NubiS SysAdmin Manual
Version 1.0
July, 2015

Preface

Table of contents

1. Overview

NubiS is a complete data collection tool developed from the ground up by USC. NubiS runs on any server, PC, laptop or netbook as well as on android tablets or smartphones.

The NubiS system includes all the traditional modes of data collection like self-administered, face-to-face, and telephone interviewing. NubiS can also collect continuous information from smartphones, tablets or other external devices like accelerometers, GPS devices, blood pressure meters and other biomedical devices. This type of information can be combined with survey data and used to prompt or ask questions to respondents, for example to ask certain questions only depending on location.

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:

1) A web server, for example Apache
2) A database. Currently MySQL is natively supported, support for other databases is available on request.
3) A scripting engine. NubiS is written in PHP and runs on PHP 5.3 or higher.

To access the NubiS survey programming interface, a modern browser with JavaScript is required. A device with a relatively large screen size is recommended (e.g. a desktop or laptop), but not mandatory.

Respondents can access a survey using any modern browser (for example Internet Explorer 9 or higher, Firefox 23 or higher, Safari, Opera, Chrome). Also, JavaScript must be supported and enabled within the browser. There is no requirement in terms of device or operating system. Note: limitations may be imposed of course by the survey design itself (for example displaying a large table on a smartphone screen), but this is not inherently imposed by NubiS.

1.2 Component details

NubiS builds on a variety of open source browser technologies:

1) Bootstrap (version 3)
2) Jquery (version 1.10)
3) Plugins (main plugins):
    a) JQuery Validator: http://jqueryvalidation.org/
    b) JQuery Bootstrap-select: http://silviomoreto.github.io/bootstrap-select/
    c) JQuery Inputmask: http://robinherbots.github.io/jquery.inputmask/
    d) JQuery Bootstrap-slider: https://github.com/seiyria/bootstrap-slider
    e) JQuery AppendGrid: https://appendgrid.apphb.com/
    f) JQuery DataTables: https://www.datatables.net/
    g) JQuery DateTimePicker: https://github.com/Eonasdan/bootstrap-datetimepicker
    h) JQuery Moment: http://momentjs.com/
    i) CodeMirror: https://codemirror.net/
    j) Bootstrap-switch: http://www.bootstrap-switch.org
    k) Highcharts: http://www.highcharts.com/
    l) JQuery UI: https://jqueryui.com/
    m) JQuery DirtyForm: https://github.com/snikch/jquery.dirtyforms
    n) JQuery TextComplete: https://github.com/yuku-t/jquery-textcomplete

1.3 Installation

To install NubiS follow the steps below:

1) Set up the MySQL database in which NubiS will store its data and metadata.
2) Download the tar/zip file (depending on your platform).
3) Extract it into the web accessible directory of your server (e.g. /var/www/html/ on Linux for Apache). NubiS will by default extract into a sub-directory called 'Nubis'. Feel free to change the name of the directory to be more meaningful keeping in mind that the resulting URL is the one at which respondents will access the survey.
4) Location the conf.php file in the NubiS root directory and ensure that 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, that is, after entering those you can simply click the 'Finish' button to complete the installation. NubiS will then use a default configuration. Alternatively, advanced users may want to explore the different tabs following the 'Database' tab to look at the different configuration options (most of which can also be modified later on within NubiS).
7) Upon completion of the wizard you will be provided with the NubiS login screen. You can log on with the default account sysadmin/sysadmin and follow the steps in section 4 to further setup NubiS.
8) Make sure your web server no longer has permissions to write to 'conf.php' in the root NubiS directory.

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 like uploading a sample, updating respondent information, and tracking progress of data collection (e.g. in terms of number of completed interviews). Due to the fact that every data collection project takes place in its own unique context with its own 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 like adding questions, defining skip patterns, downloading data files, and so on. The survey system is designed to be comprehensive in nature in terms of what is needed for programming a survey while ensuring sufficient flexibility for any custom requirements (such as a custom way for a respondent to provide an answer to a question)

The SMS and survey system can be used in conjunction OR the survey system can be used in a stand-alone fashion. You can also use the survey system with your own SMS if you ensure that this sample management system interacts with the NubiS survey system in a prescribed fashion. The choice in this matter depends on the scope and nature of your project.

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 that has been designed to facilitate the collection of data in a variety of interview modes and/or languages. An **interview mode** (or mode for short) expresses the manner in which a survey is being administered. NubiS supports four types of mode:

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 in a mode is English but this is customizable. It is possible to use both character based (like Japanese or Chinese) and non-character based language (like Spanish or French).

2.2 Sample management system concepts

As mentioned earlier, the NubiS sample management system allows one to manage the sample of a data collection project. 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 e.g. that individual respondents are grouped within a household, their data is marked in such a way that they can easily be associated with the data of other respondents within the same household, and that 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, in NubiS tracking of the sample is done on a respondent level, not on a household level.

During the data collection period there will be interaction with respondents. In order to keep track of such interactions NubiS utilizes the notion of contacts. Each **contact** captures an interaction in terms of for example its type (e.g. start of an interview), 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.

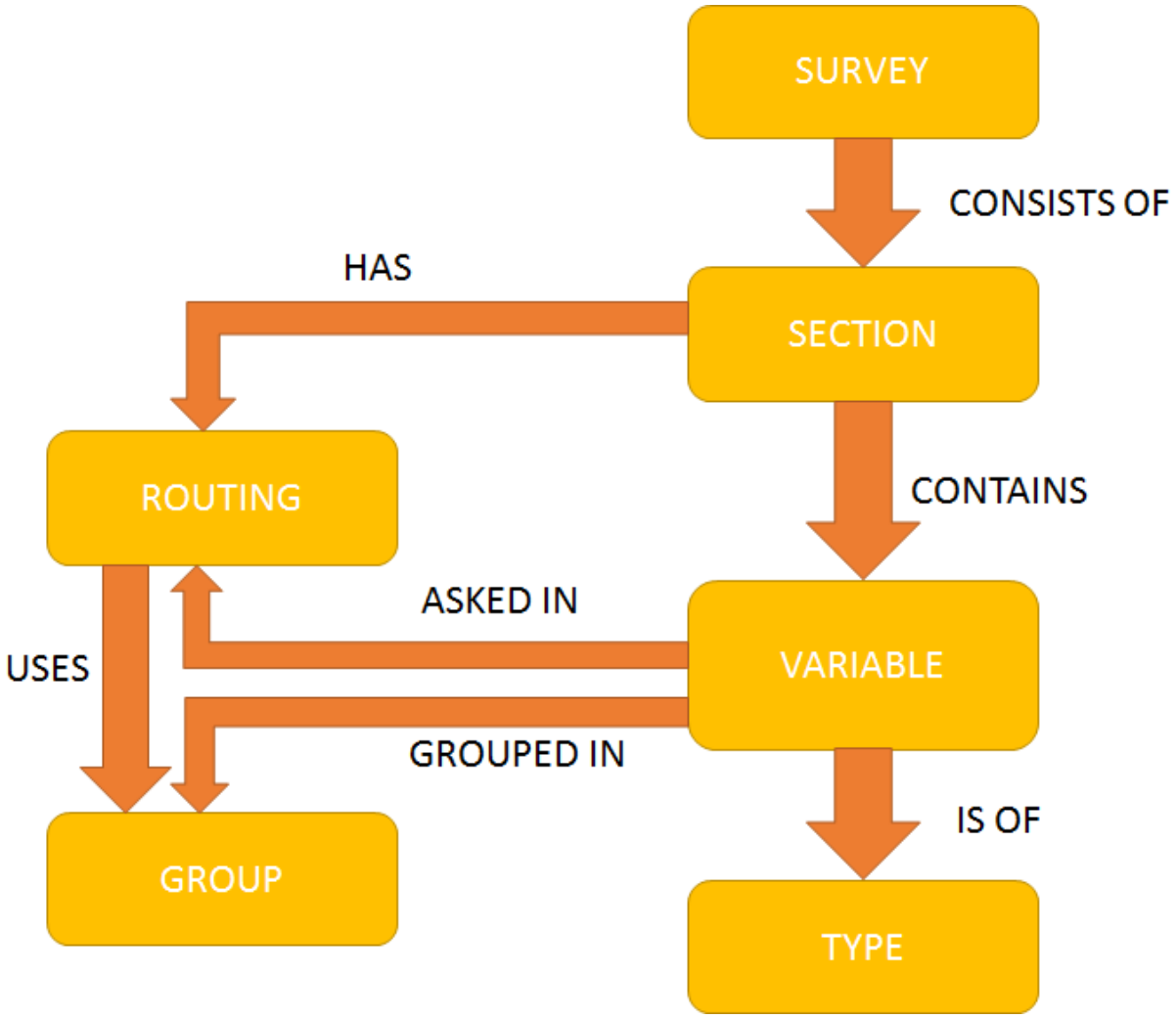
An **interviewer** represents a person responsible for conducting interviews with respondents, be it by talking to them on the phone or by meeting them in person. Interviewers are managed by a supervisor. A **supervisor** represents the person within the data collection project responsible for tracking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

Other roles within the context of sample management that are represented in NubiS are nurses and researchers. A **nurse** represents a person who is responsible for example entering biomarker information, capturing height/weight, and etceteras. A **researcher** represents a person interested in the outcome of the data collection effort, i.e. in access to the data.

2.3 Survey system concepts

The survey system in NubiS is designed to operate on and utilize the different types of object displayed in the diagram below. As the diagram shows 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 questionnaire. A **survey** comprises all the components required to program and run a survey, consisting of 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 may want to ask, some text you want to present on the screen, a placeholder for internal purposes (e.g. performing a calculation), or for deriving so-called dynamic text (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, for example the same 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 which the survey system must follow to administer the survey to a respondent. These instructions can include skip patterns, loops, and groups to combine multiple variables on the same screen. A **group** in this regard defines the collection of variables to be combined, the layout to be used, and a variety of other characteristics to be applied.

Lastly, not shown in the diagram above, each of the concepts discussed has associated with it a 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. We will return to this in detail in Section 16.

3. Running example

In the remainder of this manual we will use an example to illustrate the concepts discussed so far and the manner in which they are realized and implemented in NubiS.

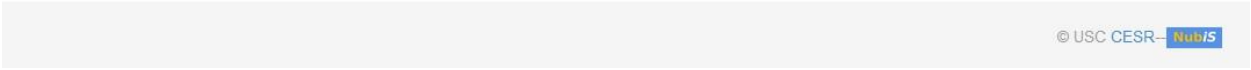
For this purpose we will use the Health and Retirement Study. This study is a longitudinal panel study that surveys a representative sample of approximately 20,000 Americans over the age of 50 every two years. The study comprises a series of interrelated surveys that collect information about income, work, assets, pension plans, health insurance, disability, physical health and functioning, cognitive functioning, and health care expenditures. Detailed information about the HRS can be found here.

We will concentrate foremost on 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 illustrative purposes we may also cover parts of other sections of the HRS.

4. Setting up your first survey

After you have successfully completed the installation of NubiS you can start working on the development of your survey. To do so open your browser and open the location in which you installed NubiS (or if you still had this screen open after . For example, if you installed NubiS in the /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 you will see is the login screen:

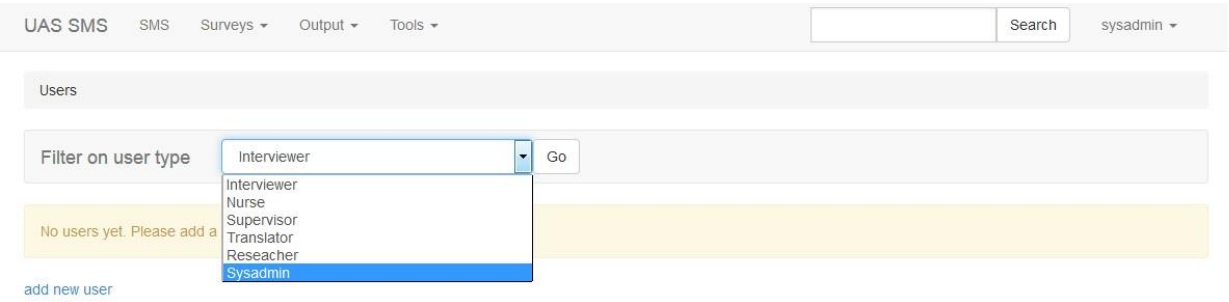


As mentioned in the installation procedure, by default NubiS comes with a single system administrator account:

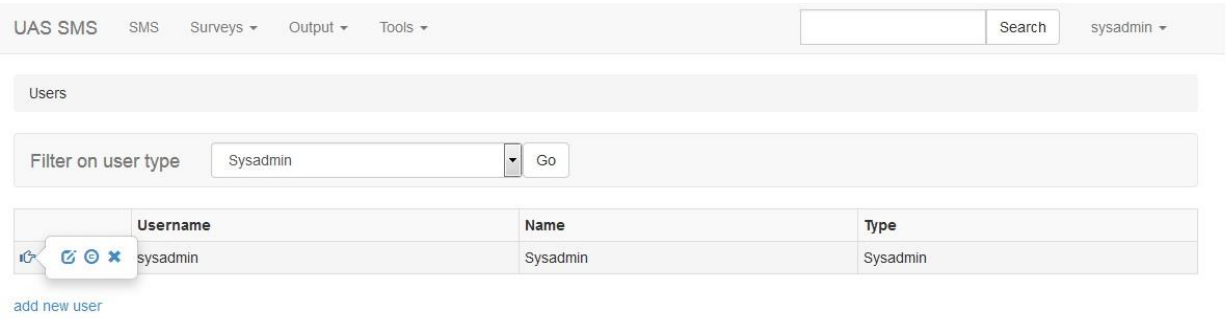
Username: sysadmin
Password: sysadmin

Log on with the above credentials. The NubiS system administrator interface will open:

The first thing you will want to do is update the default password. To do this, click the Users option in the dropdown in the top right corner:



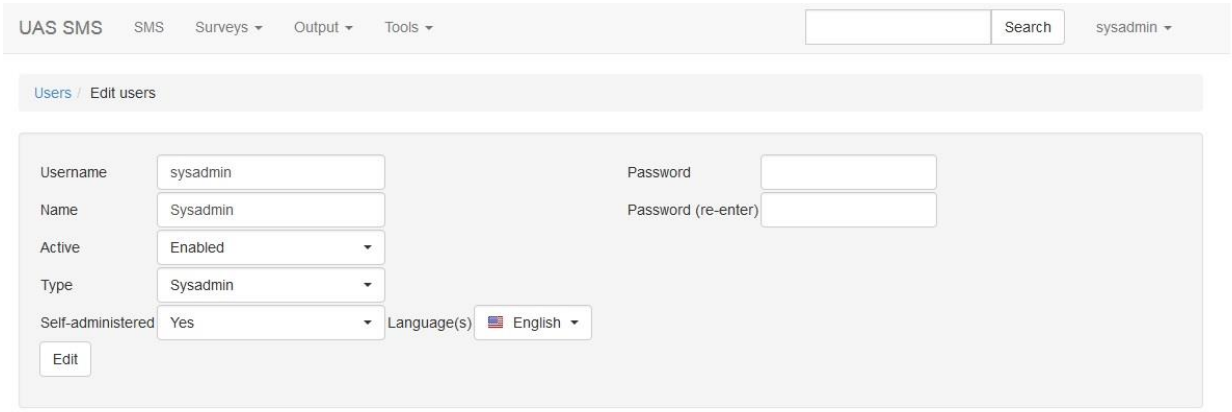
In the ‘filter on user type’ drop down, select ‘Sysadmin’ and click ‘Go’. This will bring up the overview of all known system administrators (in this case the single default account):



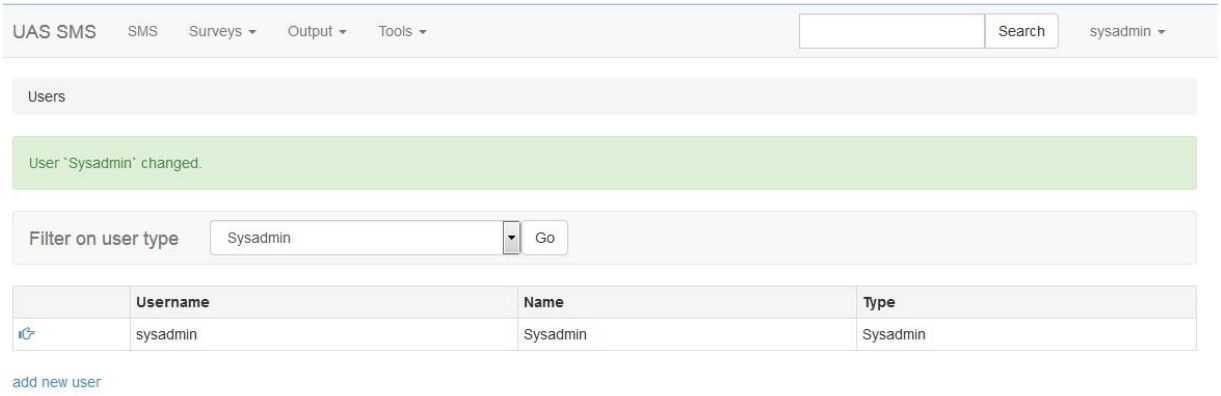
You will see that as you hover over the pointing finger icon

several options appear. You will find this consistently throughout the NubiS interface, for example 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. We will be introducing several other symbols as we start adding sections and variables.

For now, let's return to editing the sysadmin account by clicking the pointed finger symbol (or the pencil symbol):



As can be seen there are a variety of settings available. We will ignore these here and return to them in detail when discussing user management in NubiS. For now, let's enter a new password in the password boxes on the right and click 'Edit'.



A confirmation message will appear that the sysadmin account has been updated. Confirmation messages are always in green in NubiS. We will also encounter yellow messages (providing information) and red messages (representing errors).

