option was presented with what value. For example, some respondents saw *Completely satisfied* as Option 1, but for others the first option was *Completely unsatisfied*. To account for this, derandomization must construct a variable that defines its values independent of the randomizer.

For this reason, NubiS provides an alternative for displaying answer options in random order. Add new variable B000_order as an integer array and set *Keep* to yes.

Return to *B000_* and change the answer options to the original *"1 Completely satisfied"* to *"5 Completely unsatisfied."* Then, select the *Display* tab and enter *B000_order* for the options order:



Finally, edit the routing to:

```
if random_order = empty then
    random_order := mt_rand(1,2)
    if random_order = 1 then
        b000_order := array(1 => 1, 2 => 2, 3 => 3, 4 => 4, 5 => 5, 6 => 6)
    else
        b000_order := array(6 => 6, 5 => 5, 4 => 4, 3 => 3, 2 => 2, 1 => 1)
    endif
endif
b000_
```

Going line by line, NubiS picks a value for *random_order* and, based on that value, sets *B000_order* to an array from 1 to 6 or from 6 to 1. Then, upon showing *B000_*, NubiS uses this array to determine the order of display of the answer options. On screen the effect is the same, but in the data, crucially, answer codes are tied to their answer options. That is, regardless of the value of *random_order, completely satisfied* always is represented by a value of 1. Note: the described procedure works for any answer type with answer categories (for example radio button, check box, and rank).

12.1 Ask Multiple Questions in a Random Order

The types of randomization discussed so far in NubiS are fairly common. But a rarer form of randomization may occur, which has to do with asking a series of questions in a random order. Because HRS does not ask questions in such a manner, an artificial example will be used to illustrate. Use these three HRS questions:

- B105_: "Did you have asthma growing up?"
- B106_: "Did you have diabetes growing up?"
- B107_: "Did you have a respiratory illness such as bronchitis growing up?"

Typically, these questions would be asked in order. But suppose you are worried that the order in which you ask the questions impacts the manner in which the respondent answers. For example, respondents may be more fatigued and say *No* to later questions. Or maybe as more questions are asked, the respondent more accurately recalls his/her childhood. Whatever the reason, how should you ask these three questions in random order? One way is to use what you've learned so far: Introduce a randomizer, let its value vary between *1* to *6* (to accommodate all six sequences) then use that in the routing to ask the questions in a certain order:

```
    random_order := mt_rand(1,2)
endif
FLOptions.FILL
B000_

if random_sequence = empty then
    random_sequence := mt_rand(1,6)
endif

if random_sequence = 1 then
    B105_
    B106_
    B107_
elseif random_sequence = 2 then
    B105_
    B107_
    B106_
elseif random_sequence = 3 then
    B106_
    B105_
    B107_
```

Due to length, the above screenshot doesn't show all sequences: One can imagine that the addition of a fourth variable would make this method even more burdensome as there would be more combinations to consider. A more efficient solution is:

Variables | Routing

```
if random_order = empty then
    random_order := mt_rand(1,2)
endif
FLOptions.FILL
B000_

FLQuestions := array(1 => "B105_", 2 => "B106_", 3 => "B107_")
if random_sequence = empty then
    random_sequence := shuffleArray(array(1 => 1, 2 => 2, 3 => 3))

endif

for cnt := 1 to 3 do
    FLQuestions[random_sequence[cnt]].INSPECT
enddo
```

Understandably, this looks somewhat daunting at first. But it helps to break it down before adding the required variables:

- *FLQuestions*: a string variable that is an array.
- *random_sequence*: a radio button variable that is an array and has *Keep* set to *Yes* (after all, this is a randomizer), with answer options *1 B105_*, *2 B106_* and *3 B107_*.

- *B105_*: *"Did you have asthma growing up?"* A radio button variable with answer options *Yes*/*No*.

- *B106_*: "Did you have diabetes growing up?" A radio button variable with answer options *Yes*/*No*.

- *B107_*: "Did you have a respiratory illness such as bronchitis growing up?" A radio button variable with answer options *Yes*/*No*.

With those in place, now turn to the routing:

```
FLQuestions := array(1 => "B105_", 2 => "B106_", 3 => "B107_")
if random_sequence = empty then
 random_sequence := shuffleArray(array(1 => 1, 2 => 2, 3 => 3))
endif
```

It starts by assigning a string array to *FLQuestions*: The first item with key *1* is *B105_*; the second is *B106_,* tied to key *2* and *B107_* is linked to key *3*. Then, a random order is created, contained in *random_sequence*. Notice how the condition for checking if the sequence was set before is slightly different as for non-array questions:

```
if random_sequence = empty then
```

(Note: we could also specify "if isArray(random_sequence) = 2) then" to accomplish the same thing. Here 'isArray' is a Nubis function that returns '2' if the input parameter is not an array OR the size of the array is zero; otherwise it returns '1'). In this regard, NubiS considers *random_sequence* to be empty if no array has ever been assigned before (or a value assigned to a specific array instance; *e.g.*, random_sequence[1]) or if the array is empty. So the first time NubiS encounters this line, the condition is met, so the assignment on the next line is performed:

```
random_sequence := shuffleArray(array(1 => 1, 2 => 2, 3 => 3))
```

Here shuffleArray is an internal NubiS function that takes an array as its input and shuffles the order of the elements within. So in this example, you may get back the order *3*,*1*,*2* in positions, respectively, 1, 2 and 3. Then follows a *for* loop responsible for asking the questions:

```
for cnt := 1 to 3 do
 FLQuestions[random_sequence[cnt]].INSPECT
enddo
```

The line within the *for* loop provides NubiS the instruction to ask the question. First, NubiS interprets this:

FLQuestions[random_sequence[cnt]]

Taking the first loop as example, NubiS will look up the value of *random_sequence[1]* (assume this is *3*). Then, it uses the result, *3*, to look up the instance of FLQuestions[3], which is *B107_*. Finally, NubiS takes this string and attempts to find a variable with that name, as instructed by the .INSPECT statement. If found, NubiS asks the question within that variable, here *B107_*. Supposing the order *3*,*1*,*2*, NubiS will ask *B107_*, *B105_* and, finally, *B106_*.

Although this method may be more convoluted, it also is more powerful. Adding a fourth question is as easy as extending the arrays assigned to *FLQuestions* and *random_sequence*, then increasing the loop maximum to 4. Why use two arrays when this easily could've been done with one array (*FLQuestions*) and its order randomized with shuffleArray? Again, the reason has to do with data output. Had *FLQuestions* only been used, in the outputted data there would have been three variables: FLQuestions_1_, FLQuestions_2, and FLQuestions_3_, with possible values *B105_* to *B107_*. Using *random_sequence* and defining it as a radio button answer type can specify labels to subsequently be included in the data set. Looking at those variables then would allow, for example, doing a Stata tab to quickly see the distribution of which question was asked in what place of the order.

A last thing to observe about the above is that when arrays were defined in the routing, they were specified as, for example:

random_sequence := shuffleArray(array(1 => 1, 2 => 2, 3 => 3))

That is, explicit keys for the array were defined to start at *1* rather than *0*. The reason mostly is one of convenience. By default, NubiS will store an array with the indices provided, and because PHP starts arrays at index *0*, this means the random sequence numbers would be stored as random_sequence[0], random_sequence[1] and random_sequence[2]. Consequently, you would have to adjust the routing to:

for cnt := 0 to 2 do

FLQuestions[random_sequence[cnt]].INSPECT

enddo

This would work here, but the reason it is specified to start at *1* is because loops in NubiS most often are of the form:

for cnt := 1 to 3 do

Q1[cnt]

enddo

And starting such a loop at *0* would look odd because, conceptually, the survey is asking, for example, about the first person, the second person and so on, rather than starting with the "zeroeth" person and proceeding from there.

Or, there could be a situation in which an array is used with fill text:

fills := array("hello", "world")

Then, if this is being used in a loop question, the ideal sequence is:

for cnt := 1 to 2 do

Q1[cnt]

enddo

And Q1's question text is specified as, for example, ^fills[cnt]. But doing so would lead to problems because the fills are stored at *0* and *1*, not at *1* and *2*. So you would have to adjust the fill reference to ^fills[cnt-1]. As this is an extra step to remember, it is easier to just define keys explicitly to start at *1* or use NubiS' built-in function to do so:

fills := incrementArrayIndices(array("hello", "world"))

As a final note: in all the array examples numeric indices were used for anything stored in the array. As in, for example:

random_sequence := shuffleArray(array(1 => 1, 2 => 2, 3 => 3))

or implicitly leaving it to PHP in:

fills := incrementArrayIndices(array("hello", "world"))

It is also possible to use non-numeric, or associative, indices. For example, we could have stated:

random_sequence := shuffleArray(array('a' => 1, 'b' => 2, 'c' => 3))

While this would make it hard to use *random_sequence* in a loop, it would not prevent the use of it in, for example, fill reference such as ^random_sequence["a"]. Lastly, it is important to remember that if you are going to use associative arrays, in NubiS ^random_sequence[1] is not the same as random_sequence["1"]. Therefore, care should be taken not to use them interchangeably.

12.2 Ask Multiple Sections in a Random Order

Rather than randomly ordering questions it may be desired to randomly order sections instead. Though this does not occur in the context of the HRS situations can be imagined where one set of questions would influence answers on another set of questions. For example, asking about job status and income and then about subjective well being may yield different answers from doing it the other way around for people who are struggling economically.

One way is to use what we described for randomly ordering questions: Introduce a randomizer, let its value vary between *1* and *2* (to accommodate the two sequences) then use that in the routing to ask the sections in a certain order:

The above is quite straightforward and very similar to what we have seen so far. But of course if we want to randomly order a large number of sections, we run into the same issue as we encountered for ordering a large number of questions. The solution for this problem looks, not surprisingly, very similar to the one in the previous section:

Variables | Routing

```
1  FLSections := array(1 => "jobstatus", 2 => "wellbeing", 3 => "retirement", 3 => "family")
2  random_order.keep
3  if random_order = empty OR sizeof(random_order) = 0 then
4     random_order := shuffleArray(array(1 => 1, 2 => 2, 3 => 3, 4 => 4)
5  endif
6
7  for cnt := 1 to 4 do
8     FLSections[random_order[cnt]].INSPECTSECTION
9  enddo
10
```

Breaking down the required variables again:

- *FLSections*: a string variable that is an array.
- *random_order*: a radio button variable that is an array and has *Keep* set to *Yes* (after all, this is a randomizer), with answer options *1 jobstatus*, *2 wellbeing*, *3 retirement* and *4 family*.
- *jobstatus* to *family*: the four sections we are asking (each with their own routing)

With those in place, now turn to the routing:

FLSections := array(1 => "jobstatus", 2 => "wellbeing", 3 => "retirement", 3 => "family")

random_order.KEEP

if random_order = empty OR sizeof(random_order) = 0 then

random_order := shuffleArray(array(1 => 1, 2 => 2, 3 => 3, 4 => 4))

endif

It starts by assigning a string array to *FLSections* containg the four section names. Then, a random order is created, contained in *random_order*. Notice how the condition for checking if the sequence applies to ensure that the order either has not been set (empty) or is empty (sizeof):

if random_order = empty OR sizeof(random_order) = 0 then

(alternatively, we could use "if isArray(random_order) = 2 then")

Then follows a *for* loop responsible for asking the questions:

for cnt := 1 to 4 do

FLSections[random_order[cnt]].INSPECTSECTION

enddo

The line within the *for* loop provides NubiS the instruction to carry out the routing in a section. First, NubiS interprets this:

FLSections[random_order[cnt]]

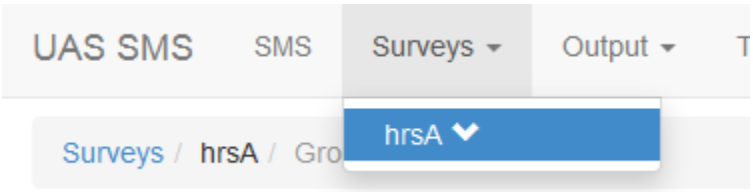
Taking the first loop as example, NubiS will look up the value of *random_order[1]* (assume this is *3*). Then, it uses the result, *3*, to look up the instance of FLSections[3], which is *retirement*. Finally, NubiS takes this string and attempts to find a section with that name, as instructed by the .INSPECTSECTION statement. If found, NubiS carries out the instructions in the section.

As with questions, this method may be more convoluted, but it also is more powerful. Adding a fifth section is as easy as extending the arrays assigned to *FLSections* and *random_order*, then increasing the loop maximum to 5. Again, why use two arrays when this easily could've been done with one array (*FLSections*) and its order randomized with shuffleArray? Once more, the reason has to do with data output. Had *FLSections* only been used, in the outputted data there would have been four variables: FLSections_1_, FLSections_2, FLSections_3, and FLSections_4_, with possible values *jobstatus* to *family*. Using *random_order* and defining it as a radio button answer type can specify labels to subsequently be included in the data set. Looking at those variables then would allow, for example, doing a Stata tab to quickly see the distribution of which section was asked in what place of the order.

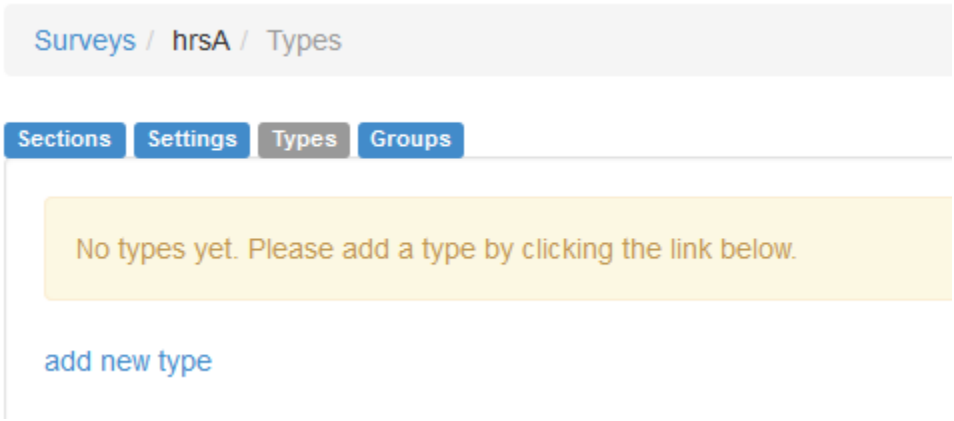
13. DEFINING AND MANAGING SIMILAR QUESTIONS

Over the past few sections, on multiple occasions you have added variables whose answer categories were *Yes* and *No*. Each time you added those categories, you created an additional location in which these categories were defined. While doing that, you may have wondered if there is a better way. For example, suppose someone now comes to you and asks, "Can we please use answer code 5 for *No* instead of answer code 2?" As things stand, the only way to accomplish this is by adjusting each and every variable with those answer categories. The batch editor may be used to do this to reduce the effort involved, but nonetheless it would require some effort.

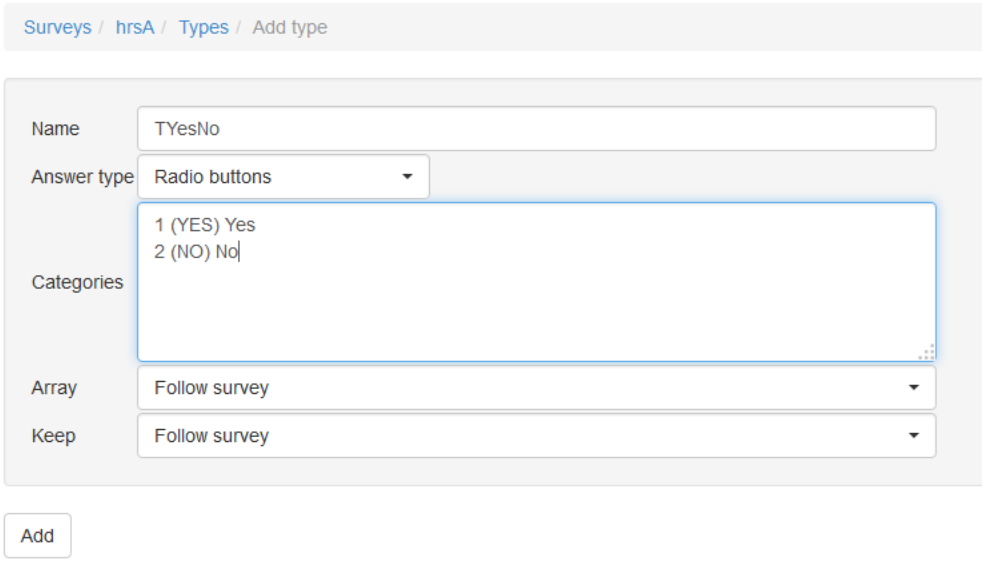
A better way is to use NubiS' type system. A type can be thought of as the blueprint for a set of variables, specifying a set of shared characteristics. Then those characteristics can be augmented with variable-specific settings, such as answer categories or inclusion/exclusion in the data output. Furthermore, characteristics can be overridden to customize certain features. To illustrate, add a type to define *Yes*/*No* questions. First, go back to the top level of the survey:



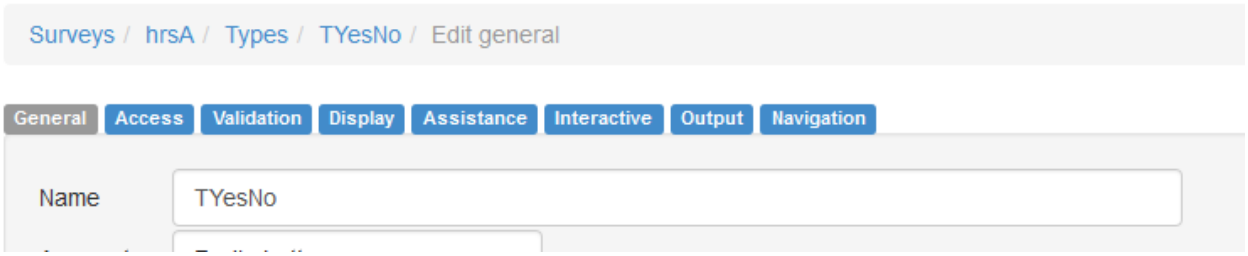
Next, locate and open the *Types* tab:



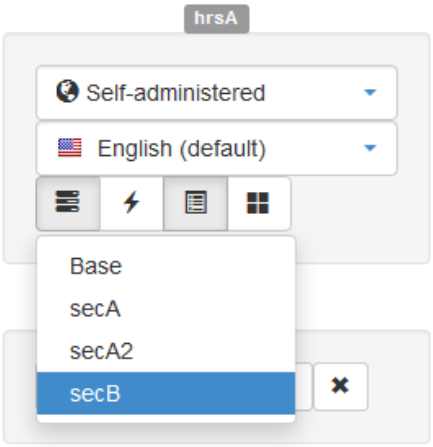
And click **add new type**:



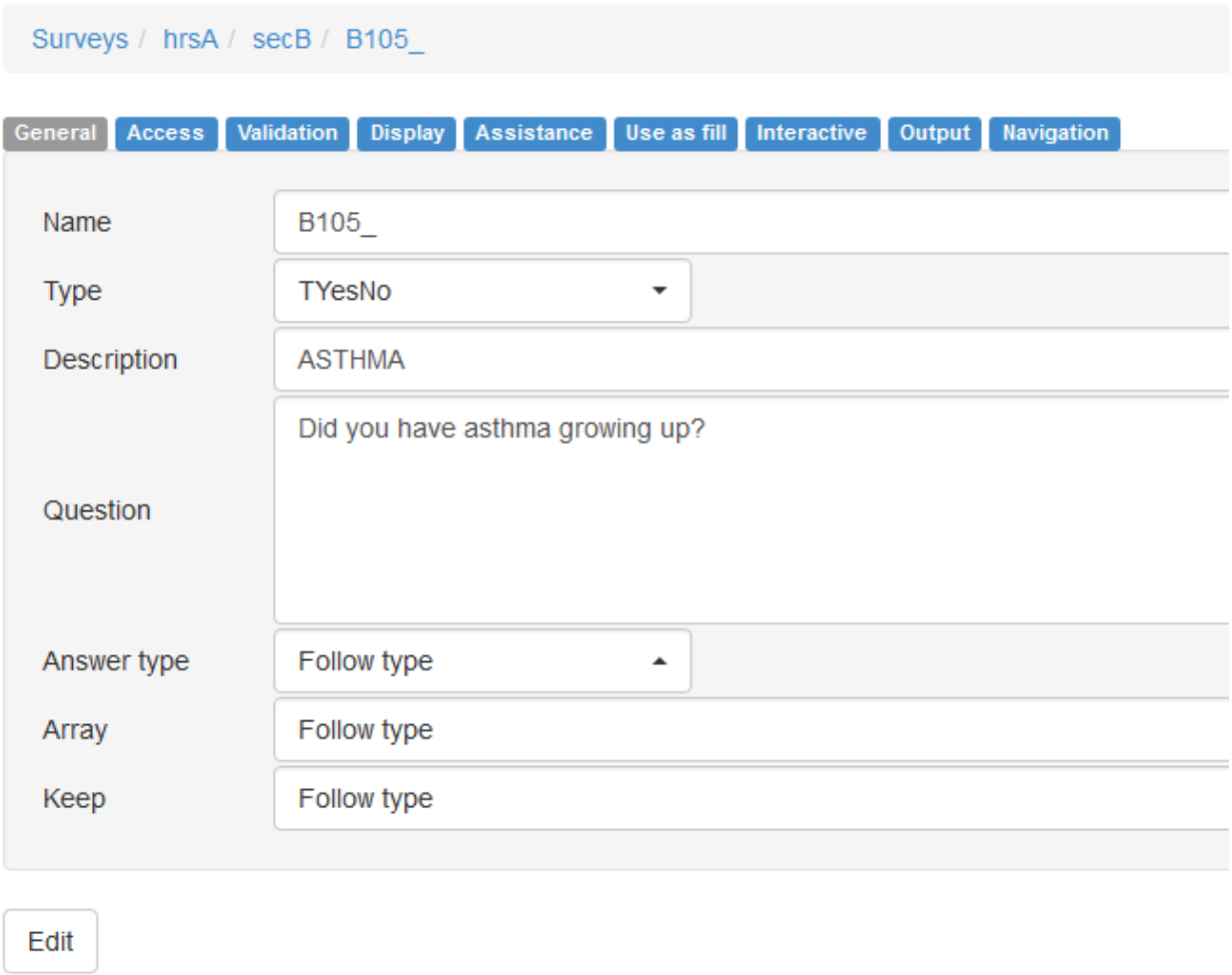
As you can see, the basic definition of a type is very similar to that of a variable, except it lacks variable-specific items such as question text. If you edit your new type, you see that a type can also hold the other types of setting discussed for variables:



You are now ready to start using your type. Enter *secB*:



And modify *B105_* on the *Variables* tab:



Now you see a *type* setting that wasn't there before. In it, choose TYesNo. Then set *Answer type*, *Array* and *Keep* to follow the type's setting. Note that if you wish to deviate from the type for any of those properties, you can just specify a value.

The effect of this on respondents is non-existent: They will still see the same answer categories in the same way. The added value is for programmers, who now can mass-update a set of variables of a particular type by modifying the type, rather than have to adjust each individual variable. As you will see later on, this is extremely useful also when translating a survey into different interview modes and/or languages.

14. CONFIGURING THE DISPLAY OF QUESTIONS

In Sec. 8, it was explained how a set of questions can be grouped on the same screen using *GROUP* statements. The example used was asking for the date of birth. The demonstration focused on the mechanics of getting NubiS to ask the questions, but not too much attention was paid to their display (with the exception of some helpful labels in front of the input boxes and dropdown).

Overall, in the display of groups of questions, NubiS adopts a templating approach in which commonly used templates have been pre-defined. (If needed, custom templates can be defined as well.) To illustrate some pre-defined templates, slightly update the routing for *secB*:

```
12
13 group.TShow
14 for cnt := 1 to 3 do
15    FLQuestions[random_sequence[cnt]].INSPECT
16 enddo
17 endgroup
18
```

Save

The goal is to group the randomly-ordered questions on the same screen. Right now, the survey looks like:

Did you have asthma growing up?

- Yes
- No

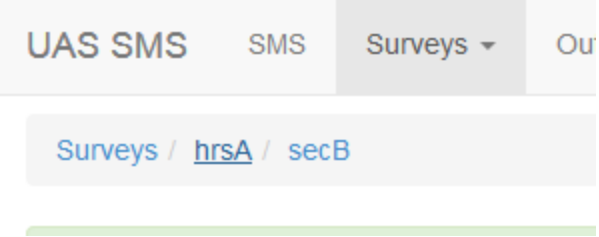
Did you have diabetes growing up?

- Yes
- No

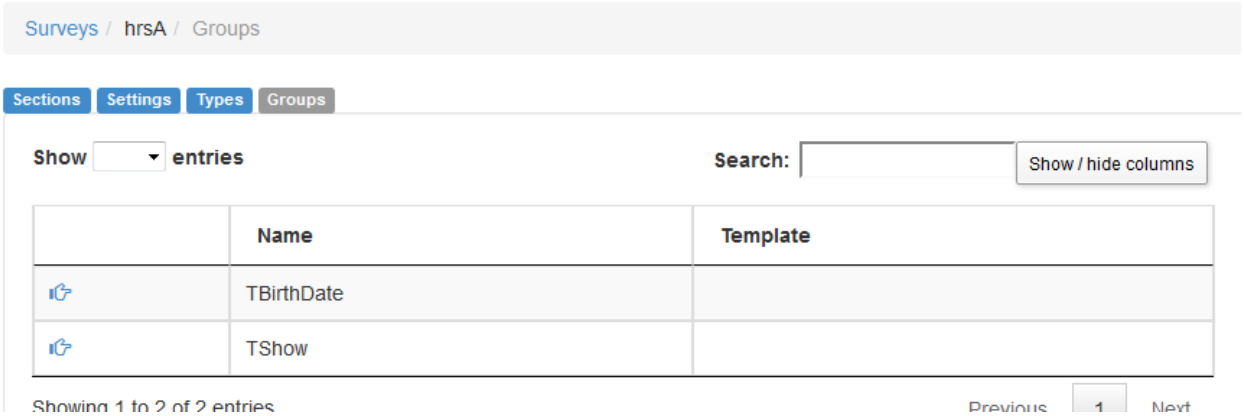
Did you have a respiratory illness such as bronchitis growing up?

- Yes
- No

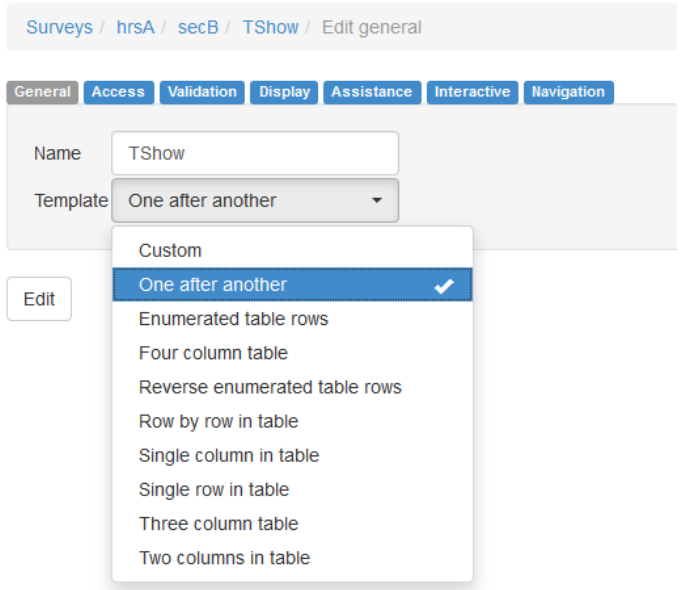
There is nothing wrong with this, but you might want to try another display. Using the *hrsA* link towards the top…:



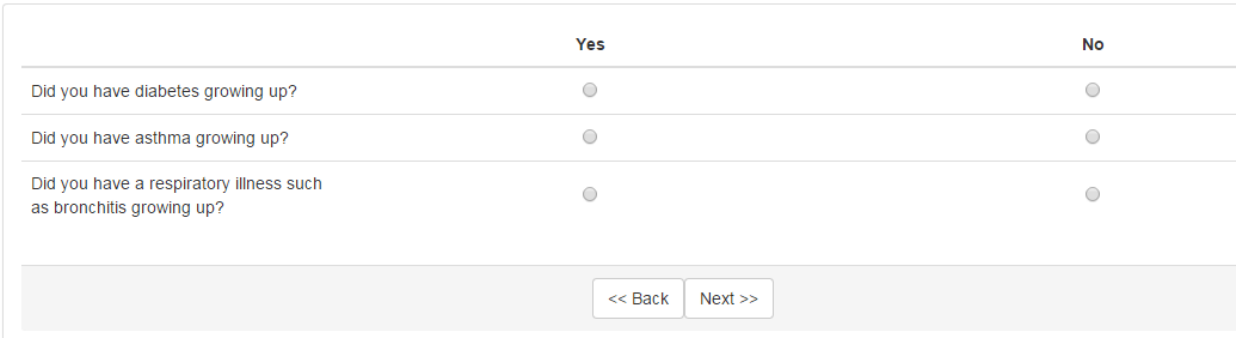
…navigate to the top of the survey and select the **Groups** tab:



The list shows the two groups used in the routing so far. (NubiS created **TShow** when saving the routing for *secB*). Edit *TShow* and choose as a template **Enumerated table rows**:



Now save and return to our survey, where you should see following display:



If you want, this could be refined more by, for example, adding borders or row highlighting. Or you could choose the **Reverse enumerated table rows** template:

Now the questions appear as column headers instead of the answer options. Perhaps you have introduction text to be added above the table (*e.g.*, **"We would now like to ask you about any diseases you may have had during your childhood."**) At first, this should seem not too difficult: create a *no input* variable (call it **b_intro**) then add it to the routing:

```
group.TShow
b_intro
for cnt := 1 to 3 do
   FLQuestions[random_sequence[cnt]].INSPECT
enddo
endgroup
```

However, the result is this:

What happened? Well, NubiS was told to use the *Enumerated table rows* template for *TShow*, so NubiS also took the new variable **B_intro** and tried to enter it into that template. To work around this, you need to slightly extend the GROUP statement:

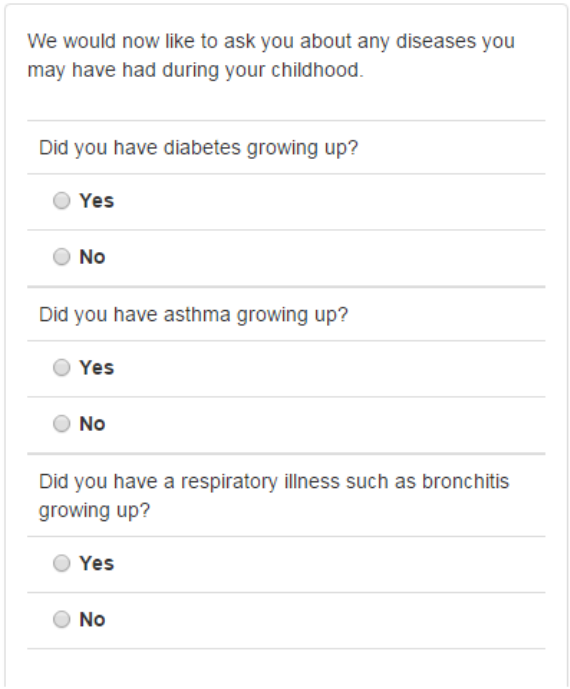
```
group.Toverall
b_intro
subgroup.TShow
for cnt := 1 to 3 do
   FLQuestions[random_sequence[cnt]].INSPECT
enddo
endsubgroup
endgroup
```

This fragment introduces the notion of subgroups. SUBGROUP statements are just like GROUP statements, with the difference being that they can only occur within a GROUP. Also it is not possible to have a subgroup within a subgroup. The main benefit of subgroups is how it allows NubiS to apply different templates to different parts (and enforce different validation criteria, as you will see shortly). In this case, Toverall is a template NubiS made upon saving the routing, which follows the default of placing one question after another. Then, *TShow* for the new subgroup will apply the *Enumerated table rows* template to the three questions. The result is:

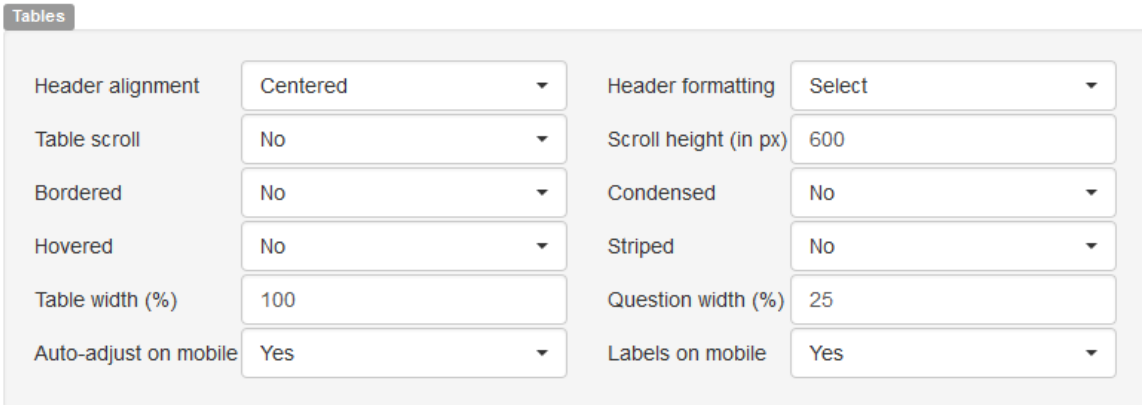
We would now like to ask you about any diseases you may have had during your childhood.

	Yes	No
Did you have diabetes growing up?	◉	○
Did you have asthma growing up?	○	◉
Did you have a respiratory illness such as bronchitis growing up?	◉	○

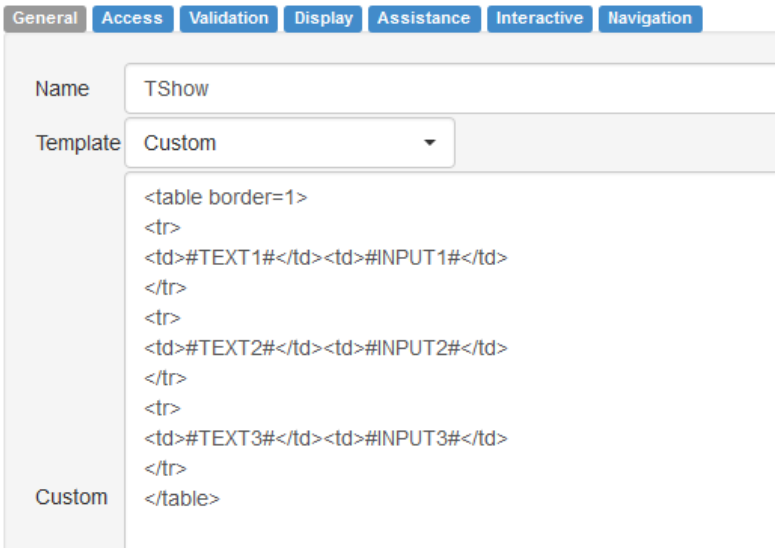
It should be noted that the display of tables like the above is different on screens smaller than 640 pixels, such as mobile phones. On such screens, these tables can be presented in a vertical format:



Whether this look should be adopted can be set on a per-question/group basis, as well as for a survey as a whole. For example, on a survey level you will find 'Auto-adjust on mobile' and 'Labels on mobile' on the *Display* tab:



Another thing to note is that NubiS' pre-defined templates are intended to cover some frequently used types of display, but are not intended to be exhaustive. For this reason, it also is possible to define custom templates. If Custom is selected as the template for *TShow*, appearing is an input box in which the template can be defined:



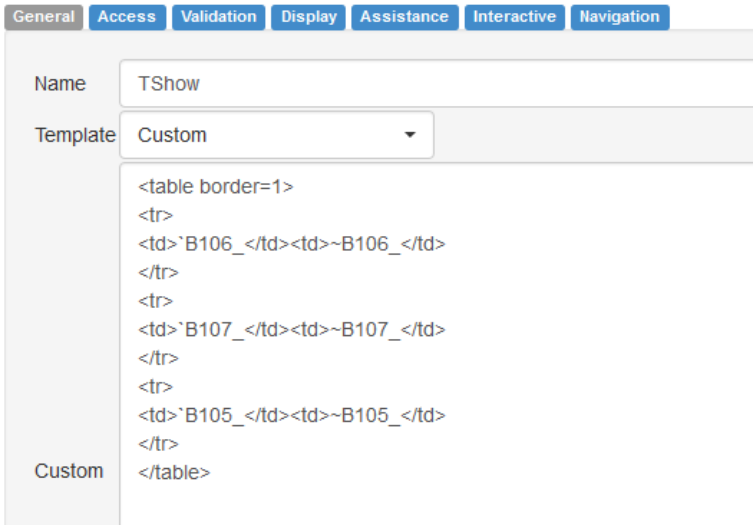
This is a simple template created using NubiS' placeholders. **#TEXT#** placeholders represent question text, whereas **#INPUT#** represents input boxes. In the above, NubiS is instructed to take the three questions and put them in an HTML table with a border:

Did you have diabetes growing up?	◉ Yes ○ No
Did you have asthma growing up?	○ Yes ◉ No
Did you have a respiratory illness such as bronchitis growing up?	◉ Yes ○ No

Note that the order in which NubiS inserts the questions is directly related to the order in which they are encountered while processing the routing instructions. That is, supposing the random order of the questions was *B106_*, *B107_* and *B105_*, then *B106_* would correspond to *#TEXT1* and *#INPUT1#*.