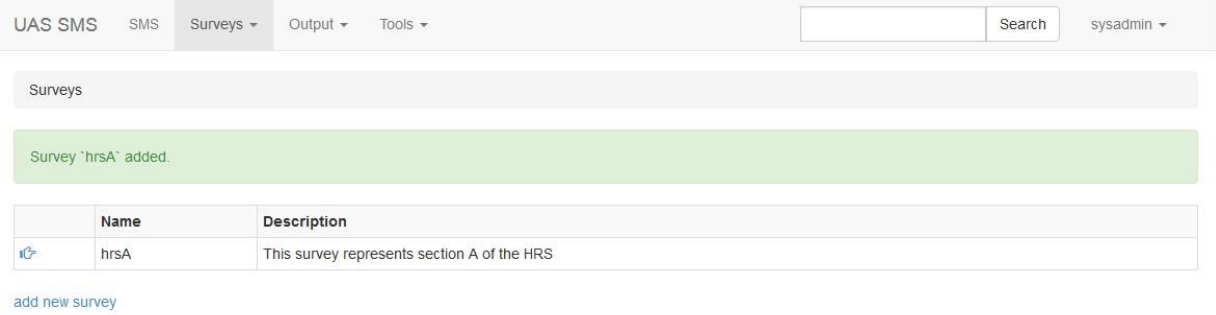
Now that we have updated the sysadmin account let's turn our attention to starting our survey. On installation NubiS comes without any pre-defined surveys. To add one we locate the 'UAS SMS' in the navigation bar on top:

Here we click on ‘Surveys’ to show the list of current surveys (empty right now):

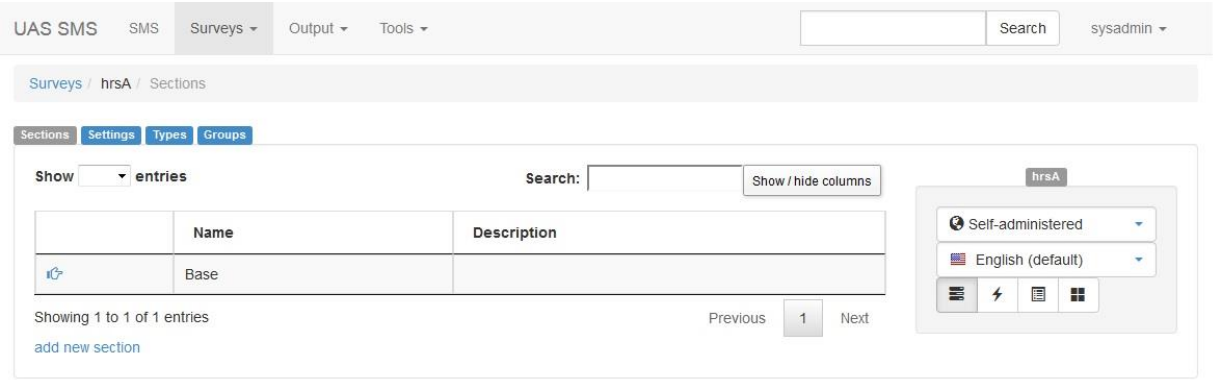
Next we click ‘Add new survey’:

In this screen we can 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 you are programming the survey and when filling out the survey), and a description. Note that if this were a second survey we were adding, we would also have the option to indicate whether the survey is the default survey. The ‘default survey’ in this regard is the survey which Nubis will launch if no other indicator is provided on which survey is to be started.

For now, for our example we will name the survey ‘hrsA’ as it will represent section A of the Health and Retirement Study. On clicking ‘Add’ NubiS will create a new survey:

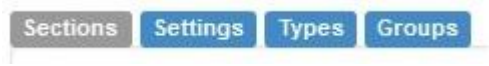


To start editing our new survey we click the pointed finger (options to copy or delete a survey are also available on hovering over the pointed finger). This brings up the following screen:



There are several things worth pointing out on this screen. As a more generic statement the screen reflects the manner in which most NubiS screens dealing with adding/editing/removing components are laid out. 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 interview mode and language you are currently working in as well as shortcuts to other types of components. For example, in the sections list screen above it can be seen that we are currently in the self-administered interview mode in English (which are NubiS' default interview mode and language). The icons below it (shown below) give access to a survey's sections, settings, types and groups respectively:

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



For now we will focus on the sections list itself. We will return to the other types of component as well as how to develop surveys for multiple interview modes and/or languages at a later stage.

In the sections list we find that there already is one section. This section is the '**Base**' section. It comes standard with NubiS and it cannot be deleted (nor can its name be edited). This standard section contains a set of variables that NubiS uses to administer surveys.

Let's explore for a moment what is in the Base section by clicking the pointed finger. This will open the contents of the section. As an aside, note that this is the only instance in NubiS where clicking on the pointed finger does not equal editing 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 will look 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

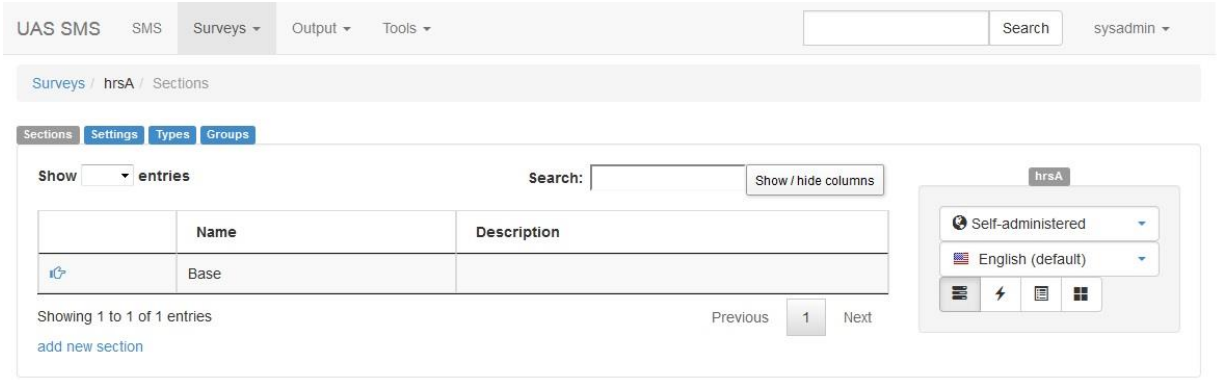
As the screen shows, there are two main tabs: variables and routing (also accessible through the second right hand side menu with its header 'Base'). For now let's focus on the variable list and briefly discuss the different standard variables that NubiS uses. We will return to the routing tab of the Base section (as well as user-added sections) once we start creating our own survey content. 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.
- 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 when entering the survey is only allowed from a web site external to NubiS
- execution_mode: the mode of survey execution, which can be normal or test mode
- 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.

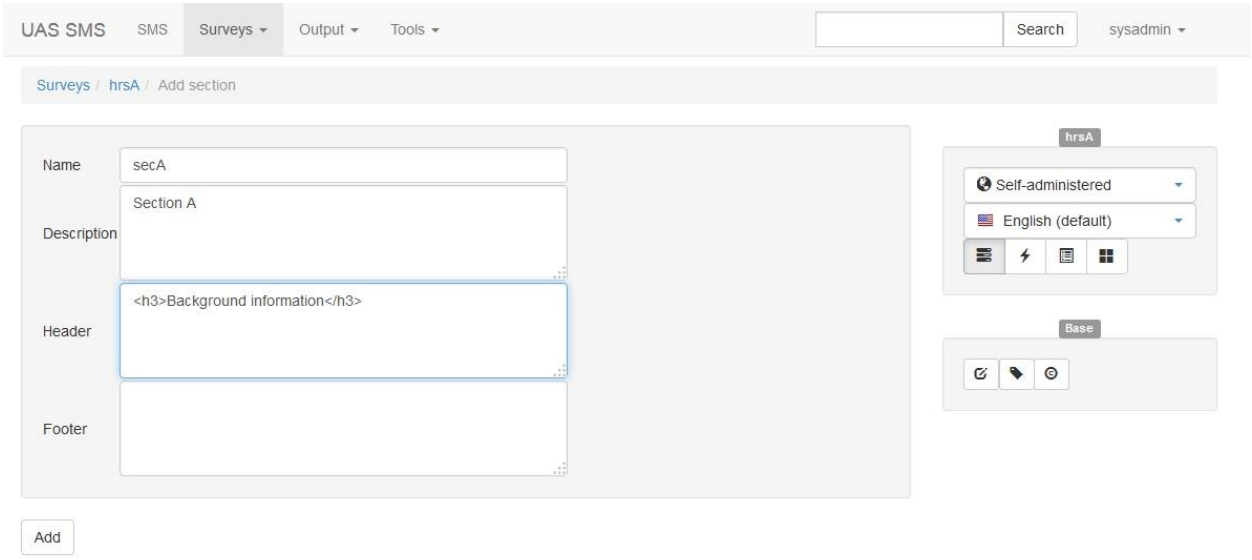
5. Adding your own content

Though the Base section is very useful, it obviously does not contain any content needed for creating our own survey. In order to do that we need to start adding sections and variables. Let us begin by first adding a new section that will group the variables we want to ask together. Note: in principle it is possible of course to simply add any of our variables to the Base section. However, for programming convenience and the purpose of clarity it is advised to add your own section(s); and moreover break up your survey into multiple sections if it is relatively long.

Having said that, to add a new section let's move back to the sections list by clicking on 'hrsA' in the breadcrumb like navigation links just below the top navigation bar:



On this screen we click 'Add new section':

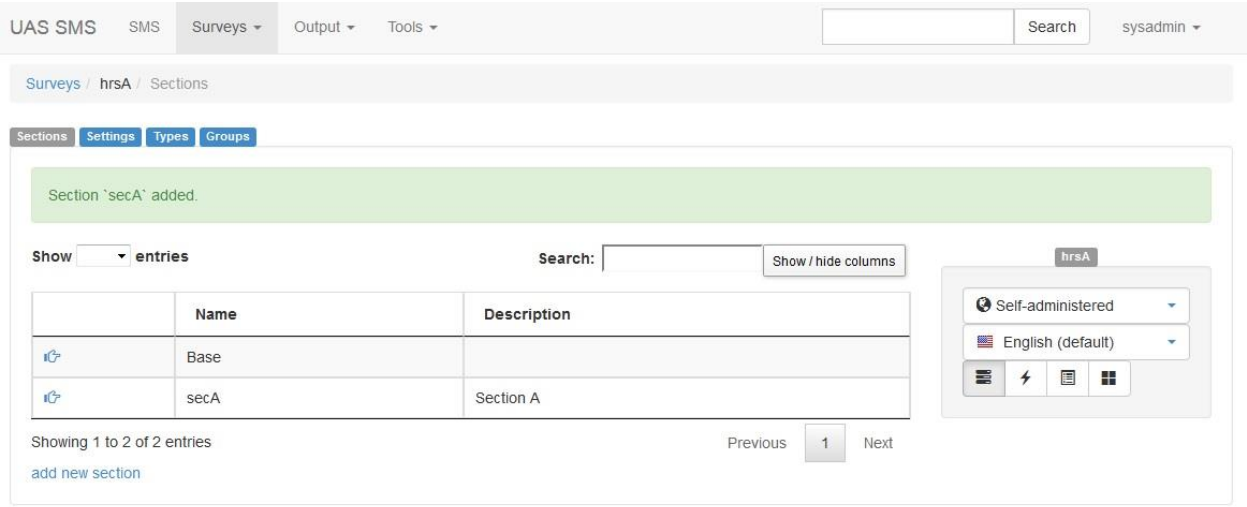


Since we are focusing on adding section A and section A2 of the HRS, we will start by creating a 'secA' section. We give a short description as well, which will make it easier to recognize what

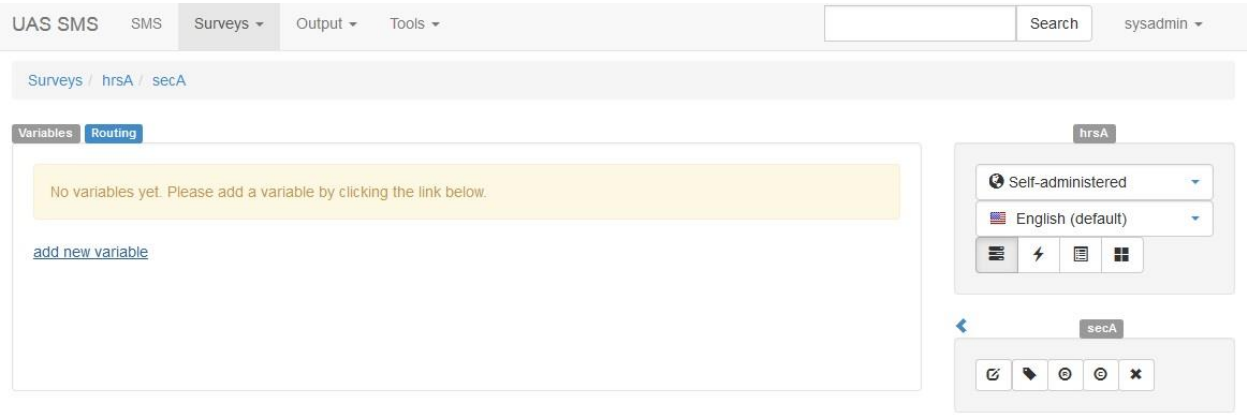
a section is about in the sections list. Lastly, we specify a section header. This header will be inserted by NubiS into the survey screen at the top of the screen. This will help respondents to keep track of which part of the survey they are in. Note that we used HTML tags here (<h3></h3>). Nubis treats any HTML tags in properties like a section header or a question text (or Javascript snippets or any other web based syntax) as regular text and consequently it gets outputted as-is. In this case our section header will appear something similar to:

“Background information”

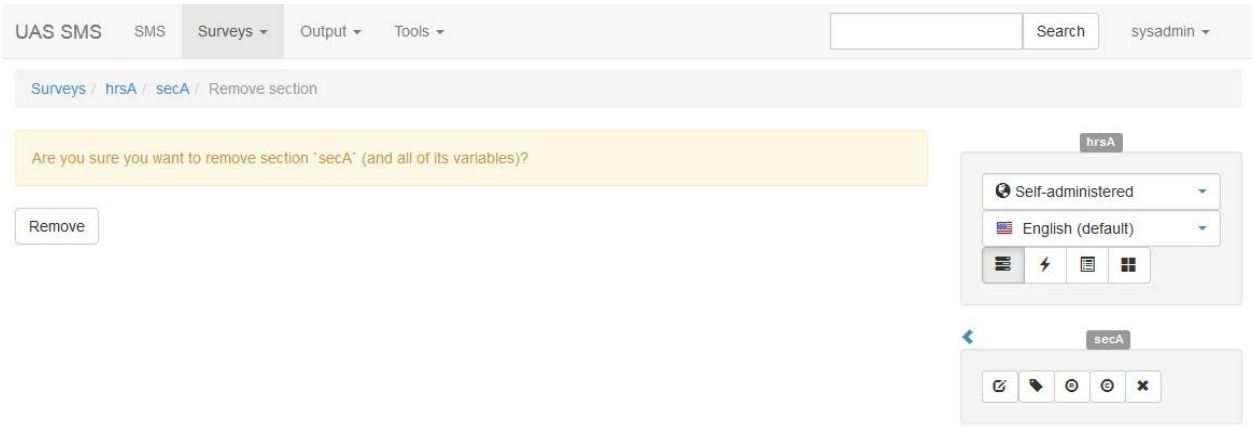
After having added the section, NubiS will confirm and show the updated sections list:



We can then go into our new section (by clicking the pointed finger in front of secA):

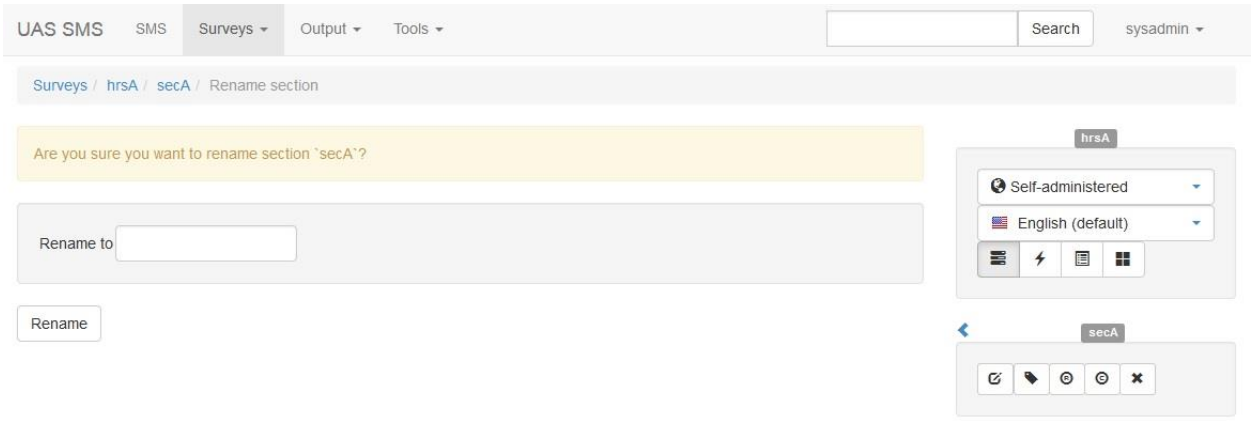


Unsurprisingly, there are no variables present yet. Before we add one, as a side note that in the right hand side menu for ‘secA’ there are two extra options compared to the ‘Base’ section. One is the refactor option (as represented by the circle with an R in it) and the other is the delete option. The delete option will prompt a confirmation:



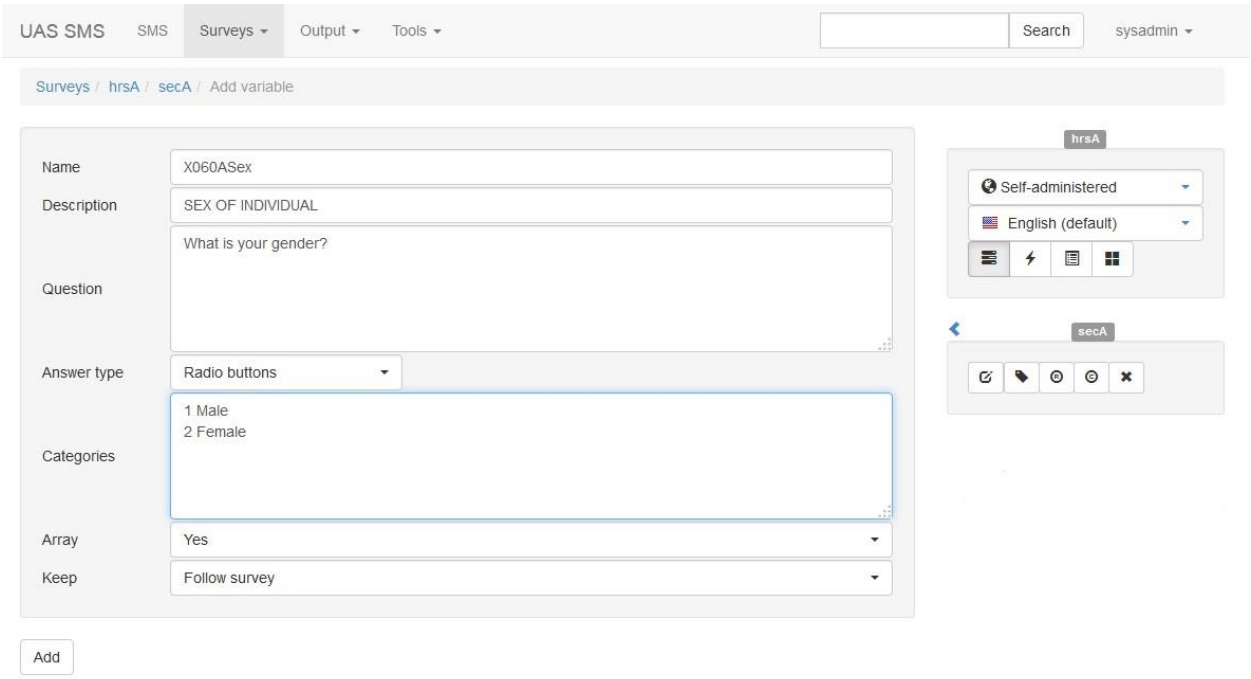
Only upon clicking ‘Remove’ the section will be removed (as well as its variables). The same applies to any delete option within NubiS, that is, all removal actions must be confirmed.

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



Refactoring, like deletion, must also 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).