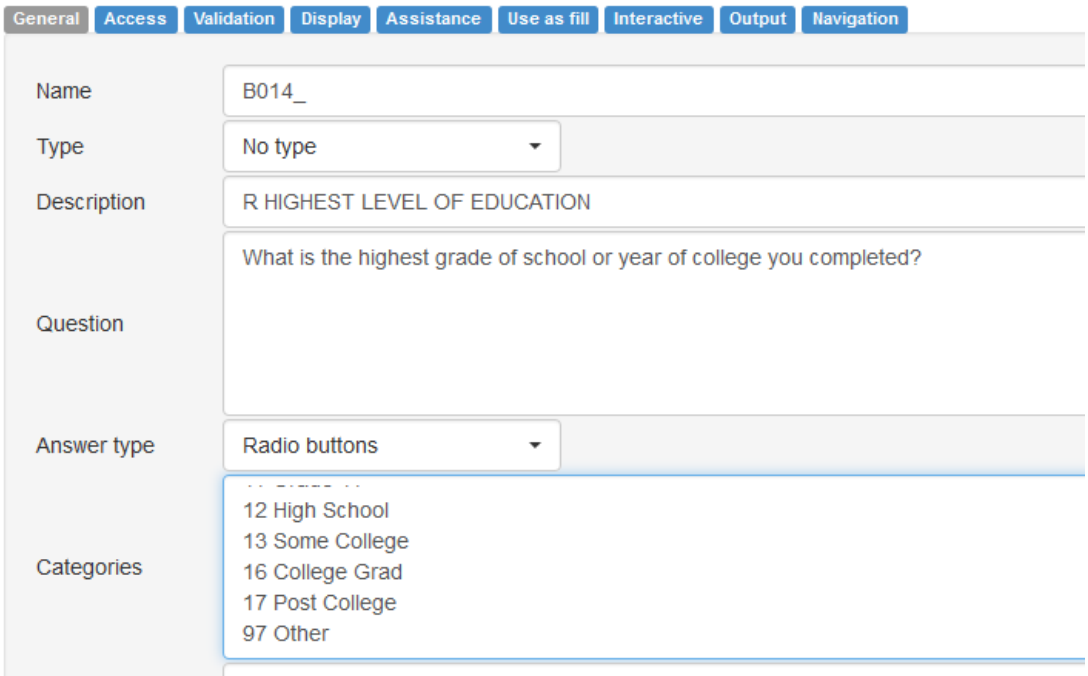
This linkage between question order and display may not always be desirable; perhaps the display of a certain question should be fixed on screen. For this reason, NubiS also supports an alternative manner for creating custom templates using **variable references**. These are different from variable value references in that they instruct NubiS to insert a part of the variable (question text or input

box) rather than its value. Their notation is shown in the custom template below, which mandates the order of *B106_*, *B107_* and *B105_*:

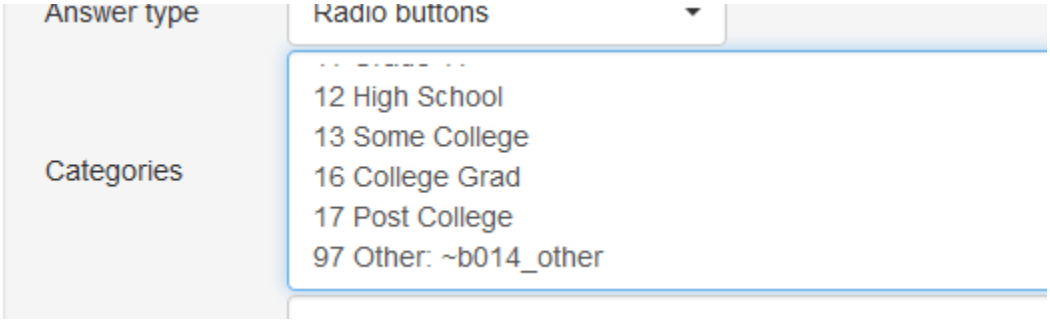


Here **`B106_** instructs NubiS to insert the question text of *B106_*, whereas **~B106_** tells it to insert the input box (here, radio buttons).

The usage of variable references is not limited to custom templates. You also can use them in question texts or answer options. To illustrate the latter, add a variable to *secB* to ask the respondent about their highest level of education:



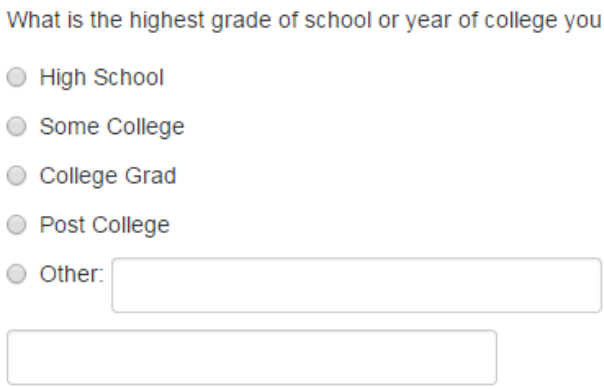
Note **Other** as a possible answer. Deviating from HRS for a moment, suppose you want to give the respondent the opportunity to specify something for *Other*. To do so, there are three steps. The first is to add a string question **B014_other**. The second is to revise the answer categories for **B014_** to include a variable reference:



And the third is to write the routing that will instruct NubiS to put *B014_* and *B014_other* on the same screen:

```
22 group.Toverall
23 B014_
24 B014_other.INLINE
25 endgroup
26
```

Now, NubiS will create a normal GROUP statement in which *TOverall* is a group and following the default template showing each question below one another. However, a special instruction is included: **.INLINE**, after *B014_other*. NubiS' default assumption is that any variable listed in a GROUP statement must be displayed on the screen no matter whether it was referenced anywhere through a variable reference. As such, leaving out the *.INLINE* would result in:



That's one text box too many. But *.INLINE* avoids this by telling NubiS it should not display *B014_other*:

What is the highest grade of school or year of college you completed?

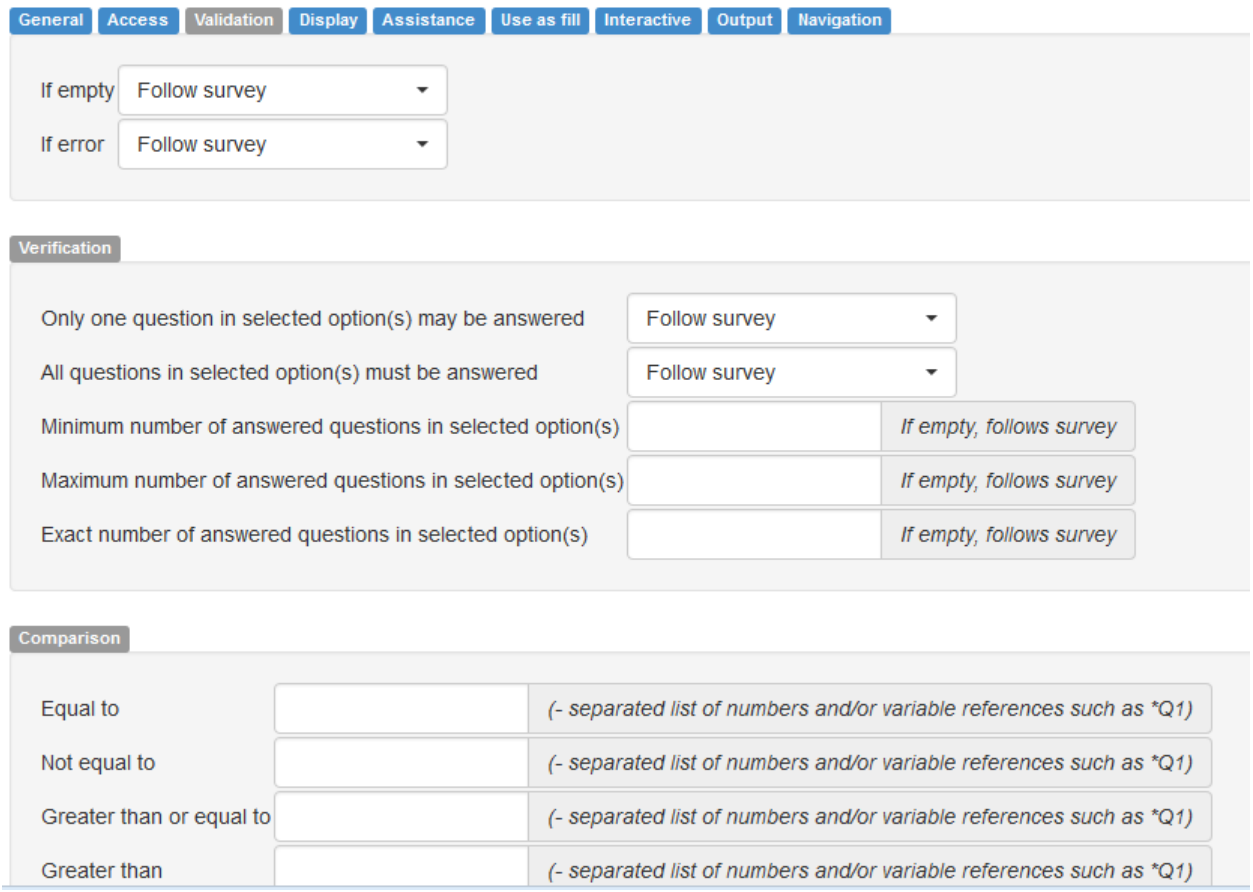
- ○ High School
- ○ Some College
- ○ College Grad
- ○ Post College
- ○ Other:

15. VALIDATING RESPONDENT ANSWERS

Now that you know variables, you can start asking all the questions you want. But how can you ensure the answers received actually are useful? Of course, to a large extent this is dependent on the respondents, and as such beyond your direct control (besides non-technical solutions such as incentives). Still, NubiS carries several mechanisms to help.

These tools fall mainly into two categories: validation mechanisms and input prevention mechanisms. Validation mechanisms within NubiS provide the ability to define what an invalid answer is and check the respondent's answer against that definition. This feature is accompanied by the capability to specify which assistive error message(s) should be displayed. Access to these mechanisms is provided via the tabs *Validation* and *Assistance* of a variable, type or group. Variables and types share the same validation and assistance settings, whereas groups complement those with their own.

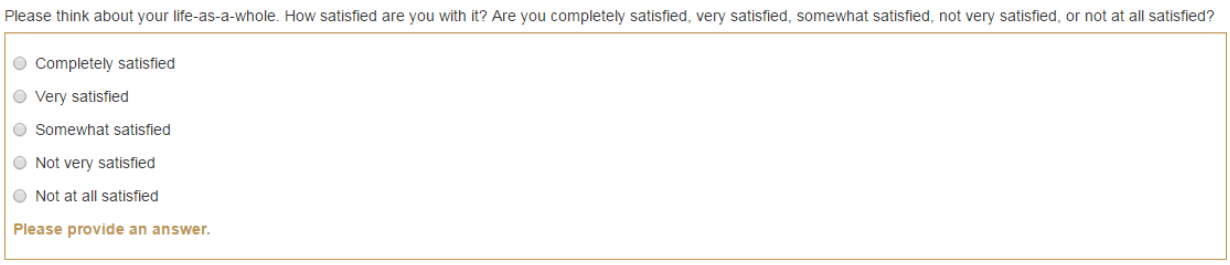
First, open *B000_* and go to the *Validation* tab:



Here you'll see a number of validation-related settings. Some settings depend on the answer type (for example, because *B000_* is not a range variable there is no need for minimum and maximum input boxes). Two settings that always will be present are **If empty** and **If error**, which instruct NubiS on what to do if a respondent, respectively, does not provide an answer or provides an erroneous answer. (Note that NubiS views not giving an answer as distinct from giving an incorrect answer.) Each setting has the same three options:

1. Allow to continue: NubiS will not check
2. Don't allow to continue: NubiS will check and, if a problem is found, force the respondent to correct it.
3. Display one-time warning: NubiS will check and show a warning, but if the respondent then clicks *Next* again, they are allowed to continue.

The default survey level values are **Display one-time warning** for *If empty* and **Don't allow to continue** for *If error*. For the example *B000_*, if no answer is provided NubiS produces:



If the respondent clicks *Next* again, though, the survey will continue. But if the value is changed to *Don't allow to continue*, NubiS will keep showing the same message.

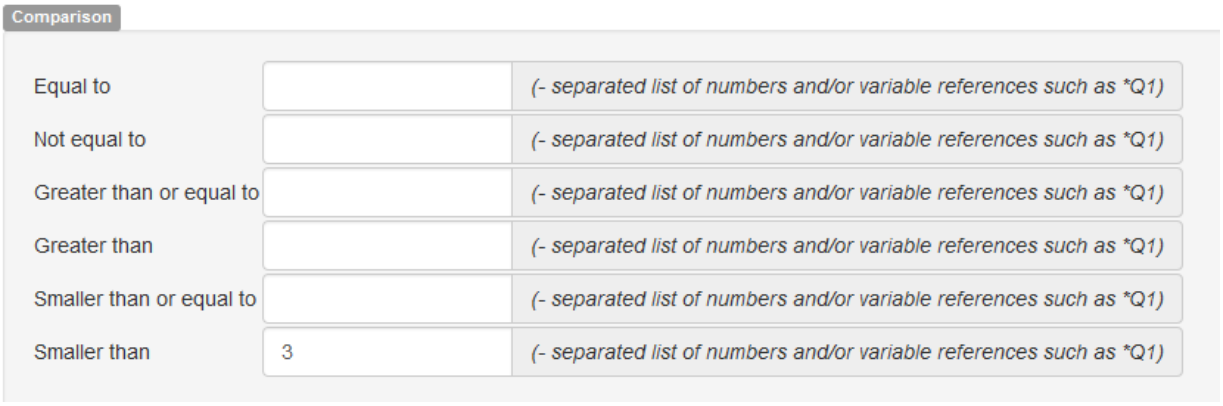
The differences between the two are referred to as a soft check versus hard check. Soft checks are employed more often, because preventing the respondent to continue can lead to frustration and, subsequently, survey break-offs (where the respondent quits the survey altogether). On occasions, though, a hard check is appropriate, such as asking the respondent for their consent to participate in the survey. Another instance would involve a question whose answer is crucial for the skip patterns defined in the routing afterward.

Erroneous answers typically are addressed with hard checks. That is, the respondent can't continue until a correct answer is given. Again, a balance must be kept between obtaining qualitative good data and avoiding excessive respondent frustration.

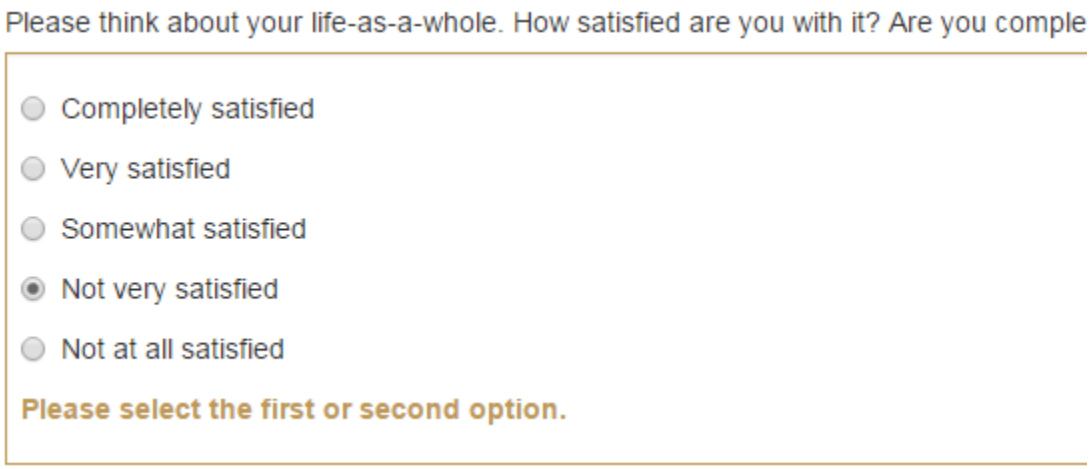
The definition of what constitutes an erroneous answer is to some extent built into NubiS, but to a large extent it must be specified. Automatic validation behavior occurs for the following answer types:

- Integer: NubiS checks if the provided answer is an integer.
- Real: NubiS checks if the provided answer is a real number.
- Range: NubiS checks if the provided answer is within a range even if no user-defined range has been specified.

Beyond the above checks, any additional error checking must be user-defined. To illustrate, define a validation criteria for *B000_* :



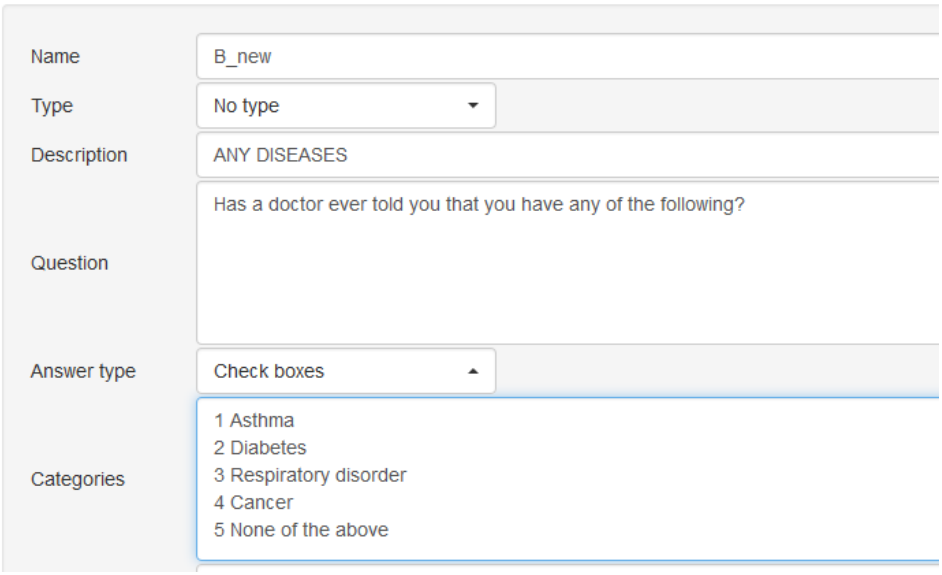
This conveys that the answer to *B000_* must be smaller than 3. (Specifying multiple values or referencing the value of other variable(s), such as by stating 2#3 also could have achieved this). Looking at this question in the survey, the respondent can still select all options – but if the third, fourth or fifth option is picked, an assistive error message comes up (which we can set in the Assistance tab to e.g. "Please select the first or second option"):



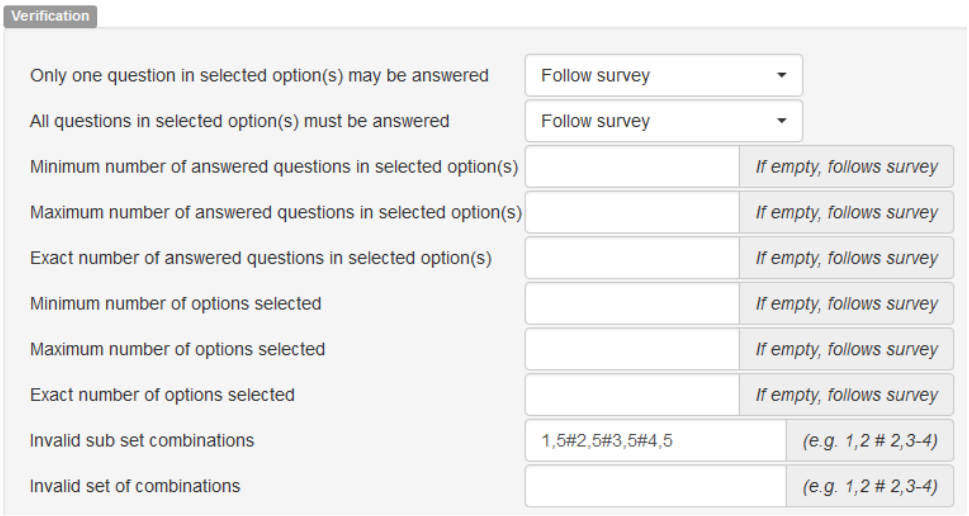
In other words, the respondent can only select *Completely satisfied* or *Very satisfied* to continue to the next screen. Numerical comparisons such as these are available for integer, real, range,

radio button and drop down variables. String and open variables, as well as date/time variables, have their own comparative variants.

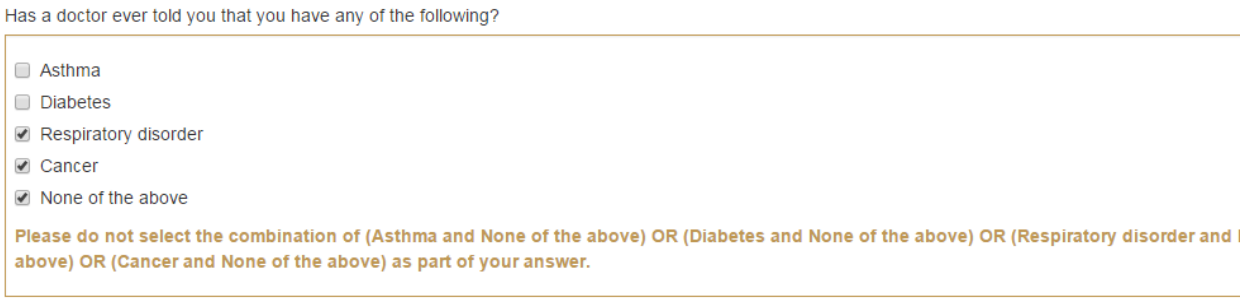
Another form of checks pertains to set of enumerated and multi-dropdown variables. An often-occurring pattern with such questions is that they ask the respondent to select one or more applicable options, or select None of the above. These obviously are mutually exclusive, and so this should be enforced in a survey to prevent ambiguous answers in which a respondent selected some options but also said none apply. To show how this can be accomplished in NubiS, first add a set of enumerated variable:



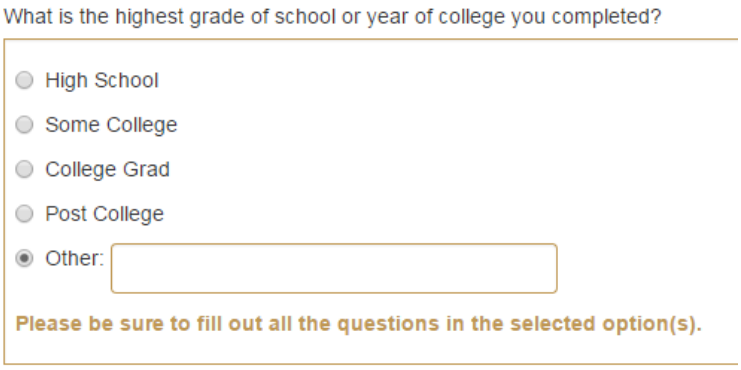
Then go to the *Validation* tab and enter:



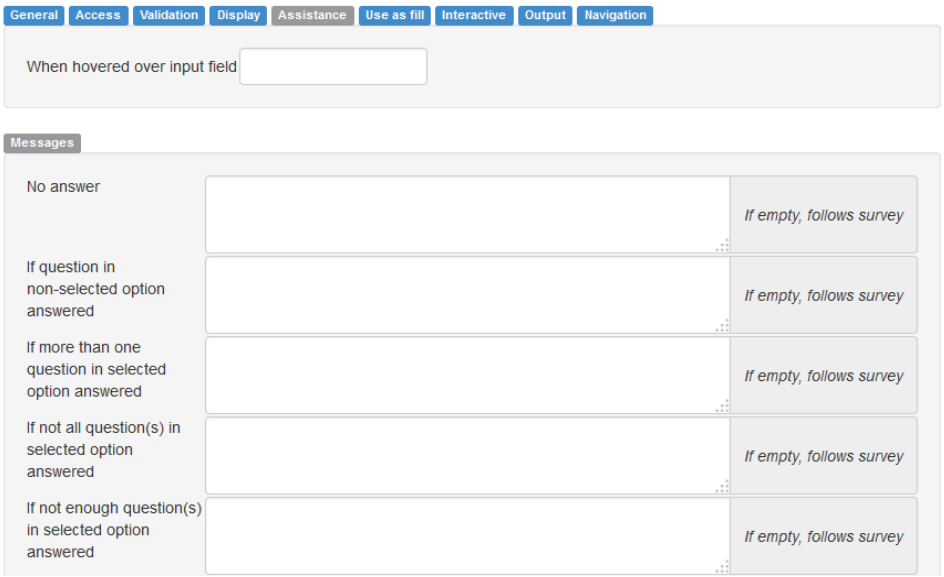
In the survey, NubiS uses this to detect invalid combinations (assuming that B_new was placed on the routing somewhere, such as in *secB*):



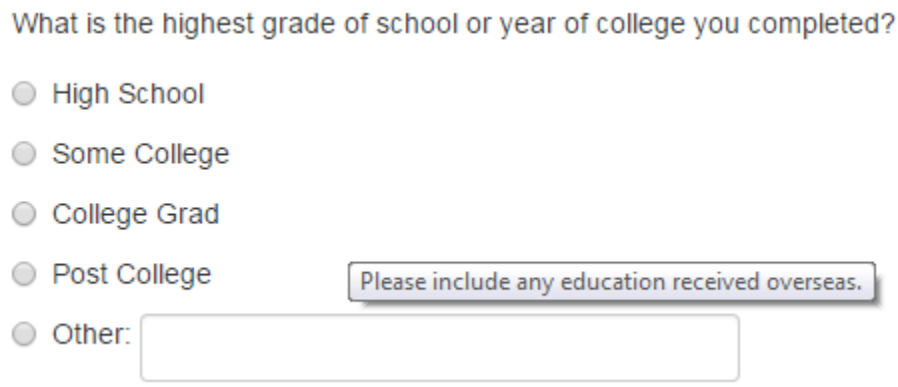
Some error checks are by default automatically enabled. For example, for *B014_* if the *Other* option is selected but nothing is specified, the following error results:



Of course, detecting an error is only half of what is needed. The second part is to help the respondent resolve the error. For this, useful error messages are indispensable, and such messages can be set on the *Assistance* tab:



One way to provide information is to have text shown if the respondent hovers over the input box. For example, as extra information it could be stated, **"Please include any education received overseas."**

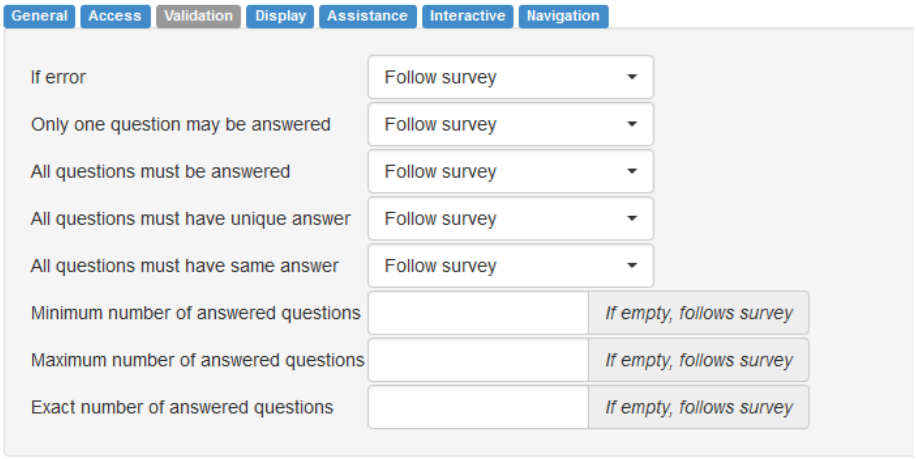


Another option is to have text appear before or after the input box (only applicable for certain answer types).

In addition to providing information up front, NubiS also can define messages to be shown when an error is detected. The screenshots above only show a few of the error messages that can be set. Each error that can be used is accompanied by a customizable message. These messages take default values as specified in the corresponding settings on the survey level.

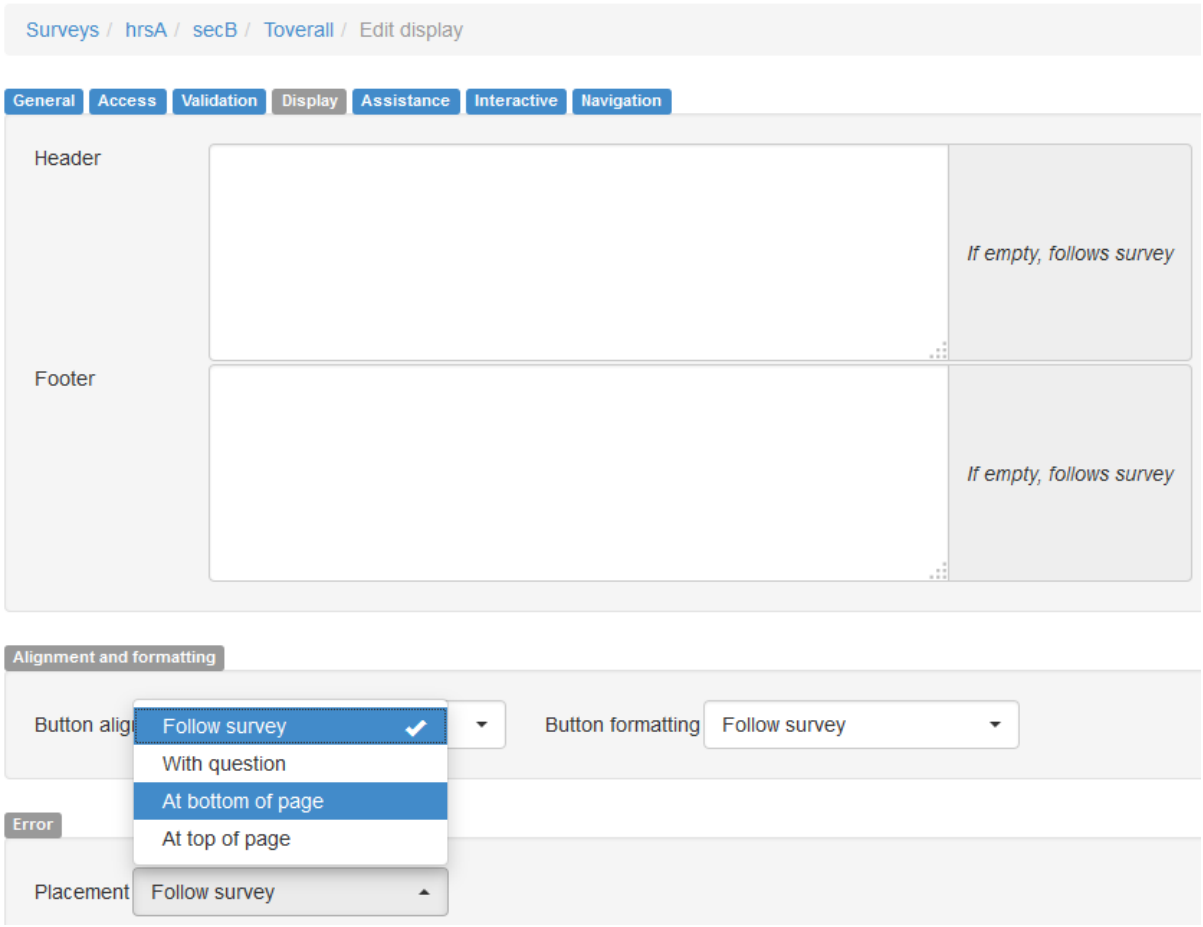
Also, it is possible to refer to variables in most validation and assistance settings. For example, it could be indicated that the value for *B000_* must be lower than that of of another variable.

Beyond validation settings for variables and types, groups come with their own settings. These operate on the group as a whole, for example to indicate that at most two questions in the group may be answered. Settings such as these can be found on a group's *Validation* tab:



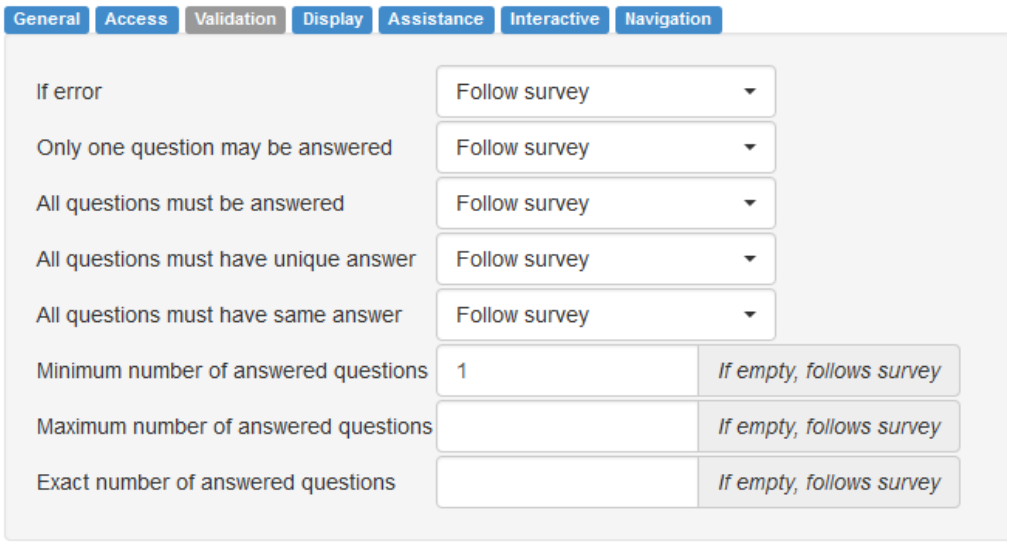
Suppose, for example, you want all questions to have the same answer. After the survey is programmed as such, a respondent trying to enter different answers on this screen gets:

Note that the error message appears as if linked only to the first question, whereas it actually refers to all questions. To alter this, the placement of the error messages can be changed in the *Display* tab of the main group used (*TOverall*):

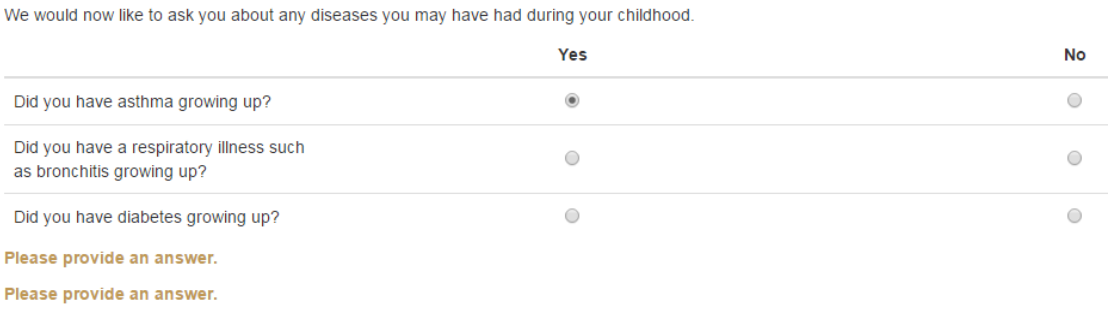


If error placement is set to At bottom of page, the result is this instead:

One aspect of group validation settings is their interaction with variable validation settings. For the most part, they complement one another – but in some cases, they can be contradictory. Suppose the validation settings for *TShow* are changed to:



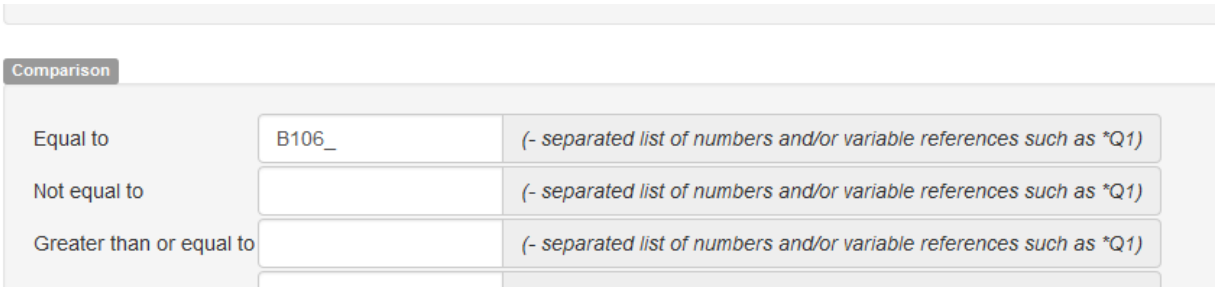
This instructs NubiS to ensure one question at minimum is answered. Now, if the respondent is on the screen displaying your three questions, the following might appear:



Why is NubiS showing error messages when the respondent met the requirement of answering at least one question? The reason is that although the group validation criteria are met, the individual question criteria are such that they instruct NubiS to detect any empty answers. In order to change

this, you must set the individual variables' *If empty* settings to Allow to continue. For this reason, typically when group validation settings are used to mandate the number of questions to be answered, *If empty* of the variables involved should be set to *Allow to continue*.