Of course, right now we are more interested in adding a new variable. In order to do so we click ‘Add new variable’:



Specifically, we will add a question to ask the gender of the respondent. The screen above shows the specifics of the question. We specify 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/or should be kept (we will explain in detail what 'kept' implies in Section TODO).

Let's start with the name. Here we named our question 'X060ASex', which follows the name of this question in the HRS. It may not always be the case however that the survey specification lists names for its questions. In that case you will need to devise a naming convention. As a general rule the most important thing with defining a convention is consistency, that is, it should be adhered to throughout the survey. In terms of how to come up with names a choice is often made between more thematic names (such as 'gender' for our question above), a more logical scheme (e.g. naming questions in the form secA_001, secA_002), or a combination of both (like above).

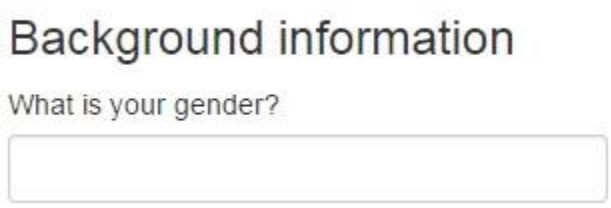
NubiS imposes no restriction on what name is chosen expect that it must start with a letter and otherwise consists only of letters, numbers and underscores (_). There is no maximum length but for usability concise names are of course preferred, but the name has to be unique within the survey. That is, it is not allowed to name a section and a variable the same. In this regard NubiS is case insensitive, i.e. 'secA_001' is the same as 'SECA_001'. Attempting to add a variable, section, group, or type with an existing name will result in an error:

The description that we put for our variable is ‘SEX OF INDIVIDUAL’. Note that the capitalization here is by choice, it is not imposed by NubiS. The reason is to improve readability of this text because it is used as a variable label when creating a STATA data file for our survey. For this reason, 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?’. Note that if we had wanted to we could have also used for example HTML markup.

Less straightforward is the next characteristic, which is the answer type. Here we have selected ‘radio buttons’ to show the answer options to our question: male or female. In general, the following default answer types are available:

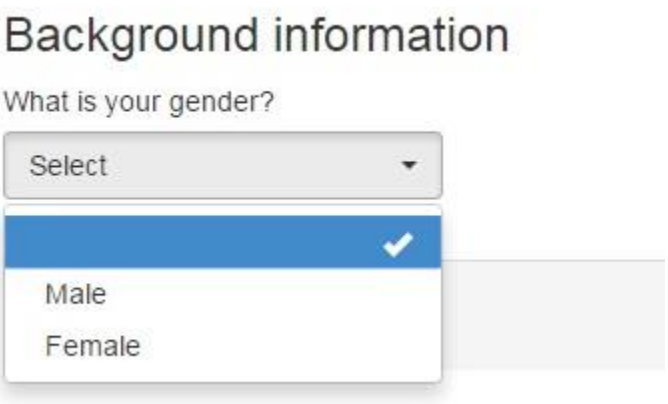
- String: allows a respondent to give a short textual answer.



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

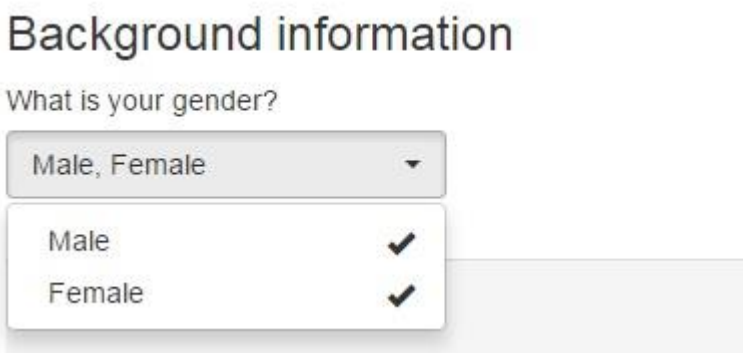
- Radio buttons: allows a respondent to choose one of the pre-defined options.

- Drop down: allows a respondent to choose one of the pre-defined options from a dropdown.



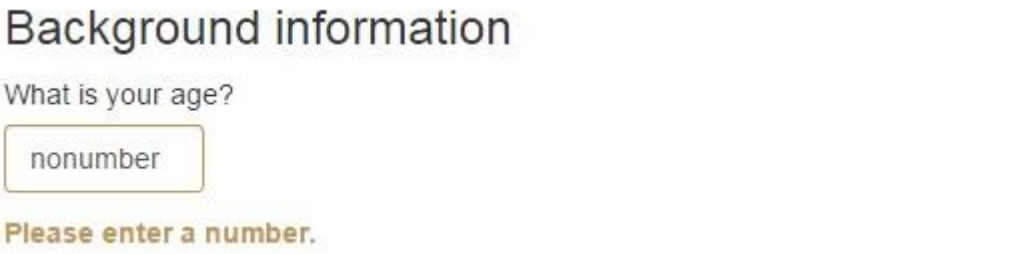
- Checkboxes: allows a respondent to choose one or more pre-defined options.

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



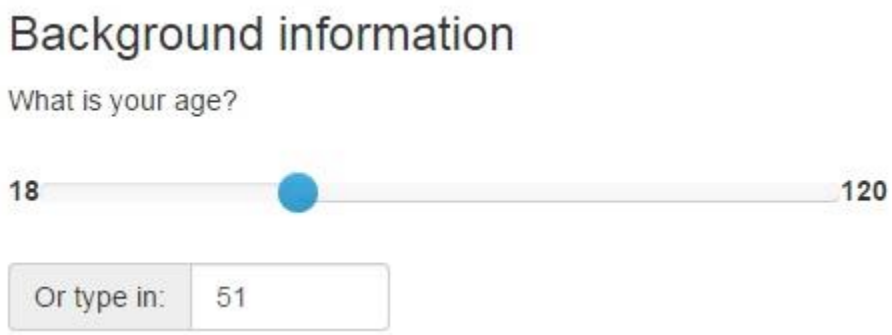
- Integer: allows a respondent to enter a whole number. Provides automatic prevention of entering anything other than a whole number (if enabled) and automated error checking.

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



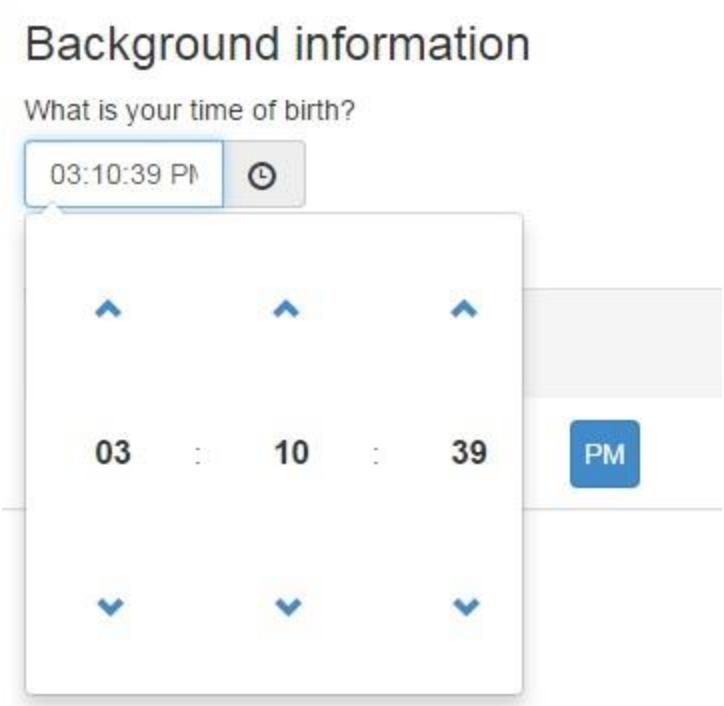
- Range: allows a respondent to enter a (whole) number in a range . Provides automatic prevention of entering anything other than a whole number (if enabled) and automated error checking.

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

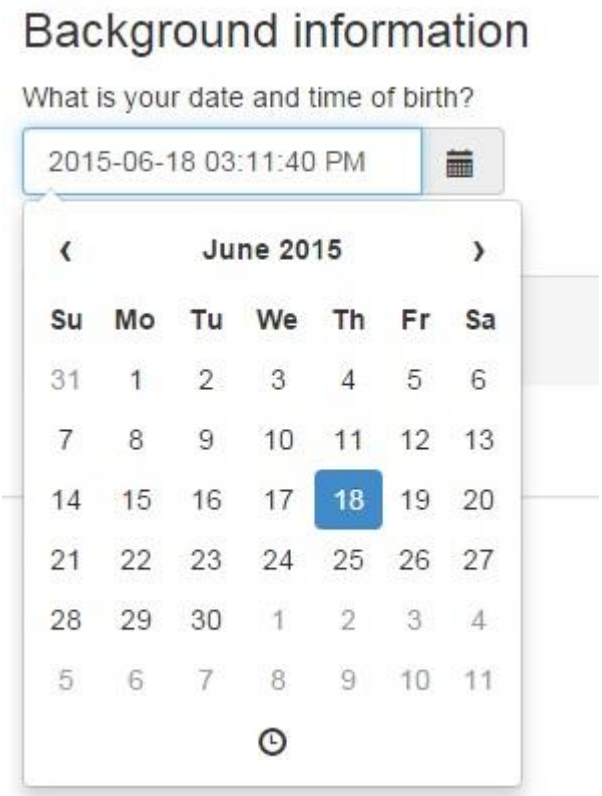


- Date picker: allows a respondent to choose a date.

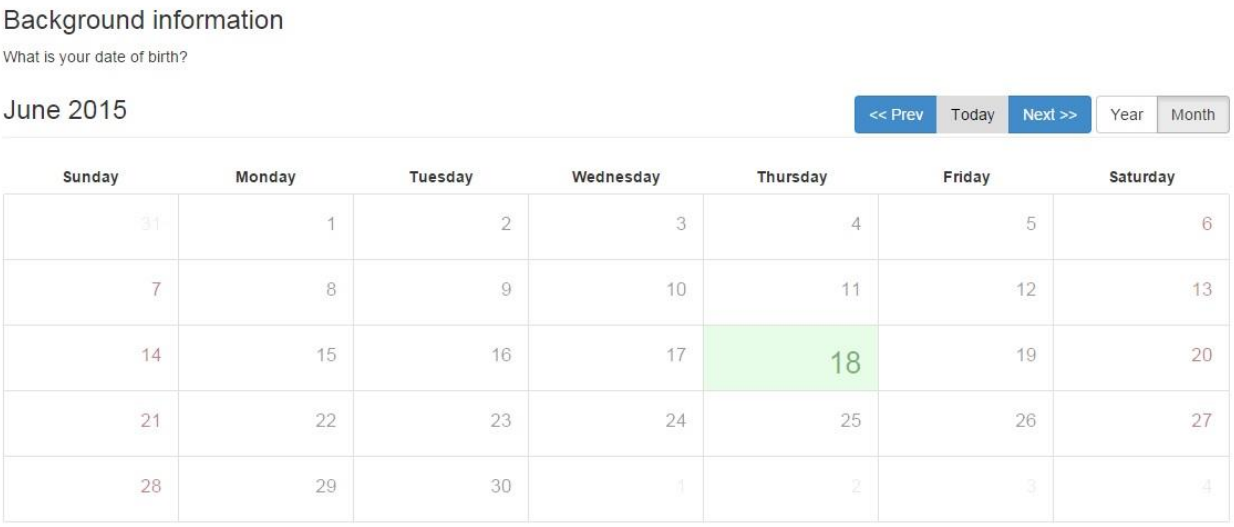
- Time picker: allows a respondent to choose a time.



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



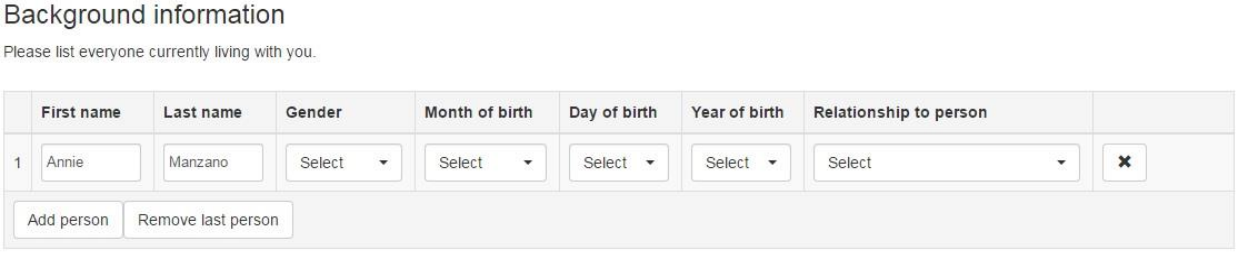
- Calendar: allows a respondent to choose a date on a calendar.



- 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. Variables of this type are not directly shown to a respondent. We discuss variables of this type in section 17.1.

- Custom: allows to define custom code to construct the manner in which a respondent should answer. This answer type is for advanced usage only and is discussed later on in this manual. An example of a custom answer type is below.



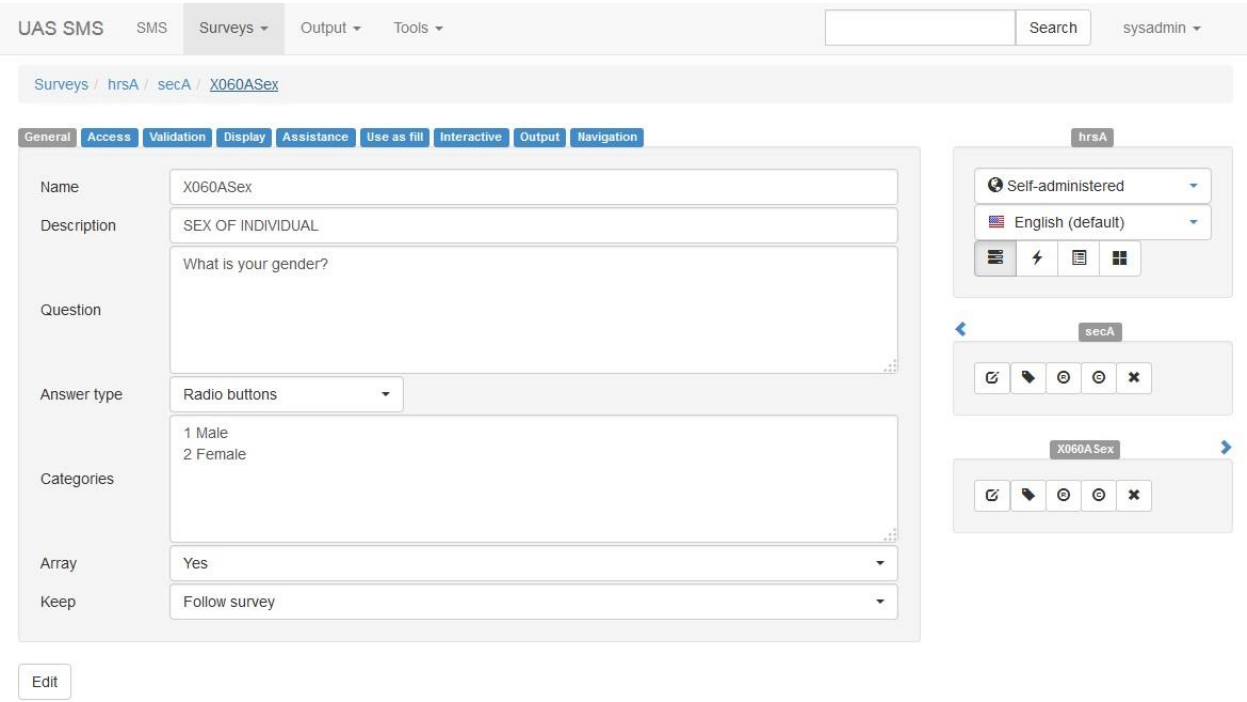
Beyond the answer type there are two other properties that can be set when adding a variable: whether the variable is an array and whether it should be kept. if the array property is set to Yes, then this means that NubiS will treat this variable as if it can have multiple instances. For

example, when asking gender we might want to ask the same question of the respondent as well as any other people in the household. Rather than constructing separate variables for each person we can declare that ‘X060ASex’ is an array allowing us to capture the gender of multiple people through a single defined variable. We will return to this in greater detail once we discuss 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 we will later see, this is useful to preserve the values of randomizer variables.

6. More settings for variables

In the variable ‘X060ASex’ that we added just now we specified several characteristics. There are a wide variety of other settings that can be defined for a variable. Let’s take a look at these settings by opening up our variable again by clicking the pointer finger in the variable list overview of section ‘secA’. The screen that appears is the following:



On first glance we see the same characteristics again as when we added the question. NubiS labels these characteristics as ‘General’, which by and large simply means that 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. Before we have a look at them, it should be mentioned that not all settings will be available for all variables. Rather, it is dependent on the answer type of the variable.

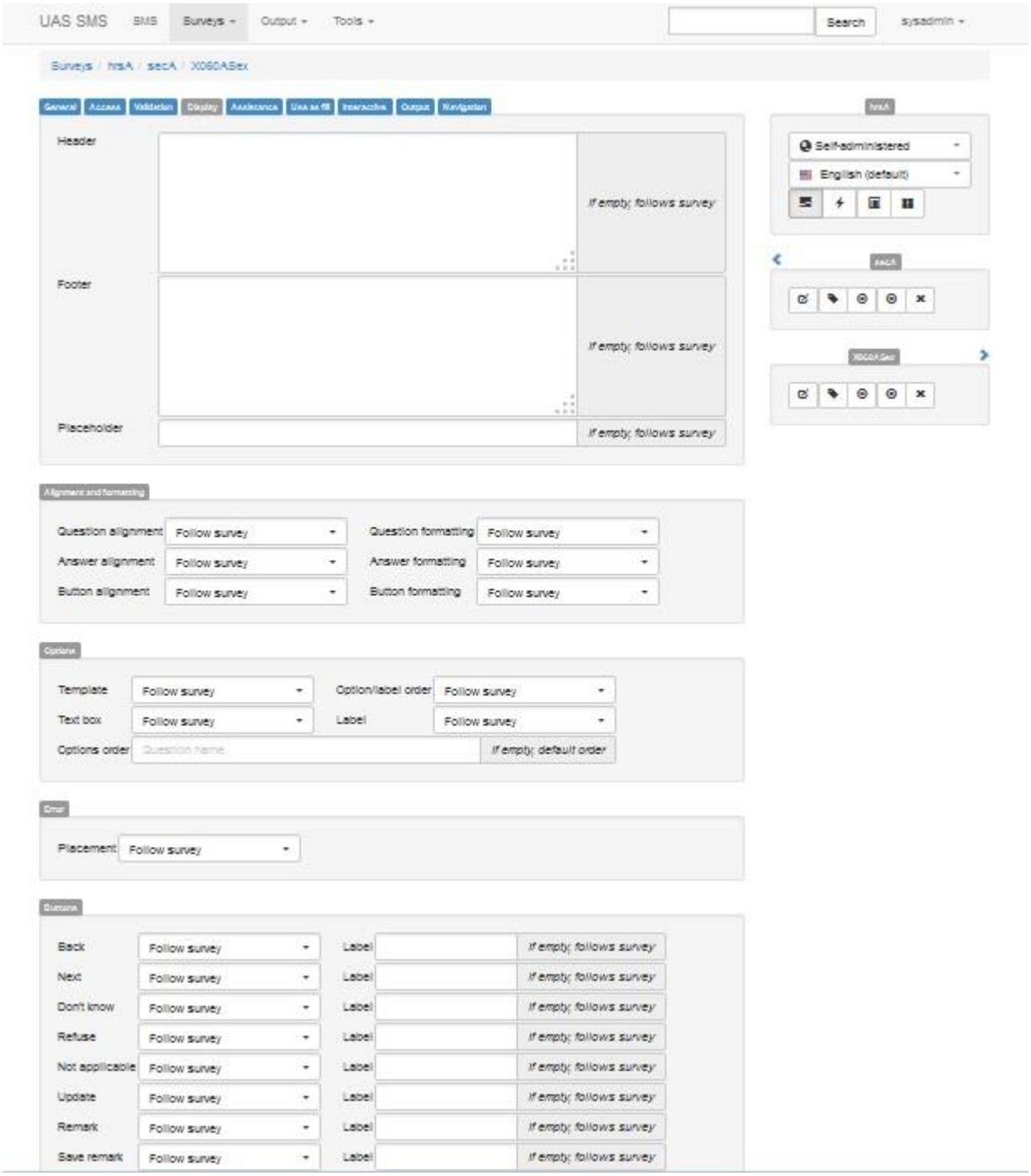
With that in mind, the following types of setting are available in NubiS:

- Access: governs the behavior in terms of if and how the survey can be accessed.
- Validation: allows to specify 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 e.g. JavaScript to implement some 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.