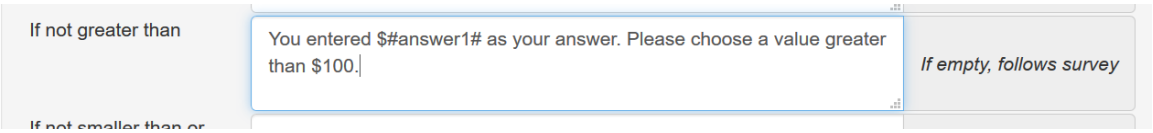
An interesting scenario is when there are multiple questions shown on the same screen and you want to ask NubiS to enforce validity between the answers. To illustrate, first clear out any group level validation criteria specified for *TShow*. Then go to the *Validation* tab of *B105_*:



Here it is stipulated that whatever answer is chosen for *B105_*, it must be the same one as for *B106_*. If this is not followed:

Obviously, the NubiS message here is not very informative, and so you would want to enter a custom message on the *Assistance* tab of *B105_*. But it showcases that you can leverage these criteria to get NubiS to enforce dependencies among questions on the same screen (in addition to referring to earlier values using variable value references of the form *B106_).

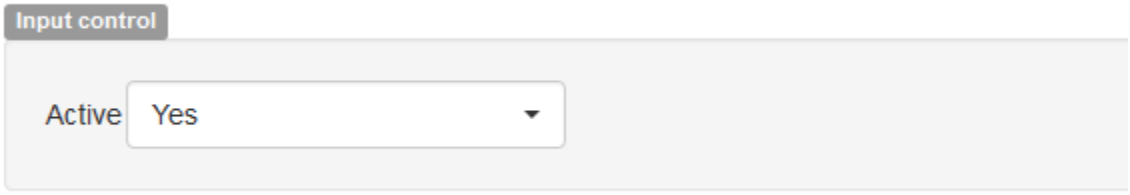
In the reactive style of error checking it may be confusing to respondents that they can't enter certain characters, and so it can be useful to add explanatory instructions on how certain questions must be answered. One important aspect of this is including the answer given by the respondent in the error message. In order to facilitate this NubiS uses placeholders similar to those within custom group templates. These placeholders take the form of "#answer#" and "#answer_value#", for example:



The **#ANSWER#** placeholders instruct NubiS to insert the answer for the corresponding question as given by the respondent in a pre-processed manner. Numerical answers are inserted as-is (integer, double, range, slider, knob). The same goes for string, open, and date/time picker answer types. Radio button and dropdown answers are displayed with their associated value label. Checkbox and multiple select dropdown answers are inserted as comma separated lists of any selected options.

The **#ANSWER_VALUE#** placeholders do the same with the difference that NubiS inserts the raw answer. Just as for the placeholders in custom group templates, the numeric identifiers at the end of the above placeholders correspond to the ordering of the questions within a group statement. So if "q1" is listed second in a group statement, then any references to its answer in an error message would use '2' as its numeric identifier (i.e. #ANSWER2# and #ANSWER_VALUE2#).

Now, the validation mechanisms discussed so far are all reactive in nature. That is, when a respondent enters an answer and clicks *Next*, NubiS verifies the answer's validity. But an alternative method available – the input-prevention mechanism – disallows incorrect answers by providing real-time input control. The nature thereof is dependent on the answer type. To demonstrate, go to the *Validation* tab of *B_new* and enable real-time input control:



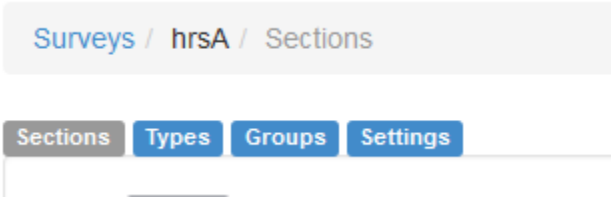
Now in the survey, a respondent will notice that upon selecting *None of the above*, any other selected options will be deselected automatically. In a similar style, if you activate real-time input control for your integer variable *numberofchildren*, a respondent would find that only numbers can be entered and any other character is not accepted.

Input control in this fashion is quite powerful, but a caveat should be made on its proper functioning on some platforms. Particularly, Android devices running older versions of Chrome or Firefox will prevent a respondent from properly entering characters when it comes to integer and

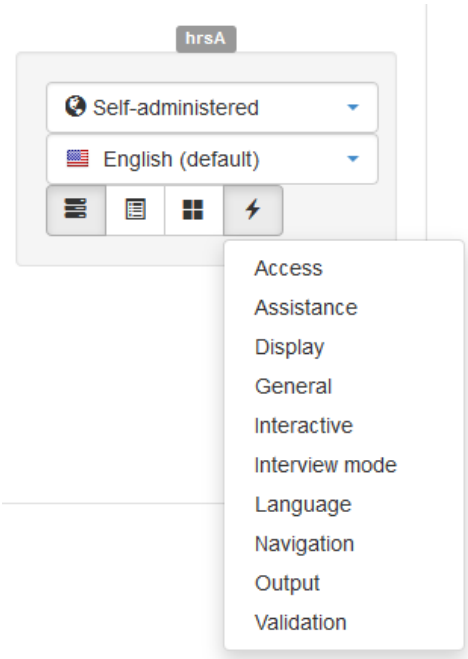
real answer types. The same goes for browsers on Kindle devices. This is not to diminish the power of input control, though. For example, in circumstances in which the device and browser are known (such as face-to-face interviewing), real-time input control can be leveraged without concern.

16. ADDITIONAL INTERVIEW MODES AND/OR LANGUAGES

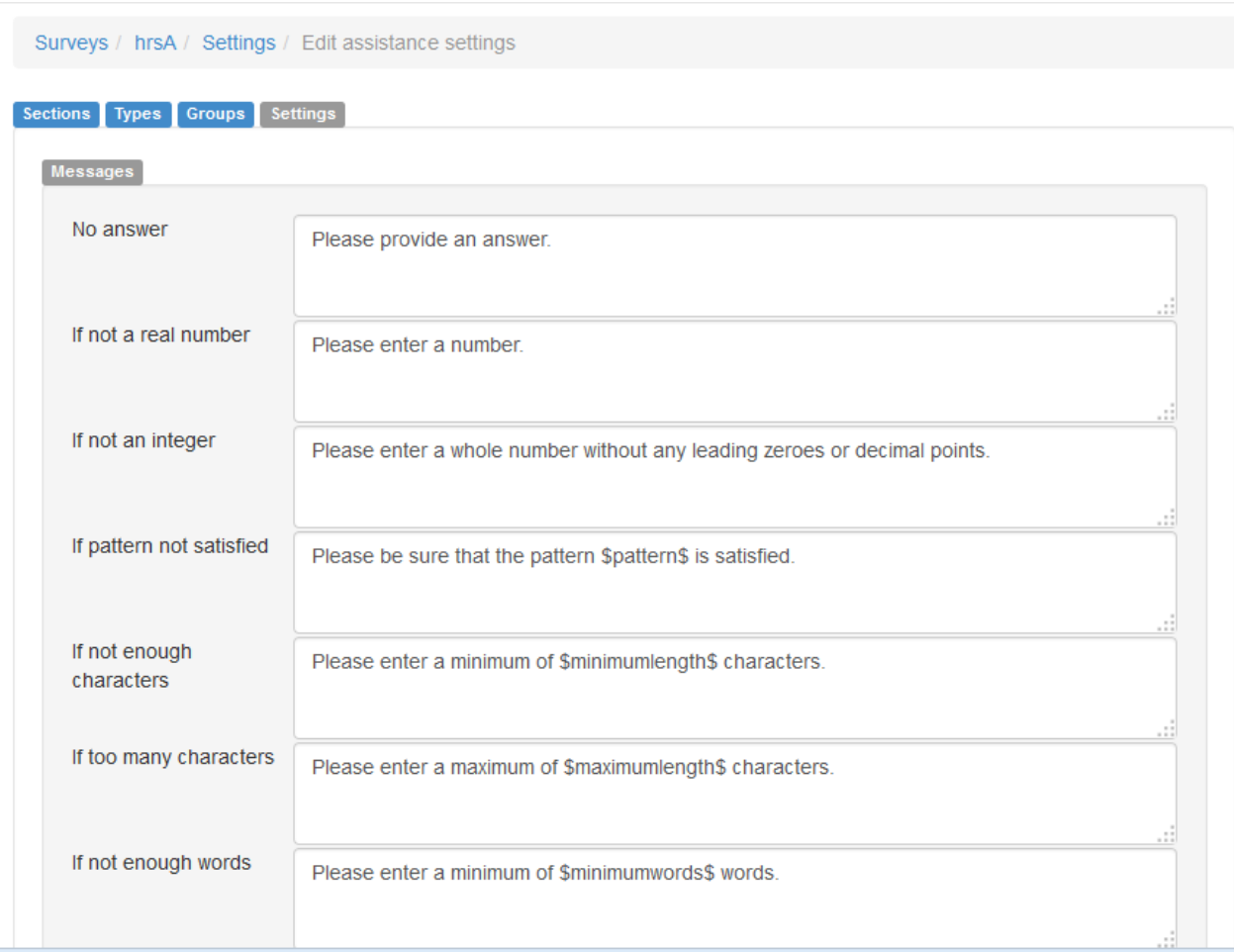
While you busily were adding sections, variables, types and groups, and tying those together in the routing, it was mentioned on several occasions that many of the encountered settings can also be set on a survey level. These survey-level settings then permeate downwards, where they may be overridden, if needed, on a case-by-case basis. These settings can be accessed via the *Settings* tab in the survey overview screen:



They also can be found in the *Settings* dropdown in the right-side menu:

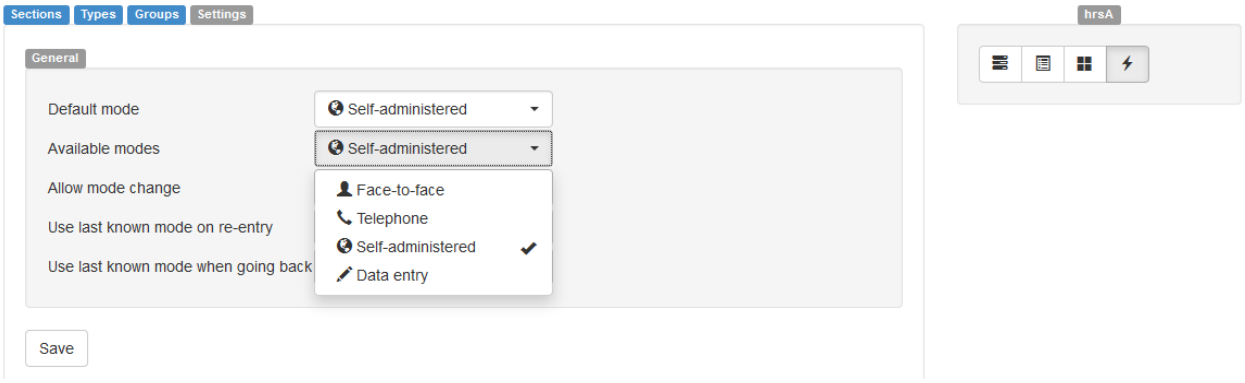


These are same types of setting, for most of which you already have seen examples. Each corresponds to a single screen, listing all settings available across all answer types in NubiS. For example, in the *Assistance* screen, there is a long list of default error messages, such as the *No answer* message.



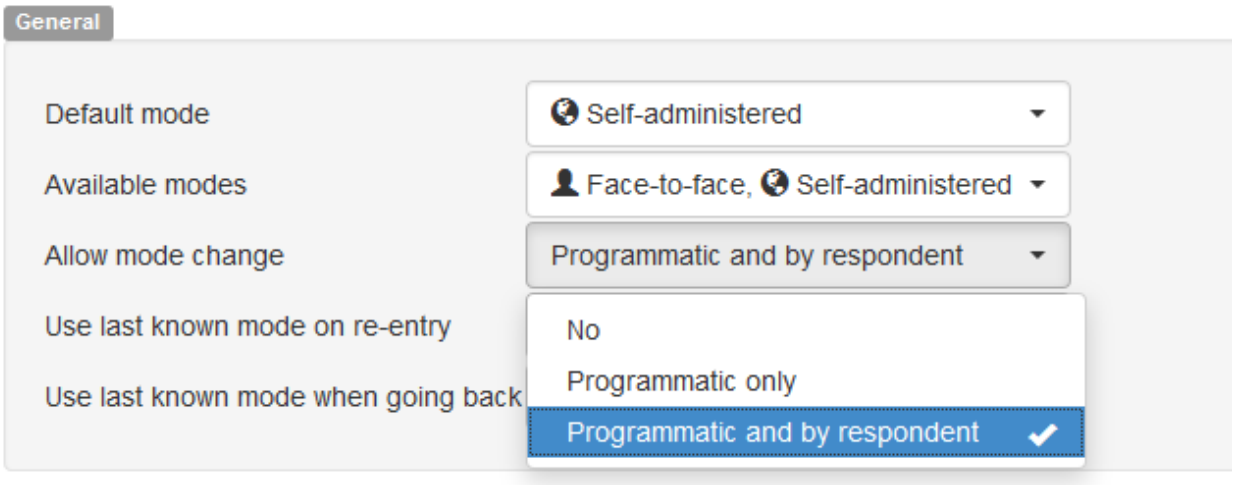
Most of the messages are straightforward enough, but some contain cryptic fragments such as $minimumlength$. These are placeholders in which NubiS inserts whatever was specified in a corresponding setting. For example, if it had been specified at least 300 characters must be used for a String question, then if fewer characters were entered this number would appear in the error message.

Some forms of setting that have not yet been discussed yet are access, interactive, navigation, and output settings. These will be addressed later. Instead, you now will focus the settings for interview mode and language. These settings control the manner in which NubiS handles multiple interview modes and/or interview languages. The overall dynamic between the two is that each interview mode can have more than one language. That is, languages are specified per mode instead of modes per language. This becomes evident upon opening the interview mode settings screen, because on the right side there are no options to switch between modes or languages.



There are several mode settings available. First and foremost, the default mode can be pre-set so NubiS adopts it to be used in the absence of any explicit instruction to start the survey in another mode. Similarly, if a setting (like a question text) was not explicitly specified for an interview mode that is not the default, then NubiS will use the question text that has been defined in the default mode.

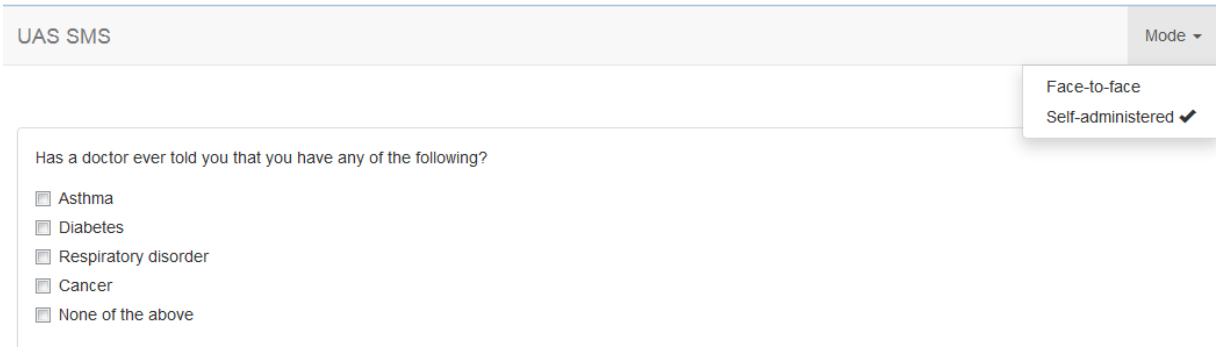
Upon the initial setup, NubiS has **Self-administered** as the default mode (it's also the only active mode). To activate more modes, toggle **Available modes**, which then allows you to choose another default mode. You also can indicate whether NubiS is allowed to change interview mode. This can be set to the following values:



1. **No**: The interview mode cannot be changed and the survey always will start in the default mode.
2. **Programmatic only**: NubiS can be instructed to start the survey in a particular mode, but the respondent cannot change the interview mode within the survey.
3. **Programmatic and by respondent**: NubiS can be instructed to start the survey in a particular mode and the respondent can change the interview mode within the survey.

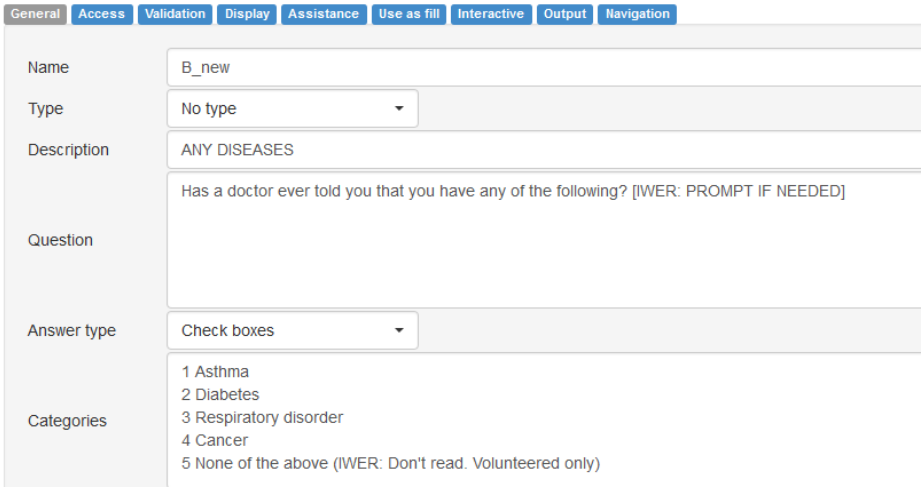
Related, the last two settings concern changing modes within a started survey. For example, you may wish to have the respondent start your survey in a self-administered mode and complete it over the phone. But upon re-entry in the survey, should NubiS use the last known mode (self-administered) or not? Similarly, if a respondent changed the interview mode, as allowed, then clicked *Back*, should NubiS act like any previous questions also were answered in the new mode?

For now, activate the **Face-to-face** interview mode while also setting *Allow mode change* to *Programmatic and by respondent*. Assuming that *secB* still is the section on the *Base* routing and *B_new* is the first question in the *secB* routing, you should see this:

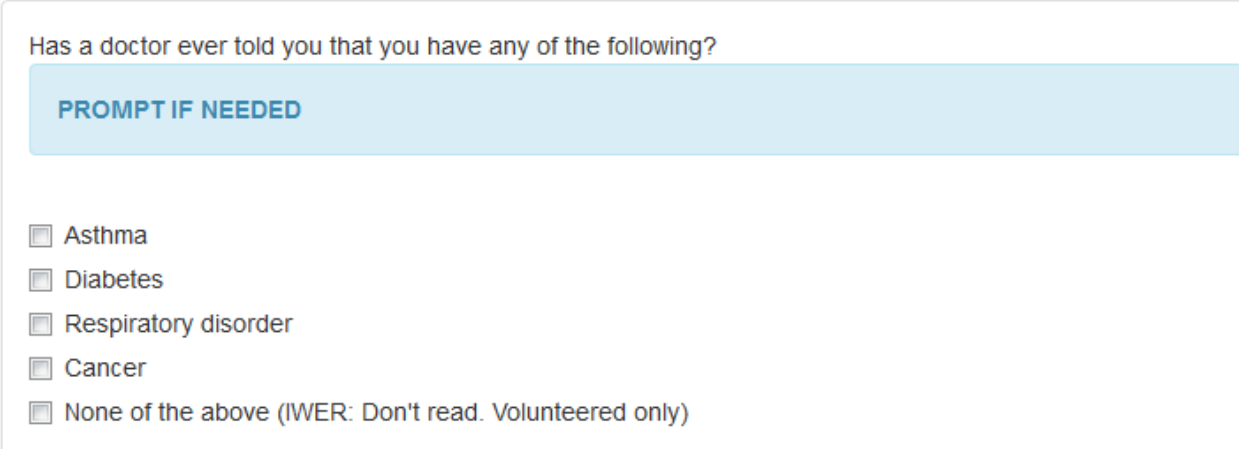


Now you have a dropdown to toggle the interview mode. Select the first answer option, *Asthma* and switch to *Face-to-face*. What happens is, nothing happens – which makes sense because NubiS was not told that anything of this question should be different between the two modes. That means NubiS fell back on the default mode to find out what text to use. For a more exciting result, exit test mode and go to *B_new*:

Switch to *Face-to-face* and, in the resulting screen, change the question text and the answer categories:



If you now switch modes in the survey, you get:

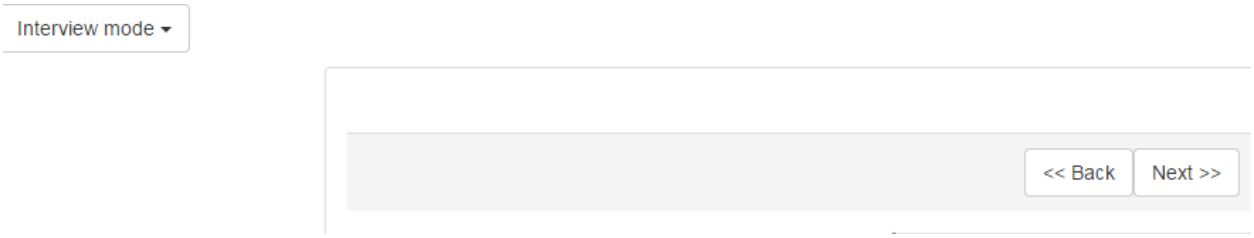


Here is the text specified for the face-to-face mode for *B_new*. (The blue box is created by NubiS, which adds styling to **[IWER:]** statements. You'll learn more of this in Sec. 18 when discussing advanced usage scenarios for NubiS.)

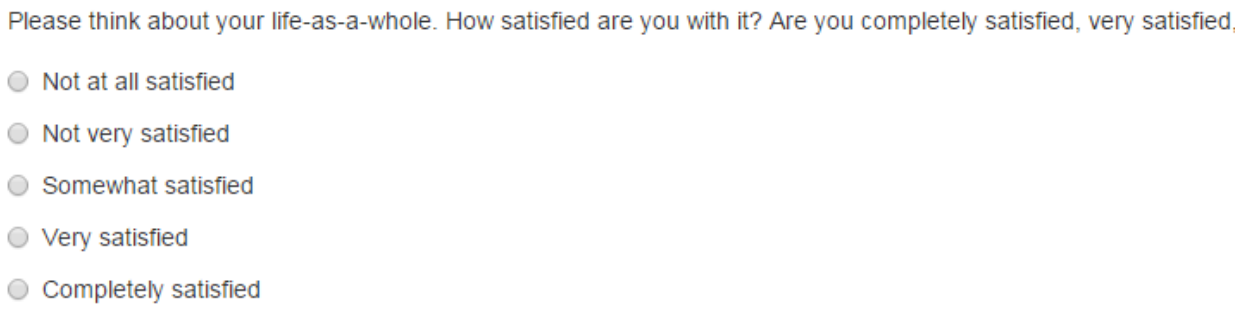
You prepare a survey for one or more additional interview modes mostly by adding settings for defined survey components (such as variables) that override the default mode's specification(s). If there is no need to change anything from the default mode, then it is not needed to specify anything in additional modes. This is due to NubiS' lookup strategy when handling multiple modes, which is that if it can't find anything in the mode of the survey then it falls back to the default mode. That explains why when you initially switched interview mode, you still saw the question and answer options.

All of this brings up a related point, which is that **changing the default mode is best done at the beginning**, before any survey components have been added. Otherwise, if you start in a default

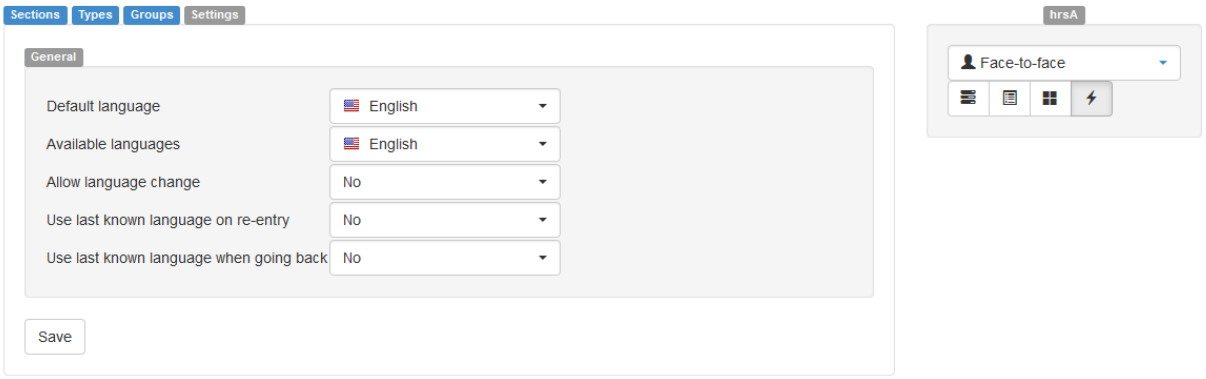
mode (*e.g.*, self-administered) and add variables, yet later change the default mode (*e.g.*, to face-to-face), NubiS also will look in this more recent default mode for matters such as question text, answer option labels or array indicators. Often, this will lead to NubiS finding nothing because everything already was defined for the old survey mode. This can be illustrated with a temporary change of the default mode to *Face-to-face*. Upon a test run of your survey the first question, *B_new*, appears fine because a new text and and answer options were defined. But if *Next* is clicked:



Switching back to *Self-administered*, the correct screen now should appear:



You'll learn later how NubiS can start a survey in a mode different from the default mode. For now, change the default mode back to *Self-administered* and turn your attention to the language settings:



The language settings are virtually the same as the aforemention mode settings, the only difference being the option to set them per interview mode (which can be switched between modes using the dropdown in the right-side menu). NubiS comes with a standard list of available language names; if a language is not listed, it can be added. NubiS does not impose limitations on the language used, be it character-based (*e.g.*, Chinese or Japanese) or not (such as English or Spanish).

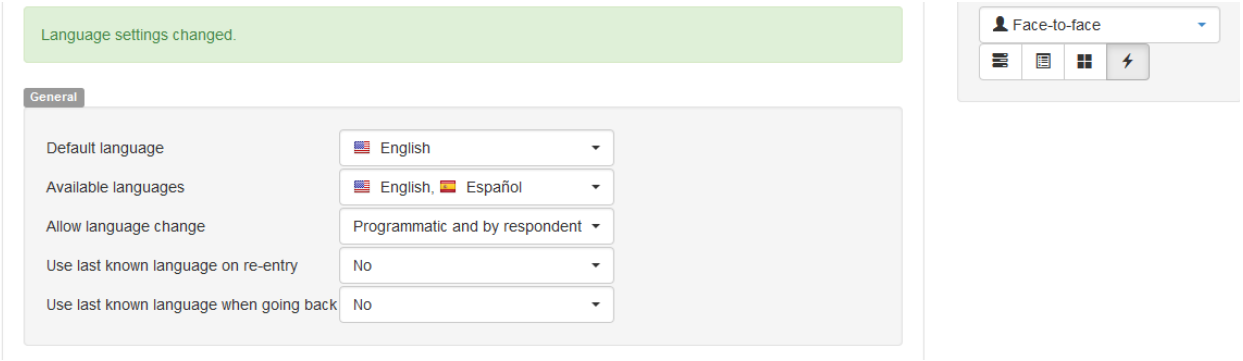
As with interview modes, all characteristics discussed for variables, sections, types, and groups also can be configured per language. So if you added Spanish as a second survey language, the supported combinations are:

- English/self-administered
- English/face-to-face
- Spanish/self-administered
- Spanish/face-to-face

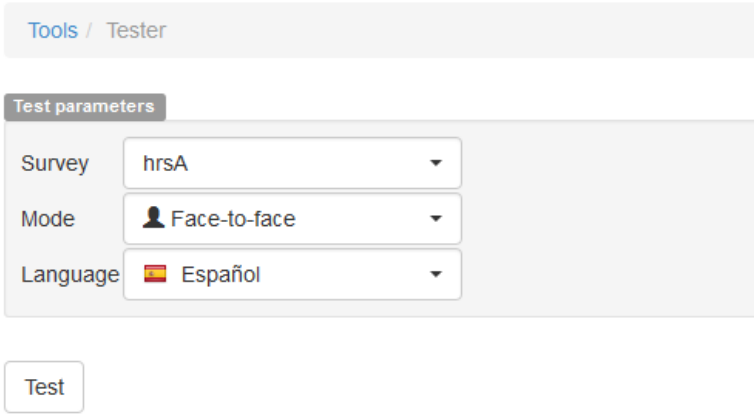
Also, as with interview modes, NubiS will fall back on the default language if nothing has been specified in the language used for the survey. When combined with multiple interview modes, the order is as follows:

1) NubiS checks the current interview mode and current language of the survey; but,
2) If nothing is found, NubiS checks the current mode with the default language of the survey; but
3) If nothing is found, NubiS checks the default mode with the current language of the survey; but,
4) If nothing is found, NubiS checks the default mode with the default language of the survey.

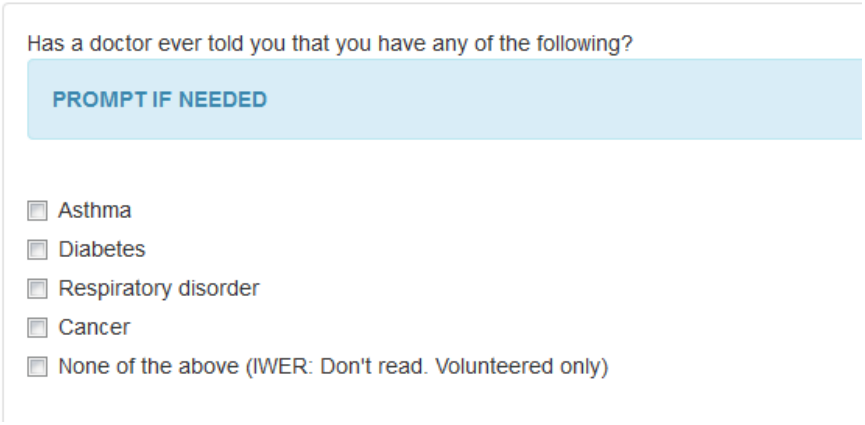
To illustrate all this, add Spanish as another language:



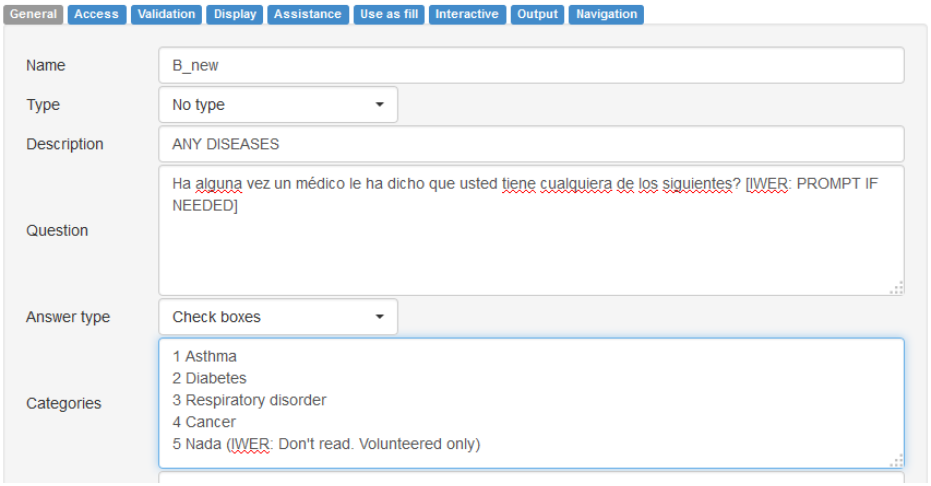
Now, open the tester:



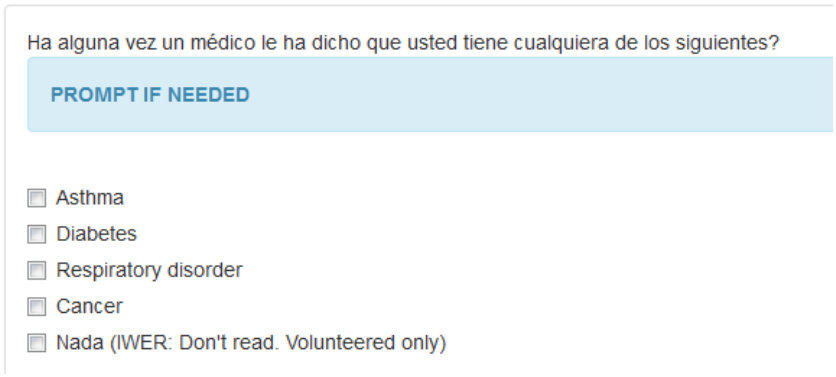
And start the survey in Spanish for face-to-face interviewing. The screen you should get is:



This is expected because you did not yet define a translation. If you do so, you would get the Spanish equivalent:



Which then shows as:



Note that if you were to switch now to *Self-administered*, you would get the English self-administered equivalent again, because NubiS simply ignores that you were in Spanish at the moment of switching, as no translation in this language exists for the question in the *Self-administered* mode.

Of course, it may not always be desirable to have the person programming the survey also do the translation. For this purpose, NubiS allows the creation of translator accounts. These accounts can be configured such that a translator can enter translations for the selected language(s) – and only translations. You will learn in Sec. 21, User Management, how to add such accounts. (For detailed information on NubiS' translator mode, please refer to the **NubiS Translator Manual**.)

So far, each time you wanted to switch language or interview mode you used the dropdowns available. However, these typically are present only in test mode. But what if you want to give the respondent the ability to choose? Doing so is relatively straightforward, because *language* and *mode* are variables just like any other, and as such they can be included in the routing. For example, on survey start the first question you might ask would be:

language

// the other questions

The respondent then can choose a language, and upon clicking *Next* the survey will continue in that language. Note that an underlying assumption here is that **Allow language change** has been set to **Programmatic and by respondent**. If not, then NubiS simply will ignore the choice the respondent makes. A similar prerequisite applies to asking *mode*.

Another manner in which the *language* and *mode* variables can be employed is in skip patterns. To illustrate, you might wish to ask only Spanish speakers whether English is their second language.

This can be easily accomplished. Assume here that *2* represents Spanish and specify the routing as:

if language = 2 then

//ask Spanish speakers only

endif

17. ADVANCED FEATURES

In addition to the functionality discussed so far, there are several other options worthy of learning even though their usage may be less frequent.

17.1 Nested Loops and Variables of Type Section

In the discussion of how to instruct NubiS to ask questions, introduced was the notion of loops to ask the same question multiple times. In that context, the simplifying assumption was made that only one loop was needed (*e.g.*, to ask the name of each child). Although this covers most cases, there are circumstances where this would not be sufficient. For example, suppose you wished to ask about the children of multiple families, such as those living in a neighborhood. One option would be to introduce separate variables for each family, but this would effectively be the same as defining separate variables for each child. A better option would be to introduce a second loop whose routing would look like this:

```
for outercount := 1 to 5 do
for innercount := 1 to 50 do
    A208ANewPerson[outercount,innercount]
    if A208ANewPerson[outercount,innercount] = YES then
    ChildX060ASex[outercount,innercount]

    GROUP.TBirthDate
    ChildX004AmoBorn[outercount,innercount]
    ChildX005AdaBorn[outercount,innercount]
    ChildX067AYrBorn[outercount,innercount]
    ENDGROUP

X060ASex[outercount,innercount] := ChildX060ASex[outercount,innercount]
X004AmoBorn[outercount,innercount] := ChildX004AmoBorn[outercount,innercount]
X005AdaBorn[outercount,innercount] := ChildX005AdaBorn[outercount,innercount]
```

X067AYrBorn[outercount,innercount] := ChildX067AYrBorn[outercount,innercount]

else

exitfor

endif

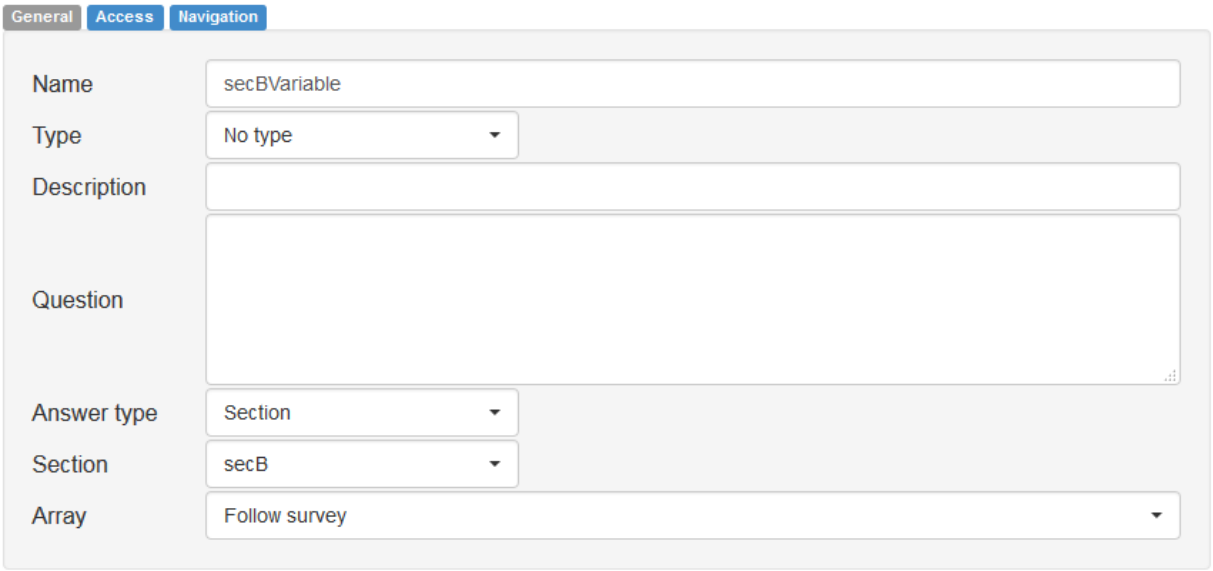
enddo

enddo

Here you see the original loop asking about children was placed it inside another loop. Then, the *[cnt]* statements were modified to track not only the original loop but also of the new outer loop. There are no limitations to the level of nesting loops, although in practice a nesting of three is typically the maximum required.

Though nested loops are useful in situations such as asking about the children of multiple families, it does require keeping careful track of the indices used in the routing (*e.g.*, ChildX004AmoBorn[outercount,innercount]). An alternative is to use variables of type section in combination with loops.

To illustrate the notion of a variable of type section, add a variable in the *Base* section of that type called secBVariable (no question text is needed):



In the *Section* dropdown that appears, choose *secB* here. Next, open the routing of Base –but instead of stating *secB*, instead state *secBVariable*:

Variables | Routing

```
//secA
//secA2
//secB
secBVariable
```

If the survey is run now, the result is the same as if *secB* was used. But what if you want to ask *secB* multiple times? One way is to add a *for* loop in the routing of *secB*, then update all references, as such:

for cnt := 1 to 5 do

B_new[cnt]

...rest of section here

enddo

Another option is to make the *secBVariable* variable an array and state the following on the routing of the *Base* section:

Surveys / hrsA / Base

Variables | Routing

```
//secA
//secA2
//secB
for cnt := 1 to 5 do
    secBVariable[cnt]
enddo
```

Once again, when the survey is run,, the result is *B_new*. However, in the background, NubiS now stores this variable as **secBVariable[1].B_new** instead of *B_new*. Consequently, this allows NubiS to distinguish between answers for the first loop and those in any of the subsequent loops. And this was done without needing to update the routing of *secB* with **[loop counter]** indices. If it were the case that *B_new* was an array variable, then its answers would be stored as **secBVariable[1].B_new[1]**.

In the above, example you may wonder about the necessity to write long fill references such as **^secBVariable[1].B_new[1]**. The answer is yes and no. If the fill reference is in a variable within the same section (*e.g.*, in variable *B000_*), then NubiS will handle prefixing the reference and you can just state **^B_new[1]**. However, if you are referencing *B_new[1]* from another section, the full reference must be included. The reason is that NubiS has no way of knowing which instance of

B_new[1] is being referenced. For example, there may be **secBVariable[1].B_new[1]** and **secBSecondVariable[1].B_new[1]**.

17.2 Using *While* Loops Instead of *For* Loops

Now you know how to use loops to ask a series of questions multiple times. This suffices for cases in which you want to do something a finite amount of times. However, there may be cases in which the maximum of the loop is less well defined. A good example is that of a conditional loop, in which there is an attempt to elicit a response while allowing the respondent the option to correct the response. To get information about a payment, you might have this:

```
FOR cnt := 1 TO 25 DO
        WHILE (correct[cnt] != 2) DO
                // ask details
        GROUP.TStandard
        payment_time[cnt]
        payment_amount[cnt]
        ENDGROUP
        // ask if details are correct
        correct[cnt]
ENDWHILE
morepayments[cnt]
IF morepayments[cnt] = 1 THEN
        EXITFOR
ENDIF
ENDDO
```

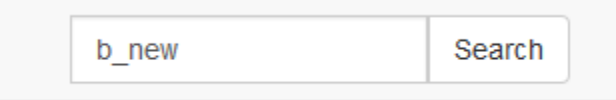
In the above snippet, NubiS cycles through a loop, between one and 25 times, asking about payments. Almost immediately upon entering the loop is the *while* construct. This instructs NubiS to do whatever is in the *while* loop until the condition of the loop is no longer met. Here, that is when the variable **correct** is no longer equal to *2*. Obviously, going in the first time, *correct* is not set yet – so the *while* loop engages and asks the first payment's time and amount. After clicking

Next, asked is the *correct* question, which here would show the answers given and ask "Is this correct?"

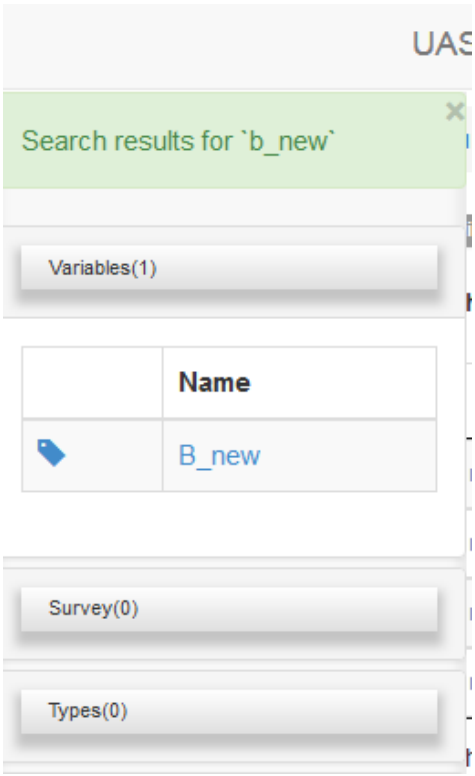
Assume the answer is negative due to a mistake in the amount. Another click of *Next* advances NubiS to the end of *while* (indicated by *ENDWHILE*) and returns it to the first *WHILE* statement. So the same payment screen is shown again, because the condition is still true (*correct* does not equal *2*). After fixing the error in the amount, click *Next* and now say "Yes, this is correct." NubiS, once again, reaches the *ENDWHILE*, and the loop cycles back to the *WHILE*. But now the condition no longer is true. So NubiS exits the *while* loop and moves on to the **morepayments** question. If there are more payments, the *while* loop engages; if not, the *for* loop is exited.

17.3 On-Screen Interactive Behavior

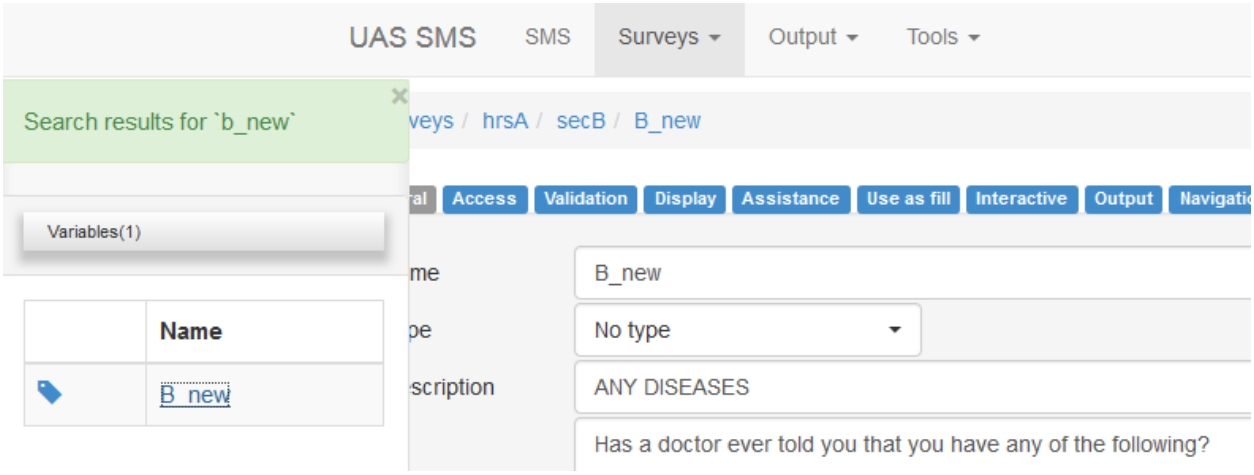
NubiS has the capability to implement, on a question screen, forms of interactive behavior. One example was mentioned earlier: In a "Check all that apply" type question, checking "None of the above" triggers the deselection of other box checked. Beyond these built-in types of on-screen interactive behavior, you can develop custom behaviors. This can be done on a per-question basis by leveraging the settings accessible through the *Interactive* tab. For an example, using the search function mentioned in Sec. 4, return to *B_new* in section *secB*:



This brings up any results with *b_new* in it (the search is case-insensitive), leading to a result displayed on the left side:



The display shows, per type of component, matches for the search term. Under **Variables (1)** is *B_new*. A direct link is provided to open this variable (as well as an option to add it to *Batch editor*):



Close the search display and open the *Interactive* tab:

The tab is divided into two parts: The upper part contains settings for implementing interactive behavior, whereas **Formatting** allows custom styling.

For programming interactive behavior, NubiS supports three methods, all using JavaScript web-scripting language statements:

1. Catering for inline statements
2. After page load scripts
3. Referencing external files

Inline statements are called as such because they are placed inside the input element by NubiS. For example, a checkbox for *B_new* has the following HTML equivalent in NubiS' output:

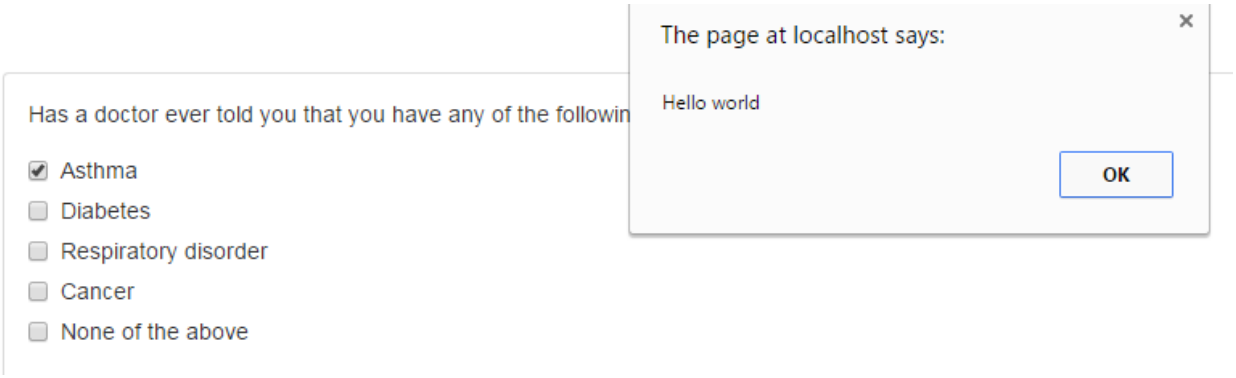
> *<input data-msg-required="Please provide an answer." data-msg-invalidsubselected="Please do not select the combination of (Asthma and None of the above) OR (Diabetes and None of the above) OR (Respiratory disorder and None of the above) OR (Cancer and None of the above) as part of your answer." type="checkbox" id="answer1_1" name="answer1_name[ ]" value="1">*

This is a typical HTML definition of a checkbox (ignoring for a moment the HTML tag attributes dealing with validation, such as **data-msg-required**). Suppose, if one of the checkboxes is clicked, you want to show a "Hello World" prompt. To do this, specify:

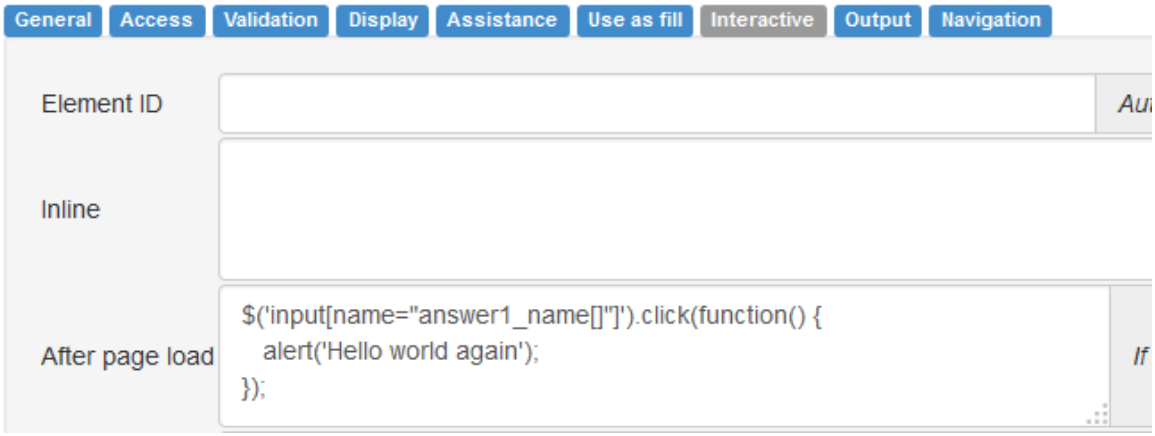
This will lead to:

> *<input onclick="alert('Hello world')" data-msg-required="Please provide an answer." data-msg-invalidsubselected="Please do not select the combination of (Asthma and None of the above) OR (Diabetes and None of the above) OR (Respiratory disorder and None of the above) OR (Cancer and None of the above) as part of your answer." type="checkbox" id="answer1_1" name="answer1_name[ ]" value="1">*

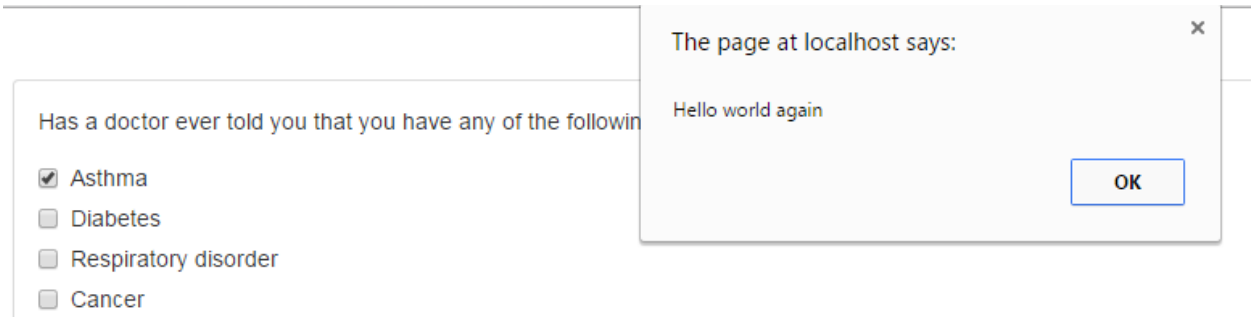
…and a prompt if a checkbox is clicked:



The second manner in which this can be accomplished is with an after page load script (which uses a more modern style of programming):



There is no need to specify script tags (such as **<script type="text/javascript>** and the corresponding **</script>**), as NubiS automatically adds those. This produces:



At the source, the following code was included:

```
<script type="text/javascript">$('input[name="answer1_name[ ]"]').click(function() {
 alert('Hello world again');
});</script>
```

Close inspection also reveals the script is placed only after the definition of the checkboxes. The reason is that when the browser interprets the above, it will attempt to link the code to the input checkboxes with the name answer1_name[ ]. If none yet exist, this link will not be established – but even with this, there is no guarantee the browser will do things in the correct order. To make sure it it does, required are jQuery on ready statements:

```
$( document ).ready(function() {
        $('input[name="answer1_name[ ]"]').click(function() {
 alert('Hello world again');
});
});
```

This will tell the browser not to attempt linking until the entire page is loaded.

Note above that the prompt will appear for any checkbox clicked. But what if you only want it for the third checkbox? This is easy to accomplish:

```
$( document ).ready(function() {
$('#answer1_3').click(function() {
 alert('Hello world from me only');
});
});
```

Here the instructions are to link to the input checkbox only with the identifier #answer1_3 and no other. But which identifier to use? To find out, look in the HTML source NubiS produces:

> *<input data-msg-required="Please provide an answer." data-msg-invalidsubselected="Please do not select the combination of (Asthma and None of the above) OR (Diabetes and None of the above) OR (Respiratory disorder and None of the above) OR (Cancer and None of the above) as part of your answer." type="checkbox"* ***id="answer1_1"*** *name="answer1_name[ ]" value="1">*

NubiS adds identifiers to any HTML input element definition it outputs. By default, these are of the form *answer,* and a number corresponding to the order in which it was encountered on the routing. So if there were a group with two questions, the second question would have identifiers starting with answer2. In this case, NubiS also added *_1* to ensure each checkbox has its own unique identifier. (Otherwise, attempting to look up an element with JavaScript would lead to ambiguous results.) NubiS does this for questions that have radio buttons or checkboxes: taking the numerical answer option code and appending it with an underscore to the normal identifier.

Unfortunately, it might not always be straightforward to know which identifiers would be assigned. For example, take a group statement like this:

```
group.TExample
if cnt = 1 then
                Q1
Q2
else
        Q2
        Q1
endif
endgroup
```

Here, *answer1* and *answer2* each depend on the value of *cnt*. One solution is to make the identifier a fill, and define the code as such:

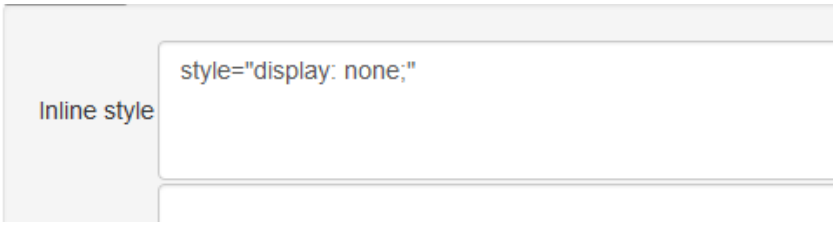
```
$( document ).ready(function() {
$('#^myfill').click(function() {
 alert('Hello world from me only');
```

});

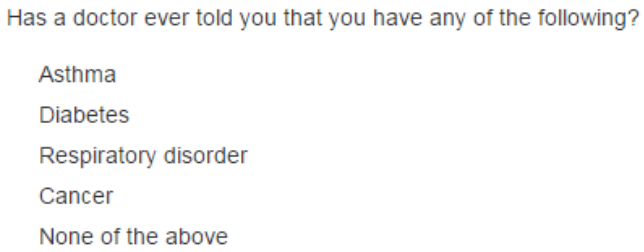
});

Indeed, this is a viable option. But available is a more straightforward solution, which is to specify an identifier for the variable:

The formatting options provided on the *Interactive* tab operate in a similar fashion as those for interactive behavior. And thanks to NubiS' inline style setting, you can use CSS in HTML coding. For example, you could state:



…resulting in no checkboxes:



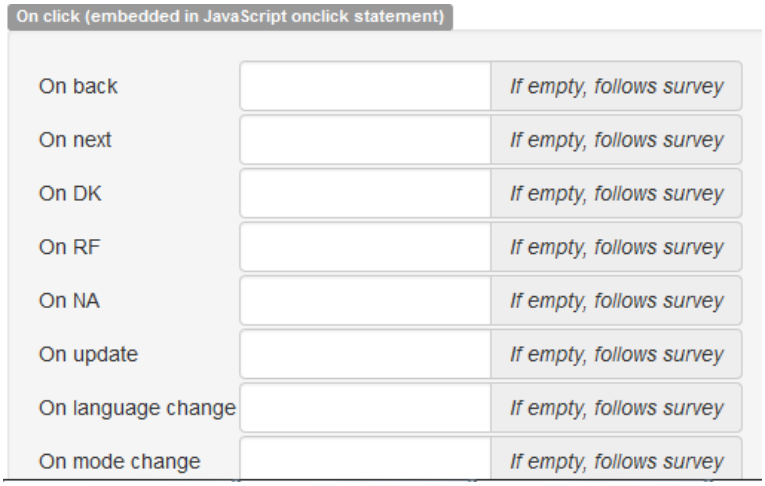
Similarly, in Page style, you could specify style sheets fragments to do the same:

Finally, JavaScript can be included by referencing external files. For example, in External scripts you could have:

<script type="text/javascript" src="js/myscript.js"></script>

This script could include function libraries or some other code needed for what you are trying to accomplish.

There is another way to add interaction: the buttons. For this, use the on click settings:



Any code entered must be valid JavaScript and not use single quotes. For example, if you would want to call a function mySubmitFunction, when you click *Next*, you can specify that function in the *After page load* setting and specify mySubmitFunction(); as the On next setting.

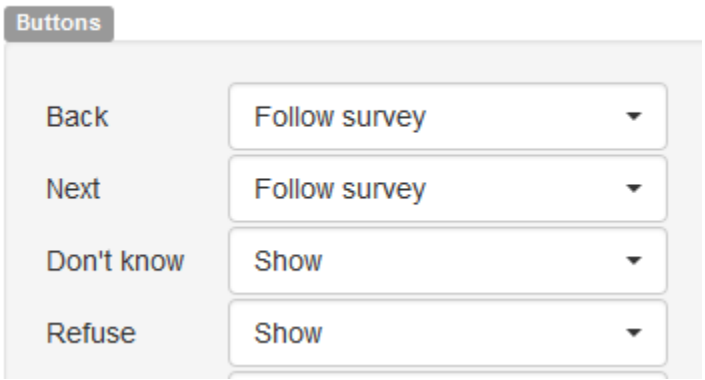
Lastly, it should be noted that a discussion of the above and of JavaScript/jQuery and CSS in general is beyond the scope of this manual. Rather, the examples are intended to be illustrative in nature only of the different options available to include interactive behavior.

17.4 Don't Know, Refuse and Not Applicable

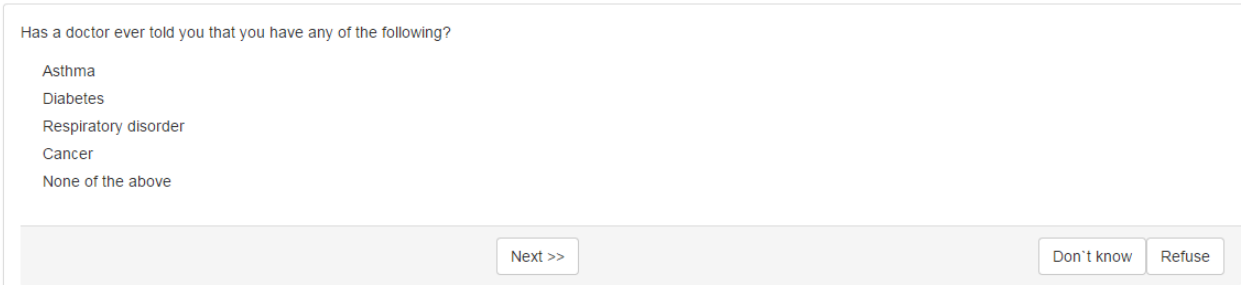
While learning the options for validating respondent answers, you spent time on the settings for what NubiS should do if an answer was empty or erroneous. Left unexplored, though, is what it means for an answer to be empty. From a purely programming point of view, empty is when no answer was given. But from an analytical approach, it is unclear. Does an empty answer mean the respondent didn't know the answer – or didn't want to give one? For example, suppose the survey asked the respondent about their children's names and all of them were empty. This would likely mean that s/he refused to say. But if you asked how large is their investment in stocks, then an answer of "Don't know" can be exactly that: the respondent doesn't know, in the vein of "Not applicable." Or perhaps the respondent is refusing to say.

In the context of self-administered mode, the loss of this distinction is usefully remedied by adding explicit options of Don't know, Refuse and Not applicable. This is straightforward with radio button or checkbox questions, as it just involves adding an option. For other types of variables, though, it is necessary to introduce separate variables that can capture an answer of *Don't know*, *Refuse* or *Not applicable*. A possible way to do so could be to have two checkboxes – but you would want to make sure that if, say, *Refuse* was checked, then *Don't know* gets unchecked as well. This is possible with what you've learned so far, particularly by leveraging the interactive options reviewed in the previous section. But NubiS provides an easier way.

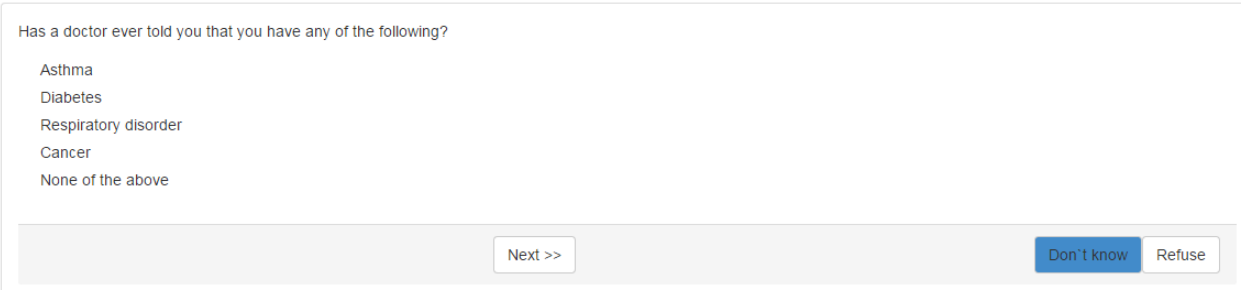
The first option is a per-question screen mechanism. To activate this, go to the *Display* tab of *B_new* and do the following:



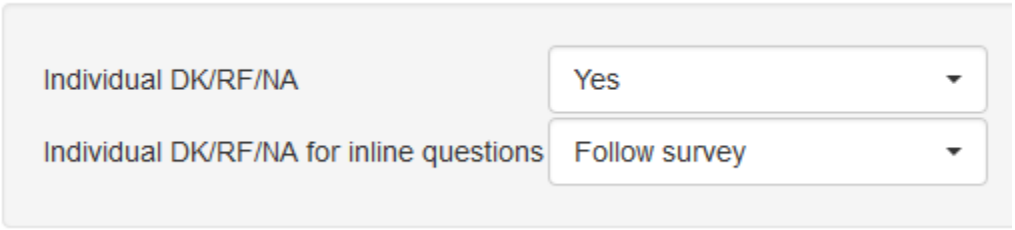
Now, if you go to *B_new* in your survey you get:



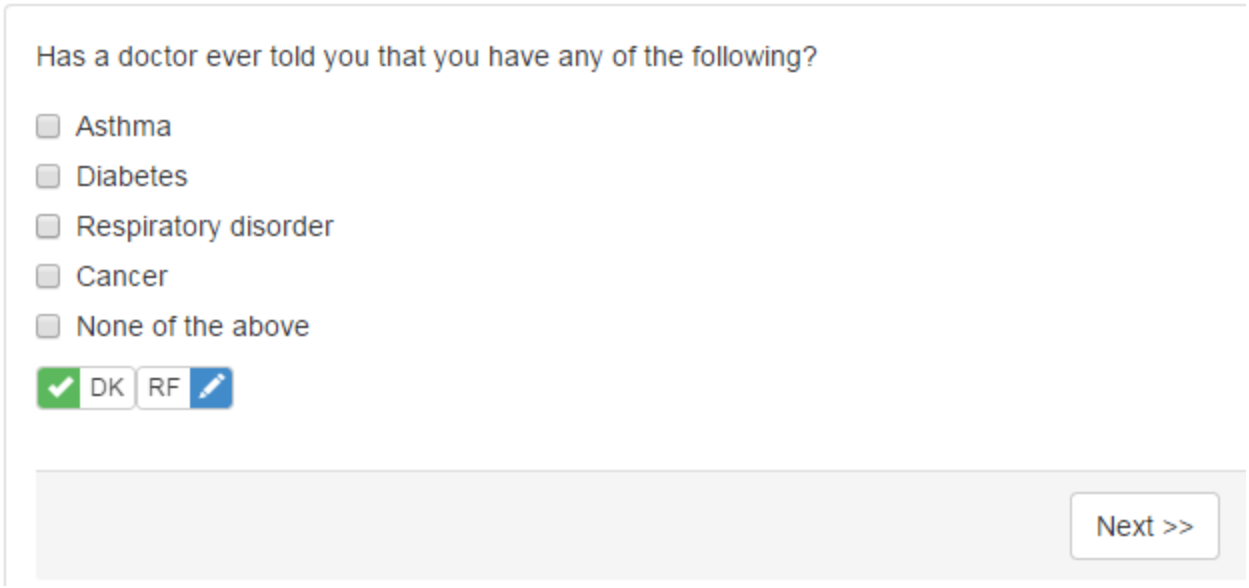
Clicking either one of the two new *Don't know/Refuse* buttons will prompt NubiS to proceed to the next screen. But if you go back, your choice remains selected:



The advantage of this approach is how it integrates with the usual buttons available, making it easy to use. But the downside is that clicking *Refuse* means refusal to answer any and all questions on the screen. Typically, this is not an issue, as explicit *Don't know* and *Refuse* options often are employed only in face-to-face interviews. And in face-to-face interviewing, the custom is to only display one question per screen. However, there may be a need for *Don't know* and *Refuse* options that can be applied on a per-question basis. If this is the case, then you can activate the **Per question** mechanism on the *Navigation* tab of *B_new*:



Looking at *B_new*, you see:



Now the *Don't know* and *Refuse* options are placed close to the question. Moreover, clicking one of them does not result in going to the next question screen. Rather, if **RF** is clicked, then **DK** automatically is deselected. Similarly, a box is checked, *RF* would be deselected. So this mechanism is akin to the one just informally described.

Once a survey starts to use explicit *Don't know* and *Refuse* options, keep in mind how NubiS interprets the routing. For example:

```
B_new
if B_new = response then
  if random_order = empty then
     random_order := mt_rand(1,2)
  endif
  FLOptions.FILL
  B000_
endif
```

Before, a response simply meant the respondent gave an answer: for *B_new* to check one or more boxes. But in this context, do *Don't know* and *Refuse* constitute answers? No – and you can see this by trying out the above. An answer only is considered a response if it is not a non-answer (which is nothing, *Don't know* or *Refuse*). But look at the following:

```
B_new
if B_new = empty then
  if random_order = empty then
     random_order := mt_rand(1,2)
  endif
  FLOptions.FILL
  B000_
endif
```

If you answer, for example, *Refuse*, NubiS says that this is an answer; that is, *B_new* is not considered to be empty. This leads to the more general statement:

if B_new = response then

is not the same as:

if B_new != empty then

if being used are explicit *Don't know*, *Refuse* and/or *Not applicable* options. You can see this by answering *Refuse*, which satisfies the second condition (and NubiS asks *B000_*), whereas the first condition is not satisfied. In other words, response and empty are not symmetrical in definition when using "*Don't know*, *Refuse* and *Not applicable*. As a final note, observe that a *Not applicable* button was not activated here, but it would work in the same fashion as described above for *Don't know* and *Refuse*.

17.5 Update Button

In addition to the three "empty answer" buttons just described, there is another type of button you can activate: Update. If this button is shown on the screen and clicked by the respondent, the current page is updated and shown again. You may wonder in what scenario you would need such functionality. Suppose you are asking the names of the respondent's children, using the following routing:

```
group.Toveral
max := 5
introChild // no input variable showing the question text
for cnt := 1 to max do
    ChildX058AFName[cnt] // string variables that asks the name of the child
enddo
endgroup
```

This would show five input boxes on the screen in which the respondent can enter names:

But perhaps you want the option to have another five input boxes appear for the eventuality that the respondent has more than five children. One option would be to do the following:

group.Toverall

max := 5

introChild // no input variable showing the question text

for cnt := 1 to max do

ChildX058AFName[cnt] // string variables that asks the name of the child

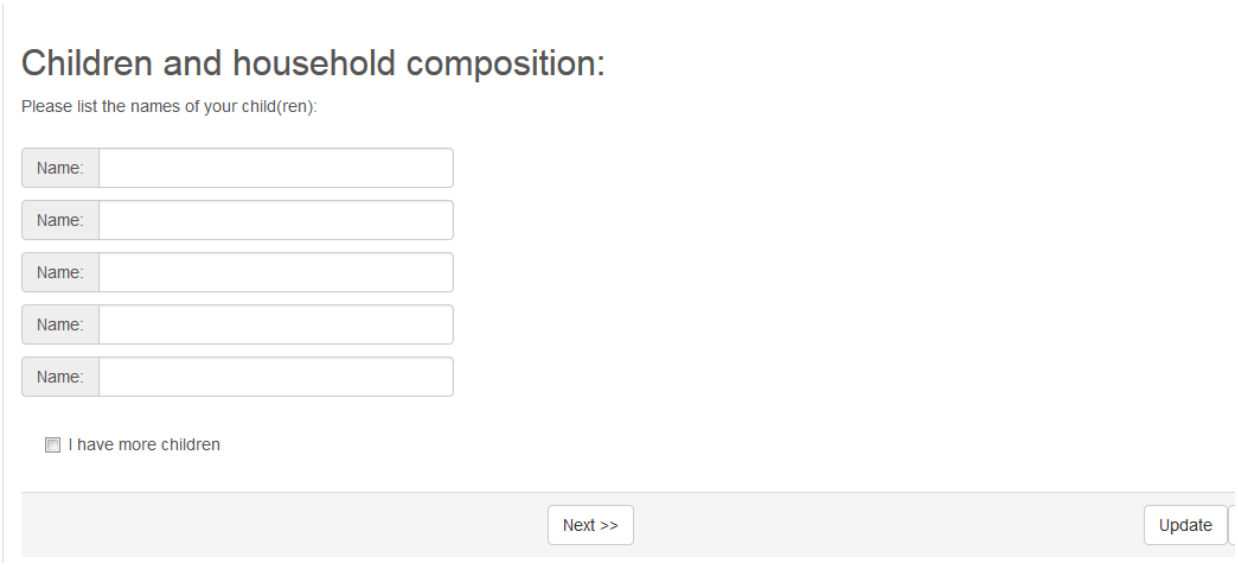
enddo

// set of enumerated variable with one checkbox that can be checked to indicate more kids

morekids

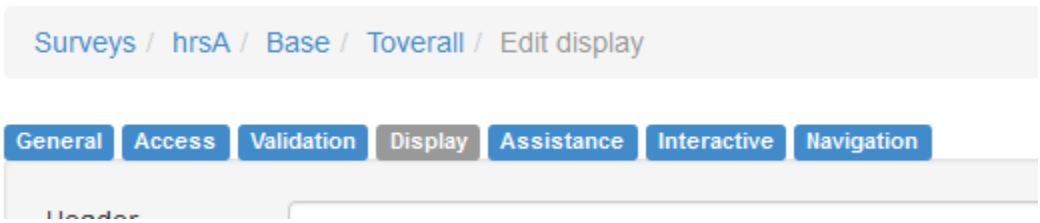
endgroup

In the survey, this would appear as:



With this in place, there are two ways of asking the respondent about more children if they check the box "I have more children." The first is to let them click *Next* and follow up with a similar screen. The downside is that the respondent can't see the names already entered (unless they are shown as fills in the question text of the follow-up screen).

Another way is to utilize the *Update* button, which can be activated by editing a variable or a group. In this case, you will edit it on the *Display* tab of group *TOverall*. The reason is that the display of a button or a progress bar is a setting applicable to the screen as a whole. As such, if multiple variables are displayed on the same screen, you need to specify settings for the group of variables as a whole.

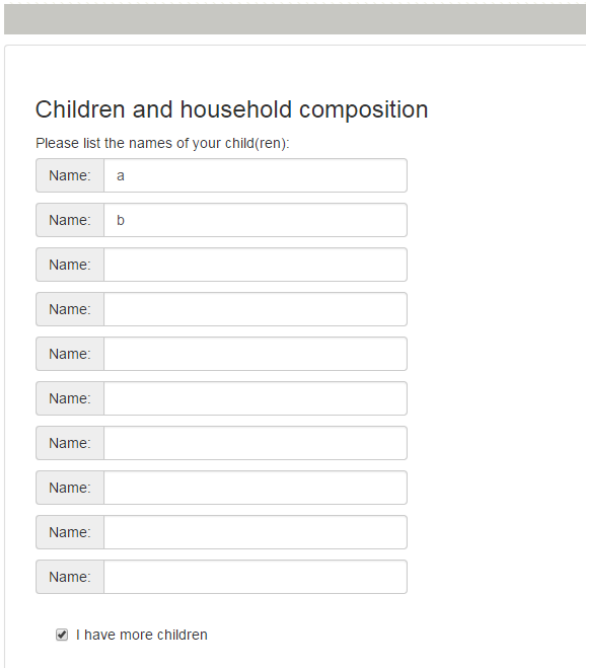


To enable the *Update* button, simply select *Show* instead of *Follow survey* and save the change.