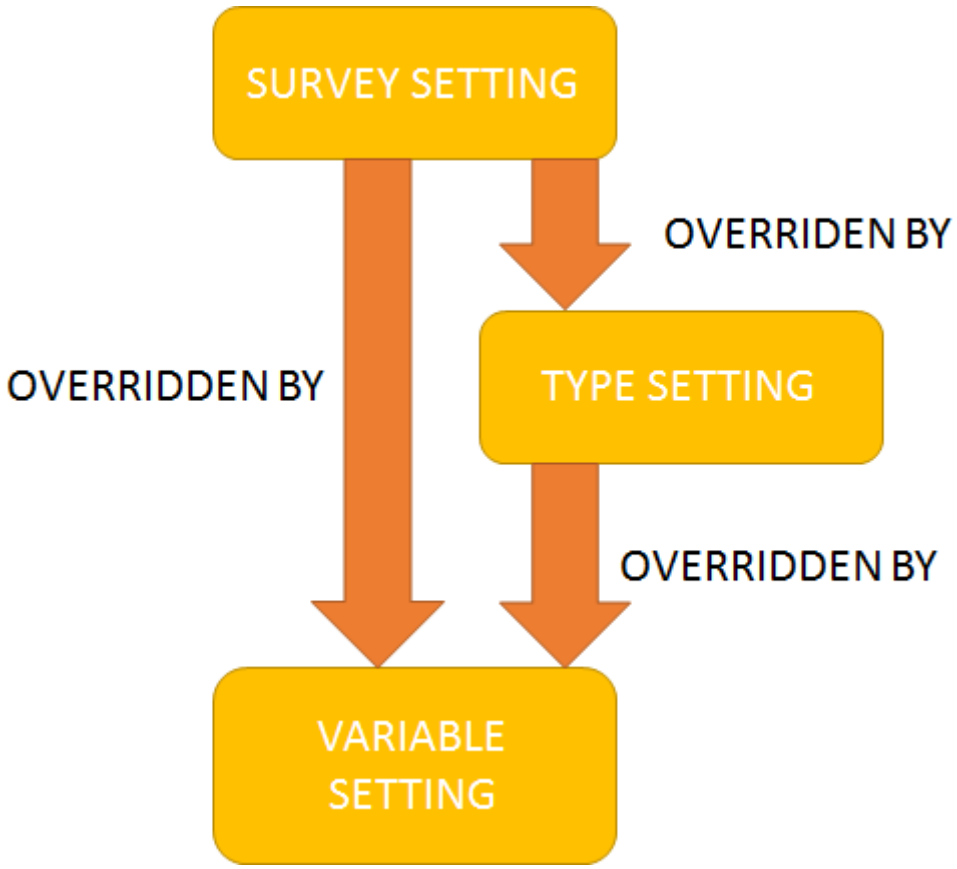
The observant reader will notice that one tab is missing in the above list, which is the ‘Use as fill’ tab. This tab does not really contain settings, but instead provides functionality for using the variable as dynamic text. We will return to this topic later.

For now, to showcase some of the specific settings in the above list, let’s click the ‘Display’ tab:



There are a few things to note here. Firstly, NubiS has a large number of settings. The goal of this manual is not to discuss each and every setting, but rather provide some examples to illustrate how settings can be defined and how NubiS interprets what is specified. If you are interested in a particular setting please refer to the NubiS reference manual.

Secondly, by and large the settings that can be set for a variable can also be set on a 'higher' level for types and 'higher' yet for the survey as a whole. We use 'higher' in this context to reflect the 'inheritance' style structure that NubiS employs. That is, a value for a setting can typically 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. The figure below conveys this:



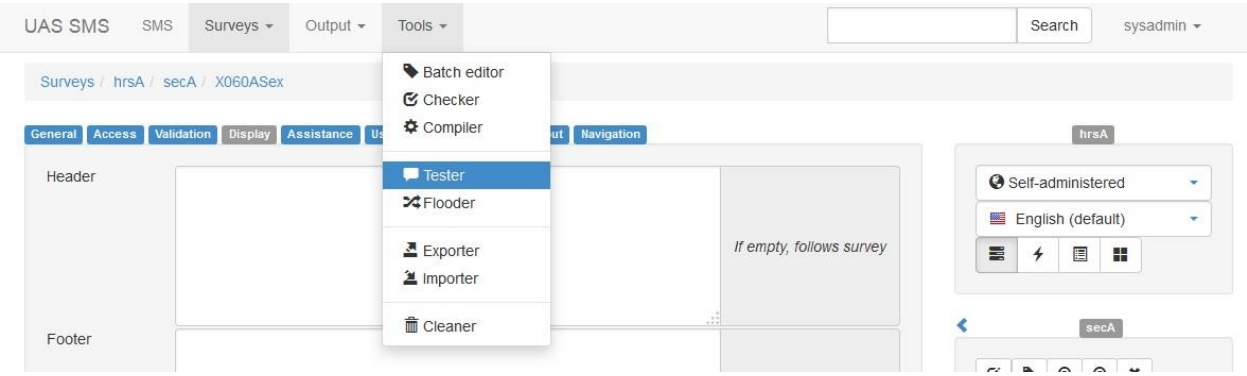
For settings with certain options (like Show or Hide) this is conveyed by the option 'Follow survey'. This is for example the case for the 'back button' setting towards the bottom:

As an aside, if our variable had been of a certain type instead, then instead of the ‘Follow survey’ option it would have been the ‘Follow type’ option.

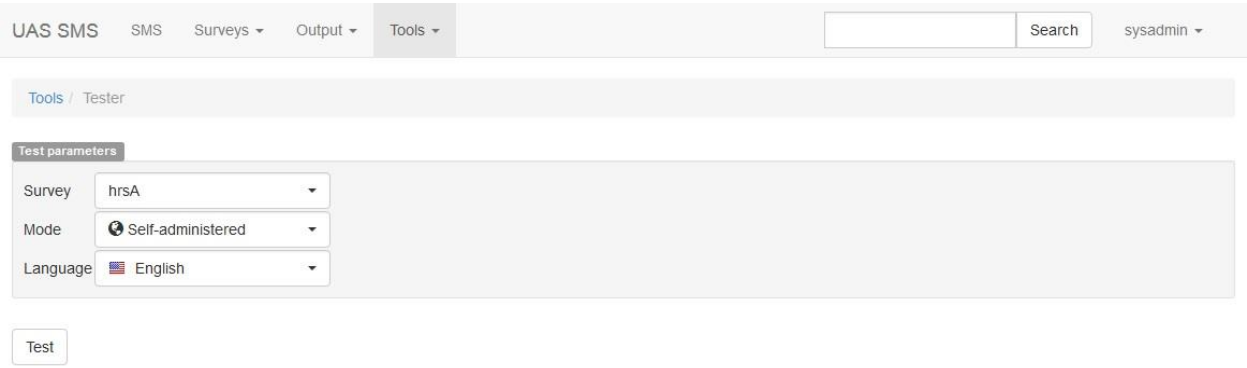
There are also some settings that are not of a ‘drop down’ variant, but instead allow for the free text specification of some text. An example is the ‘Back button label’ setting. This kind of setting is also inherited. The inherited value is used UNLESS a non-empty value is specified. So to override the normal back button label (which is ‘<< Back’) we 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 our question about the respondent’s gender. Note that 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 not have it appear for our variable, we would have to specify a non-empty string in the header setting textbox; the best candidate for which is   (which is HTML’s no break character and is not displayed in a browser screen).

7. Getting NubiS to ask your questions

So far we've discussed how to add a variable to NubiS and subsequently edit its settings. Now we turn our attention to the matter of how to get our survey to ask our question about the respondent's gender. You might say 'but haven't we already added our variable and isn't that enough?'. We can simply find out by trying to test our survey. To do so we can use the 'tester' tool located under 'Tools':



The resulting screen shows our test options:



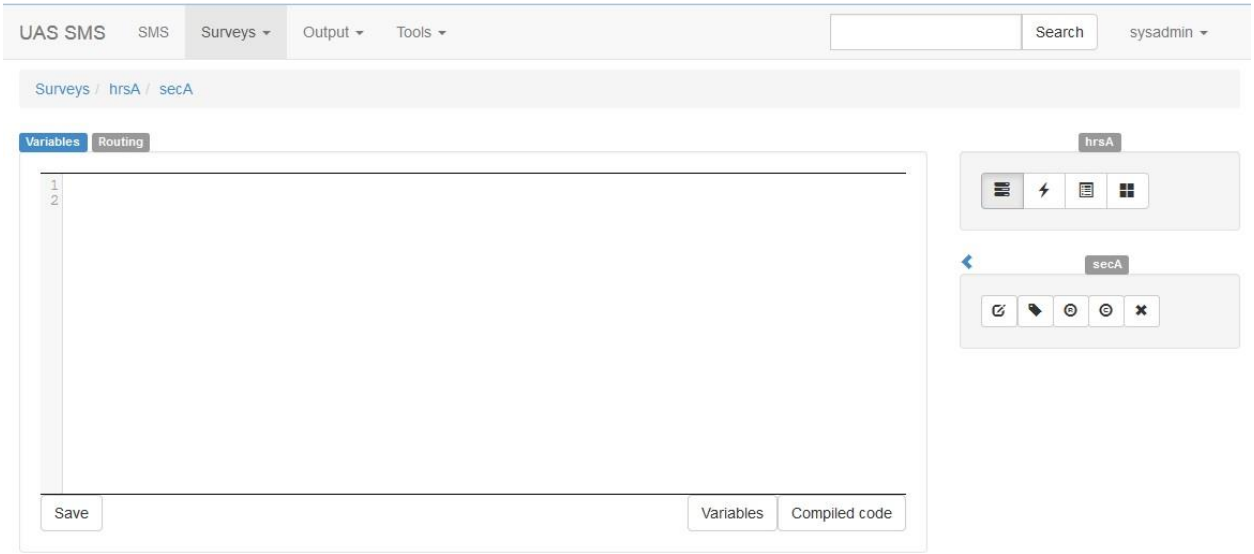
We hit 'Test' and we get:

In other words, we are back inside NubiS's area for developing surveys. So what happened? Well, adding a variable to NubiS is in itself not sufficient for it to be asked in a survey. Rather, we need to explicitly instruct NubiS to ask questions. Now that we didn't, NubiS starts the survey after we hit 'Test', finds there is nothing to ask, and so completes the survey and returns back to where we were since we did the survey in test mode. If we had tried to run our survey in normal mode as a respondent, we would have gotten a similar result:

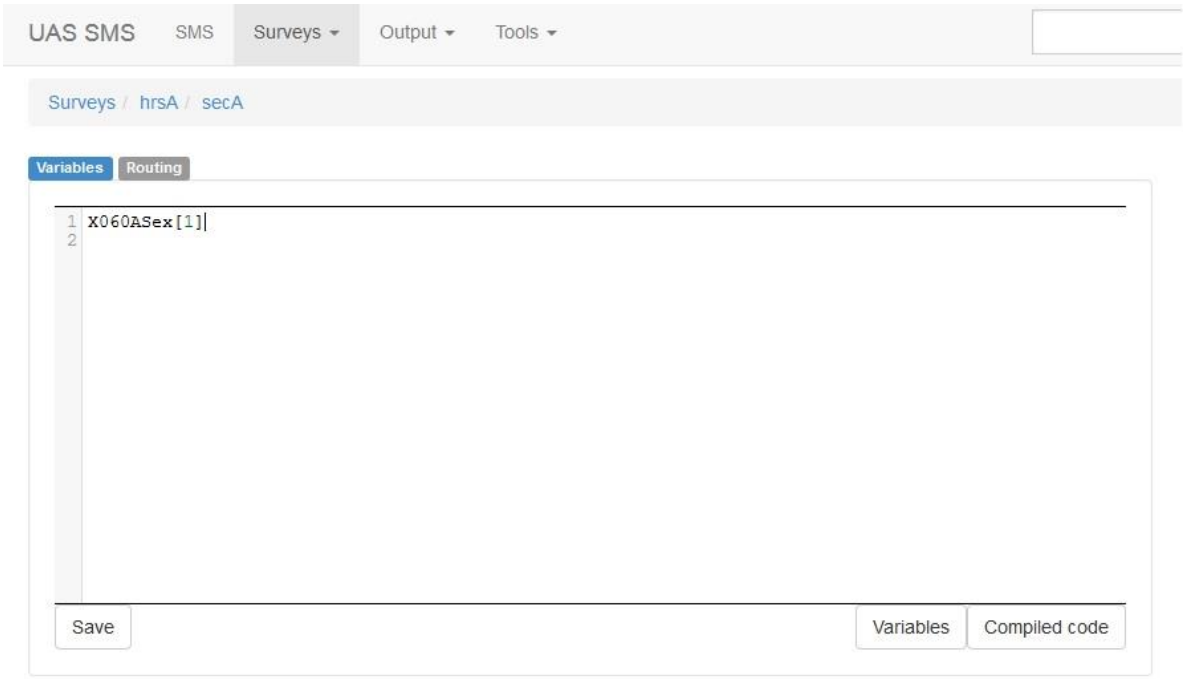
Here as well, NubiS went through the survey, found nothing to ask and so finished with the 'Thanks' screen.

So how do we get NubiS to ask our question about gender? For that we need to include it in the routing of the survey. Survey routing can be thought of as as series of statements that define which variables NubiS should use during a survey and in what manner. Routing is defined per section, that is, each section has its own routing. Those can then be integrated into the survey as a whole by incorporating a section into the 'Base' section routing.

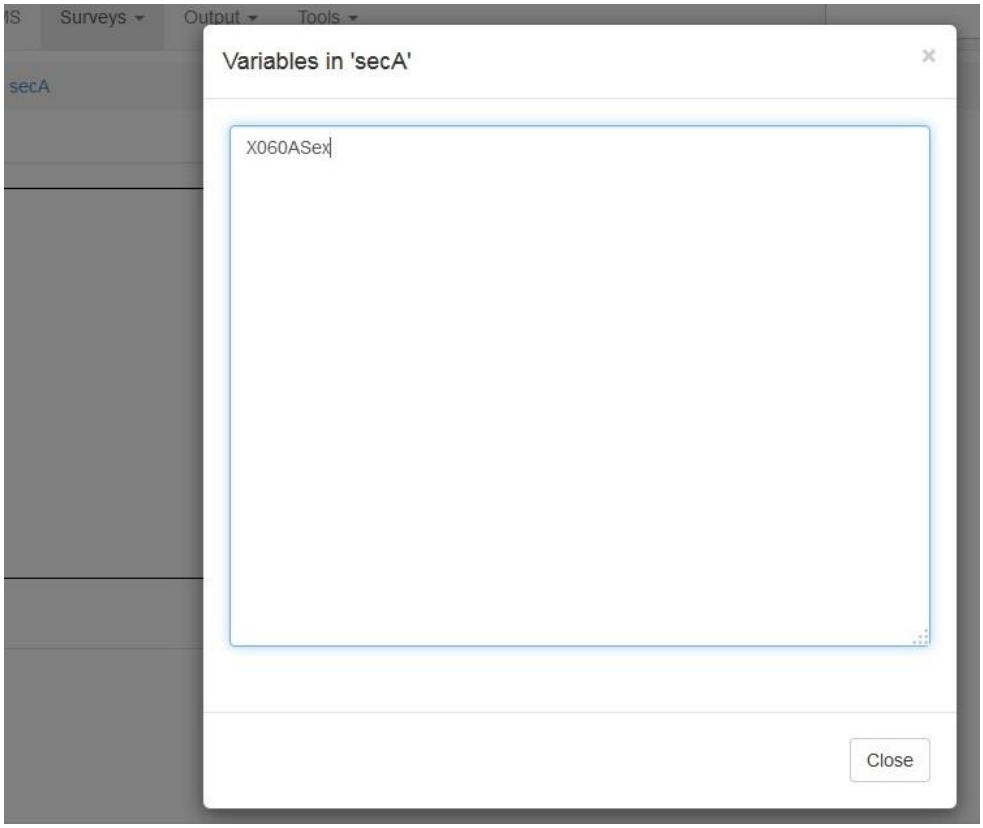
In order to do this let's click 'Surveys' in the screen we got as a result of our test, then click the pointed finger in the survey overview to open up our survey. Next, if not selected yet, choose the sections tab and click the pointed finger in front of the 'secA' section. Then, click on the 'Routing' tab (next to the 'Variables' tab) leading to the following screen:



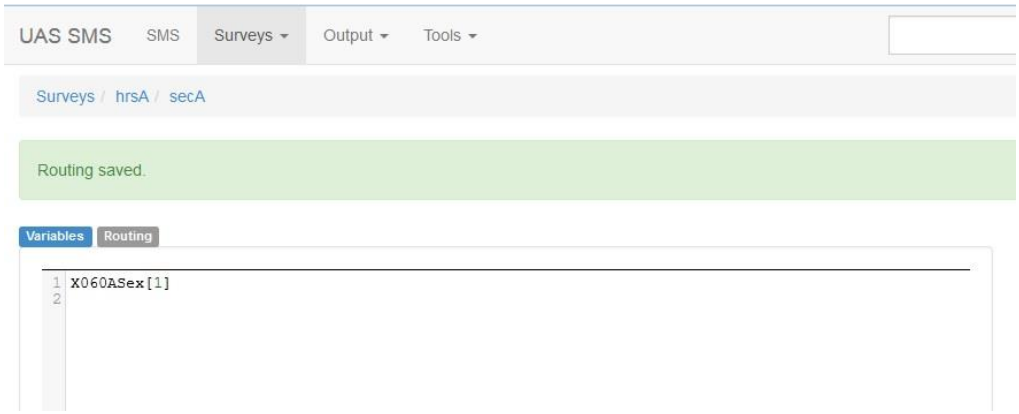
The routing screen basically consists of a single text box in which routing instructions can be entered. To instruct NubiS to ask a question simply use the variable's name and add it on a line:



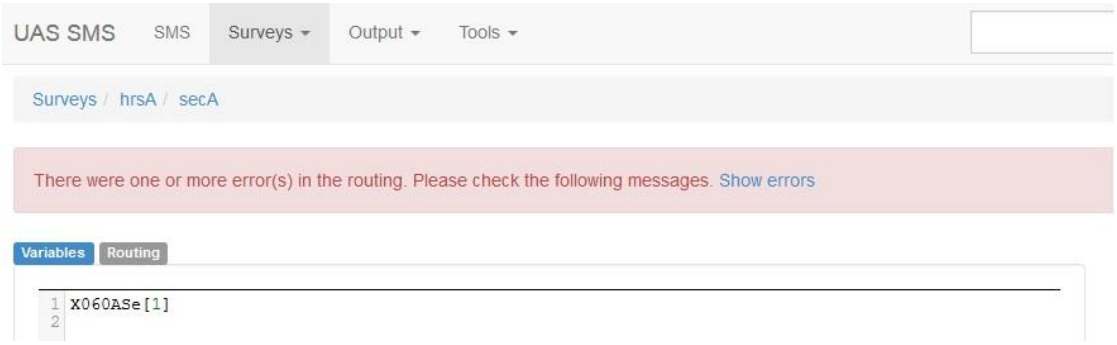
In this specific case we enter ‘X060ASex[1]’; we will explain in a little bit why we are adding the ‘[1]’. In case you forgot the name(s) of specific variables you can open a list of variables in the section by clicking the ‘Variables’ button below the routing text box on the right hand side:



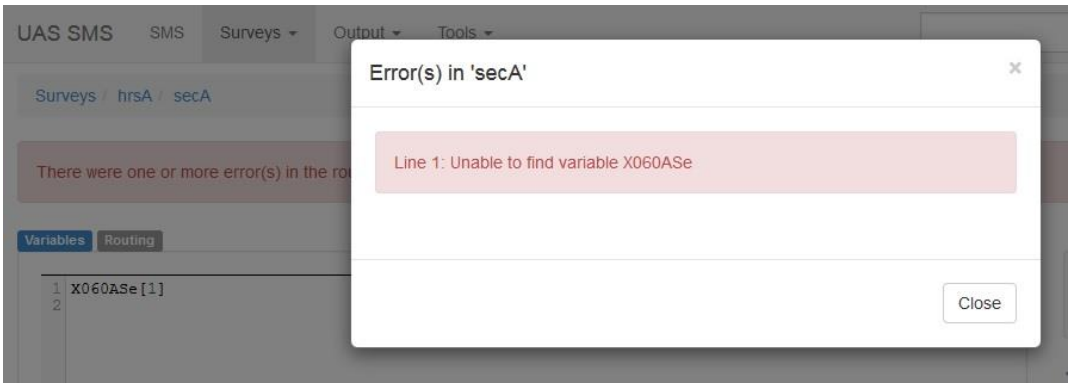
To save the routing just click the ‘Save’ button on the bottom left. NubiS will verify the instructions you entered and confirm the changes:



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

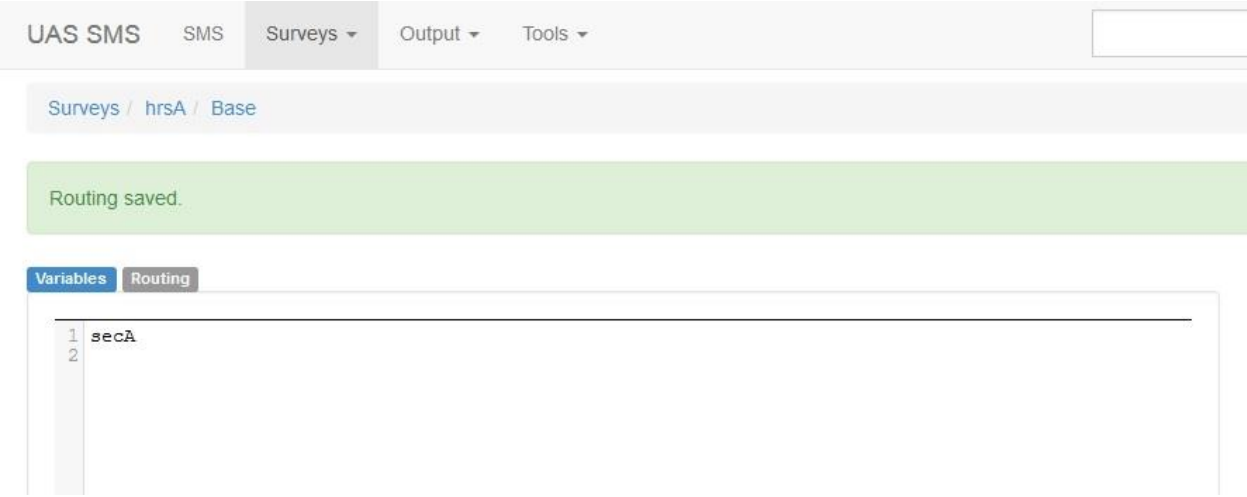


You can then click ‘Show errors’ to see which line(s) contain error(s):



In this case we misspelled ‘X060ASex’ by forgetting to ‘X’ at the end and so NubiS can’t find this variable.
So, now we have added our variable to the routing of ‘secA’. Surely if we test the survey now, it will appear? The answer to that question, however, is no. The reason is that although NubiS

now knows that if we ask ‘secA’ we want to ask ‘X060ASex[1]’, it does not know yet that we want to ask ‘secA’. In order to accomplish this we open a routing screen again, but this time the one belonging to the ‘Base’ section. Here we type in ‘secA’ on the first line and save the routing.



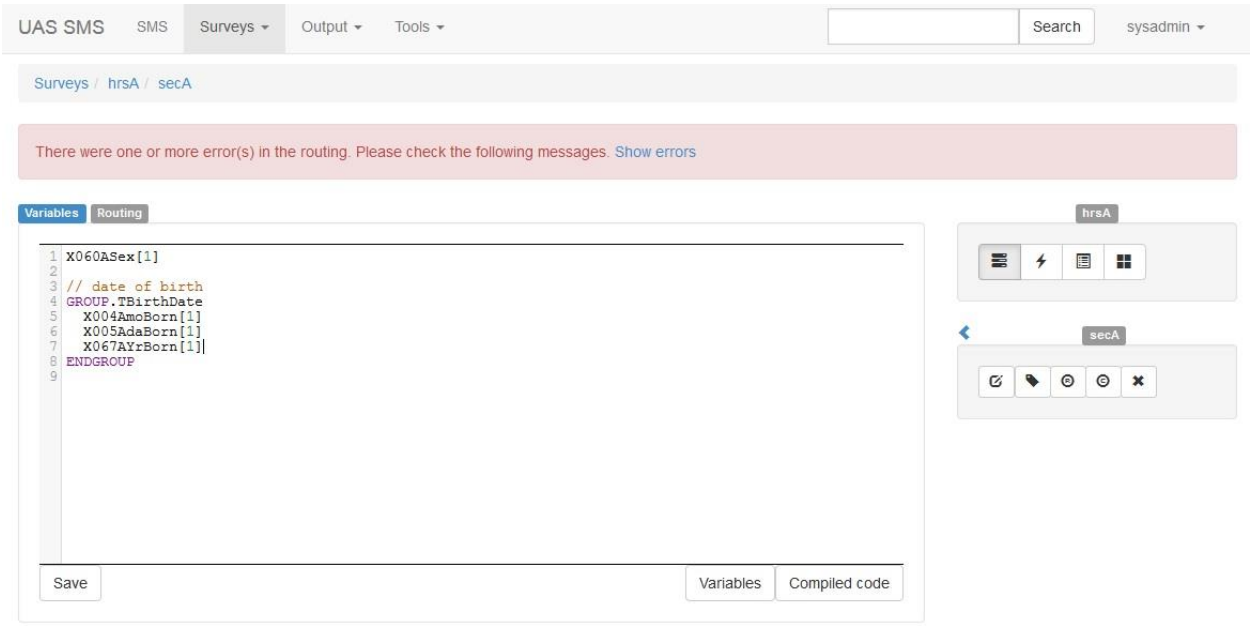
If we now go to ‘Tools | Tester’ and hit ‘Test’ we get:

Here now we finally see our question and the available answer options as radio buttons. We also see the header we specified for our section ‘secA’ (to recall, this was <h2>Background information</h2>, which the browser interprets into a nice heading), a progress bar (which is completely full here because there is only one question, and so we’ve reached the end of the survey), and a ‘Next>>’ button. There is no back button because this is the first screen, so there is no previous screen to go back to.

In this particular case, because we are running the survey in test mode, we also have a bar in the top with a dropdown in the right hand corner. This bar does not usually appear, but can be used here to exit test mode to go back 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

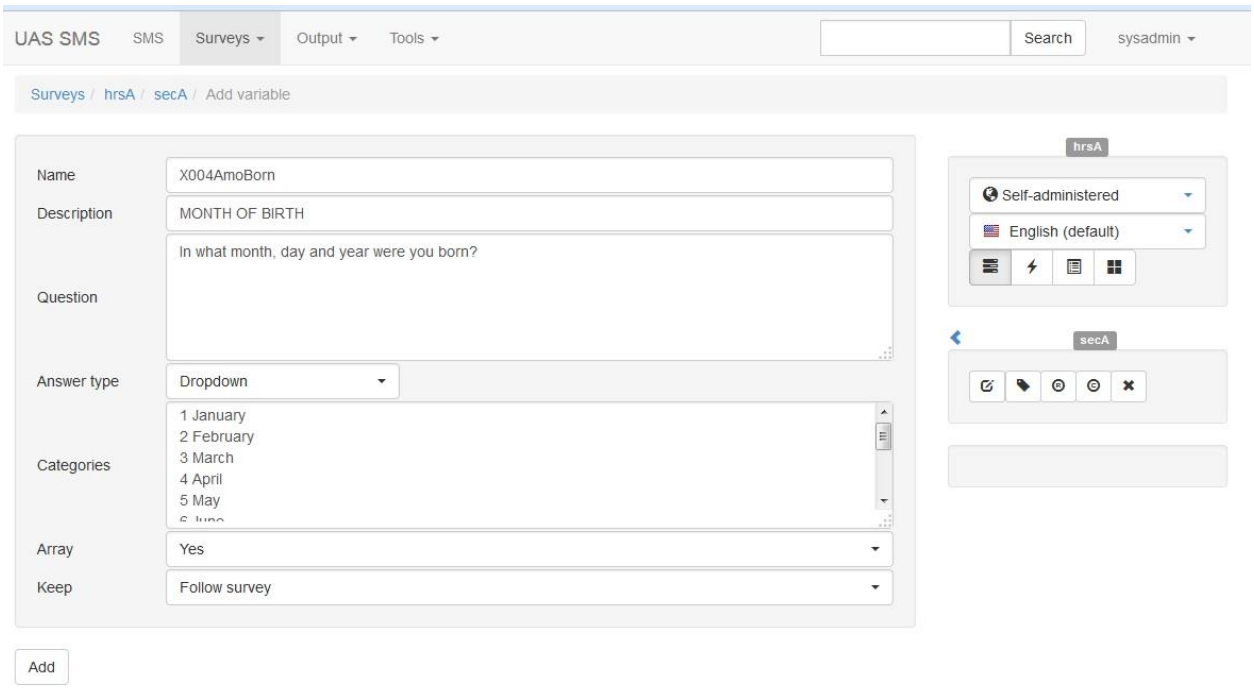
Obviously being able to ask a question is useful, but typically we will want to do more than that. For example, we might want to ask the birth date of the respondent in order to calculate their age. We could do that by first asking the year of birth, then go to the next question screen to ask the month of birth, and then 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 so-called **group** statement:



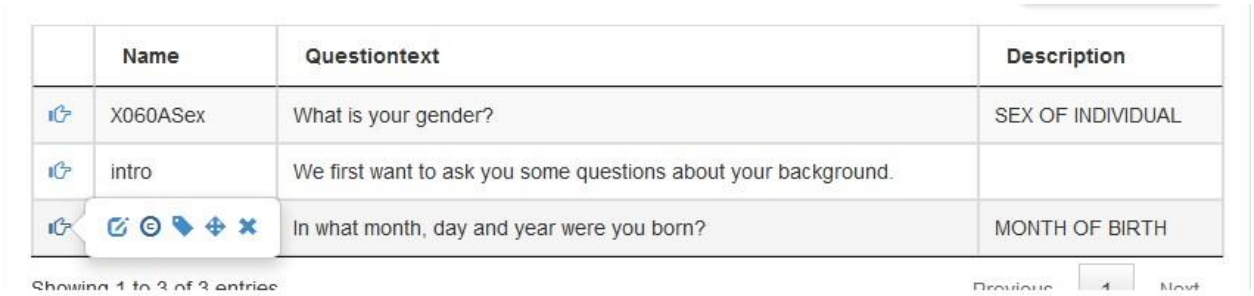
We can see the syntax used in the above code snippet. Ignoring for a moment the errors prompted by NubiS (for not adding the questions we refer to in the group statement), we see that we start a group statement by saying GROUP and close it by saying ENDGROUP. One can think of these as being similar as to how you would open and close HTML tags.

The opening GROUP statement mandates the presence of a group name, so in this case we specify 'TBirthDate'. This group does not exist yet, but when we save the routing NubiS will create it automatically for us (we will see where we can edit groups in a bit). Next, we list the questions to be grouped together (here three variables representing month, day and year of birth) after which we finish with ENDGROUP.

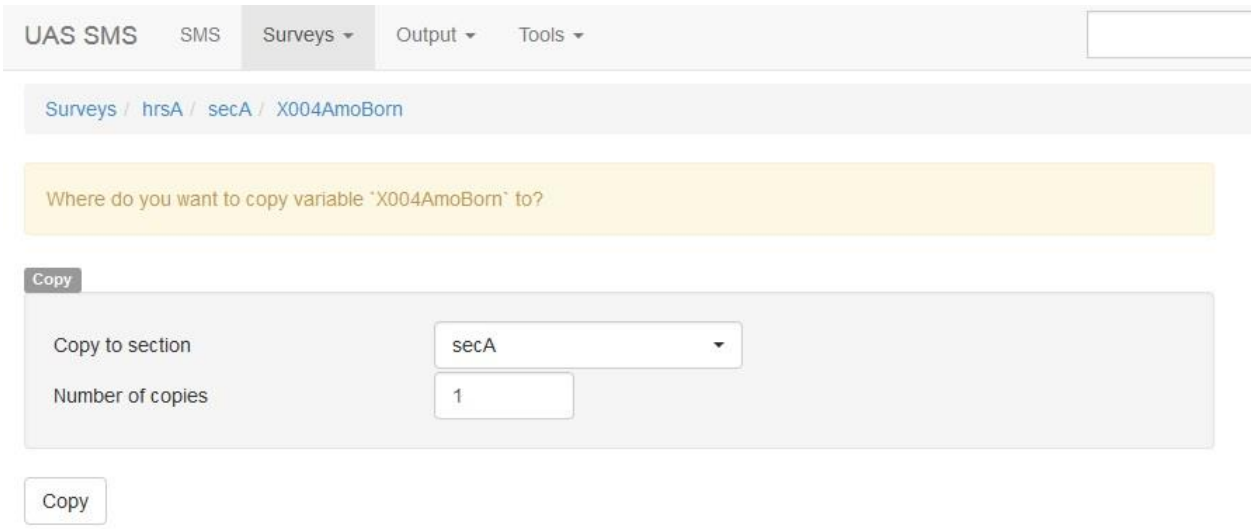
In order to be able to see the effect of this let's quickly add these three questions. We switch back to the 'Variables' tab and click 'add new variable'. Let's start by adding month of birth which we call 'X004AmoBorn'.



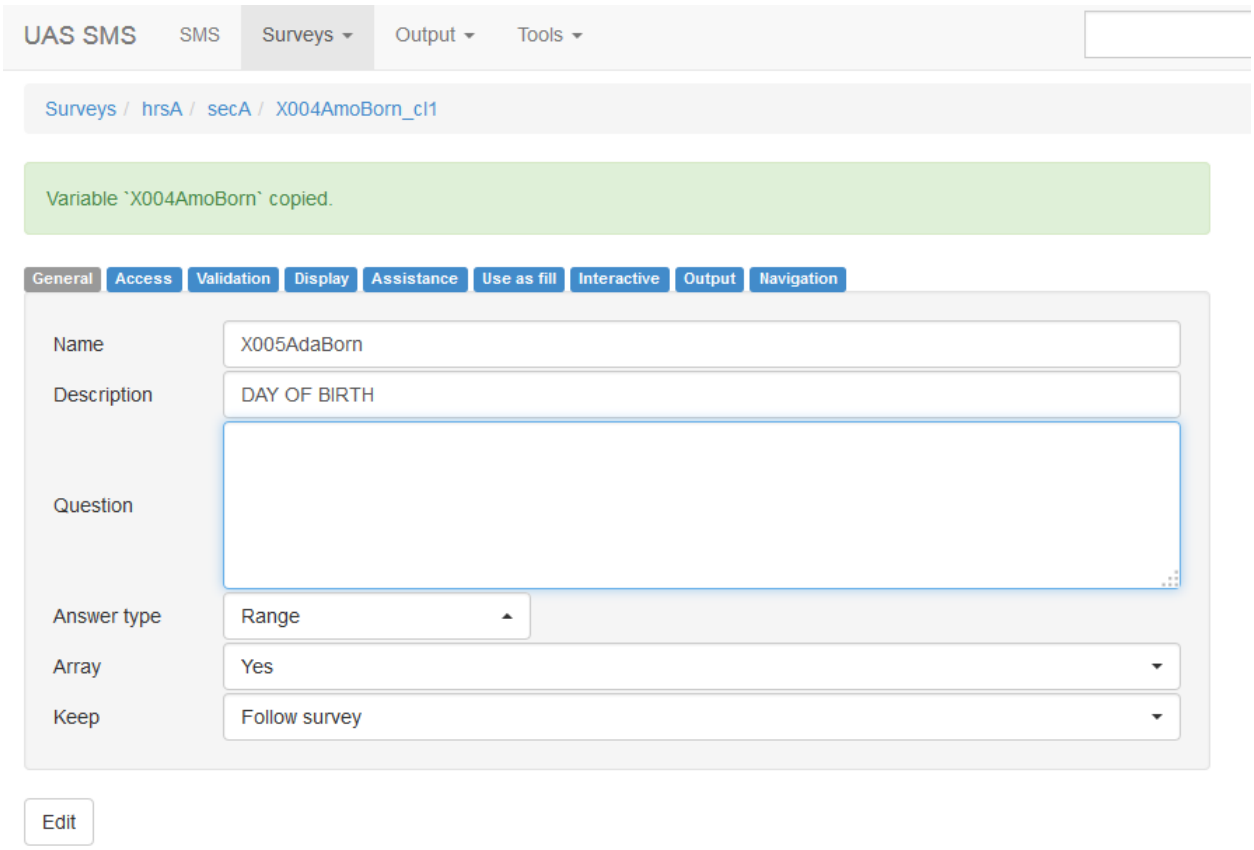
This variable is very similar to the one we added on gender, the only difference being we make it a dropdown; which we do to reduce the space taken up by the answer options. Next, we add day of birth as a range variable. For ease we do so by making a copy of 'X004AmoBorn' by clicking the copy icon in front of it:



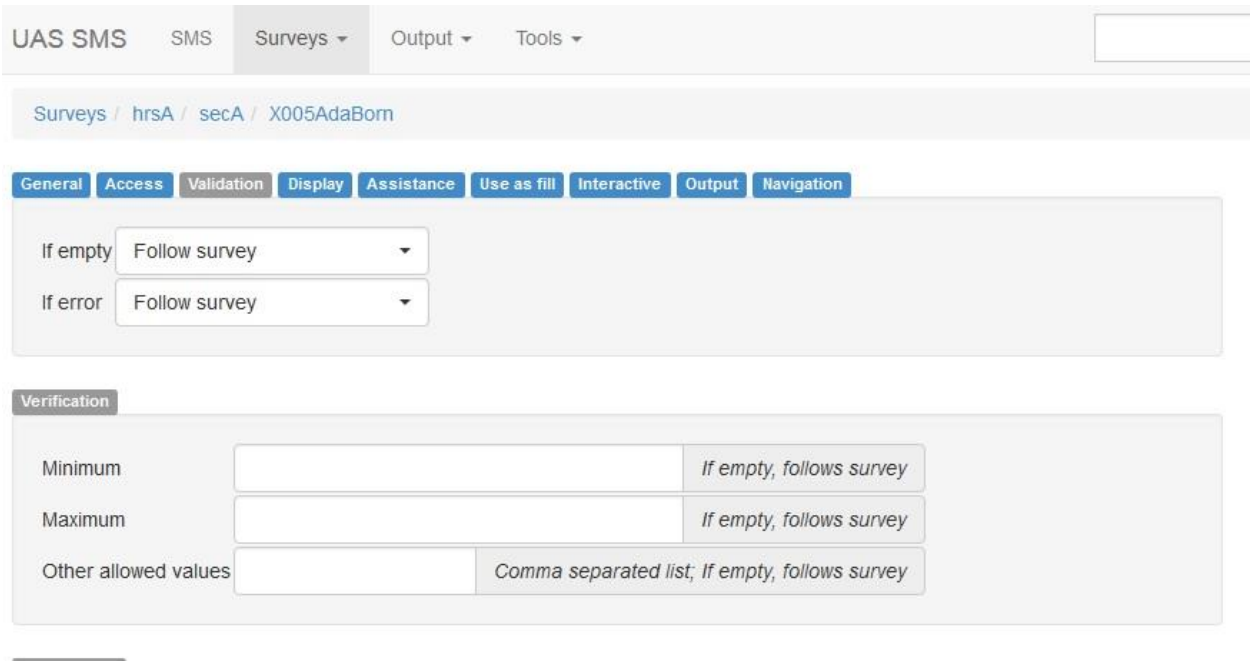
The following screen asks us where we want to make a copy and how many copies we want:



Here we want the copy to be in the same section ‘secA’ and we only ask for one copy. In the resulting screen we change the name of the variable to ‘X005AdaBorn’, change the description to ‘DAY OF BIRTH’, remove the question text and change the answer type to range.

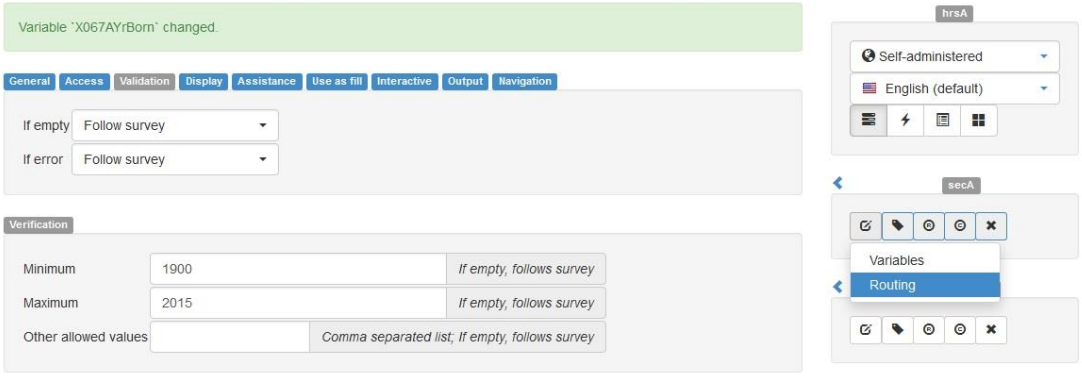


Given that we want to know the day of birth the answer a respondent gives to this question should be between 1 and 31. In order to enter this range we click ‘Validation’:



There is a variety of settings we can specify here, but we will focus for a moment on minimum and maximum. As the screen informs us, if we leave these empty it will follow the survey level setting. This setting itself defaults to 0 and 9223372036854775807 (which is the maximum integer value in PHP), which is clearly not what we want. So we change this to 1 and 31.

The third question to add is for the year of birth. We also want a range here, so we make a copy of 'X005AdaBorn' and call it 'X067AYrBorn'. We also adjust the variable description and save our changes. We then go to the validation tab again and specify 1900 as a minimum and 2015 as a maximum. Note: the observant reader will remark that obviously the maximum for the year of birth should change once the next year starts. As we will see shortly NubiS facilitates this through its support for dynamic text and capability to leverage PHP functions.

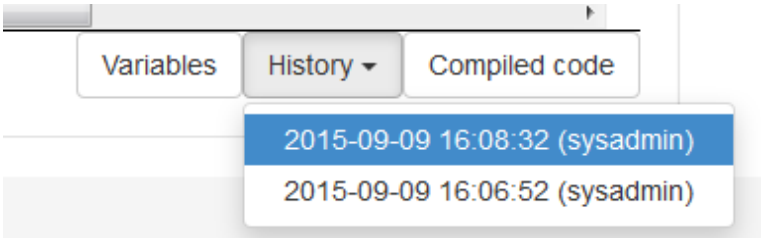


For now, let's go back to our routing:

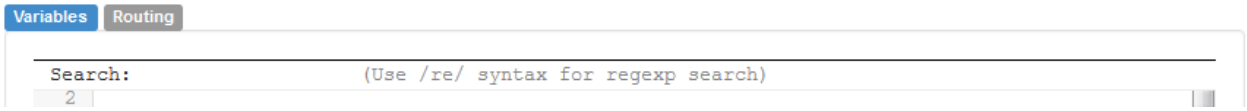
Here we hit save again to update the routing. Note that this is partly because we want to make sure our routing is now correct. More importantly, though, re-saving gives NubiS the opportunity

to re-assess our code in order to be able to interpret it correctly when running a survey. Failing to do so may lead to NubiS attempting to use the result of assessing an older and flawed set of routing instructions.

Before we go and test our survey again, let's take a quick moment to notice an additional option appearing underneath the routing text box:

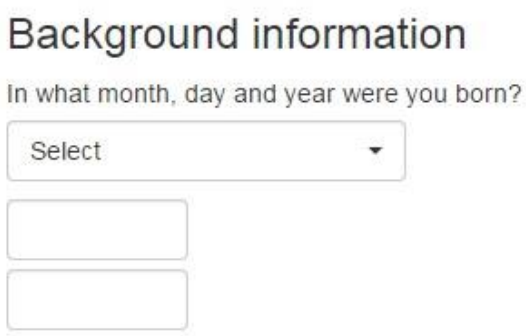


Every time the routing is saved, NubiS logs an entry of the routing at that point in time. These snapshots are available through the 'History' dropdown. This allows to quickly revert to a previous version of the routing. Another useful feature is to search for occurrences of certain text in the routing, e.g. to quickly locate some 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:

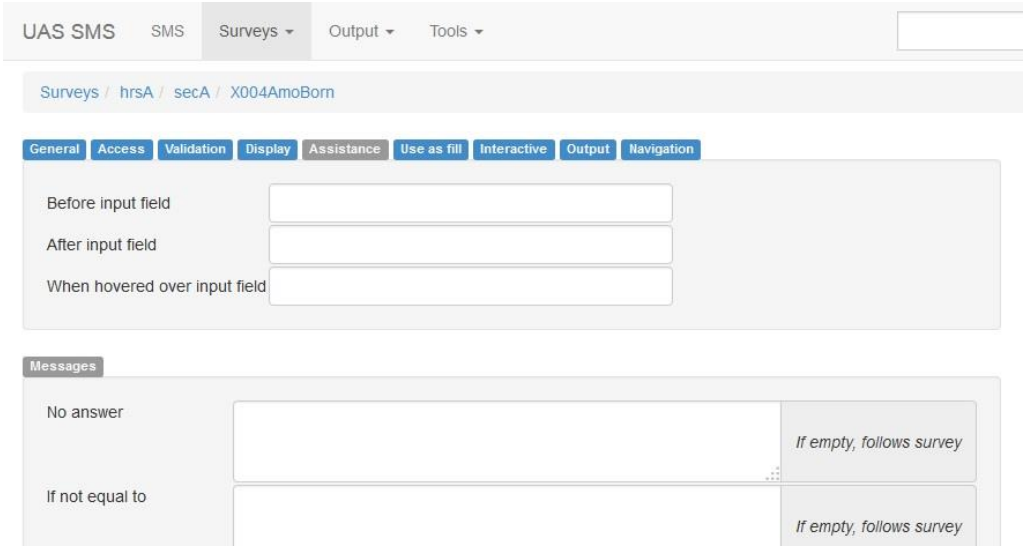


Here we can enter the term for which we want to search (the search is case insensitive). On pressing Enter each occurrence will be highlighted. We can then move from one occurrence to the next by using Ctrl+G (or Cmd+G). Similarly, we could use Shift+Ctrl+F (or Shift+Cmd+F) to replace certain occurrences. Note that the latter is advised to be used when not renaming a variable. If this is what is intended, then using NubiS built-in refactoring option is best. Alternatively, we can also search for the occurrence of some text in the routing of any and all sections by utilizing NubiS generic search functionality.

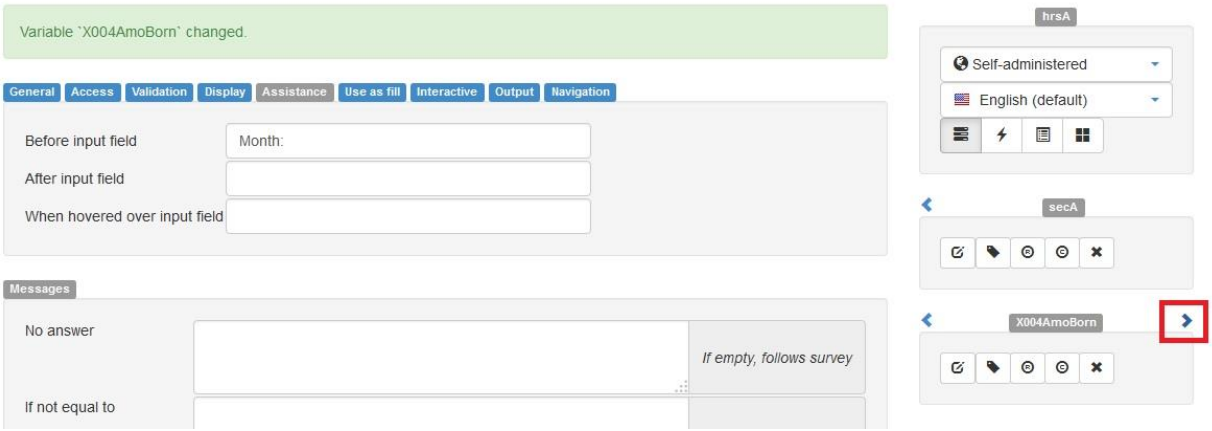
With all that, let's return to the topic at hand and go to test our survey again (under Tools | Tester). We give an answer to the first question about gender, and then click 'Next':



Above we see the result of our work. We see our question and the three answer boxes. One is a dropdown (for the month of birth), the other two for day and year of birth. But which one is which? To make this a bit easier for the respondent let's exit test mode and go back to 'X004AmoBorn'. There we click the 'Assistance' tab:



Ignoring all other settings available to us here, we focus on 'Before input field'. This setting allows us to enter some text, which will then be inserted before the answer box (in this case the dropdown). Let's enter 'Month:' here and save our changes. We can easily do the same for our day and year of birth questions. To go to 'X005AdaBorn' we can use the navigational arrows on the right hand side (highlighted in a red box here):



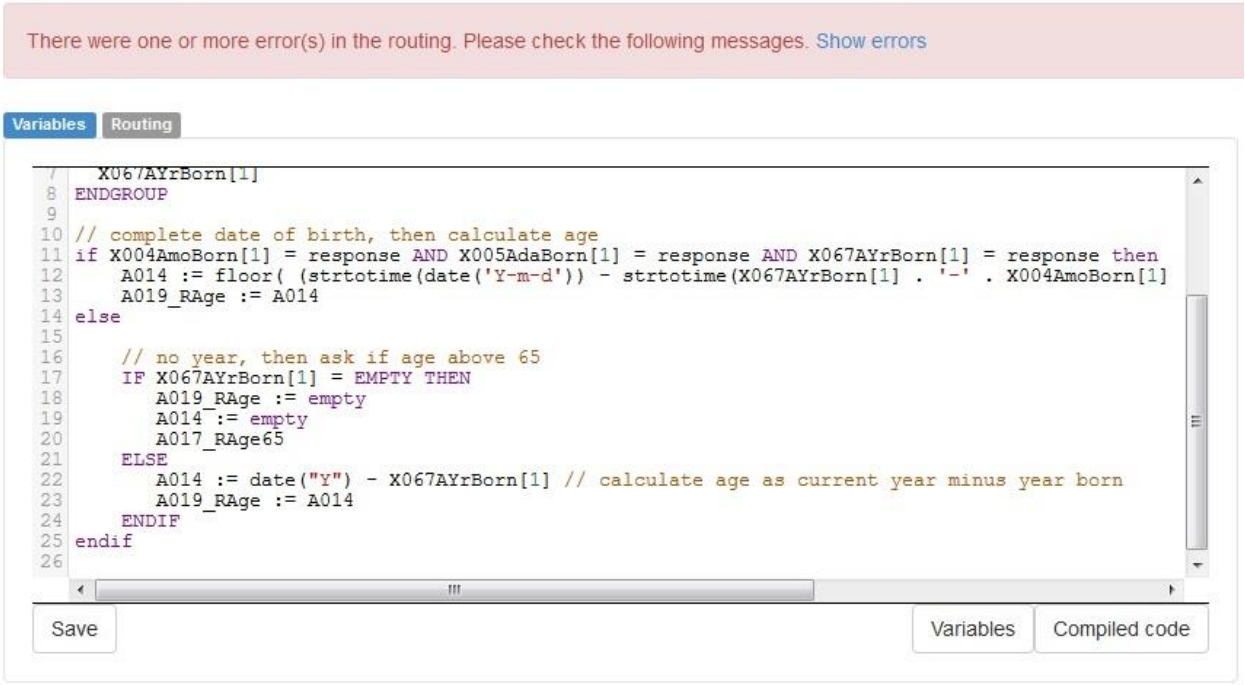
These navigation links allow us to quickly move from one variable to another (a similar mechanism is in place to go from one section to another). Using these links we add 'Day:' and 'Year:' in the 'before input field' settings of 'X005AdaBorn' and 'X067AYrBorn'.

Testing our survey again gives us:

Notice that we deliberately attempted to enter 2016 here as a year of birth and clicked 'Next' to show that NubiS will detect this and provide an error message (similarly, we see it also detected that we left the month and date of birth empty).

9. Asking questions under certain conditions

Besides the capacity to group questions, another useful routing construct is skip patterns. Skip patterns in NubiS are realized through **IF THEN** statement. These can be 'positive' in nature by stating 'if this is true, then ask this question' or 'negative' in nature (if this is not true, then ask this question). To illustrate the usage of skip patterns let's see how we can employ them to calculate the respondent's age based on the birth of date details s/he provided. For this purpose we are adding the following snippet of routing instructions to the routing of section 'secA':



At first glance what we see looks quite overwhelming. And although there is a lot going on, the intuition of each line is quite easy to understand. First, let's look at the basic structure. On line 11 we see an **IF** statement specifying a condition. If that statement is true, then we instruct NubiS to perform the actions specified on line 12 and 13. If it is not true, then NubiS notices our **ELSE** statement. Such **ELSE** statement can be employed to instruct NubiS what it should do when the IF statement (or IF statements if multiple) that came before the **ELSE** were not true. In this regard the actions listed in the **ELSE** can be thought of as the default behavior we would like to happen if the conditional behavior is not applicable.

Zooming in on the **IF** statement on line 11 we see the names of the variables we just asked the respondent about in a series of logical expressions combined by **AND** operators. Informally, what we are saying here is that if the respondent gave us a month (X004AmoBorn[1] = response), a day (X005AdaBorn[1] = response) and a year (X067AYrBorn[1] = response), we

have all the information needed to calculate the respondent's age. If we don't, then we'll need to do something else (which is specified in our **ELSE** statement starting at line 14).

Let's assume for a moment the respondent gave us a full date of birth. This means the conditions on line 14 are satisfied, and NubiS attempts to perform the actions on line 14 and 15:

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 variable specified to the left of the ':='. Assignments can be as simple as line 13, in which we assign the value of 'A014' to 'A019_Rage' (which are two variables we have not added yet, but we can think of as two integer variables that will hold the age of the respondent). However, they can also be more complex. For example, 'A014' itself 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 we leverage existing functions in PHP to determine how much time in milliseconds has passed between now and the date on which the respondent was born. We divide the result by 31556926 to get the equivalent period in number years. We then floor the result to obtain the age as a whole number. For example, if the respondent had entered June 23, 1978 as their date of birth, then the calculated age would be 37 if today was June 24, 2015. This usage of existing PHP functions is very natural in NubiS and in fact extends to the usage of user defined functions (a topic to which we will return in Section ?).

Of course, all these calculations to deduce the respondent age presupposed that we have a complete date of birth. If we don't, then we need to find another way to determine the age. This is where the ELSE comes into play. If we look inside the ELSE, we find 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 there tells NubiS that if it doesn't have a year of birth (IF X067AYrBorn[1] = EMPTY THEN), we have nothing to base our calculation on (EMPTY is a reserved word here indicating no answer was given). So we ask it to perform the following actions:

A019_RAge := empty
A014 := empty
A017_RAge65

These three lines boil down to instructing NubiS to set the values for ‘A019_RAge’ and ‘A014’ to empty, and then ask ‘A017_RAge65’, which is used in HRS section A to ask if the respondent is older than 65 or not.

If we do have a year of birth, then we tell NubiS to calculate the 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 saw in the above two lines and in the code snippet we introduced that we inserted some expressions like *// calculate age as current year minus year born*. Any text in the routing that is preceded by // is considered to be a comment by NubiS (multi-line comments are also supported by starting with /* and ending with */).