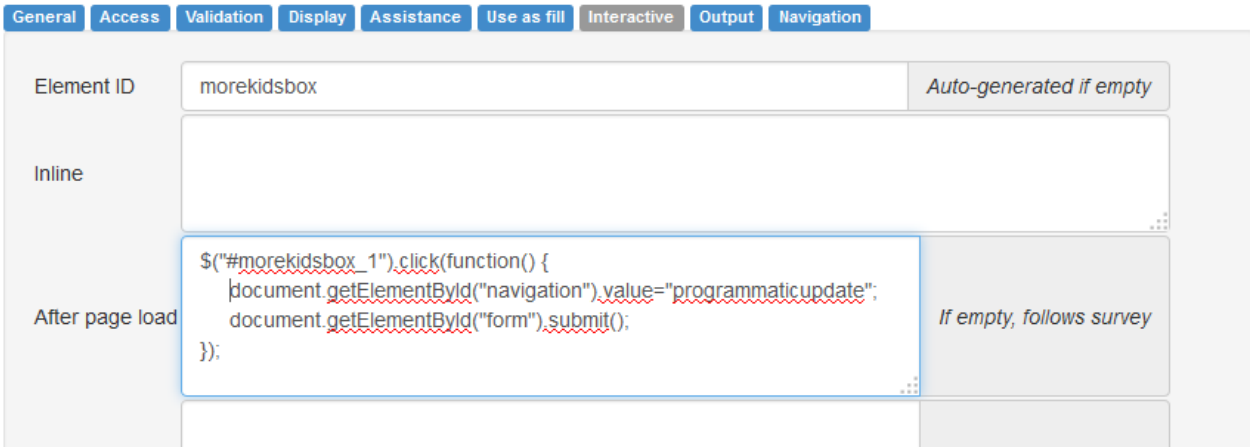
When the respondent clicks the *Update* button, NubiS refreshes the current screen while storing any answers given prior to the respondent having clicked *Update*. In this example, if the box "I have more children" is checked and *Update* is clicked, NubiS will store this answer to morekids then show the same screen again. But how does that help? Modify the routing to:

```
group.Toverall
if morekids = 1 then
max := 10
else
        max := 5
endif
introChild // no input variable showing the question text
for cnt := 1 to max do
        ChildX058AFName[cnt] // string variables that asks the name of the child
enddo

// set of enumerated variable with one checkbox that can be checked to indicate more kids
morekids
endgroup
```

If the respondent checked the box and clicked *Update*, NubiS would store the answer to *morekids* (which would be *1*, given it was checked) and run the routing again to display the same screen. That is, it will again execute the group statement. Now, instead of *max* being set to *5* it would be set to *10*, leading to the inclusion of five extra instances of ChildX058AFName:



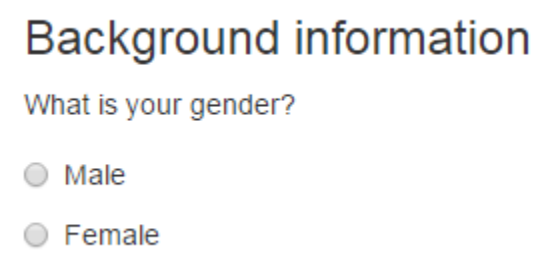
What if you want to reduce the number of actions the respondent must perform to refresh the screen? For one, the screen could refresh when the respondent clicks the checkbox. This can be accomplished using NubiS support for a so-called programmatic update of the screen. Setting it up involves a small snippet of JavaScript associated with the checkbox. On the *Interactive* tab, open the *morekids* variable and enter the following:



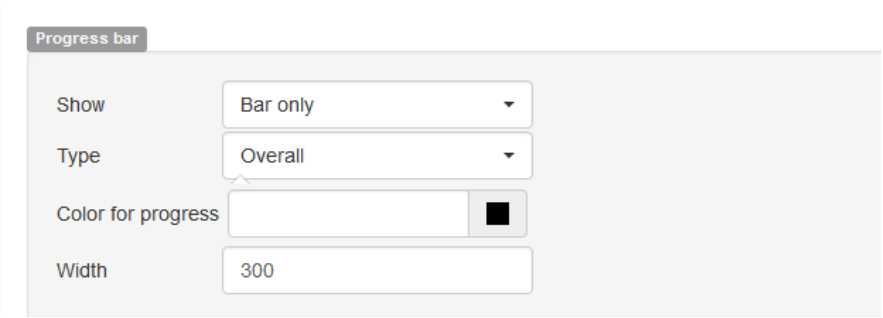
Assign an identifier to *morekids* to make it easier to reference it in the JavScript code in the *After page load* text box. Here, it is stated if the box is clicked, there is an update to the hidden input tracking the action performed. After that, the form is submitted to refresh the question screen.

17.6 Progress display

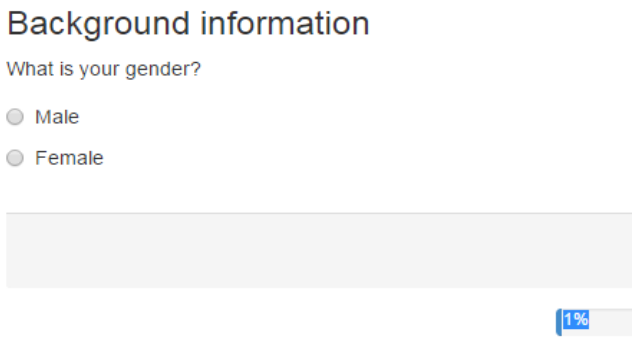
For a respondent, an important feature of a survey is knowing how much progress has been made towards reaching the end. NubiS has a few ways in which such progress can be conveyed. One is rudimentary in nature, involving using section headers to indicate in which part of the survey a respondent is. An example is when you used an HTML header in *secA*:



Another way is to use a progress measurement tool. NubiS provides two types of measurement, both of which can be activated on the survey-level *Display* tab:

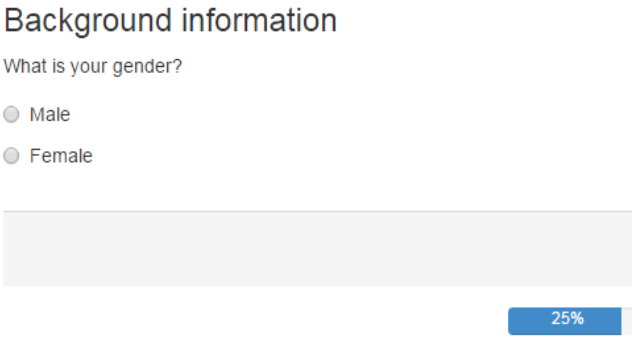


First, the *Show* setting instructs NubiS how to display the progress made. This can be a bar, a percentage, a bar and a percentage, or nothing at all. You already have seen the bar. A percentage appears as:



The color used and the bar's width can be configured with the appropriate settings.

Second, you can set the type of progress measurement. One is the default, which is **Overall**. This means that progress is measured by comparing the current point of the interview to the survey's end. In contrast, the **Per section** type measures progress using the end of the section of which the question is part. If you were to switch, you would see:



Progress has jumped from 1 percent to 25 percent, but this is only because progress now is measure until the end of *secA*. Once the end of secA is reached and the respondent moves on to another section, the progress bar will reset.

Which type – *Overall* or *Per-section* – is most suitable? For relatively simple and short surveys, the *Overall* progress bar would be appropriate. However, in a lengthy survey, the *Overall* measure can be uninformative. As such, it would be better to switch to the *Per section* type so the respondent feels progress is being made.

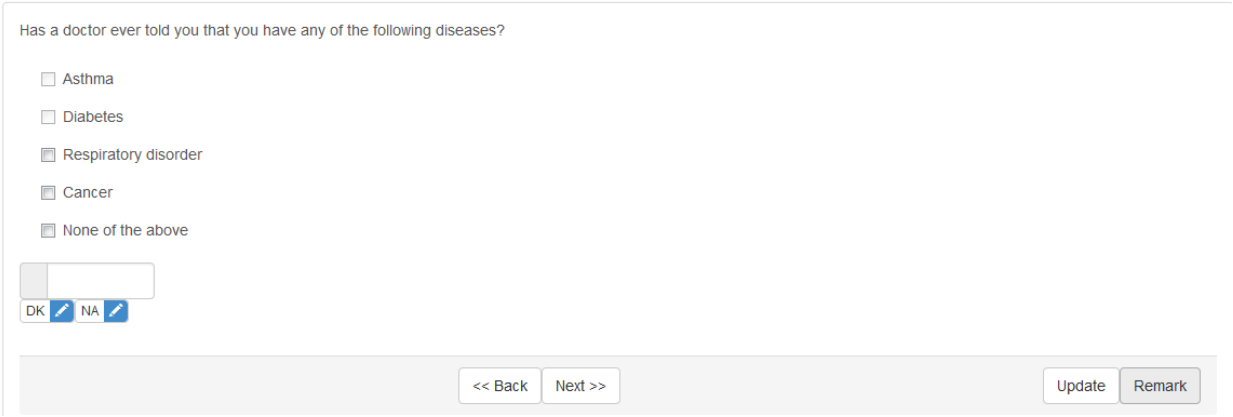
Note that generating the data for displaying the overall progress bar is relatively resource intense, and as such NubiS refrains from updating its progress bar data until the Base section routing is saved. As such, whenever you make a change to a section's routing and you wish to update the overall progress bar, you will need to re-save the Base section routing.

Lastly, it should be noted that the progress NubiS displays is an approximation. In a complicated survey, the distance until the end of the survey (or section) greatly depends upon the respondent's

specific answers. As such, jumps in the progress bar can occur when half of a section is skipped due to answers given by the respondent earlier.

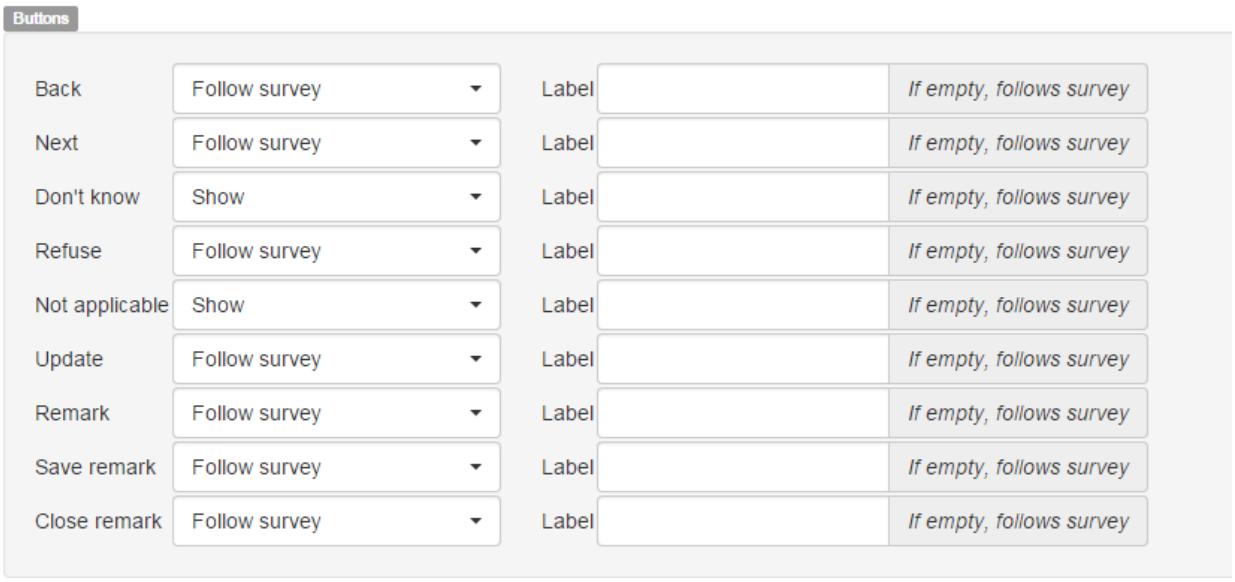
17.7 Remarks

Another useful function, particularly in a face-to-face or telephone interview mode, is the ability to add remarks. For example, take the following question:



Suppose the respondent indicates that s/he has diabetes, but also mentions to the interviewer that it is type 1 rather than type 2. As it stands, there is no way for the interviewer to capture this information. That is where the **Remark** button (seen in the lower-right corner) comes in handy.

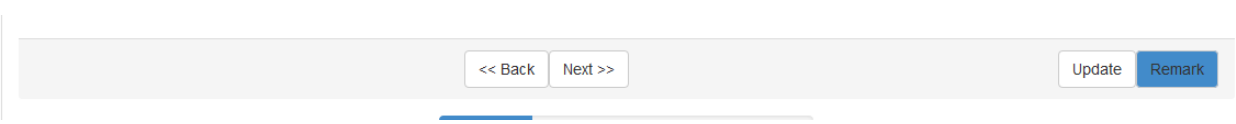
Activation of the remark mechanism is straightforward and similar to, for example, the *Update* button. That is, you can choose whether to show it on a variable's *Display* tab. To illustrate, open up *B_new* and select the *Display* tab. Here, you can decide which buttons to show:



Note that it is possible to enable the remark popup without showing the *Save* or *Close* buttons. The *Close* button is not strictly necessary because clicking outside the remark popup will also close it. The *Save* button can be disabled if it is desired, for example, to allow an interviewer to see existing remarks yet not edit them.

Clicking *Remark* brings up a popup:

Here the interviewer can enter “Diabetes type 1” and save. The popup then closes and the *Remark* button turns blue to indicate a remark is present for this question screen:



Now, on clicking *Next*, the answer to the question will be stored, as will the remark. You will learn shortly during the discussion of data output that those remarks can be extracted for use in data analysis.

17.8 Showing A Different Screen on Re-Entry

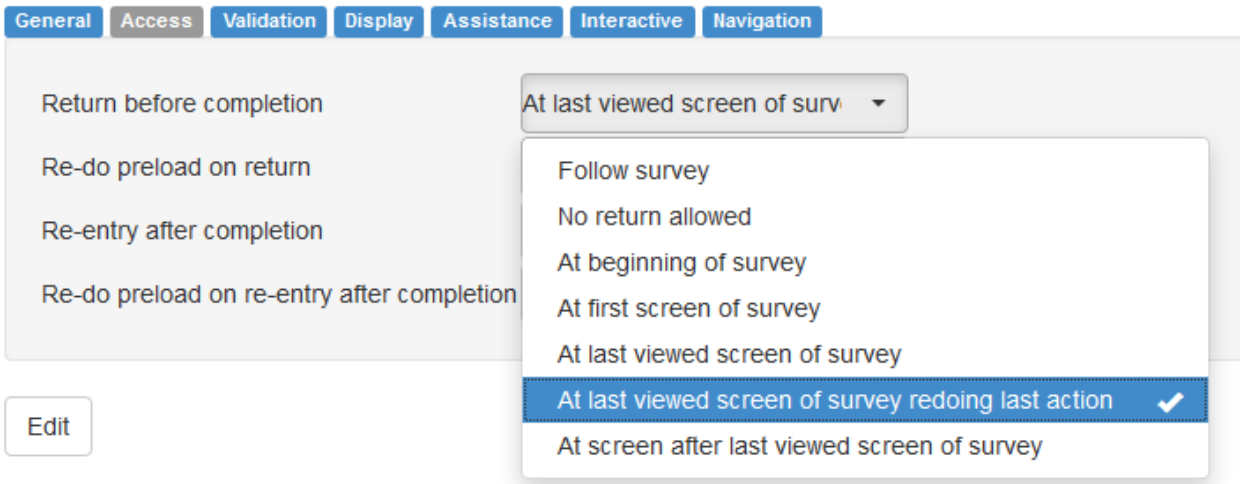
On occasion, your may need, for example, an eligibility screen informing the respondent whether they can fill out the survey. Or maybe you want to give access to respondents only after they complete another survey. How can these be accomplished in NubiS?

An example:

```
group.Toverall
othersurveycompleted := isCompleted()
if othersurveycompleted = 1 then
        continuescreen
else
        waitscreen
endif
endgroup
```

When the respondent arrives at this part of the survey, the function **isCompleted** is called, which returns *1* if the other survey was completed. (In the background this could be, *e.g.*, a query in a database.) But if the survey is not yet complete, this would lead to **waitscreen** being shown. If the respondent completes their other survey, then they re-enter the first survey at the screen where they left. (This is NubiS' default behavior, as to be explained in Sec. 19.4) However, the wait screen still appears. Why?

What went wrong is that NubiS, by default, re-shows the last screen viewed by the respondent – but it does not re-perform the last action. Here, NubiS showed *waitscreen* again without considering the assignment and *IF* condition just before it. To alter this, modify the **Access** settings of the group *Toverall*:



Specifically, NubiS is instructed to redo the last action, which will result in the function *isCompleted* to be called again and **continuescreen** to be shown (assuming, of course, the respondent completed the other survey).

18. CUSTOM FUNCTIONALITY

The options discussed so far should be more than sufficient for developing a survey. There may be the need, though, to program in features NubiS does not offer by default.

18.1 Using Custom Functions in the Routing

NubiS supports integrating custom functions. There are several avenues available, depending on the goal to be achieved. One scenario is the survey reacting to a respondent's answers. You already have seen examples, such as PHP's built-in functions calculating the respondent's age. And the usage of such functions through assignments is not limited to pre-defined PHP functions. Rather, if required, custom functions can be added to NubiS. This is done by locating the file customfunctions.php in the folder NubiS root/custom, and defining the needed PHP functions there.

Before any user-defined or PHP built-in functions can be used, NubiS requires their function names to be listed in the getAllowedRoutingFunctions array. NubiS defines a series of function names in this array by default. These MUST be kept in order to ensure proper functioning of NubiS' routing commands. Any used functions must also not be part of the array defined in the getForbiddenRoutingFunctions. Together the arrays in these two functions act as a white list and black list for PHP functions within NubiS. NOTE: it may be possible that the server on which Nubis resides prevents using certain functions through its overall PHP configuration.

Once defined and added to the white list, the function can be called on the routing:

dummy := executeMyFunction()

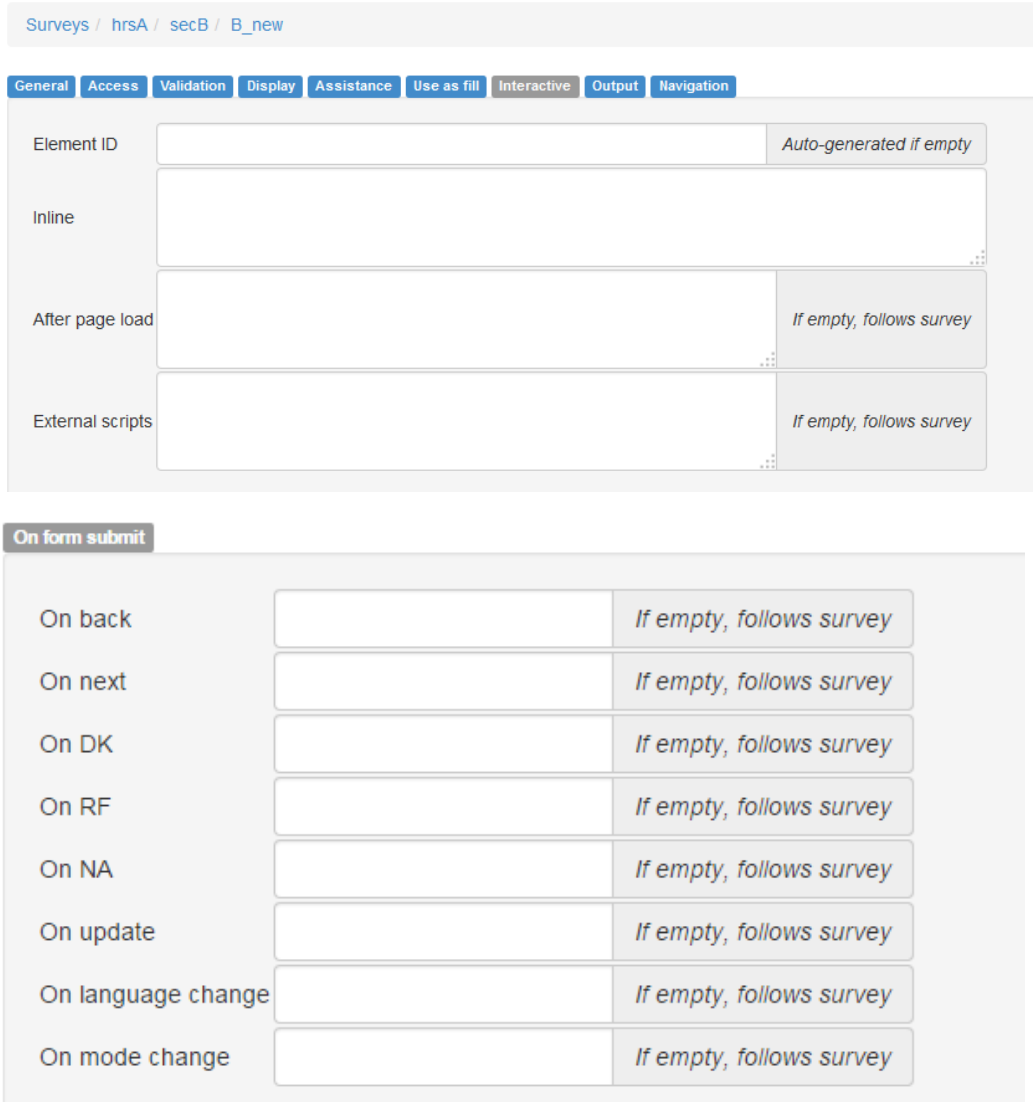
In this regard, it is possible to pass along the answers to variables by stating:

dummy := executeMyFunction(variable1, variable2, variable3)

Dummy is exactly that: a dummy variable. This is because NubiS expects to assign the result of a function call to something; in this case, *dummy*. This does not mean a custom function should return a result. Rather, it is a merely syntactical construct telling NubiS to execute the function.

18.2 Using Custom Functions on Button Click

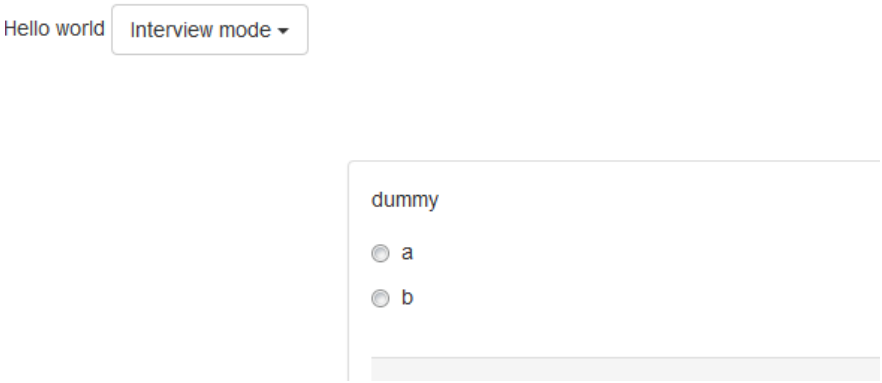
Another place where custom functions can be leveraged is on a variable's *Interactive* tab of a variable. Open it for *B_new*:



Ignoring the options towards the top, those under On form submit allow a function to be executed when the corresponding button is clicked. Consider a custom function defined as follows:

```
function mytest() {
    echo 'Hello world';
}
```

Enter mytest for the Next button value here. Then when the survey is run and, on the *B_new* screen, *Next* is clicked, appearing is:



Obviously, this example is not particularly useful, but a possible use is having the respondent's answer update an external database. You also could accomplish this by having a routing that says:

b_new

dummy := mytest()

Keep in mind that upon executing a function on going back in the survey, NubiS will take care of undoing any completed assignments – but this does not extend to whatever was done in a function call. For example, if *mytest* updated a database table, then upon the respondent going back in the survey, rolling back that change might be a good idea. Specifying a function to be performed on going back can address this issue (whereas there is no such routing equivalent in NubiS).

If desired, parameters can be passed along. These can be literal, such as "Bye" or references to variables. For example, *On next* could be specified instead as:

mytest(^prim_key)

and the function modified to:

```
function mytest($param) {
        echo '<br/><br/><br/><br/><br/>Hello world: ' . $param[0];
}
```

…which would then show as 'Hello world: ' plus the case identifier. ($param[0] is needed here because NubiS passes the parameters as an array.)

If you would rather not pass any parameters, but rather get values directly, this is also possible. To do so, add the following to the custom function:

```
function mytest() {
 global $engine;
 $param = $engine->getAnswer("prim_key");
 $displayvalue = $engine->getDisplayValue("prim_key", $param); // to get a label instead
        echo '<br/><br/><br/><br/><br/>Hello world: ' . $displayvalue;
}
```

Note that an *On submit* function is executed after NubiS' normal processing. For example, NubiS first will update its database for any answers given by the respondent, undo any assignments if going back in the survey and so on, and only then execute the custom function.

Lastly, observe that just like for custom functions in NubiS routing, before any user-defined or PHP built-in functions can be used in this context, NubiS requires their function names to be listed in the getAllowedOnChangeFunctions array. NubiS defines a series of function names in this array by default. These MUST be kept in order to ensure proper functioning of NubiS' routing commands. Any used functions must also not be part of the array defined in the getForbiddenOnChangeFunctions. Together the arrays in these two functions act as a white list and black list for PHP functions to be used on button click within NubiS.

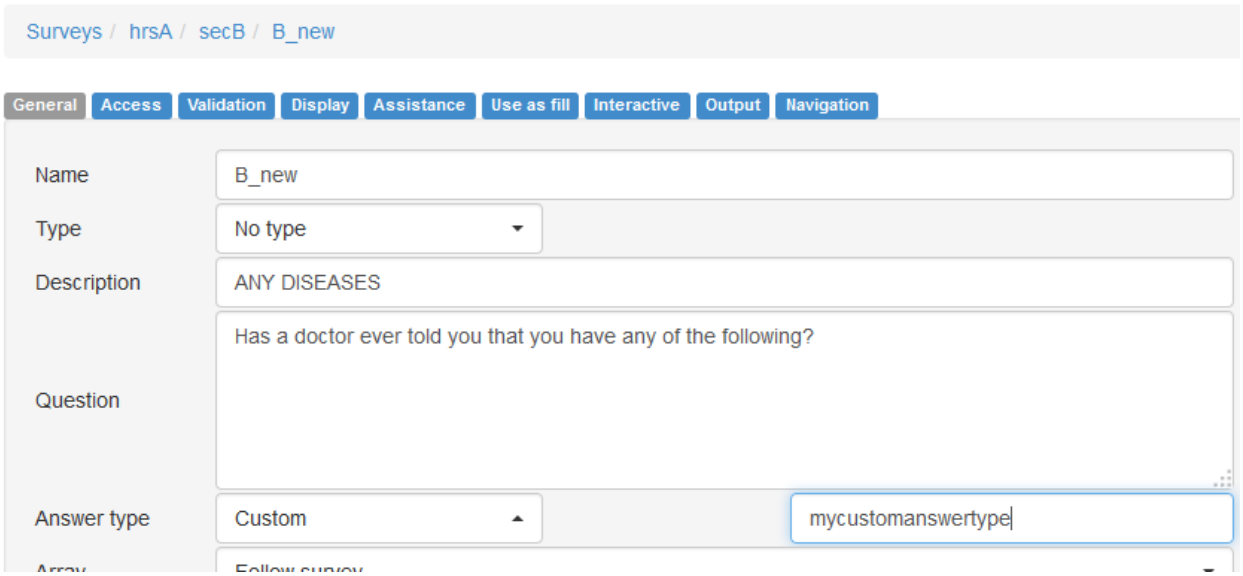
18.3 Modifying the Auto-Generated Question Display

Another ability of NubiS is altering the outputted question screens in some form. An example of this was the introduction of the *[IWER:]* syntax that created a blue box in the screen. The code to accomplish this is located in the *customfunctions.php* file in the **lastParse** function. By default, its content is:

```
function lastParse($str) {
        $patterns = array('/((?i)\[IWER:(.*?)]])/s');
        $replace = array('<br/><div class="alert alert-info" role="alert"><b>\2</b></div>');
        $str = preg_replace($patterns, $replace, $str);
        return $str;
}
```

The value that goes into this function is the entire HTML page that NubiS will output. So one way to employ this would be consistently using notation while programming the survey (*e.g.*, for interviewer instructions), then applying a certain formatting rather than having to specify such format for every occurrence.

Another manner in which the question display can be influenced is through the usage of custom answer types. A custom answer type can be activated as follows:



When NubiS encounters this question, the function mycustomanswertype will tell it how to display the manner in which the respondent is to answer the question. You have seen an example of this with the household grid. The code behind it is extensive but the important fragment here is the line linking the custom answer type to NubiS:

```
function hhGrid($variables) {
        $name = "hhGrid";
        $contentStr = '';
        $contentStr .= '<input type=hidden name=answer1 id=hiddengrid value="" />
        ....
        ....
}
```

The line needed by NubiS is the one defining the HTML input element relating to a question on its routing:

$contentStr .= '<input type=hidden name=answer1 id=hiddengrid value="" />

The key lies in name=answer1. When NubiS builds a question screen, it names the input boxes numerically, starting with answer1, where the numbers correspond to the order in which the questions are encountered. So if there is:

group.TOverall

Q1

Q2

endgroup

…then, on the outputted screen *Q1* is demarcated by *answer1* and *Q2* by *answer2*. When *Next* is clicked, NubiS will take the values of *answer1* and *answer2*, and store them for, respectively, *Q1* and *Q2*. As such, for NubiS to be able to link the answer to the custom answer type question, the code creating the custom input mechanism must define this linkage. The remainder of the code can be anything, as long as it defines a proper input mechanism that is returned as a PHP string. Just look in customanswertypes.php to see some examples.

A last manner to alter the display of questions involves display templates, which are are defined in the root NubiS/display/templates folder. Display templates specify, for example, what the header of the survey should be, how it should handle the end of the survey and so on. NubiS's default template is defined in displayquestion_0.php. Several other templates are included as examples; displayquestion_1.php is used in the Understanding America Study and displayquestion_2.php is found in the M-Teens data collection project. Among the customizable options is displaying text from right to left instead of left to right (useful for an Arabic-language survey), achieved by overriding the function that creates the main body of the screen.

Note that before any user-defined or PHP built-in functions can be used to construct a custom answer type, NubiS requires their function names to be listed in the getAllowedCustomAnswerFunctions array. NubiS defines a series of function names in this array by default. These MUST be kept in order to ensure proper functioning of NubiS' routing commands. Any used functions must also not be part of the array defined in the getForbiddenCustomAnswerFunctions. Together the arrays in these two functions act as a white list and black list for PHP functions to be used for custom answer types within NubiS.

18.4 Parallel Sections

Consider this scenario: Multiple people in a household will partispate in your survey, and you want the most knowledgeable person answering all questions on finances or family. But it is possible different people in the household may have the most knowledge of those subjects.

One option is to simply develop the survey, ask one section after another, and switch between people when needed. This assumes, though, that all relevant people are present and, moreover, available to answer the questions in the order defined. But what if the person who can answer family questions is there, but the person who can answer about finances is not? In such a case, if the routing demands financial questions are asked before family questions, then the interview is over before it can start. However, NubiS can be programmed to give the ability to jump to the family questions and return to the financial questions.

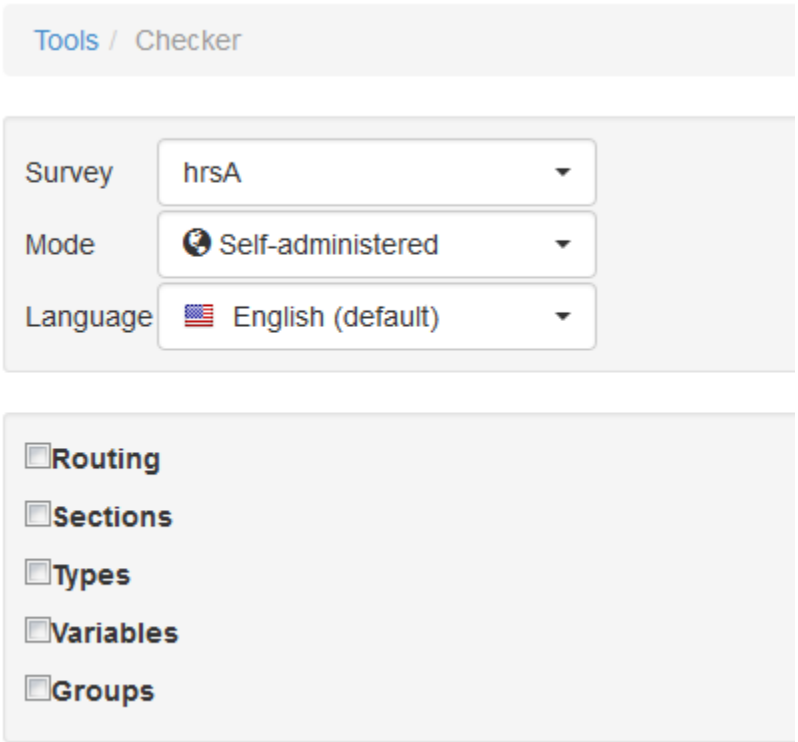
Parallel sections are NubiS' solution for this. They are parallel in the sense that they are considered to be independent from one another. If they had any dependencies, then you could not switch their order because the answers to one would influence the questions asked in the other. NubiS caters for running sections independently by allowing to launch-specific sections through an external web site. Such a site would be similar in make-up to the example test page discussed earlier, but would specify which section should start. A rudimentary illustration is provided in root nubis/parallel.php.

19. PREPARING FOR FIELDING/SAMPLE MANAGEMENT

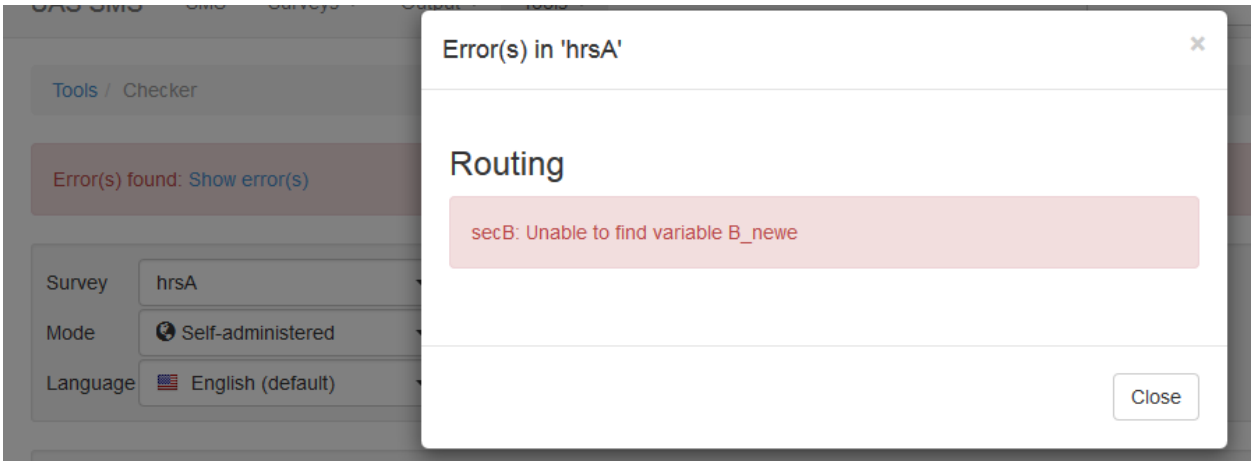
The previous sections have focused almost exclusively on developing a survey in NubiS. Occasionally mentioned was how the tester tool could be used to see how programmed features would look in the survey. And now it's time to start learning now to actually collect data by fielding a survey.

19.1 Checking and Testing the Survey

An important step towards fielding a survey is making sure the survey has been programmed in its intended manner. From the programmer's end, this typically involves two checks. The first is to use NubiS' built-in mechanism to find any syntactical mistakes. It can be accessed through the navigation bar under Tools | Checker:

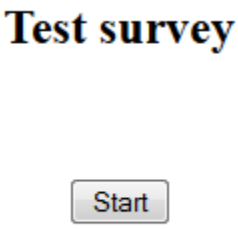


The mechanism operates on a per-survey level. After selecting the survey (in case of multiple surveys), you can choose the interview mode, language and which components to be checked. If there is an error found in any of the selected components, NubiS will provide a message to locate it. For example, in the routing of *secB*, there is a reference to B_newe rather than *B_new*. Upon running the checker, you would see:



The second step is to ensure the survey everything works properly from a "semantic" point of view, meaning the intended behavior is displayed. One part of that is the programmer testing different scenarios, skip patterns, and so on. A second part is to ask other people (for example, any researchers involved) to test the survey.

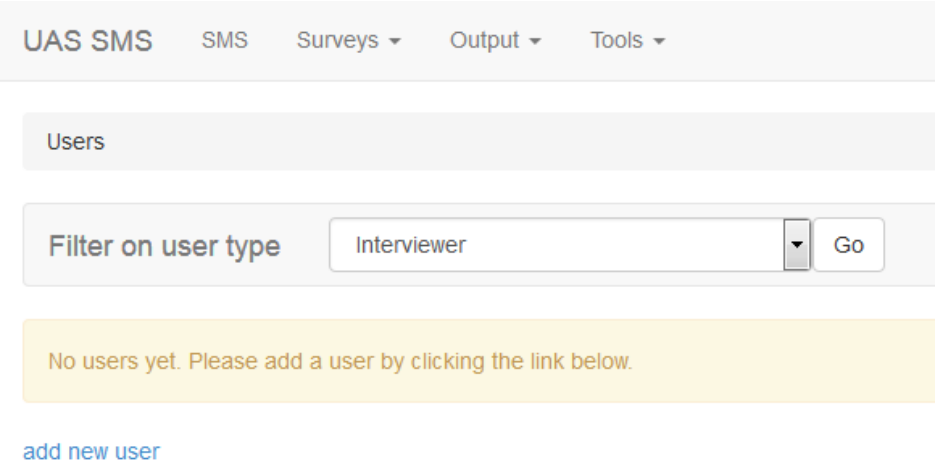
To accommodate such testing, NubiS provides both a basic and more advanced mechanism. The basic mechanism utilizes a standalone test page to which testers can be directed. A simple example of such a page is included with NubiS, located in the root NubiS/test folder. In the browser this translates into, for example, the URL www.example.org/nubis/test/index.php. Opening this page shows the very basic:



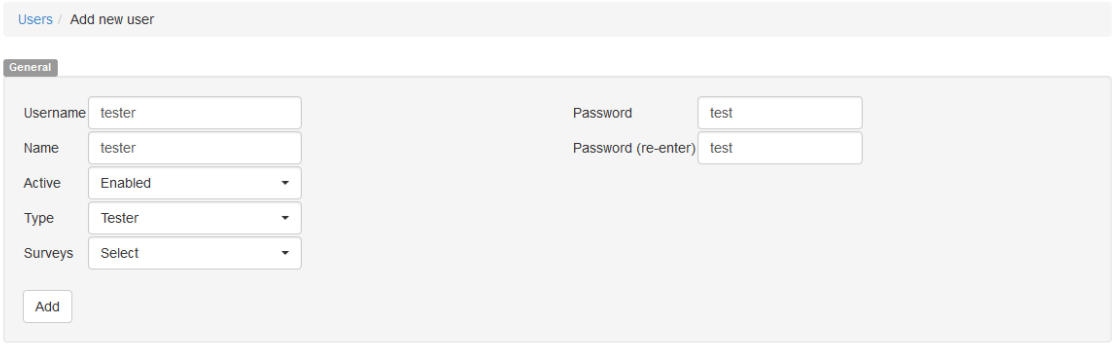
This can be modified to include more text/logos and so on, or to make a new test page based on the standard one.

Another option is to create a so-called tester account. This is a NubiS account just like that for the system administrator, but with access restricted to testing a survey and reporting any found problems. To set up such an account, access NubiS' user management:

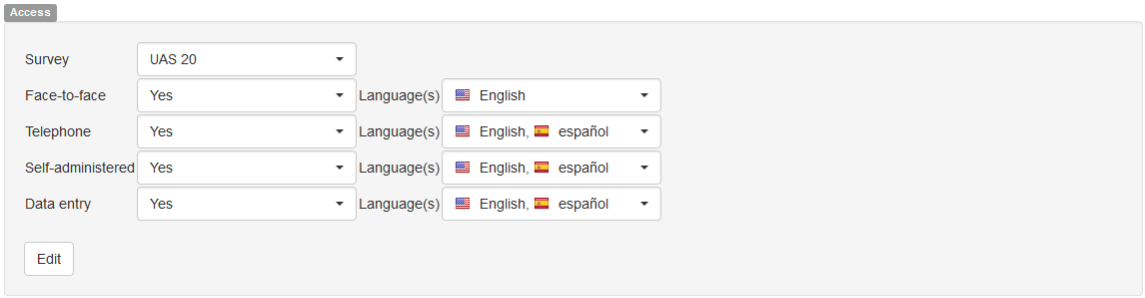
Opening the *Users* screen shows the following:



Click **Add new user**:

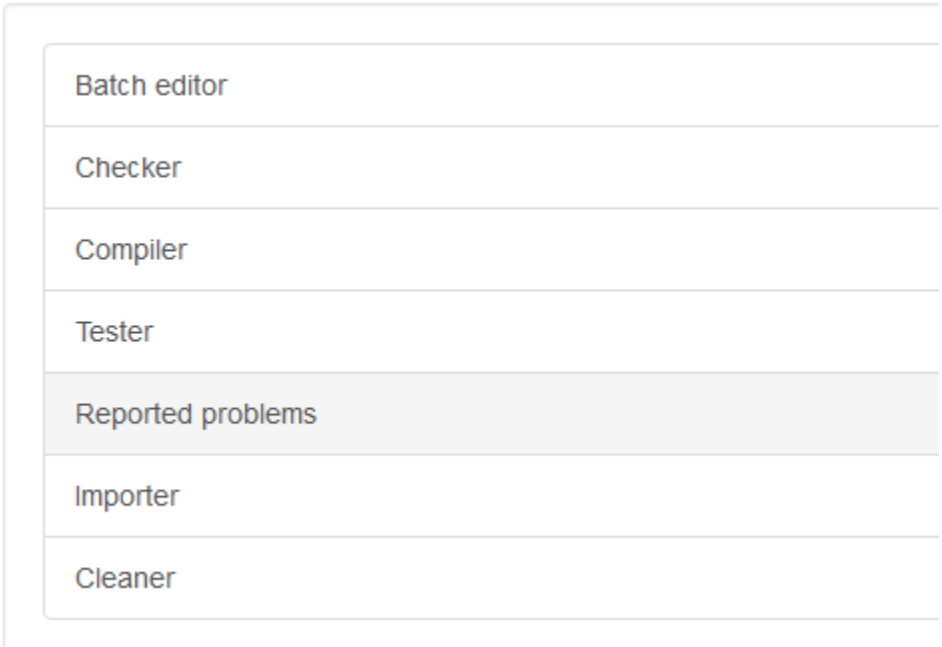


Specify a username and password, indicate that the account is active and choose as the account type *Tester*. You can also decide which survey(s) the account will be able to test. Once you completed your selections, click Add. NubiS will confirm creation of the account and show an additional section:

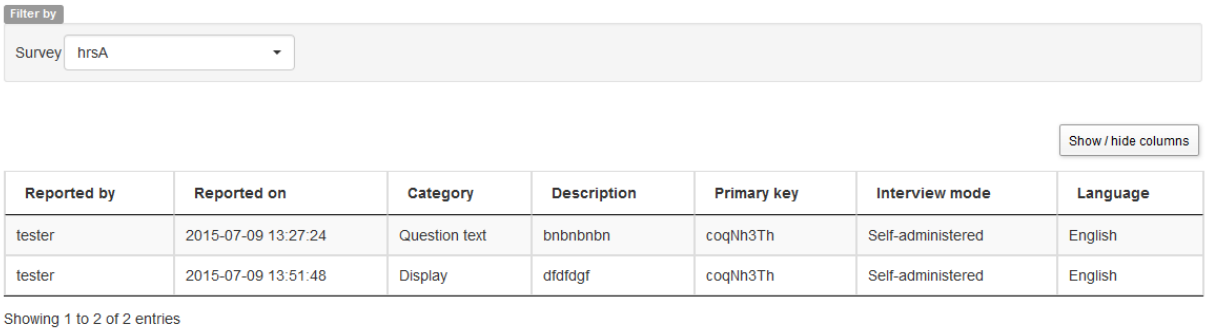


Under Access you can choose the interview mode(s) and language(s) the tester is allowed to test on a per-survey bases. For example, you may only make face-to-face available or (like here) every mode with every language. After creating the account, you only need to share the credentials with the person who is to test. For a detailed discussion about using NubiS as a tester, please refer to the NubiS Tester Manual. For more information about user management please see Section 21.

Once the tester has gone through the survey and flagged any issues, you can go over these reports. Click Tools and you will see the tools overview with the option Reported problems:



Choosing this option, you can see an overview of the reported problems:



With this information, you can analyze the problem and, if needed, make modifications to the survey.

Another useful feature that can facilitate testing is the usage of the **execution_mode** variable, which captures whether the survey is running in normal mode or test mode. Why is this useful? Because just as you can leverage *language* and *mode* on the routing, you can do the same with *execution_mode*. For example, you can include test screens contingent on being in test mode:

if execution_mode = TEST then

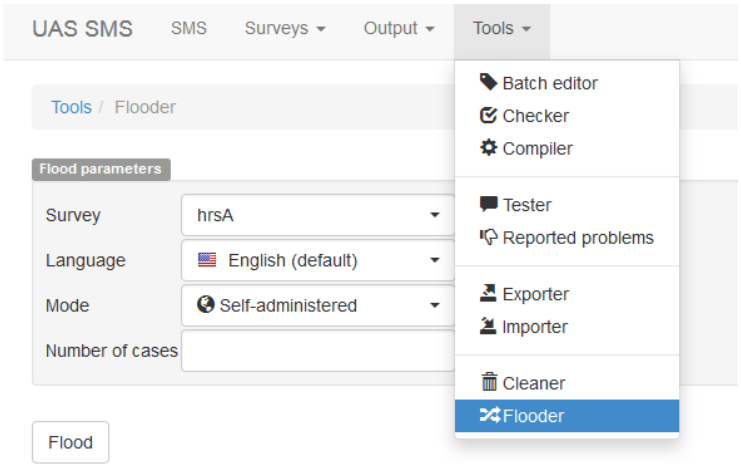
// test screen here, e.g., to pick randomization values to run through a particular scenario

endif

When NubiS encounters the above, it will be skipped if in normal mode; otherwise, the test screen is shown. This has the advantage that you do not need to remind yourself to take out the test screen once the survey is administered to real respondents. Moreover, should you wish to run additional tests when the survey is already in the field, you could still use the test screen(s) in place prior to fielding.

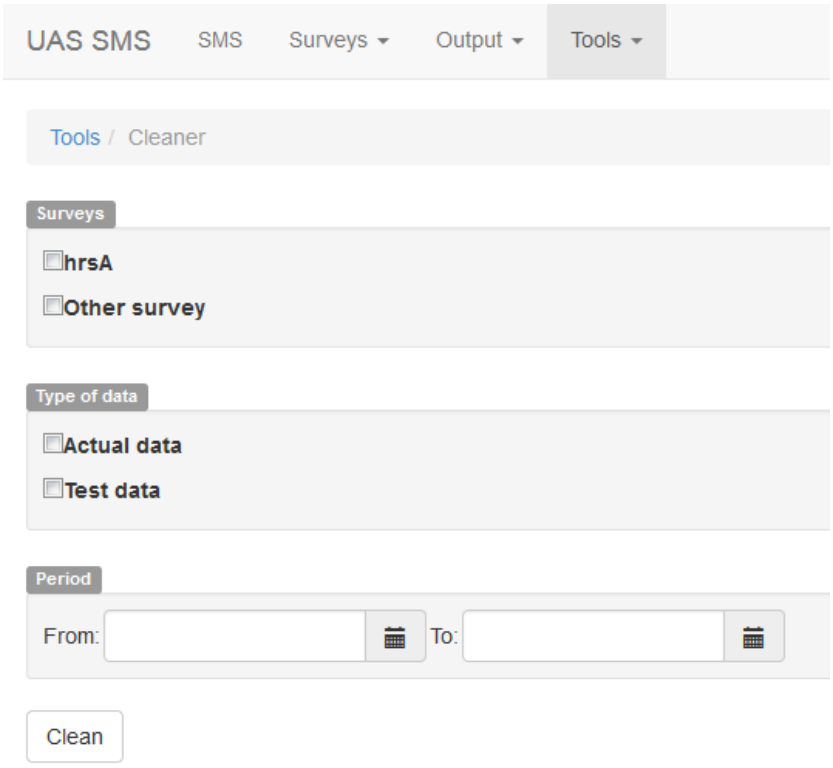
19.2 Generating and Removing Test Data

In addition to testing the survey itself, it is just as important to look at how data collected in the survey is outputted. One option is to simply test the survey multiple times and then use NubiS' output options to generate and inspect the resulting data file. Another option is to use the flooder, located in the *Tools* drop down, which allows the automatic generation of data.



In the flooder screen, you can choose the language, mode and number of cases for which data is generated. On clicking **Flood**, NubiS will go through the survey and randomly generate answers within the parameters set by the programming (for example, respecting numerical ranges and skip patterns). Note that this is an experimental feature and it may take considerable time for NubiS to generate the data if the survey is long and the number of generated cases is large.

A related aspect of managing test data is the ability to remove such data prior to actual survey fielding. It is highly recommended that test data be cleared to avoid cluttering generated data files with test data. To this end, NubiS provides the cleaner tool, which also can be found in the *Tools* drop down:



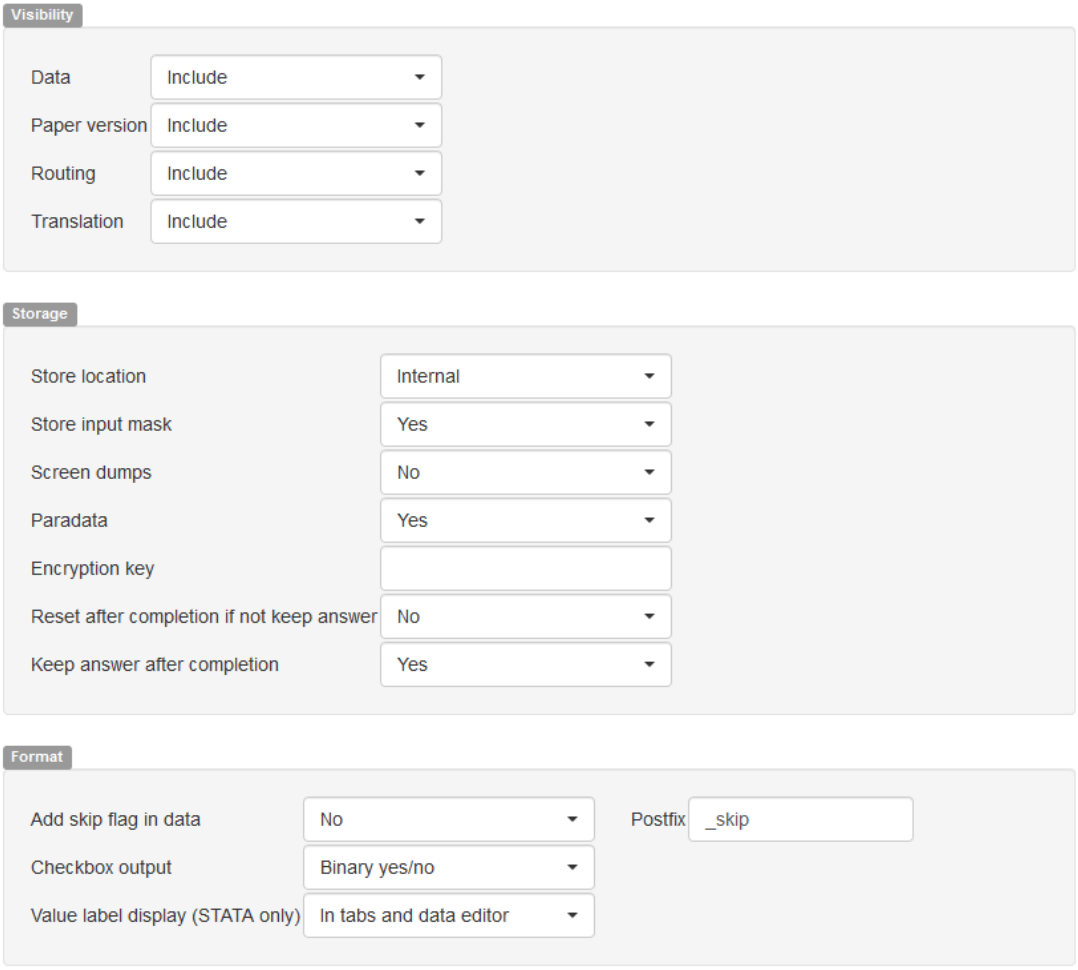
The cleaner allows for removing data from any selected survey. It can be told the forms of data that should be removed: actual, test or both. The difference between actual and test data lies in the fact that they are stored in separate database tables (where NubiS stores survey data depends on whether the survey is started in normal mode or test mode). By default, NubiS starts a survey in normal mode. However, surveys run through the NubiS tester are launched in test mode. Similarly, a survey can be started from a separate page that instructs NubiS to follow test mode by passing along a specific POST variable value. (An example can be found in the test launch page included with NubiS, located in the root/test/index.php file.)

Typically, both actual and test data can be safely removed prior to fielding – but this is, of course, dependent on the specific demands of the survey project. Before cleaning, it would be advisable to back up relevant database tables and/or generating data files containing the to-be-deleted data. Note that, if needed, a time period can be specified for which data should be deleted (*e.g.*, everything between July 1-Oct. 31, 2016).

19.3 Output Settings

Before fielding a survey, the programming should specify how NubiS is to capture data – *i.e.*, what data to collect and whether to encrypt it. All such aspects are governed by NubiS' output settings.

Just as other settings, these can be set on a survey level as well as (partially) on a variable level. First, look at the survey-level settings:



The settings fall into three categories: what to include in outputted files (Visibility), what to capture and how (Storage), and how to output data (Format). You saw an example of a visibility setting when fills were discussed. Typically, data visibility is set to to hidden, which has the effect that no data collected is outputted. The other three visibility settings work in similar manners, but have an effect on documentation created by NubiS. You will learn more about them soon.

Looking at the storage settings, you will see a variety of options – with the encryption key perhaps as the most important. By default, NubiS stores all data in a non-encrypted fashion. To activate encryption, all that is needed is to enter a key here. There are no specific requirements that NubiS imposes on the key used (*e.g.*, in terms of length or characters), but needless to say the key should be sufficiently strong to protect stored data.

Three other relevant settings control storing screenshots, capturing so-called paradata and storing data with an applied input mask.

Screenshots are screen captures that NubiS can store. If activated, NubiS stores a screenshot of a question screen when it is outputted to the browser and when the respondent hits a button moving the survey back or forward. Such screenshots can be useful in case of a problem to view exactly what a respondent saw, or as a visual for a paper or presentation.

Paradata is data collected by NubiS in addition to the answers given, capturing such details as mouse movement, keyboard strokes, leaving the survey browser window, errors encountered by the respondent and so on. This type of information is useful to gain insight into how a respondent answered a question rather than the actual answer. Depending on the amount of respondent activity on a screen, the size of the collected paradata can be small or large. It may be that the post limit size of the server must be increased in order to allow for proper processing of all paradata. Alternatively, one may want to consider collecting paradata only on certain screens, if doing so on every question screen hurts server responsiveness. This can be accomplished by setting the Capture paradata option for specific variables and/or groups.

The store location setting controls where NubiS stores its data. By default this is set to 'Internal' but it is possible to store data externally as well. This setting is configurable on a type and variable level basis, and can be used for example to store an answer only in an external location. If that is the case, NubiS will not store the answer in any logs, paradata error check results, or screenshots. NubiS will require you to specify the name of the function that implements the intended external storage behavior. This function must be defined in customfunctions.php, and its name must be listed in the getAllowedExternalStorageFunctions array and not in the getForbiddenExternalStorageFunctions array. Together the arrays in these two functions act as a white list and black list for PHP functions to be used for external storage handling within NubiS.

The input mask setting is less intuitive, but has to do with the following: Suppose you added an integer variable and applied the US currency input mask to it. In the survey, this would mean if the respondent entered a number greater than 999, a comma automatically would be added after every third digit from the left. (*e.g.*, to show 1,215 instead of 1215). The question is whether, in the stored data, the comma should be included. Its inclusion means, for example, in an outputted Stata file, the data is represented by a string format variable; whereas excluding the comma means the data is included as a byte format variable, which has consequences for how it can be used in analysis.

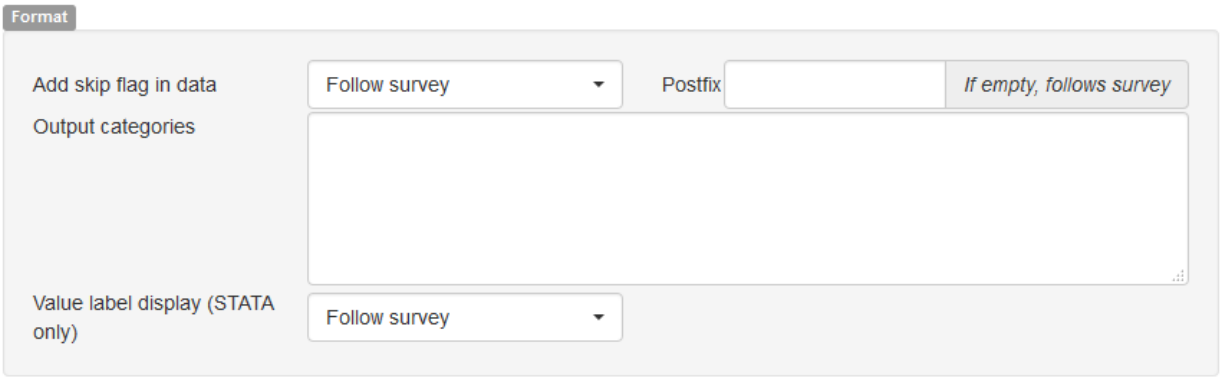
Less used are the settings Reset after completion if not keep answer and Keep answer after completion. These settings are only of use in very specific circumstances. *Keep answer after completion* determines whether the answer to a question must be kept as the survey is completed or whether it should be deleted. Meanwhile, *Reset after completion if not keep answer* enables/disables this behavior.

But why would one want to throw away data? Consider a long-term recurring survey in which a respondent is asked the same question multiple times; perhaps you intend to update basic background information on a quarterly basis. In such a survey, every three months a question such as "Did anyone else move in since last time we spoke?" would be asked – but the answer would not be kept over that interim. By default, all answers are kept for all variables (for obvious reasons, because otherwise respondents would answer but produce no data), but the setting can be specified per variable to override this standard behavior.

The *Format* settings control how to output data. One is whether NubiS adds skip variables. By default, when NubiS outputs data, it marks questions skipped by a respondent (*e.g.*, by not satisfying a condition) or not yet seen (in case of an incomplete interview) with a period, whereas questions left empty are marked with .e. With the skip variable parameter set to *Yes*, NubiS is told instead to create special skip variables (by default named variable name + _skip) and set them to *1* when a respondent skipped that question. Similarly, NubiS' behavior can altered when outputting checkbox variables. Standard, NubiS creates a binary variable for each answer option, setting it to *Yes* if it was selected and *No* if it was not. This can be changed using the *Checkbox output* setting so NubiS instead creates variables for those options, taking as their value the answer option code if selected and no value if not selected.

A third parameter that can be configured, *Value label display (STATA only)*, concerns radio button, checkbox, dropdown and multi-dropdown variables. NubiS can export those to Stata to show up only in tabulations or the data editor as well. For CSV export, this means that in Excel Only in tabs results in seeing just the numerical answer codes, whereas the default of including the data editor leads to seeing both the code and value label.

Finally, there is a type/variable level setting, *Value label display (Stata only)*, which can be leveraged for the aforementioned four types of variables: specifying alternative answer options to be used during export. For example, the value labels to be shown on screen to the respondent may be very long, and in the data file you would like them to be shorter. On the *Output* tab of a variable, labels can be defined that are shorter in the *Format* part:

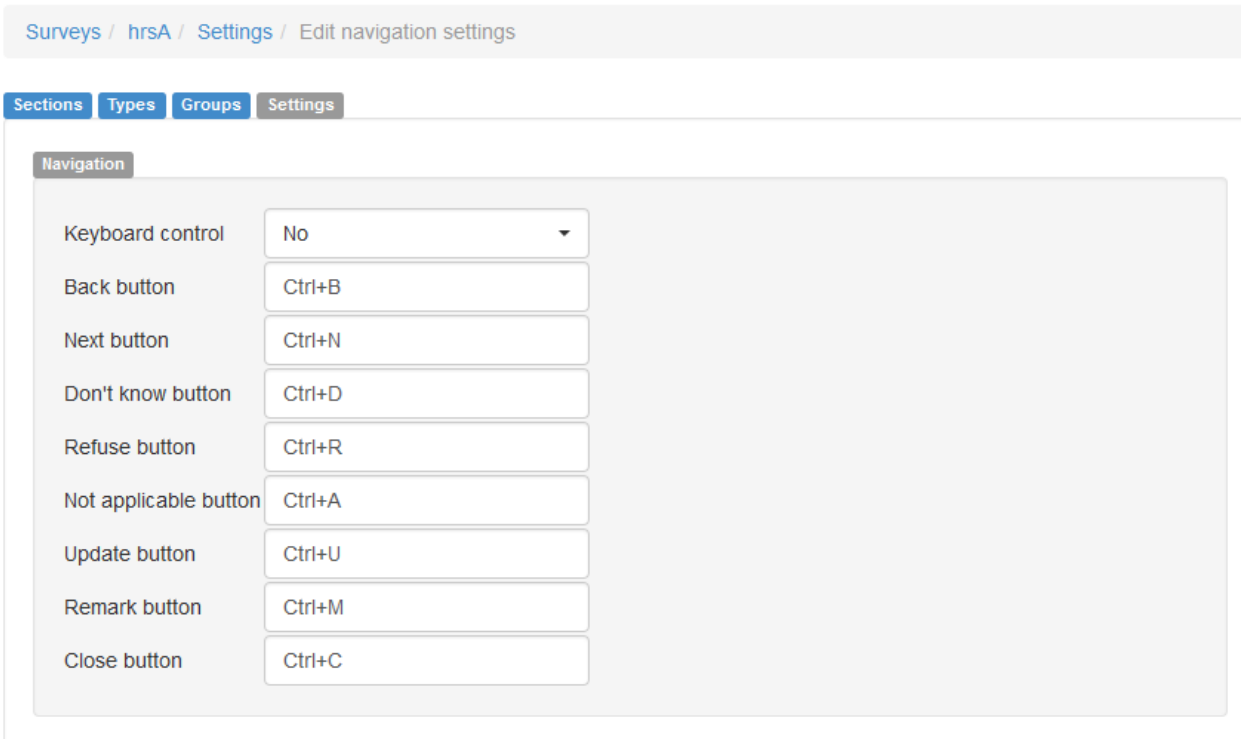


Please note this setting is not intended for re-coding variables. That is, if *1* meant, **"Yes, I have other people living with me,"** then it can define here that the outputted label should be **OTHER PEOPLE** for that answer option. However, *1* cannot be recoded to another value.

19.4 Configuring the Respondent Navigation Experience

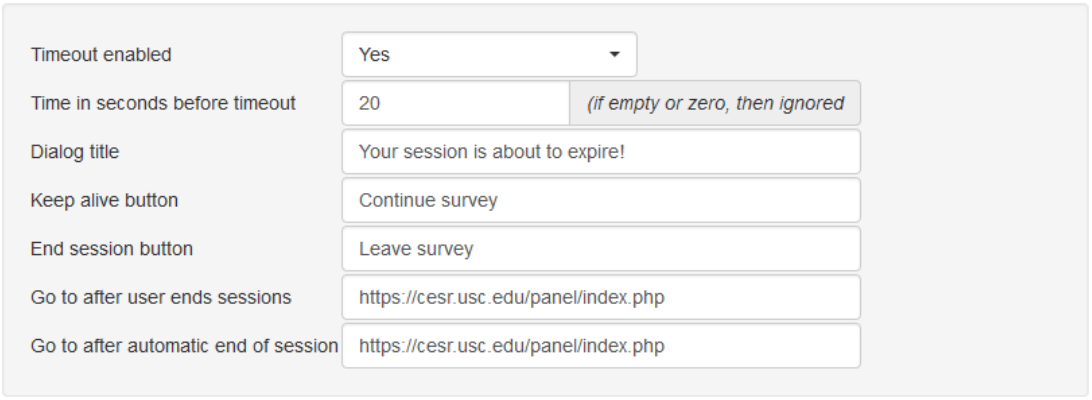
Another aspect that can be configured is how the respondent interacts with the survey. Usually, such interaction takes place via the mouse to select answers, click buttons and so on; on a tablet or mobile phone, this would be done through touch interaction. However, on some occasions navigational interaction might be too cumbersome. Particularly, in a face-to-face setting with an interviewer present, it can be more expedient instead to use keyboard-based shortcuts. (Or a mouse simply might not be possible if fieldwork is taking place in a remote area).

Keyboard-based interaction can be activated in the *Navigation* settings. You saw some of those when learning about *Don't know*, *Refuse* and *Not applicable*. On the same screen is a second set of settings:



These settings define keyboard shortcuts that can be used in place of clicking certain buttons. It should be observed that most browsers enforce their own keyboard shortcuts and do not allow those to be overridden. Most notably, Firefox allows a user to define shortcuts that override its own. As such, the above are of use in a controlled environment such as face-to-face or telephone-based interviewing, where the browser used for interviewing can be pre-determined.

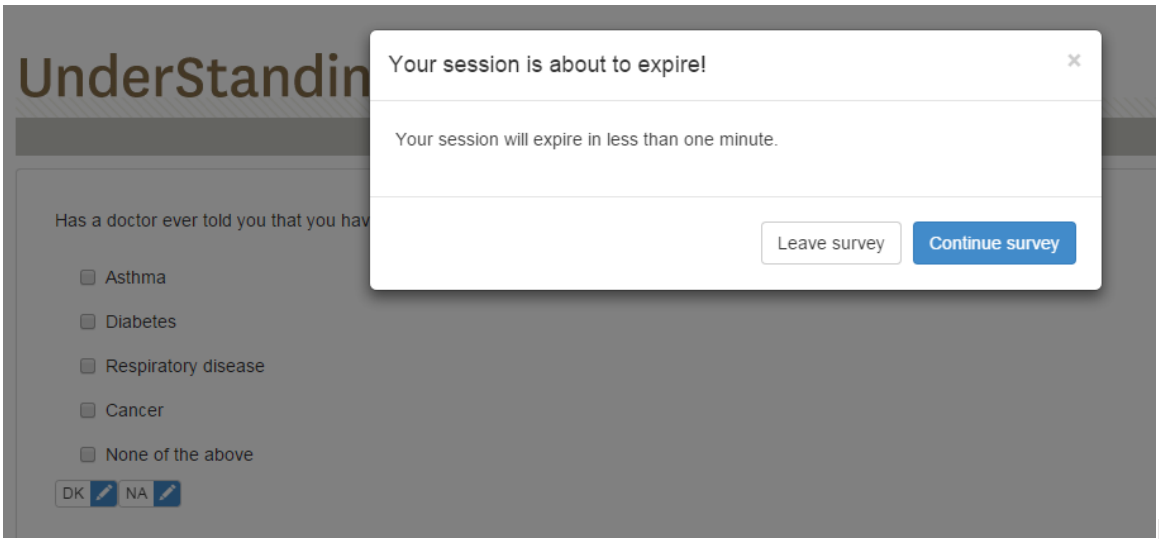
Related is the third set of *Navigation* settings, **Timeout enabled**, pertaining to whether NubiS should end a survey session after a specified span of time. After activating, you will see these settings:



As the above shows, several parameters can be defined:

- **Time in seconds before timeout**: instructs in seconds how long before a survey session times out.
- **Dialog title**: the title of the dialog showing the warning.
- **Keep alive button**: the label of the button to continue the survey session.
- **End session button**: the label of the button to end the survey session.
- **Go to after user ends session**: the location to go to if the user ends the survey session.
- **Go to after automatic end of session**: the location to go to if NubiS ends the survey session.

If this is activated with a timeout period of 20 seconds, once the survey is opened and nothing is done during that timespan, the following popup will appear:

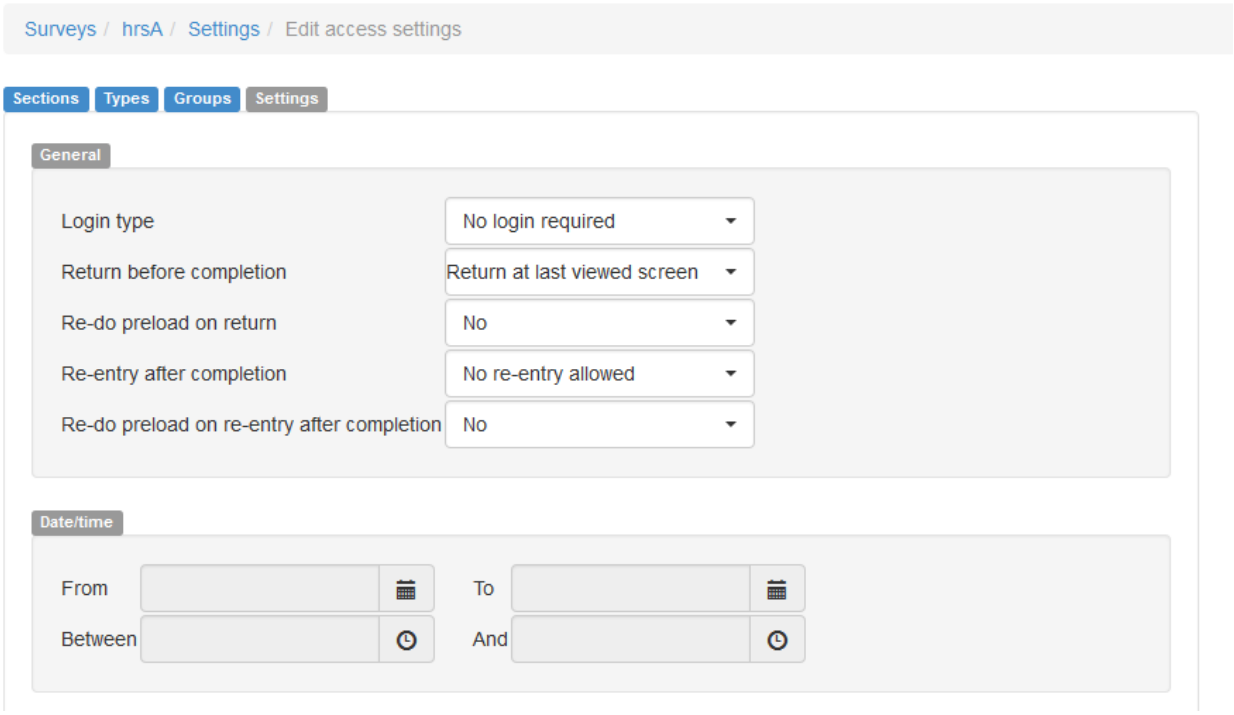


In this regard, "doing nothing" means NubiS detected no activity, such as typing in something, selecting an answer, moving the mouse and so on.

By default, NubiS will show the above warning when 60 percent of the specified timeout period has passed. This can be modified in the **root NubiS/config.php** file's function **sessionExpiredWarnPoint**. Similarly, you also can adjust how frequently NubiS should check for activity (default is every five seconds) by changing the function **sessionExpiredPingInterval**.

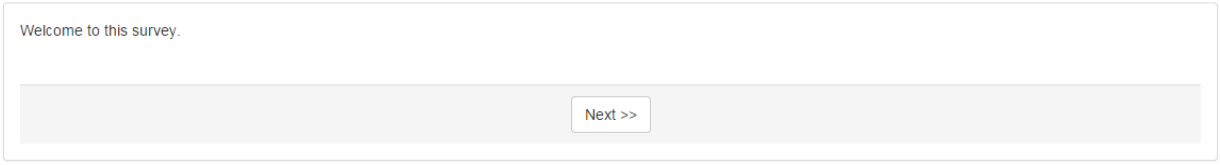
19.5 Setting up survey access

Another important part of preparing the survey for fielding is determining how respondents can access it. Under the *Access* settings, the following are available:



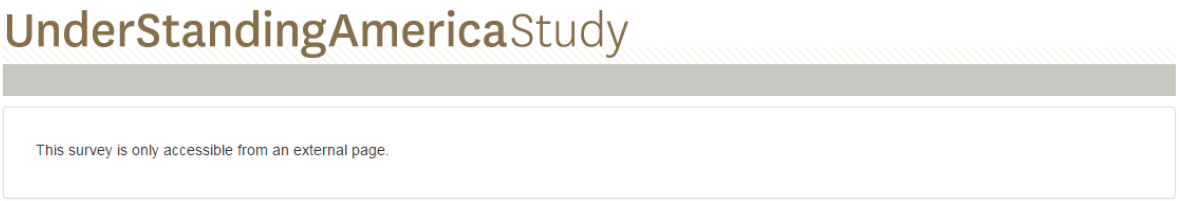
First of is the login type. Three options are available:

1. No login: If no login is required, anyone who opens the web site can access it. This is done by navigating to root nubis/index.php, which opens the default starting page of the survey:



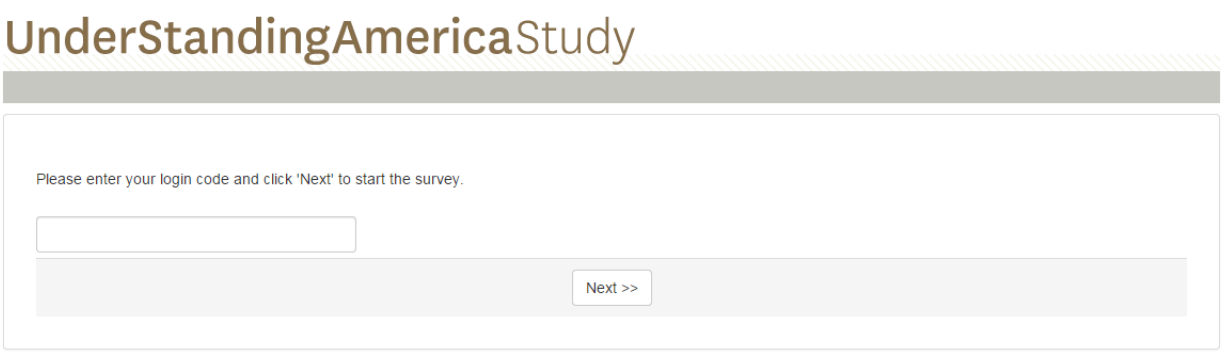
The text displayed here can be customized by changing the question text in the variable *Introduction* in the *Base* section.

2. Via external web site only: In this case, respondents only can access the survey from another web site. Trying to open the survey directly will lead to the following screen:



This text is defined in the *Direct* variable in the *Base* section.

- By login code: Respondents can log in to NubiS if they have a login code:



The text for the screen comes from the *Login* variable in the *Base* section. If an incorrect code is entered – that is, a code NubiS not find in its database – an error text is shown (which can be customized in the survey's *Assistance* settings). This option thus relies on the fact that a sample of respondents has been loaded into NubiS, which will be discussed shortly.

Another access setting is Return before completion. The default is At last viewed screen of survey, allowing respondents to continue where they had stopped. But this can be set to one of five other values:

1. No re-entry allowed: The respondent cannot leave the survey and return later.
2. At beginning of survey: Upon re-entering, the survey will start at the beginning. It will have retained any answers but starts from scratch. Moreover, any assignments and so on executed prior to the display of the first question screen will be carried out again.
3. At first screen of survey: Upon re-entering, the survey will start at the first screen. It will have retained any answers but, different from the previous option, any assignments and so on executed prior to the display of the first question screen will not be carried out again.
4. At last viewed screen of survey: Upon re-entering, the survey will show the screen resulting from the last action.
5. At last viewed screen of survey while redoing last action: Upon re-entering, the survey will show the screen resulting from the last action. If the last action was a group statement,

this allows, for example, *for* assignments to be redone and, potentially, a different screen to be shown. Obviously, this is not something you would want to do for each and every question screen in your survey. Rather, you would want to use it for specific groups.