So far we have created a survey that will ask a respondent about their gender and date of birth. The next step we want to accomplish is to ask about the respondent’s marital status. To that end we will introduce variables that ask if the respondent is married and if not, if they have a partner as if they are married. Let’s first define the necessary routing:

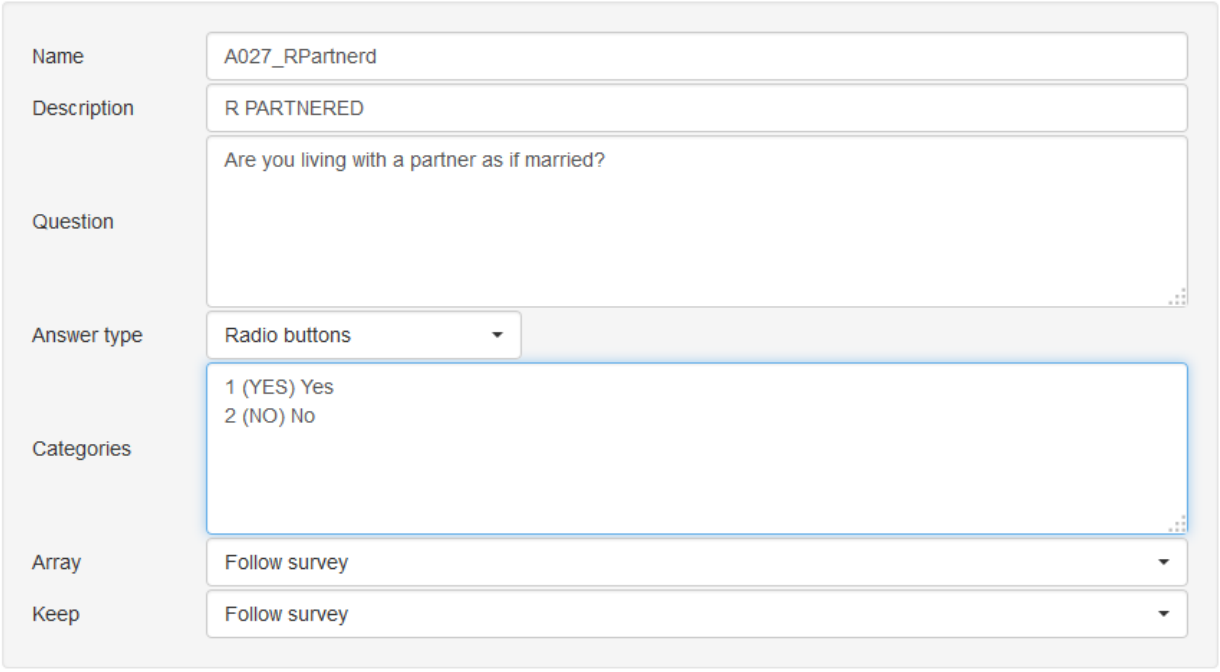
```
22          A014 := date("Y") - X067AYrBorn[1] // calculate age as current year minus year born
23          A019_RAge := A014
24      ENDIF
25  endif
26
27  // married or with partner
28  A026_Rmarried
29  if A026_Rmarried != YES then // not married
30      A027_RPartnerd
31      if A027_RPartnerd = 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 we instruct Nubis to ask ‘A026_Rmarried’ (is the respondent married). If the answer is not yes (line 29), we ask if they have a partner (‘A027_RPartnerd’). If they say yes to ‘A020_RPartnerd’ (line 31), we set variable ‘A166_A020TSameSpP_A’ to ‘YES’ (line 32) and assign the value ‘3’ to ‘X065ACouplenss[1]’ on line 33 (we will see shortly what ‘3’ means when we add ‘X065ACouplenss’ as a variable). If the respondent also did not say yes to ‘A020_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 all this is pretty much the same in terms of routing constructs as we just saw when calculating the respondent age. There are a few new things though. Firstly, line 29 introduces a new symbol ‘!=’, which means not equal to. NubiS also supports other types of

comparison like greater than, greater than or equal to, less than, and less than or equal to. Secondly, in some of our conditions and assignments (e.g. line 31 or 32) we use the word 'YES'. How does NubiS know how to interpret this word 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 a condition like we specified, it will attempt to find a definition for 'YES' in the variable used in the condition. Let's see how we can do this.

First, let's quickly save our new routing and ignore NubiS' complaints about variables it can't find. Next, we head over to the 'Variables' tab and say 'add new variable'. In the resulting screen we specify the following:



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

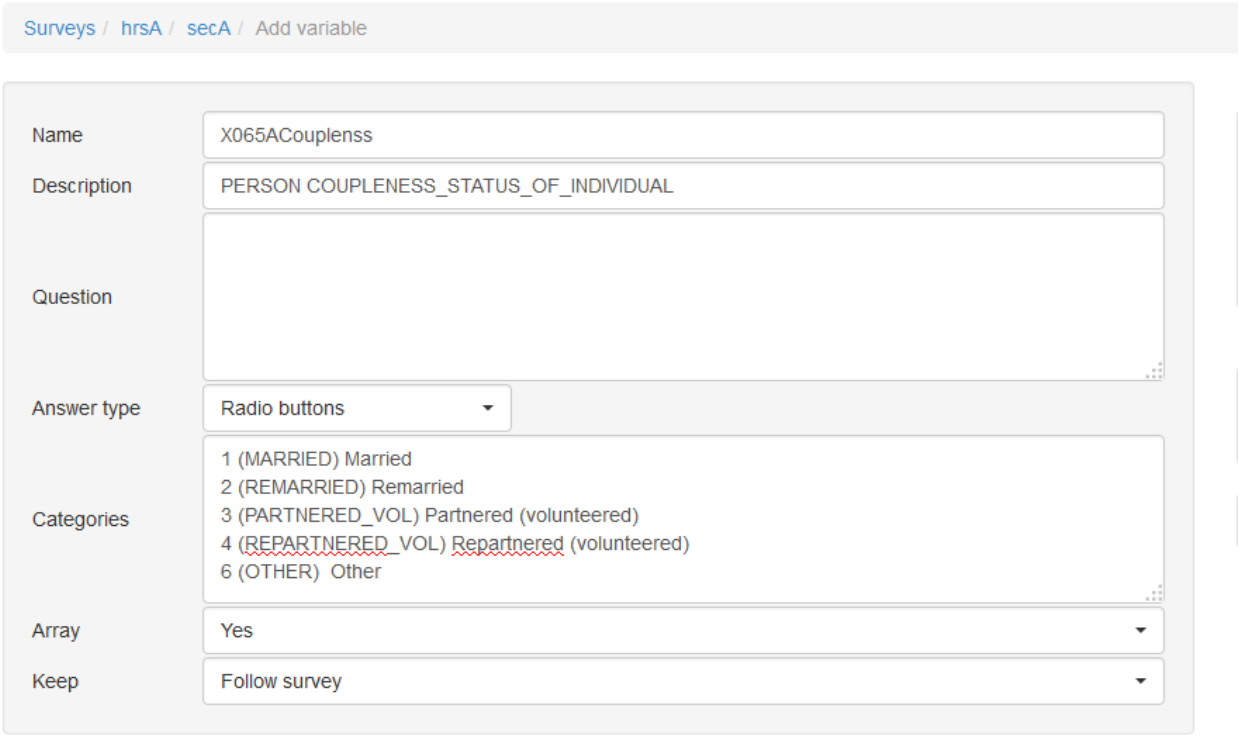
1 (YES) Yes
2 (NO) No

The proper syntax for a value label is to start off with an answer code (which must always be the case for an answer option), then a value label enclosed by (and) and then the answer option text shown to the respondent. Here we specified our value labels in capital letters. This is not mandated but typically helps to make them stand out while looking at a set of answer categories.

In a similar vein we can also 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. 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, if we want to ask a question based on this fact, we can simply say something 'IF A166_A020TSameSpP_A = YES THEN' instead of the longer 'IF A026_Rmarried = YES OR A027_RPartnerd = YES'.

After adding these three variables we can return to the 'Routing' tab and save again. If all is well, we will only have a single source of errors left, which is the non-existence of 'X065ACouplenss'. We quickly add that variable as well:



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

X065ACouplenss[1] := 3

We could have said:

X065ACouplenss[1] := PARTNERED_VOL

on line 33 of our routing, which is more readily understood than just '3' (which is the main reason for using value labels, that is, to make the routing more easy to understand).
By and large the constructs we discussed so far for specifying conditions in Nubis cover most cases. There are a few additional items worth discussing though.

For starters, let's suppose we want to ask a question to everyone who is either married or partnered with someone else while using 'X065ACouplenss[1]'. One option would be to state something like:

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

Inherently there is nothing wrong with this. But suppose now that instead of 2 eligible answers we have five. We would have to specify five conditions then separated by 'OR' to cover all of them. Luckily, there is a more concise way and it 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 we could change it further to:

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

Here we incorporate usage of the value labels in X065ACouplenss to make it easier to understand the condition.

Another construct similar to the above that is also frequently used deals with set of enumerated (or multi-drop down) variables. Suppose we have a question 'Q1' we want to ask about any childhood diseases you might have had. We provide a list of them, e.g.:

1 Asthma
2 Diabetes
3 Respiratory disease
4 Cancer

Now let's also assume we want to ask follow up questions for each selected disease. Instinctively we might want to say something 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 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 then 1 it would be '3-1'. Neither are equal to 1 or 2. To avoid this we need to specify this instead:

if 1 in Q1 then
    // ask follow up questions
endif

if 2 in Q1 then
    // ask follow up questions
endif

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

Q1 := '1-2'

Alternatively, we could also say:

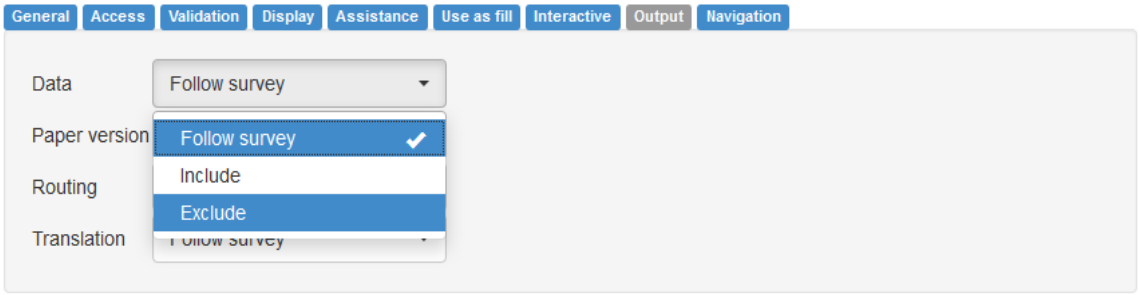
Q1_1_ := response
Q1_2_ := response

To reflect 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. We will see shortly in the next section that we can also use references of the form 'Q1_1' when using previous answers as dynamic text.

10. Using dynamic text in questions

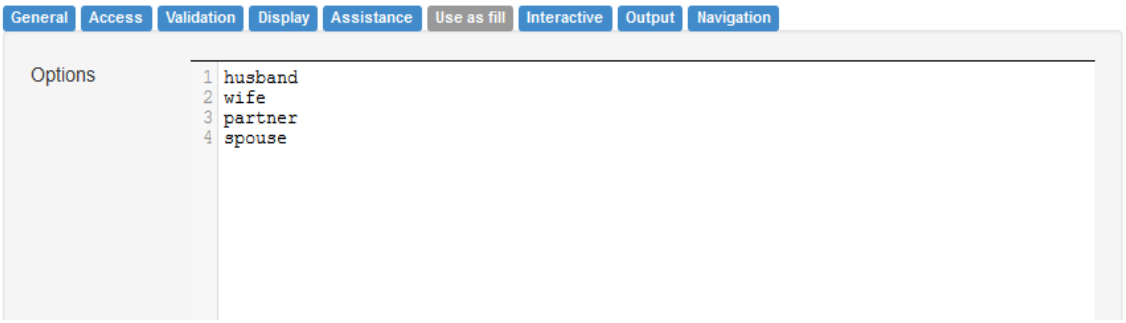
At this stage let's add some questions to learn some more about the respondent's spouse or partner (if any). One aspect we need to deal with is how to phrase our questions. Say for example we want to ask the respondent about their spouse/partner's gender. One option is to write the question as 'What is the gender of your spouse/partner?'. Although this is a valid question text, it may leave the respondent a bit puzzled why the question would not just state 'spouse' or 'partner'. After all, s/he just told us whether they are married or living with a partner as if married. Luckily in NubiS you can use this previous information to construct so-called **dynamic text**, also referred to often as a **fill**. The general idea is that 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 give it a name. Here, following HRS, we will name this variable 'FLHWP'. We can leave the description and question text empty, since this question will not be used by itself to present a question and collect data.
2. Next, edit the variable and select the 'Output' tab. In it we set 'Data' to 'Exclude'. This has no effect on the survey workings, but will instruct NubiS to not include this variable in any outputted data set.



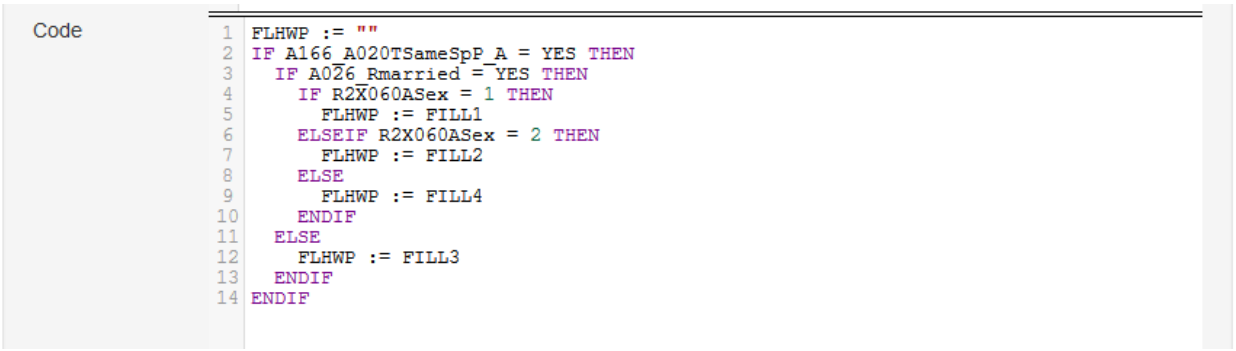
3. Open the 'Use as fill' tab. This tab consists of two parts: options and fill code. In the options part we can state the possible values the fill variable can take. Here (as can be seen in the image below), we enter the following options:

   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 we can specify under what conditions the fill variable will take which value:

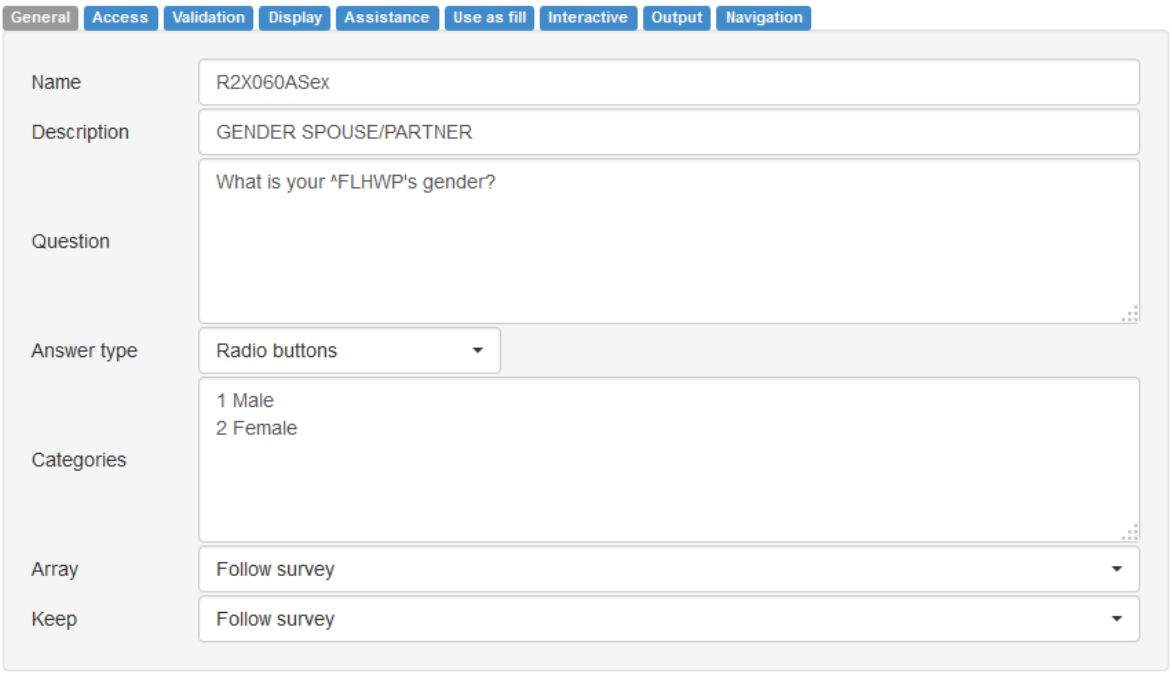


Most of the above syntax already looks familiar. This is because fill code uses the exact same syntax as normal routing with the exception that in fill code we can not instruct NubiS to ask a variable. Instead we can only instruct NubiS to perform assignment(s).

In the fill code for our fill variable we start by setting ‘FLHWP’ to an empty string (“”). Then, if our flag variable ‘A166_A020TSameSpP_A’ is equal to ‘YES’, we determine whether the respondent is married (A026_Rmarried). If yes, then if ‘R2X060ASex’ equals 1, we set ‘FLHWP’ to something called ‘FILL1’. Obviously we haven’t added ‘R2X060ASex’ yet, but this question will basically ask about the gender of the respondent’s spouse/partner. The mystery text ‘FILL1’ is not so mysterious, as it simply corresponds to the first option we specified in the options part (here ‘husband’, since 1 in ‘R2X060ASex’ is the code for ‘Male’)

Following the different conditions tells us that 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, we want to assign ‘FILL4’ (line 9). Lastly, if the respondent has a partner instead of being married, we assign ‘FILL3’ (line 12).

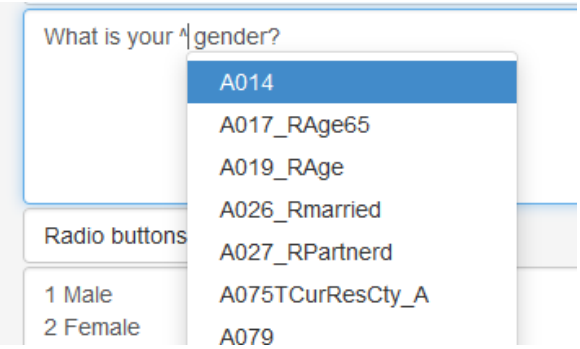
Now, let’s add ‘R2X060ASex’:



This is yet another radio button variable with answer options ‘Male’ and ‘Female’ this time. What is different though is the question text:

What is your ^FLHWP's gender?

We introduced a reference here to our fill variable ‘FLHWP’ by using the ‘^’ symbol. This 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’ was ‘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:



We can continue typing while the popup is there to locate ‘FLHWP’. For example, the moment we hit ‘F’ the list will update to show all variables starting with that letter. This auto-completion feature is available for any setting for which variable references are supported (including other types of reference which we will discuss later on). Note that if we are referencing an array question we still need to type in the instance indicator, e.g. ‘[1]’.

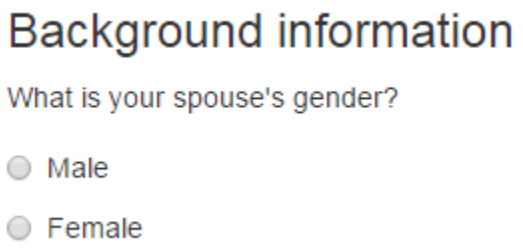
Right now, at this stage there are only two things left to do to see our fill variable in action. One is to return to the ‘Use as fill’ tab of ‘FLHWP’ and re-save the routing code. This gives NubiS the chance to update its interpretation of the fill code now that ‘R2X060ASex’ is present.

The second is to instruct NubiS to ask ‘R2X060ASex’ while using the ‘FLHWP’ fill. This is accomplished on the routing tab:

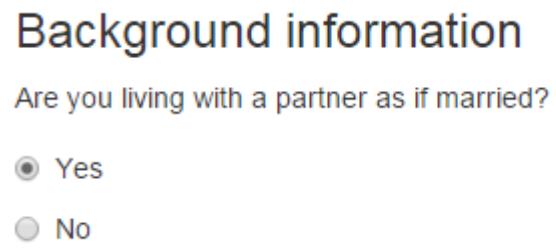
```
44 // set fills
45 FLHWP.FILL
46
47 // ask spouse/partner details
48 IF A166_A020TSameSpP_A = YES THEN
49  R2X060ASex
50 ENDIF
51
```

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

If we now run our survey again through the tester and we say we are married, then we will get the following screen:



If we say we are not married, but partnered we get:

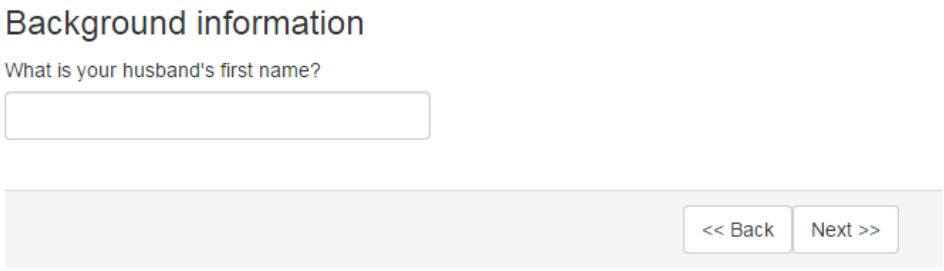


You might wonder why we don’t get ‘husband’ or ‘wife’. The reason is simple. Those fill options will only be triggered once we know the gender and we are asking the gender right now. To showcase this, let’s add another question asking about the first name. We’ll make a copy of

‘R2X060ASex’, rename it to ‘R2X058AFName’, change the label to ‘FIRST NAME SPOUSE/PARTNER’, the question text to “What is your ^FLHWP's first name?”, and lastly the answer type to String. After adding this variable we 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
```