6. At screen after last viewed screen of survey: Upon re-entering, the survey will start at the screen following the screen on which the respondent left the survey. This option is useful if, for example, the respondent stopped at a screen displaying a message such as "Please continue again tomorrow" and no *Next* button. The respondent, when re-entering, should not return to this screen, as there is no way forward. Instead, you want to show the next screen.

Related to the above is Re-do preload on return, which instructs NubiS to redo any preloading that would occur at normal survey start. Passing along an encoded string to NubiS can do such preloading when starting the survey from an external web site. For example, you could pass along values by stating the following PHP code as part of a HTML form submitted to start the survey:

```
$params = array("name" => "Respondent name");

$encoded = strtr(base64_encode(addslashes(gzcompress(serialize($params), 9))), '+/=', '-_,');

$strform = '<input type="hidden" id="pd" name="pd" value="' . $encoded . '">';
```

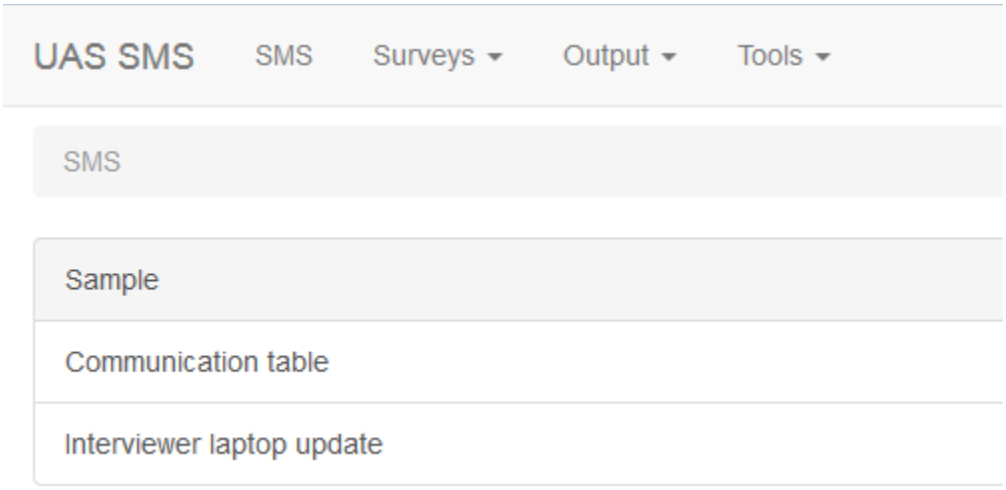
NubiS, when starting a survey, will check for a variable called pd and store any values it finds. In the above, it would set the *name* variable to Respondent name.

The next two access settings are similar, but instruct NubiS' behavior when a respondent returns after completing the survey. By default, such re-entry is not allowed – yet it can be set to the same three values as discussed above (with the small distinction that the respondent can return to the last screen viewed, but not any after it because that would constitute the survey's end). If re-entry is enabled, then it can be either with or without re-doing any preloads (achieved by configuring the setting Re-do preload on re-entry after completion).

Finally, you can set the time period when the survey will be accessible. In this context, NubiS will take midnight as the begin time and end time, unless the survey specifies access can only take place between certain hours. Note that both date and time checks here will be based on the date and time of the server on which NubiS resides. If a respondent attempts to access the survey outside the specified dates or time, NubiS will display the text defined in the *closed* variable in the *Base* section.

19.6 Adding A Sample

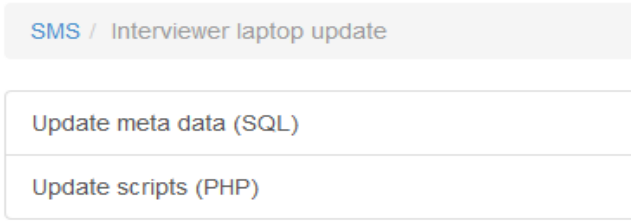
As you just saw, there are various ways in which respondents can access a survey in NubiS. With the exception of the anonymous mode, the other options require the presence of a sample. To this end, NubiS supports uploading a sample into the system by means of pre-formatted CSV files. This functionality is found in the SMS area of the interface, which can be access by clicking **SMS** in the top navigation bar. Doing so brings up the following screen:



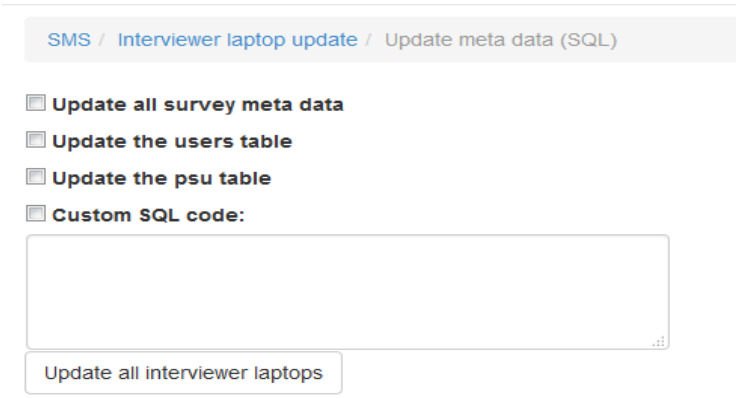
Clicking **Sample** will bring up the instructions on uploading a CSV file.

19.7 Managing Fieldwork

If you are planning to use NubiS in a face-to-face interview setting, there are several utilities available to manage the fieldwork. These are predominantly aimed at interacting with interviewer laptops (*e.g.*, to push out updates) or tracking progress. The **Communication table** and **Interviewer laptop update** options fall under the first category. Starting with *Interviewer laptop update*:



You have the option to push out database and/or code updates. Database updates constitute the following:



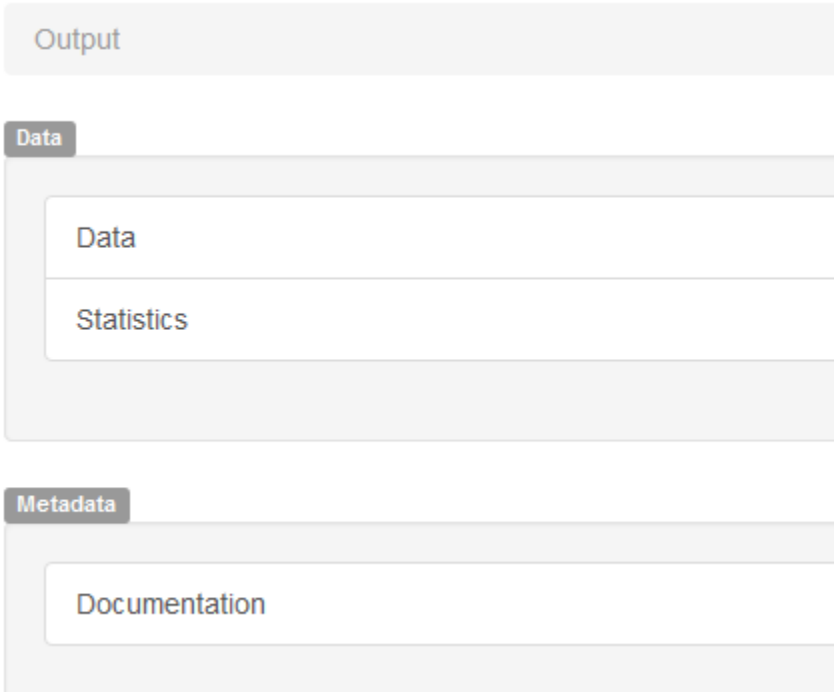
These are options to update the survey instrument, the users (for example, if a new interviewer laptop was added), the psu table (dividing the sample into sub-units) and custom SQL code (*e.g.*, to make other minor changes such as correcting erroneous data).

You also can update the code on interviewer laptops; that is, NubiS itself. This may be necessary if, for example, new custom functions were added.

Note: if you wish to turn off communication with interviewer laptops you can do so in Nubis' configuration file (conf.php) by setting 'CONFIGURATION_SAMPLE_ALLOW_COMMUNICATION' to '2' ('1' enables communication).

20. EXPORTING DATA

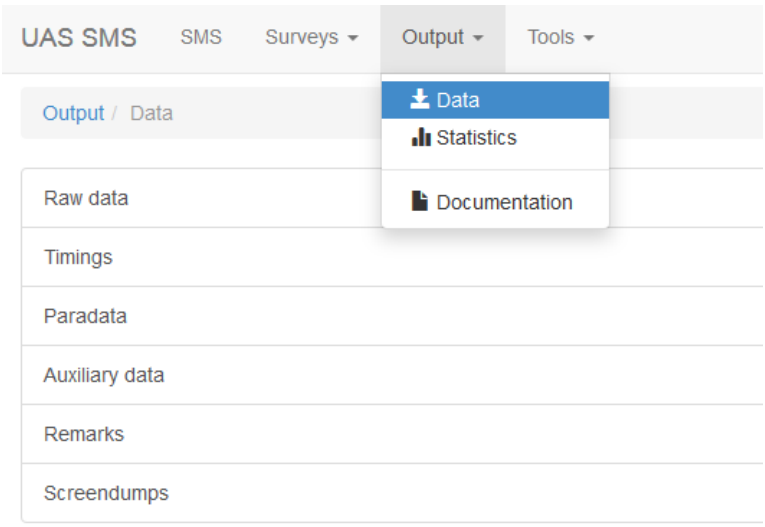
Through this point, you have learned how to program a survey, set up a sample and access, and prepare a survey for fielding. The next topic is how to collect data for analysis, for which NubiS provides options to generate real-time data and meta-data output related to the survey. (For information on data related to sample management, please refer to Sec. 19.) Data-collection settings are accessible through the *Output* menu:



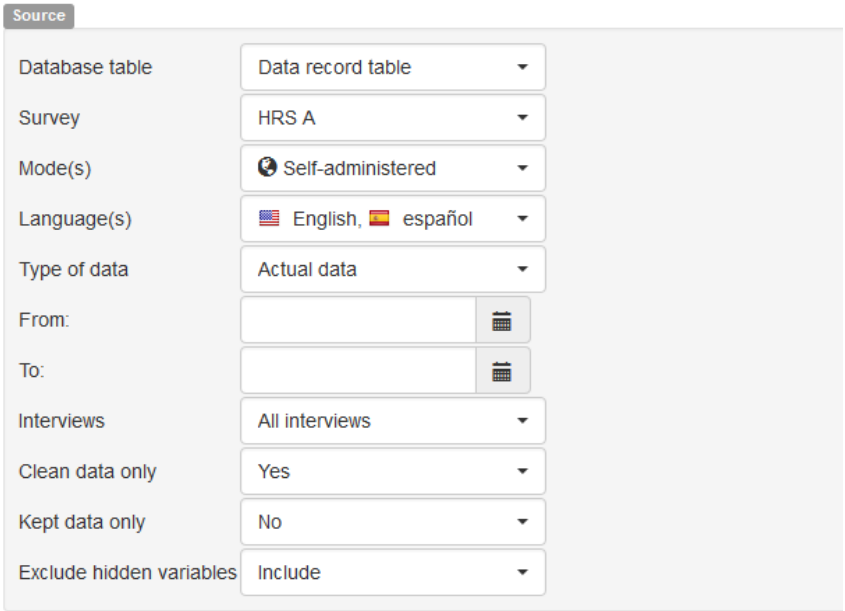
NubiS provides support to generate data files based on the raw data as well as aggregated statistics. For meta-data, the options are related to documentation.

20.1 Data

Data output in NubiS comes in the following forms:



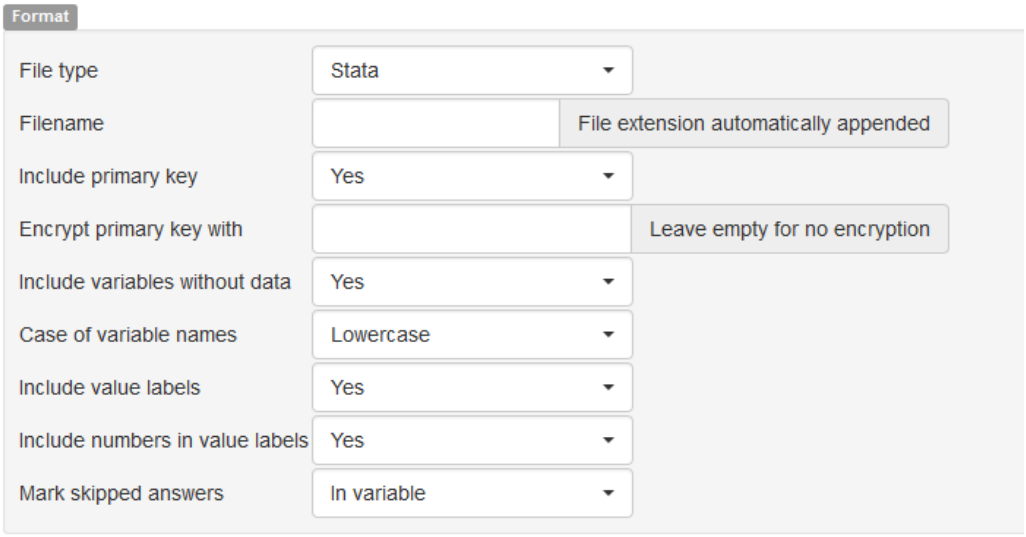
Raw data is the data directly collected through the survey in terms of answers to questions, flag variables being set and so on. On the **Raw data** screen, you can specify the output parameters NubiS will follow when generating a data file:



In part, these parameters concern from where data should be pulled and under which restrictions. First, you can choose whether to base the export on the data record or data table. In virtually all situations there is no difference, but if you were to insert data into the data table directly during your survey (and not through the usual NubiS internal storage mechanisms), you can only include that data by basing the export on the data table.

Next, you can choose the survey for downloading data, from which interview mode(s) and which language(s). By default, all available modes and languages are selected. You can choose whether you want to download real or test data, and whether to include all interviews or only those that were completed. You also can decide between clean data or dirty data. Dirty data is data that may contain answers that are no longer valid, because the respondent answered a question then went back in the survey and altered previous answers, invalidating their response. NubiS will not include such invalidated data if clean data is desired. Similarly, you can say whether you only want to have "kept" data; that is, data for variables for which answers are kept upon survey completion. Lastly, you can limit the number of outputted variables by excluding variables you instructed NubiS to hide in data output.

Upon deciding the source settings, you can specify how the data should be output:



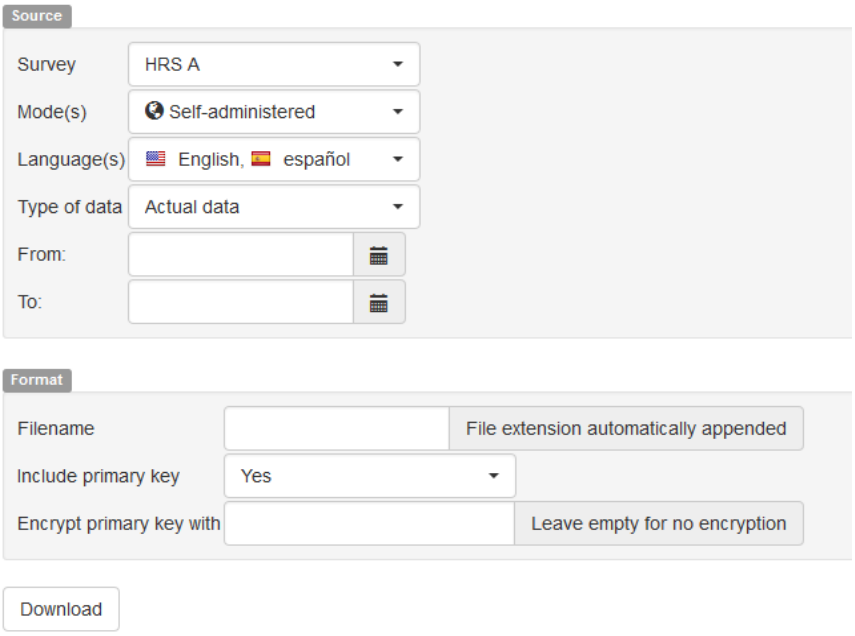
NubiS supports outputting data to Stata or CSV. If desired, you can specify a particular file name (if left empty, NubiS will generate one based on the survey title). Also, you can indicate whether to include the primary key and whether to encrypt it. You may wish to exclude the key or encrypt it if you don't want the person receiving the data to know the identifiers. You can define whether to include variables that have no data associated with them, whether variable names should be lower- or uppercase, and, for downloading a Stata file, whether to include value labels and, if so, whether to include the answer option codes in those labels.

Finally, you can instruct NubiS to mark skipped answers. To recall, a respondent can leave the answer to a question empty. If Mark skipped answers is set to *No*, then in the outputted data file this empty answer will be represented with a period in Stata and an empty value in CSV. But suppose a respondent never saw a question due to a skip pattern. When NubiS outputs data for

this question, it does not find a value and outputs it, again, as a period in Stata and an empty value in CSV. With no distinction between the two and unknown whether the respondent did not get the question or chose not to answer, this has consequences for analysis in terms of what can be done with empty values. To make a distinction, NubiS can mark empty answers given by the respondent. This can be done within the same variable, in which case they are outputted both in Stata and CSV as *.e*. This also can be done by creating skip variables, such as **Q1_skip** for *Q1* where the skip variable is set to *1* if the corresponding question was skipped, and, otherwise, to empty.

When the settings are to your liking, click **Download**. NubiS then will generate the data file real-time based on the raw data as it is at that moment. How long this takes depends upon the size of the survey and the number of interviews. When done, NubiS will prompt for the generated file to be downloaded.

You also can download a file containing detailed timings information:

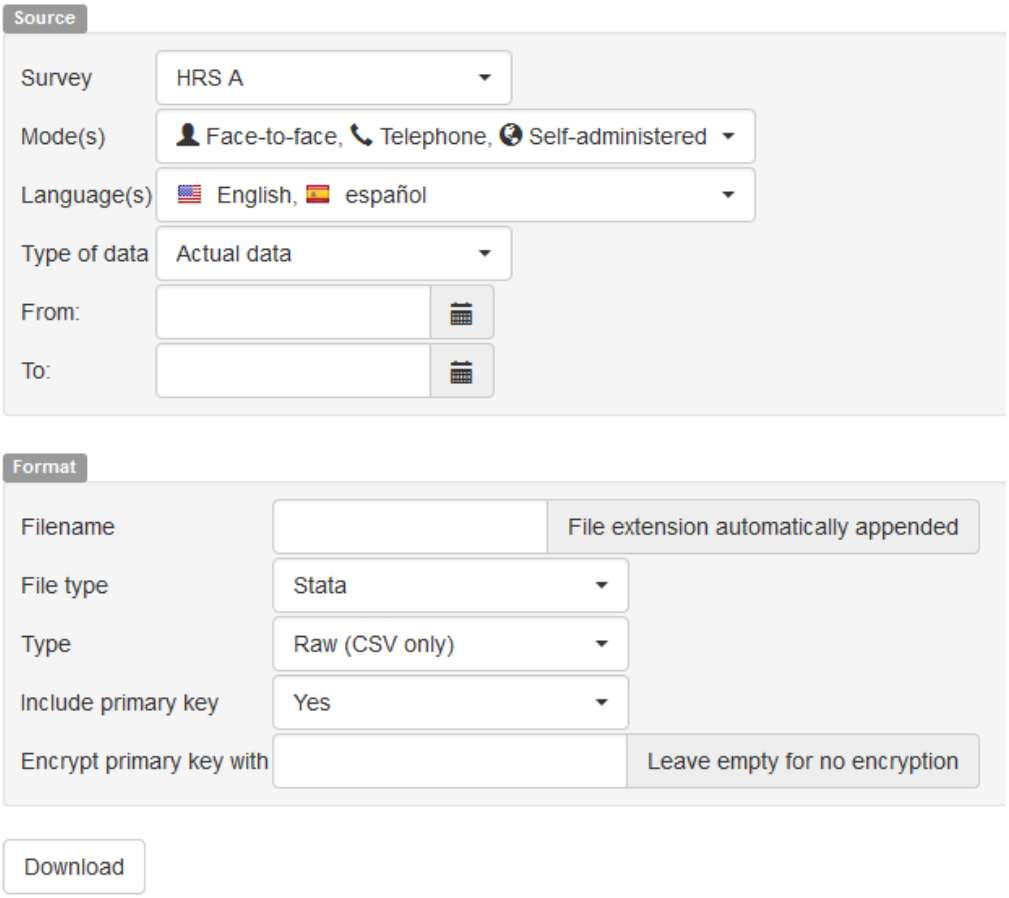


The main data sets you created already have begin and end time for each interview, but this might not give an accurate picture of how long someone took to fill out the survey. For example, they may have logged off and returned the next day, or went out for a moment and came back 10 minutes later. The timings file NubiS generates tells how long each respondent spent on each individual question screen. Based on that, it totals the overall time spent by the respondent on the survey. This information can be used, for example, to inform whether a respondent actually read the question. If the average time spent by all respondents is 45 seconds and someone answered

the question in three seconds, then it could be reasoned this individual did not properly read the question.

Note: the listing of time spent per individual question screen includes all captured time measurements. In contrast, the average time spent by a respondent and the average time spent on a question exclude any time measurement exceeding a five-minute period (to use a different cut off period, re-configure NubiS' *getTimingCutoff* function in config.php).

Other types of data files that can be downloaded contain paradata and auxiliary data (accessible by clicking, respectively, Paradata and Auxiliary data). The screen for downloading paradata is as follows:



Paradata is downloadable in one of three ways: 1) a raw listing, 2) an event counts listing or 3) an erroneous answers listing. The event counts listing are outputted as a Stata or CSV file, and it contains variables summarizing how often a particular type of error occurred for a specific question for an individual respondent. Also provided are variables capturing how often the respondent left the question screen and came back again (if applicable). The erroneous answers listing is outputted as a CSV file and lists all erroneous answers given per question per respondent.

The raw file contains all collected paradata strings. These strings follow this format:

||FI=1447279216935||ER1:answer1=1447279218058||ER1:answer2=1447279218058||MC:801:197:1:nextbutton=1447279218083

As the example illustrates, a paradata string comprises a series of actions separated by '||'. There are currently the following types of action:

- FI: browser tab displaying the survey received focus. It comes with timestamp in milliseconds passed since midnight of Dec. 31, 1969.

- FO: browser tab displaying the survey lost focus (*e.g.*, the respondent opened another browser tab or another application altogether). It is timestamped in milliseconds passed since midnight of Dec. 31, 1969.

- MC: mouse click on the survey browser tab. The first two numbers after MC indicate, respectively, the X- and Y-coordinates. The third number identifies the button pressed (*1* = left button, *2* = middle button, *3* = right button). The string after that identifies the component clicked (if the component's HTML element had a name attribute). Again, it is timestamped in milliseconds passed since midnight of Dec. 31, 1969.

- MM: mouse click on the survey browser tab. The first two numbers after MM indicate, respectively, the end X- and Y-coordinates. The string after that identifies the component clicked (if the component's HTML element had a name attribute). Lastly, it is timestamped in milliseconds passed since midnight of Dec. 31, 1969.

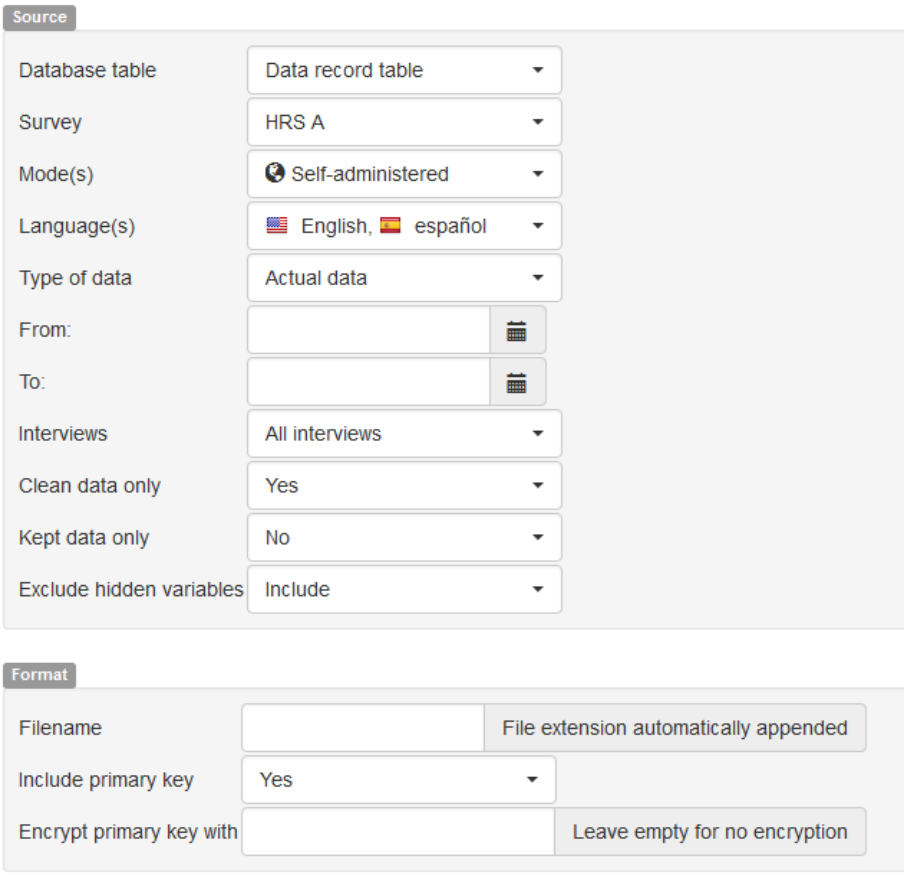
- ER: error recorded for a particular input box. Each error action comes with the component's name and a timestamp in milliseconds passed since midnight of Dec. 31, 1969. These names relate to the value of the displayed field in order of appearance. For example, *answer2* corresponds to the second field in the *displayed* list. The following errors currently are logged:

    - ER1: no answer given
    - ER2: text (string/open) question answer pattern not satisfied
    - ER5: text (string/open) question answer not enough characters
    - ER6: text (string/open) question answer too many characters
    - ER10: numeric question outside of range
    - ER11: custom question outside of range

- ER16: numeric question not a numeric answer
- ER17: integer question not an integer answer
- ER24: text (string/open) question answer too many words
- ER25: text (string/open) question answer not enough words
- ER27: set of enumerated question not enough options selected.
- ER28: set of enumerated question not exact number options selected.
- ER29: set of enumerated question too many options selected.
- ER30: set of enumerated question invalid sub-combination of options selected.
- ER31: set of enumerated question invalid combination of options selected.
- ER32: multi-dropdown question not enough options selected.
- ER33: multi-dropdown question not exact number options selected.
- ER34: multi-dropdown question too many options selected
- ER35: multi-dropdown question invalid sub-combination of options selected.
- ER36: multi-dropdown question invalid combination of options selected.
- ER37: more than one inline field answered.
- ER38: not all inline fields answered.
- ER39: not enough inline fields answered.
- ER40: too many inline fields answered.
- ER41: not exactly enough inline fields answered.
- ER42: option selected, but inline field not answered.
- ER43: more than one question in the group answered.
- ER44: not all questions in the group answered.
- ER45: not enough questions in the group answered.
- ER46: too many questions in the group answered.
- ER47: not exactly enough questions in the group answered.
- ER48: not all questions in the group have a unique answer.
- ER49: not all questions in the group have the same answer.
- ER50: invalid answer option code entered in enumerated text box.
- ER51: invalid answer option code entered in set of enumerated text box.
- ER52: question answer not equal to specified value.
- ER53: question answer equal to specified value

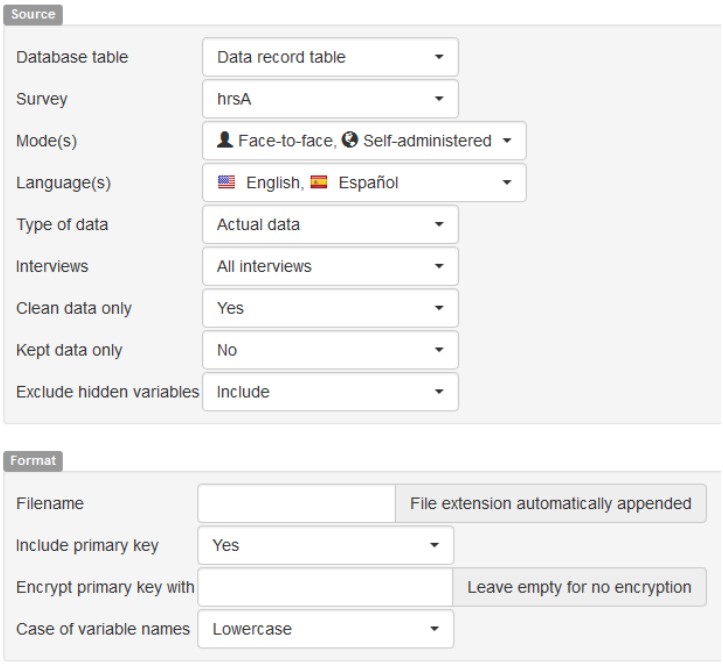
- ER54: question answer smaller than specified value.
- ER55: question answer smaller than or equal to specified value
- ER56: question answer greater than specified value
- ER57: question answer greater than or equal to specified value
- ER58: set of enumerated/multi-dropdown question answer not equal to specified value(s).
- ER59: set of enumerated/multi-dropdown question answer equal to specified value(s).
- ER60: set of enumerated/multi-dropdown question answer smaller than specified value.
- ER61: set of enumerated/multi-dropdown question answer smaller than or equal to specified value
- ER62: set of enumerated/multi-dropdown question answer greater than specified value
- ER63: set of enumerated/multi-dropdown question answer smaller than or equal to specified value
- ER64: string question answer not equal to specified value (case sensitive).
- ER65: string question answer equal to specified value (case sensitive).
- ER66: string question answer not equal to specified value (ignoring case).
- ER67: string question answer equal to specified value (ignoring case).
- ER68: date/datetime question answer not equal to specified date.
- ER69: date/datetime question answer equal to specified date.
- ER70: date/datetime question answer before specified date.
- ER71: date/datetime question answer before or equal to specified date.
- ER72: date/datetime question answer after specified date.
- ER73: date/datetime question answer after or equal to specified date.
- ER74: time question answer not equal to specified time.
- ER75: time question answer equal to specified time.
- ER76: time question answer before specified time.
- ER77: time question answer before or equal to specified time.
- ER78: time question answer after specified time.
- ER79: time question answer after or equal to specified time.

The screen for obtaining an auxiliary data file does not differ much from those seen so far:



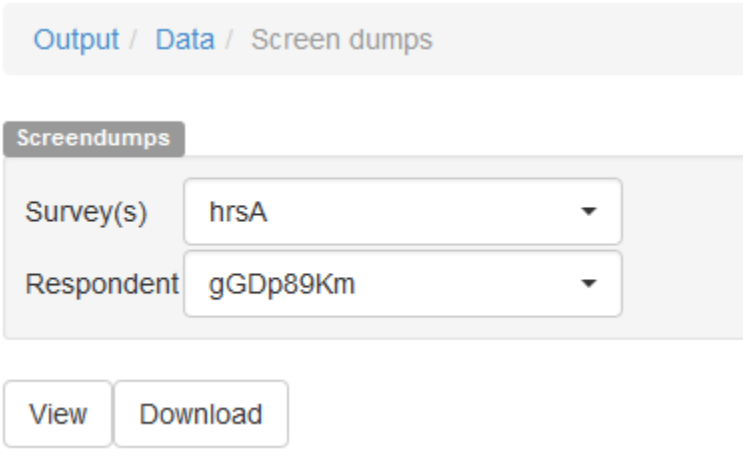
Like for raw data files, you can define the conditions for the source data to use, as well as the output conditions. NubiS will generate a CSV file for downloading, and the file will show per variable per respondent in what interview mode and language it was collected.

A different form of data is remarks provided by respondents:

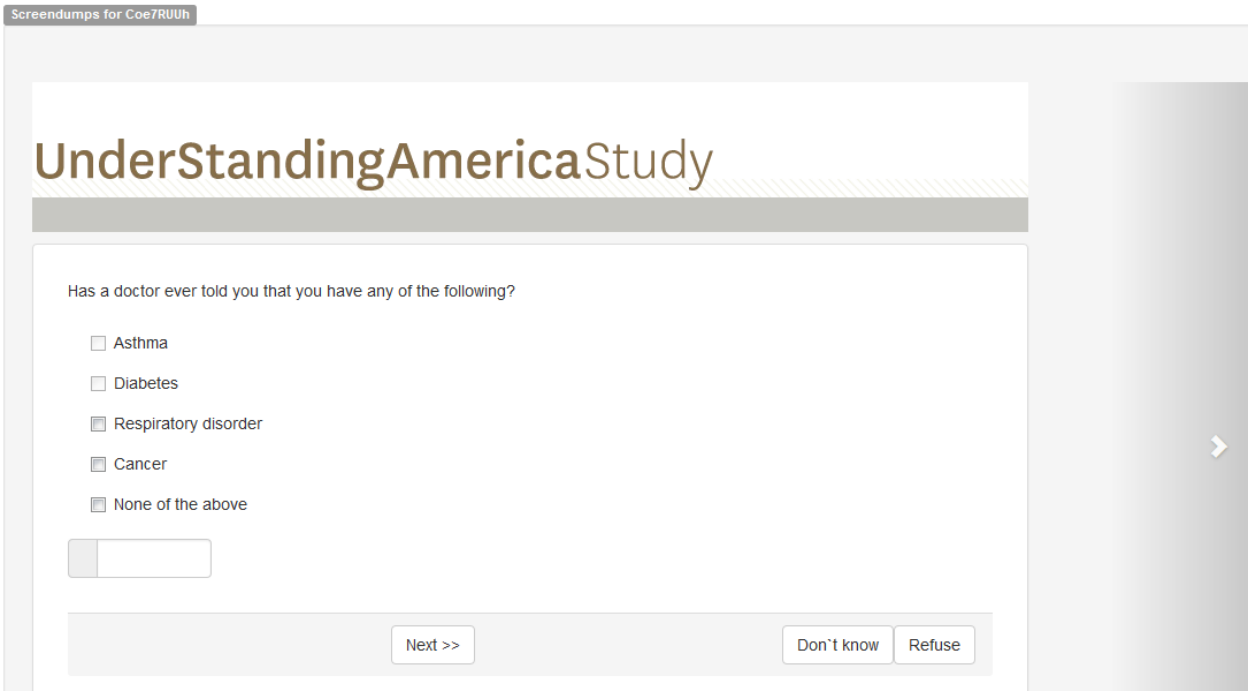


These are comments provided through the *Remark* mechanism discussed in Sec. 17.7. As before, you can choose the interview modes, languages and so on.

Finally, you can view or download screen dumps. These are screenshots captured by NubiS (if activated) during a survey:



You can choose to view the screenshots for an interview or download them in a single HTML file. If you select a case and choose view, you will see:



Beyond NubiS' standard options to extract data, it always is possible to pull data directly from the database. The following tables are used to store data that can be accessed readily through SQL queries:

- _data: contains all the raw data collected. Each entry comprises the following fields:
    - suid: identifies the survey.
    - primkey: identifies the primary key.
    - variablename: identifies the variable for which data is stored.
    - answer: the answer. May be encrypted.
    - dirty: indicates whether the entry is dirty data or not.
    - language: indicates the language of the survey for the stored value. May differ within the same interview if the respondent switched languages.
    - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched modes.
    - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched versions.
    - completed: indicates if the data is part of a completed survey
    - ts: the time at which the data was stored.
- _times: contains all the timing related entries. Each entry comprises the following fields:

- suid: identifies the survey.
- tmid: an auto-generated database table key.
- primkey: identifies the primary key.
- rgid: identifies the routing line to which the entry corresponds. It takes the same value for variables that were presented in a group on the same screen. As such, it can be used when calculating the total time spent by a respondent without double-counting such variables.
- variable: identifies the variable for which data is stored.
- begintime: the time at which the variable appeared on the screen.
- endtime: the time at which the screen was left (*e.g.*, by the respondent clicking *Next*).
- timespent: the period of time in which the variable was visible on the screen.
- language: indicates the language of the survey for the stored value. May differ within the same interview if the respondent switched languages.
- mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched modes.
- version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched versions.
- ts: the time at which the data was stored.

- _paradata: stores any captured paradata. Each entry comprises the following fields:

  - suid: identifies the survey.
  - pid: an auto-generated database table key.
  - primkey: identifies the primary key.
  - stateid: identifies the screen of which the screenshot was taken. Can be linked to the _state table to find out which questions were being displayed.
  - rgid: identifies the routing section line number corresponding to the action associated with the displayed screen.
  - displayed: identifies the variables displayed on the screen. If there are multiple variables, they are separated by ~.
  - paradata: identifies the paradata captured. Each paradata string follows the same format as described before.
  - language: indicates the language of the survey for the stored value. May differ within the same interview if the respondent switched languages.

- mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched modes.
- version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched versions.
- ts: the time at which the data was stored.

- _screendumps: contains any stored screenshots. Each entry comprises the following fields:
  - suid: identifies the survey.
  - scdid: an auto-generated database table key.
  - primkey: identifies the primary key.
  - stateid: identifies the screen of which the screenshot was taken. Can be linked to the *_state table* to find out which questions were being displayed.
  - screen: the screenshot made. May be encrypted and is always compressed with PHP's gzcompress.
  - language: indicates the language of the survey for the stored value. May differ within the same interview if the respondent switched languages.
  - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched modes.
  - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched versions.
  - ts: the time at which the data was stored.
- _observations: contains any remarks made via the Remark mechanism. Each entry comprises the following fields:
  - suid: identifies the survey.
  - primkey: identifies the primary key.
  - stated: identifies the screen of which the screenshot was taken. Can be linked to the *_state table* to find out more details.
  - displayed: lists the questions being displayed with which the remark is associated.
  - remark: the remark itself. May be encrypted.
  - language: indicates the language of the survey for the stored value. May differ within the same interview if the respondent switched languages.
  - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched modes.
  - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched versions.

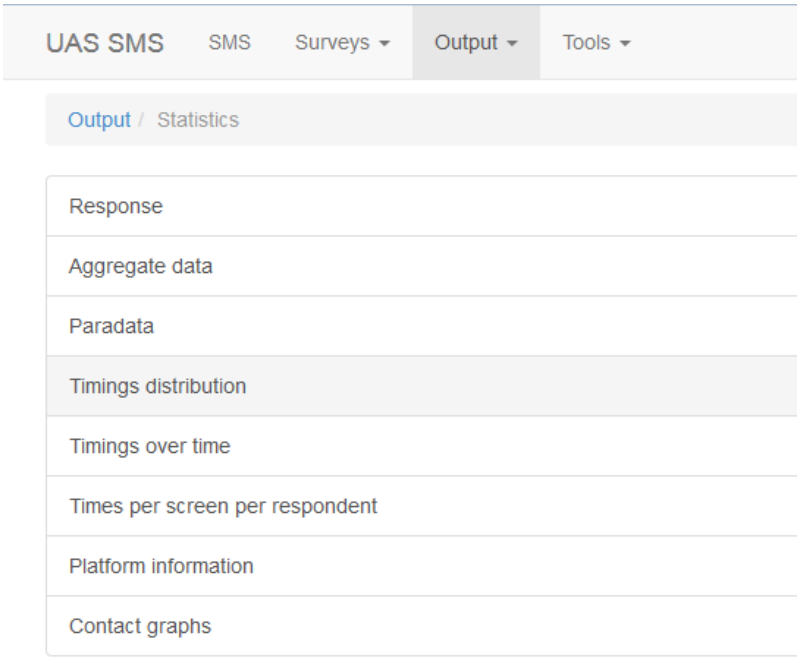
    - ts: the time at which the data was stored.
- _logs: contains all the raw data collected, including any answers changed during the survey (*i.e.*, all data instead of the final data stored in *_data*). Each entry comprises the following fields:
    - lgid: an auto-generated database table key.
    - suid: identifies the survey.
    - primkey: identifies the primary key.
    - variablename: identifies the variable for which data is stored.
    - answer: the answer. May be encrypted.
    - dirty: indicates whether the entry is dirty data or not.
    - language: indicates the language of the survey for the stored value. May differ within the same interview if the respondent switched languages.
    - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched modes.
    - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched versions.
    - ts: the time at which the data was stored.
- _actions: contains entries for each and every action made in NubiS from navigating through the survey to logging on to adding a contact. Each entry comprises the following fields:
    - asid: an auto-generated database table key.
    - sessionid: an auto-generated session id.
    - suid: identifies the survey. May be absent if the action is not related to filling out a survey.
    - primkey: identifies the primary key. May be absent if the action is not related to filling out a survey.
    - urid: identifies the user responsible for the action. May be empty if the action relates to filling out a survey.
    - ipaddress: identifies the IP address from which the action originated
    - systemtype: indicates the type of interaction with NubiS. *1* conveys a survey interaction; *2*, an administrative interaction.
    - actiontype: identifies the type of action. May be textual or numeric. Survey-related actions are numeric and correspond to the following codes:
        - 1 -> entry into a question screen.

- 2 -> exit out of a question screen using the *Back* button.
- 3 -> exit out of a question screen using the *Next* button.
- 4 -> exit out of a question screen using the *Don't know* button.
- 5 -> exit out of a question screen using the *Refuse* button.
- 6 -> exit out of a question screen using the *Not applicable* button.
- 7 -> exit out of a question screen using the *Update* button
- 8 -> exit out of a question screen using the *Change language* drop down.
- 9 -> exit out of a question screen using the *Change mode* drop down.
- 10 -> exit out of a question screen using the *Change version* drop down.
- 11 -> exit out of a question screen using a programmatic trigger.
- 12 -> initialization of the survey engine
- 13 -> entry into the survey (survey start).
- 14 -> re-entry into a non-completed survey.
- 15 -> end of the survey (survey completion).
- 16 -> return into a completed the survey.
- 17 -> left the survey window (survey window lost focus)
- 18 -> entered the survey window (survey window gained focus)

- action: identifies the specific action. In the case of survey actions, this identifies the routing number to which the action corresponds.
- params: provides a serialized representation of the $_POST array as submitted with the action.
- language: indicates the language of the survey for the stored value. May differ within the same interview if the respondent switched languages.
- mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched modes.
- version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched versions.

20.2 Statistics

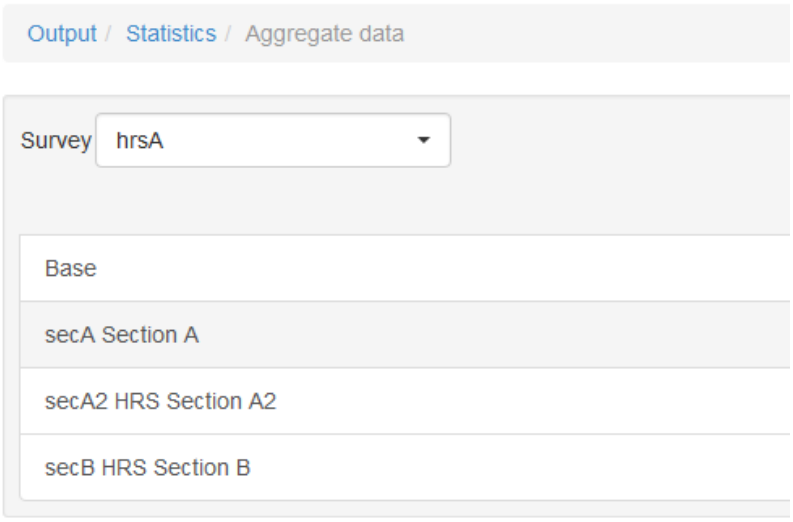
You can ask NubiS to provide aggregated data to get a quick sense of the data. Located under Statistics are the following options:



Under **Response** is the response rate for the survey:

Similarly, in **Contact graphs** are how many contacts were made during telephone interviewing.

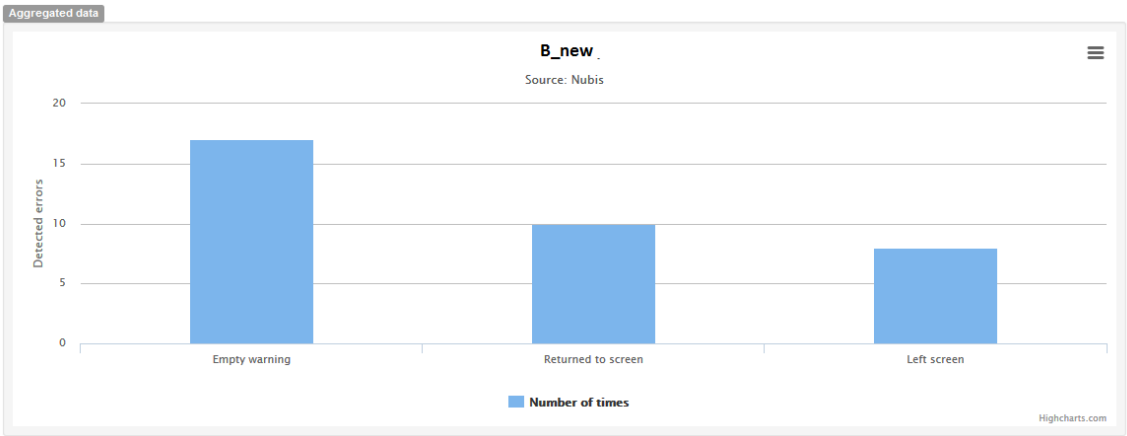
You also can get quick aggregate statistics for variables by navigating to a specific question:



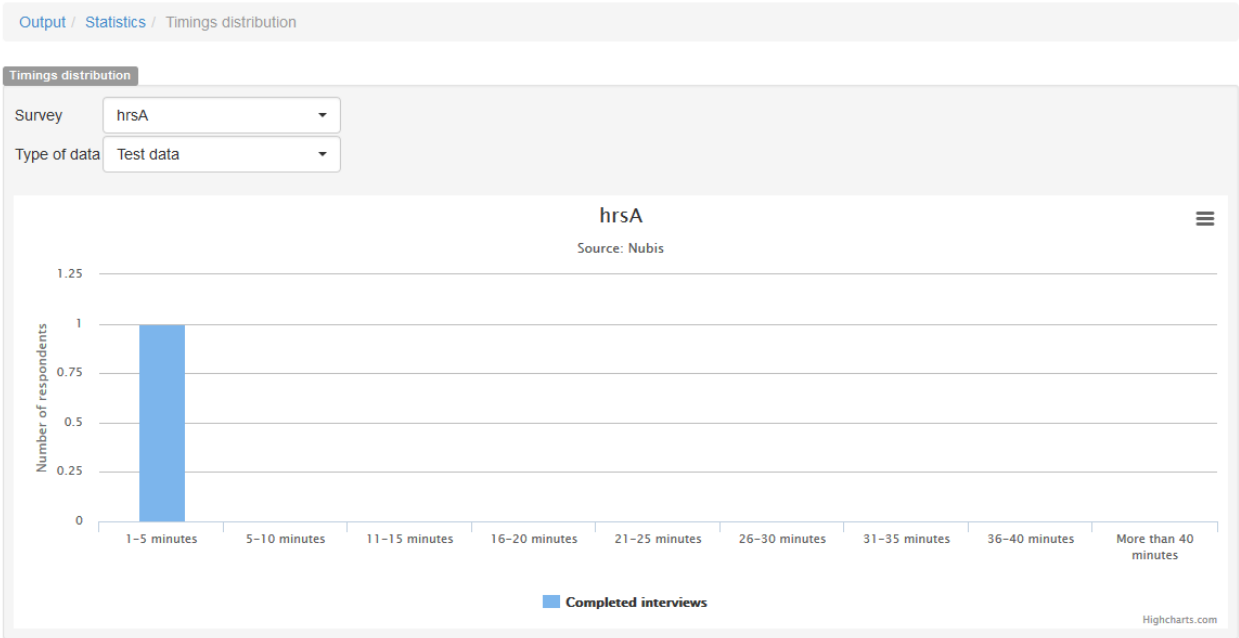
Selecting *secB* and *B_new* then will get you this, for example (depending on the data present, of course):

Though such graphs cannot replace detailed analysis, they can provide a quick sense of the distribution. Note that if the selected variable is an array, then a drop down will be included in the *Details* section to specify for which instance NubiS should show aggregated data.

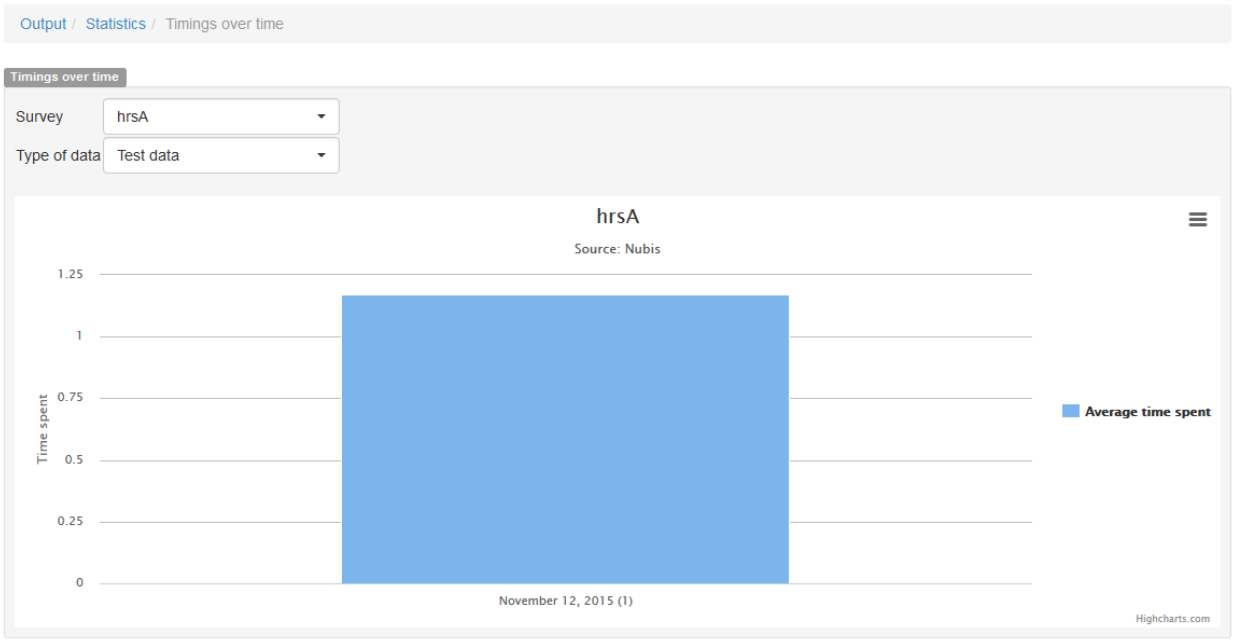
Complementary to such quick data overview are the paradata overview. These paradata capture items like the number of times respondents saw an ‘please answer the question’ warning or ‘please enter a number within the specified range’.



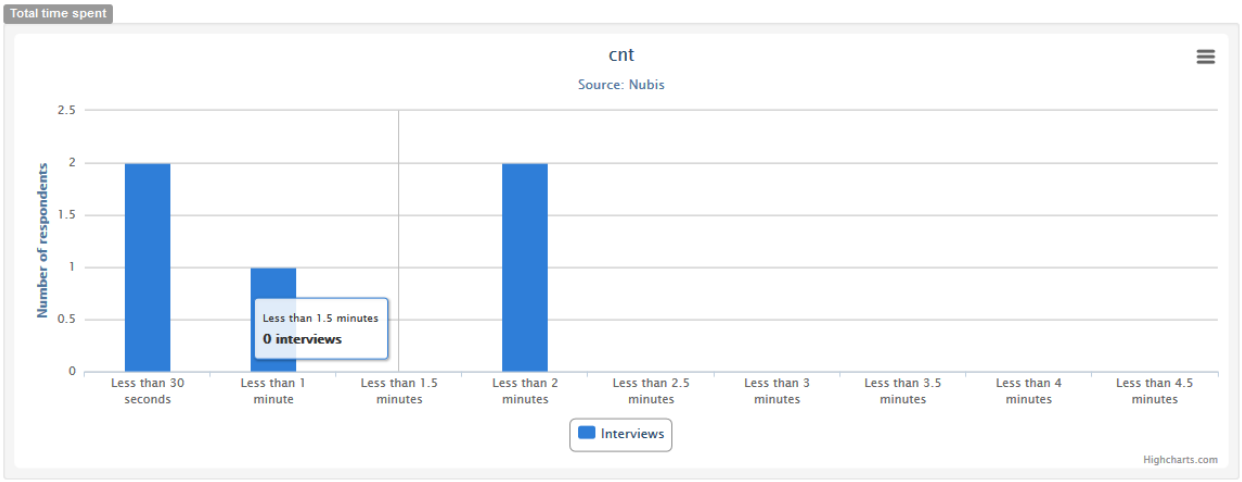
Another available option is to look at the time spent by respondents. The overall timings distribution can be accessed under **Timings distribution.** (Note that only completed interviews are included in order to have a reliable time estimate.)



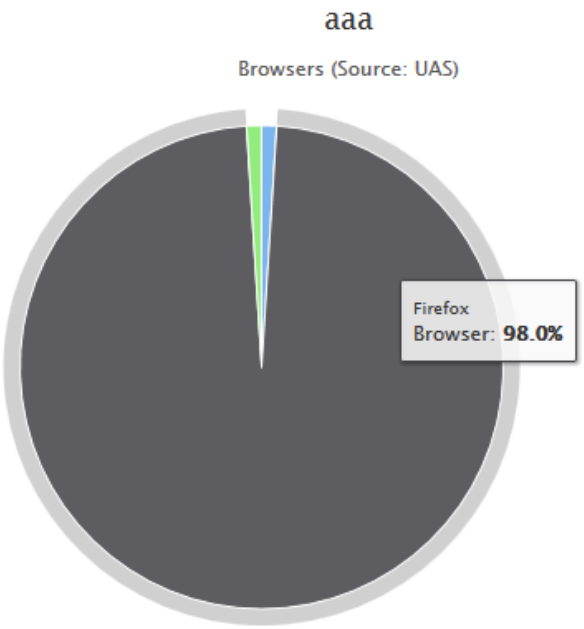
A different perspective on the same issue is to look at the timings over time. This shows the average time spent by respondents per day. This can reveal, for example, if the time spent on the survey goes down as the fielding period increases:



It's also possible to view respondents' time spent per variable. A graph conveying this information is displayed beneath the aggregate data graph for that variable:

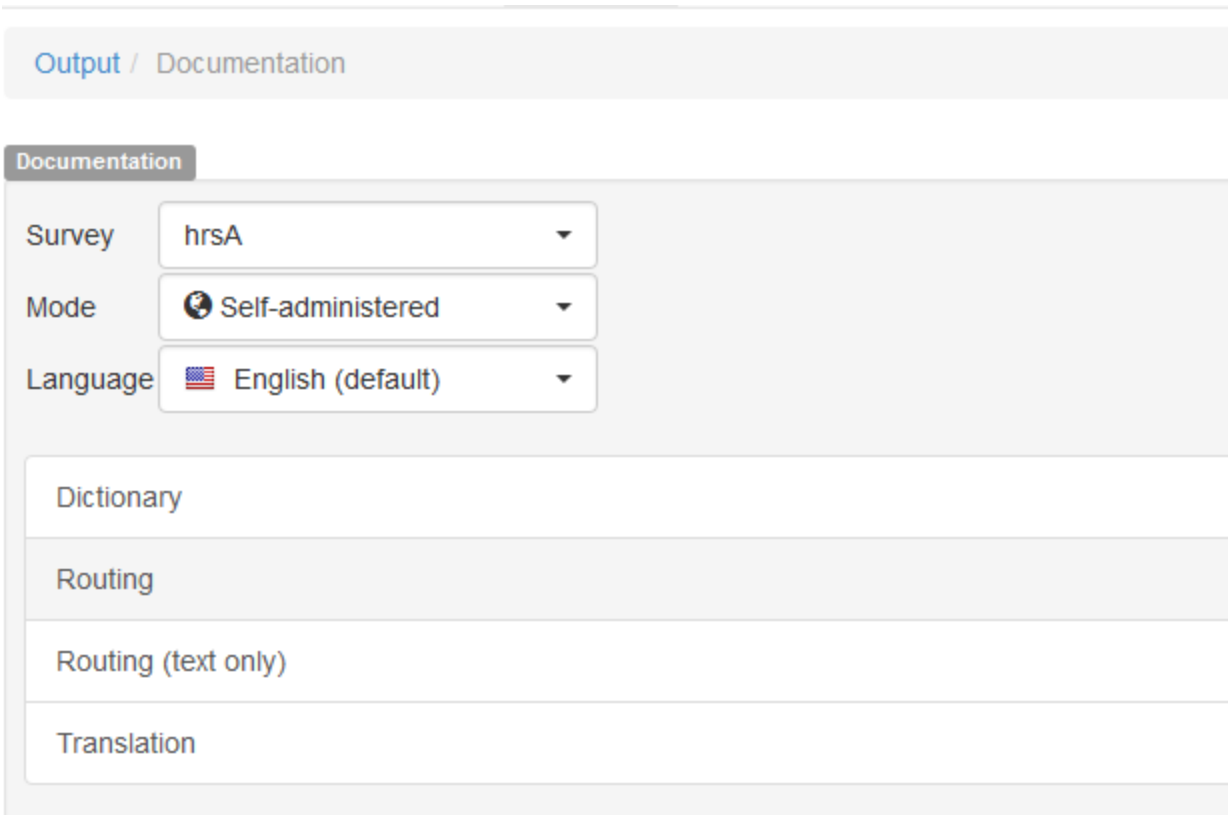


A last interesting series of graphs is to look at Platform information. Based on the user agent strings provided by respondents' browsers, these graphs give insight into the type of device, browser and operating system used by those taking the survey. (Note that the platform information graphs include both incomplete and completed interviews).



20.3 Meta-Data

Besides data, you can ask NubiS to provide meta-data. This is data about the survey, rather than data collected by the survey. The following types of meta-data are available:



The data Dictionary provides a list of all the variables in the survey in terms of their question text, answer options and label, and can be useful for understanding the variables in the data set. The Routing documents also can assist, either with graphical markup or plain text. It provides a one-on-one, more human-friendly version of the routing as it has been programmed for the survey. As such, the routing document is of use to understand and implement analysis code for skip patterns and so on.

Lastly, the Translation document gives a detailed overview of all the text in the survey that might require translation. It can be used, for example, as the basis for a first translation or to compare the original language and the translation. For the latter, you can choose which interview mode and language to generate the translation document (Note that switching these has no effect for the routing document, as it is mode -and language-independent.)

21. USER MANAGEMENT

While this manual itself focuses mainly on activities related to programming a survey, also mentioned has been other activities such as translation and testing. These different activities have been logically grouped and, when bound together, constitute user types. NubiS currently provides the following roles:

- System administrator: the focus of this manual, the NubiS SysAdmin Manual. A system administrator has access to all functionality to developing a survey including (but not limited to) programming, testing, translation, sample management and user management.

- Translator: the focus of the NubiS Translator Manual. A translator has access to all functionality needed for translating a programmed survey and testing these translations.

- Tester: the focus of the NubiS Tester Manual. A tester has access to all functionality needed for testing a programmed survey and reporting any problems.

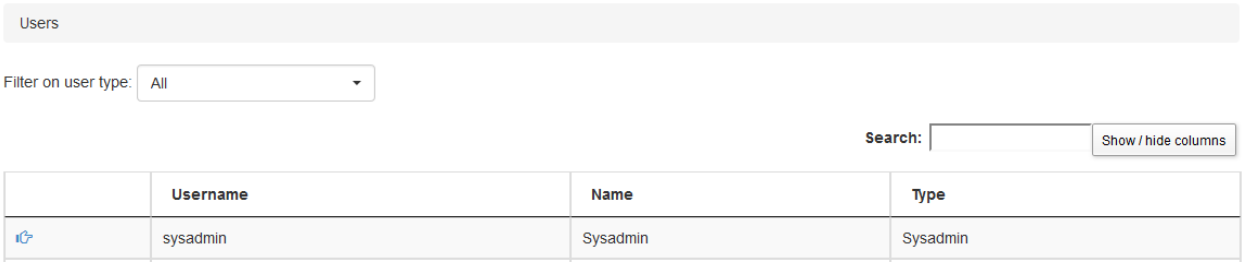
- Interviewer: the focus of the NubiS Interviewer Manual. An interviewer has access to all functionality needed for managing assigned interviews, in terms of adding contacts or conducting interviews.

- Supervisor: the focus of the NubiS Supervisor Manual. A supervisor has access to all functionality needed for assigning interviews, monitoring fieldwork progress and so on.

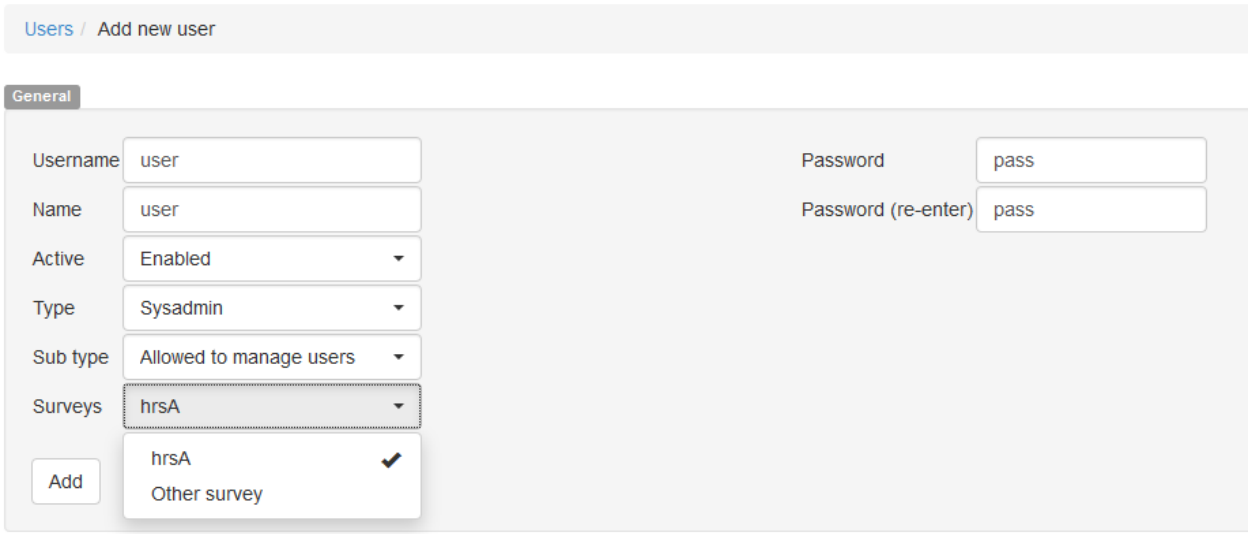
- Nurse: the focus of the NubiS Nurse Manual. A nurse has access to all functionality needed for performing data entry of, for example, biomarker data.

- Researcher: the focus of the NubiS Researcher Manual. A researcher has access to all functionality needed for downloading data and documentation.

The management of the above lies with system administrator(s) plus those granted permission to do so. That is, only a system administrator can create, edit or remove accounts. This can be accessed through the user management facility in NubiS:



As learned in Sec. 19.1, a user can be added using the *Add new user* link:



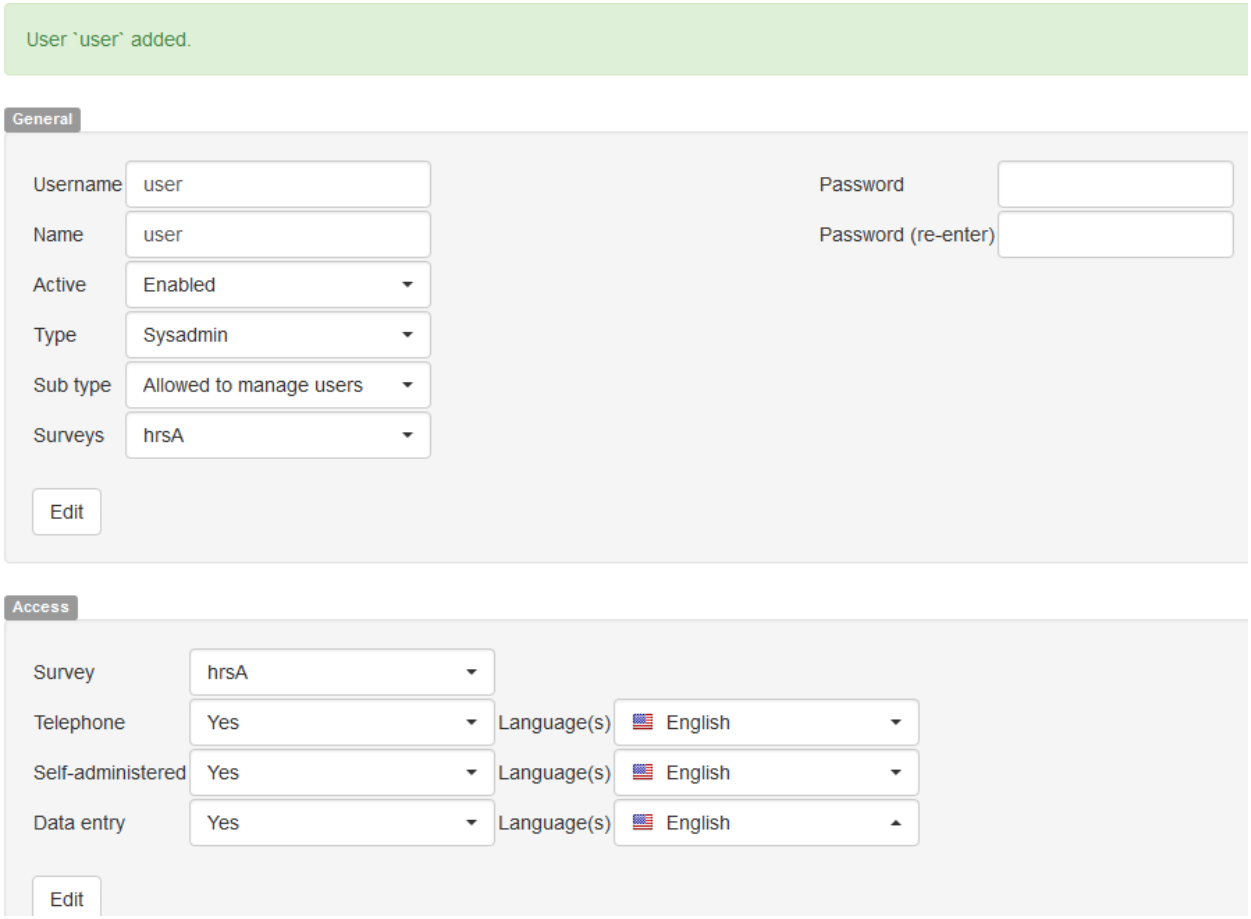
Adding a new user involves specifying a username/password combination. You also can specify a name to make it easier to distinguish between accouts. Next, you can indicate whether the account is active. For example, over time an account may become obsolete. One option is to remove it altogether, but another solution is to simply disable it so it can be re-activated if needed.

Next is the choice of user account type. If the type chosen is that of system administrator, translator or tester, you must specify the level of access in terms of interview modes and languages. Levels of access take on slightly different meanings depending on the account type, but all enforce access limited to selected interview modes and languages. For example, a translator for whom only Spanish was selected can see English during translation but will not be able to edit it. Similarly, a tester for whom only face-to-face mode was selected cannot test the survey in the self-administered interview mode.

If, instead, if a nurse account was chosen, you can specify if it is, for example, a head nurse. Head nurses have access to more functionality than other nurses. (For more information, please refer to the NubiS Nurse Manual.) Similarly, for a sysadmin account, you can indicate whether the account will be allowed to access user management. This allows you to create sysadmin accounts while at the same restricting user management. Lastly, you can pick the surveys to which the

account(s) will have access. This enables an account to have access to for example our hrs survey, but not the other survey.

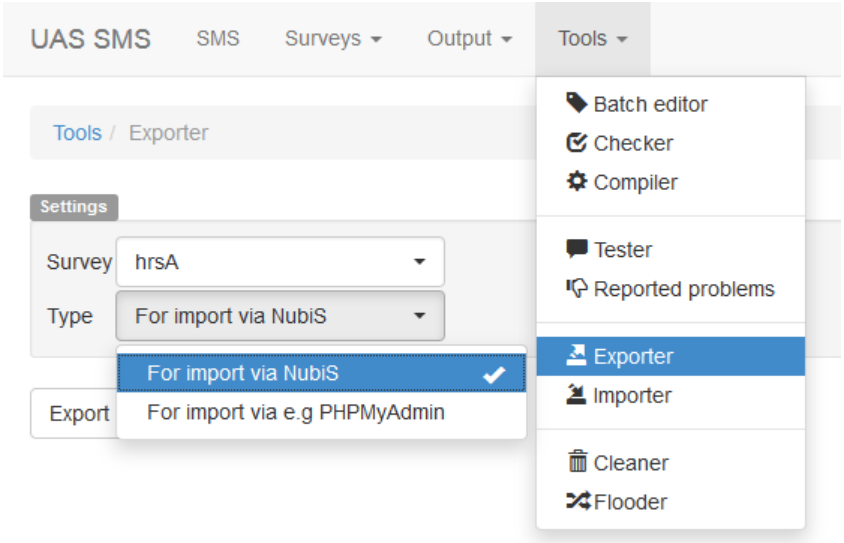
Once the account is added, the following confirmation screen appears:



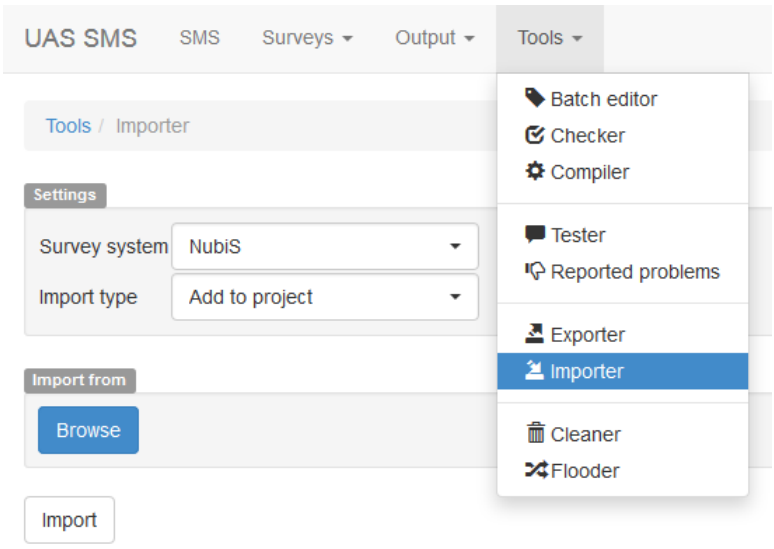
Notice the additional *Access* section below. This appears for sysadmin, researcher, tester and translator accounts, and provides the means to specify to which interview modes the account has access; and per mode. The modes and languages shown are those selected for the survey in the *Interview mode* and *Language* settings screen. To toggle between surveys, use the *Survey* dropdown (assuming the account has access to multiple surveys, as specified in the *General* section).

22. IMPORTING AND EXPORTING SURVEYS

Sometimes it may be required to move a survey in one NubiS project to another. To facilitate this, NubiS comes with an exporter and importer. The exporter can be used to generate two types of export files:

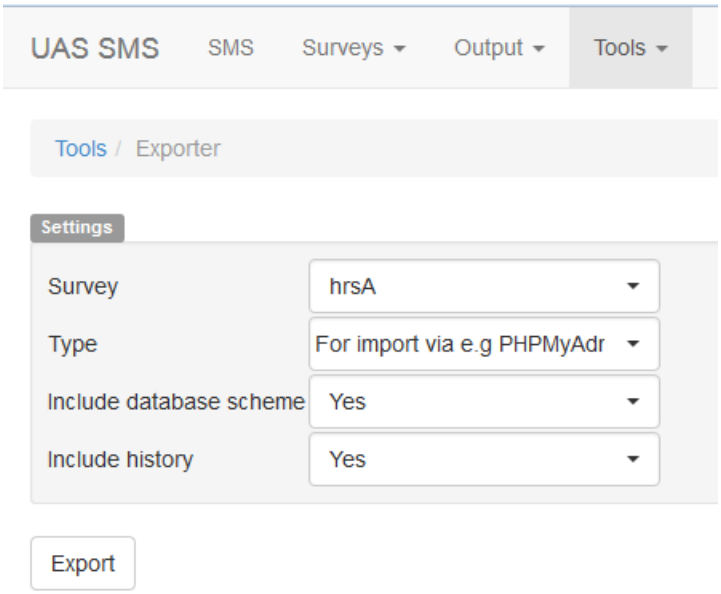


Export is per-survey; that is, if you wish to export multiple surveys, then the exporter needs to be run once for each survey. The created export file's format can be either NubiS or SQL. A NubiS export file is created by selecting the For import via NubiS option and clicking the *Export* button. The generated file can be saved and imported into another NubiS installation via the importer (note: the file contents may look garbled, but this is just NubiS' internal format for exporting and importing surveys):



Simply select the file via *Browse* and click *Import*. NubiS will ask for a confirmation by entering "IMPORT." This is to confirm importing truly is the action you wish to perform. This is particularly pertinent if the selected import type is not Add to current project, but Replace current project. If the second is chosen, then on import NubiS removes any existing surveys and their data. As such, take care when choosing this action.

It also should be noted the NubiS importer can only process NubiS export files. Surveys exported to SQL format cannot be imported using the NubiS importer:



Rather, such export files should be imported via PHPMyAdmin, the database command line or in another manner that allows for executing SQL queries against the database. Two configuration options are available in this regard. One is whether to include so-called create table statements. This should be set to *Yes* if you are setting up a new NubiS installation altogether, as it will allow you to create the entire database scheme and survey content in one go. If you are adding content, then create table statements are not required. (Note that all create table statements include an IF NOT EXISTS code to avoid conflicts with existing database tables.) Second, one can opt to include or exclude any history kept by NubiS for the survey components. In a large-scale survey undergoing many changes, this history may be substantial in size, in which case it can be a consideration to exclude the history from the export.