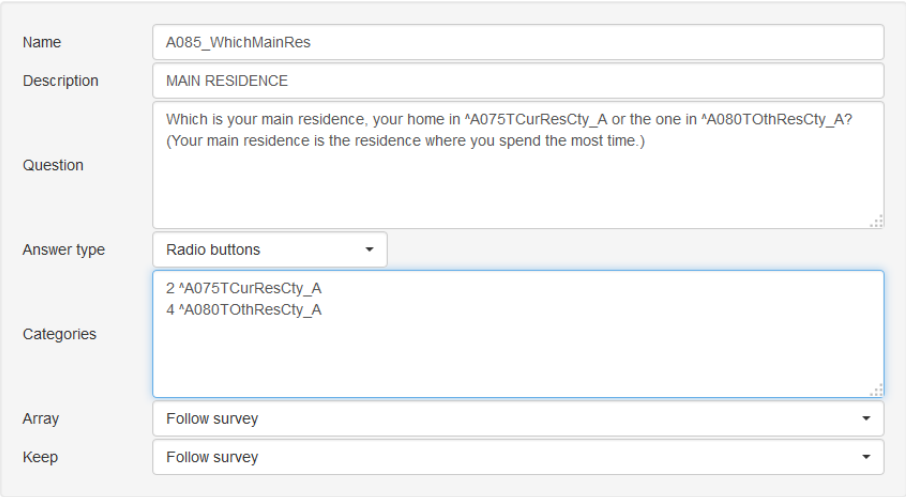
First we instruct NubiS to execute the FLHWP fill again after we have answered ‘R2X060ASex’. Then we want it to ask ‘R2X058AFName’. The reset on line 50 is to ensure that the value for ‘FLHWP’ is refreshed with the new information in mind. If we run the survey once more, we get for example:



The usage of a file like we just did is an extremely powerful way to incorporate dynamic text inside your survey. For example, let’s add a few more variables to ask which city the respondent is living. Specifically, we’ll add:

‘A075TCurResCty_A’: In what city is your residence currently located? (answer type String)
‘A079_’: Do you have any other home? (radio buttons with answer options 1. Yes and 2. No)
‘A080TOthResCty_A’: In what city is your other residence located? (answer type String)

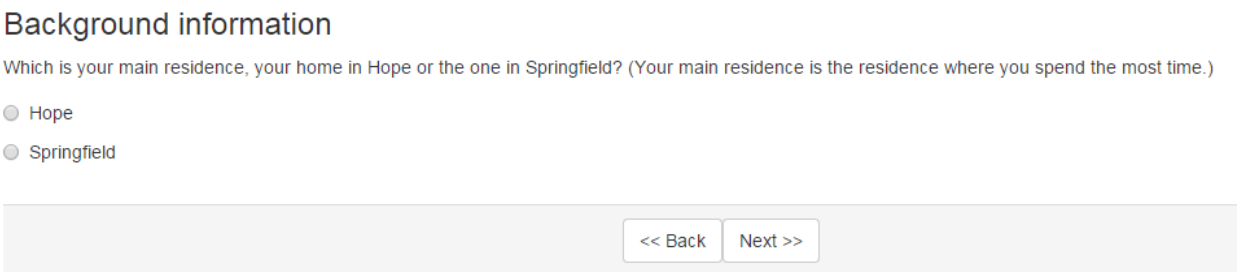
We then also add a question to figure out which one is the main residence:



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 we run the survey and reach this point, we get the following (supposing for a moment we entered a current city, said yes we have another home, and listed another city there):



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 though 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'. This is because those variables never get a value. Also, variable value references can be of an exact or interpretative nature. The latter we have seen so far and are indicated by a '^'. The end is typically demarcated by a non-alphanumeric character, such as a '+' or ',' or '.'. These

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

The ‘interpretative’ part reflects the fact that 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 for radio buttons, dropdowns, checkboxes and multi-select dropdowns it is not. Rather, for those NuBiS will look up the answer label(s) corresponding to the answer and return that text instead. To illustrate, if we were to reference ‘A085_WhichMainRes’ and the answer was ‘1’, we 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 is always inserted. In the case of ‘A085_WhichMainRes’ this would be ‘1’ if the first answer option was chosen, 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. If we recall our variable ‘Q1’ from the previous section asking about childhood diseases, suppose we want to use it as a fill in a follow up confirmation question. We can do so by using a question text like this:

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

If we 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. This is great, but maybe we had something more along the lines of a list in mind. Luckily we can also accomplish that:

You said you had the following childhood diseases:

*^Q1_1_*
*^Q1_2_*

Here we reference the individual answer options of our ‘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

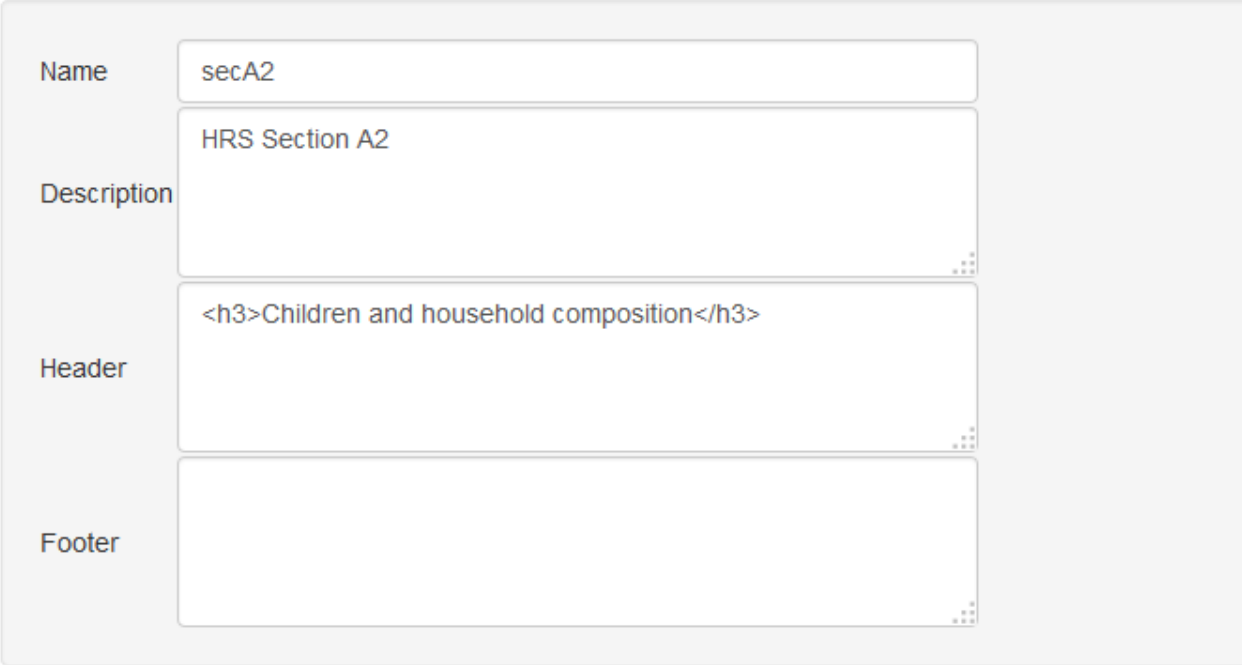
At this point let's summarize for a moment the different routing constructs we introduced. We have seen how to instruct NubiS to ask questions, use skip patterns via IF-THEN statements, and leverage assignments to assign values to variables. Something that we have blatantly ignored however is why we made some of our variables an array and others not. For example, we specified 'X060ASex' as an array and subsequently included it in our routing as 'X060ASex[1]' (instead of just 'X060ASex').

The basic reason underlying this choice is the manner in which the HRS study captures information such as gender and name of not only the survey respondent, but also any children or other members of the respondent 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 several reasons: the first is that it would mean that we would need to introduce enough variables for the gender of any people mentioned by the respondent. This is complicated by the fact that we would not know a priori during survey programming how many we should define. The second is that it would force us to include each variable individually on the routing leading to something like 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 its for loop routing construct. A for loop is basically an instruction to NubiS to carry out the actions inside the loop for a certain number of times. Let's exemplify this by adding a snippet of section A2 of the HRS to our survey that will allow us to ask if a respondent has any children, and if yes, specify what their gender and date of birth is.

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



Next, we add a variable containing our question on whether the respondent has any children. There are several ways that we can go about this. We could ask for example how many children a respondent has. We could also ask if a respondent has a child and if so, ask our gender and date of birth questions. Both are equally feasible and illustrate different parts of using a for loop, so we will do one first and then the other.

Starting by assuming we will ask 'how many children do you have', we add an integer variable to our new section 'secA2'. Since HRS chooses the other implementation, we'll just call this variable 'numberofchildren' for convenience sake. We also introduce a so-called **counter** for the loop we plan to create. This counter will be used to keep track of how often we have gone through our loop, and it is typically an integer question as well. So let's make a copy of 'numberofchildren' and rename it to 'cnt'.

With all that out of the way we are ready to specify our 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 2 lines are straightforward. Basically we ask 'how many children do you have?' and if the answer is more than zero we go into our loop. Let's take a closer look then at the different components of this loop. The first line (line 3) states:

for cnt := 1 to numberofchildren do

This reveals the general syntax of a for loop. We start with '**for**'. Then we list our counter field 'cnt' setting its value (as indicated by the ':=' symbol) to the number at which we wish to start the loop (in this case '1'). We follow this by '**to**' and the maximum number of times we wish to perform the loop. Here we use the answer to our question as the maximum (Note that we could also have set a fixed number of course, e.g. 10). We finish the loop statement by adding '**do**'.

Next, on line 4 and 6-10 we tell NubiS to ask gender and date of birth. To ensure that we capture the respondent's answers separately for each child, we 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, we close the loop by saying **enddo** (and closing our original **IF** statement with an **ENDIF**)

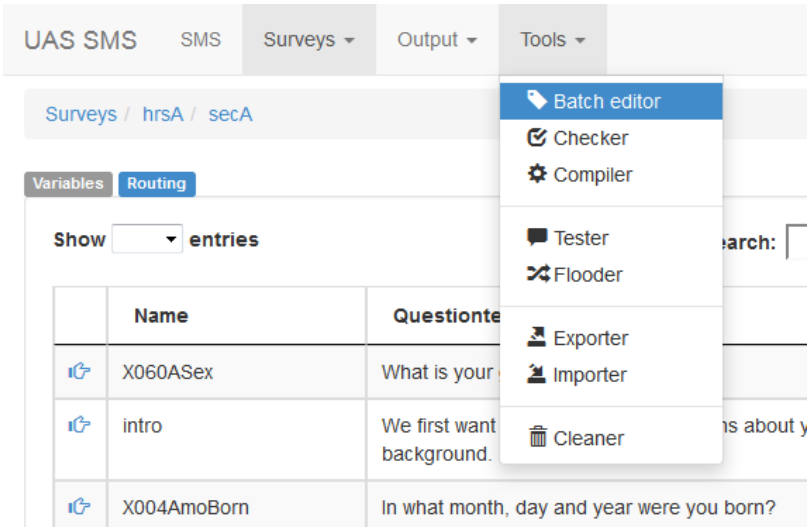
To see our new snippet in action we just need to add 'secA2' to the base section routing:

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

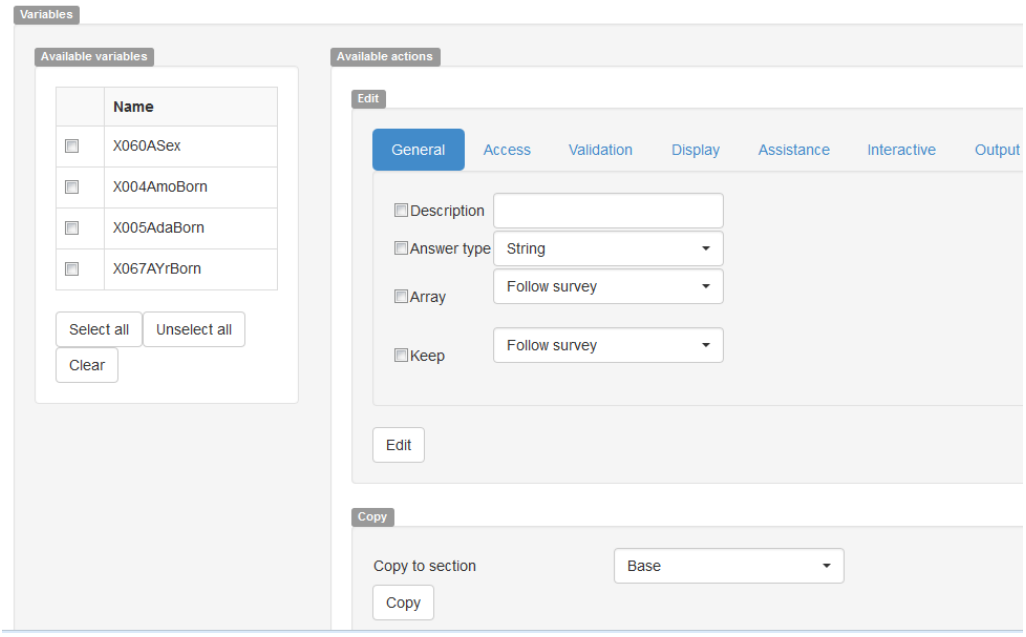
(For testing purposes we commented out 'secA' for a moment, so we can immediately start with the new part). Now, if we enter for example 2 as the number of kids, we will see the 'What is your gender' and 'What is your date of birth' questions two times before the survey concludes. Obviously the question texts are wrong, because we ask the same question twice and moreover we ask it as if we are asking about the respondent themselves. There are two ways to deal with this: one is to introduce fills that are dependent on the 'cnt' variable to change the question text. The other is to define new variables with different question text and then store the values of these variables in the ones we have now. NubiS has no preference one way or another, so the choice between them is ultimately a programming decision (typically what is most easy and intuitive to accomplish).

Here we will follow the second discourse while also showcasing the alternative way of setting up our loop. First we'll make copies of the relevant questions we want to ask. To do this in one fell swoop let's go to the 'Variables' of 'secA'. Hover over the pointed finger in front of e.g. 'X060ASex' and click the 'tag' symbol:

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



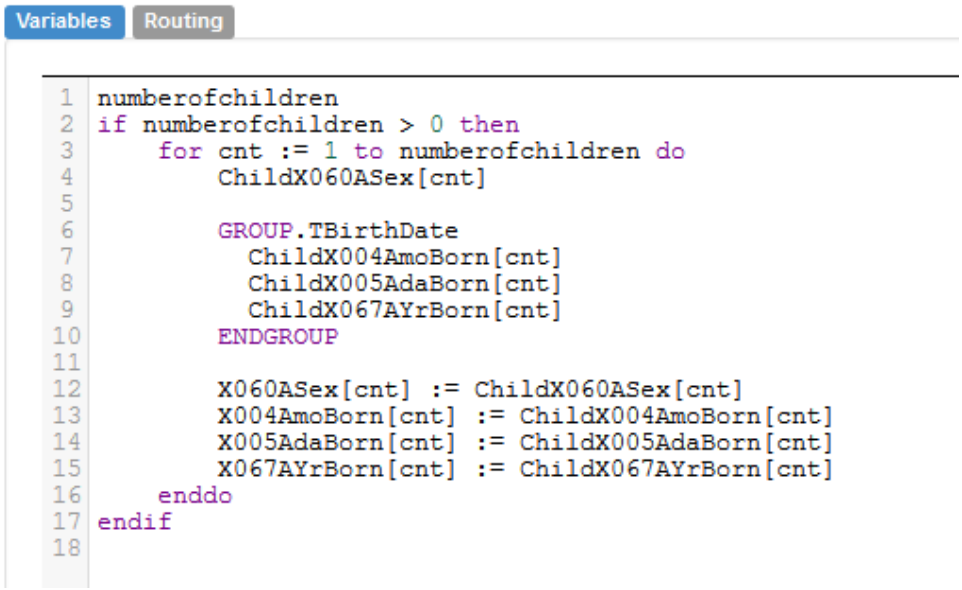
The screen that opens will look as follows:



The batch editor facilitates batch style operations on multiple variables, types, groups, and sections. For now let’s click ‘Select all’, select ‘secA2’ in the dropdown next to ‘Copy to section’ and click ‘Copy’. The batch editor will confirm your action. Now, we return to ‘secA2’ to find copies of these variables there. Rename all variables to remove the ‘_cl’ and start them with ‘Child’:

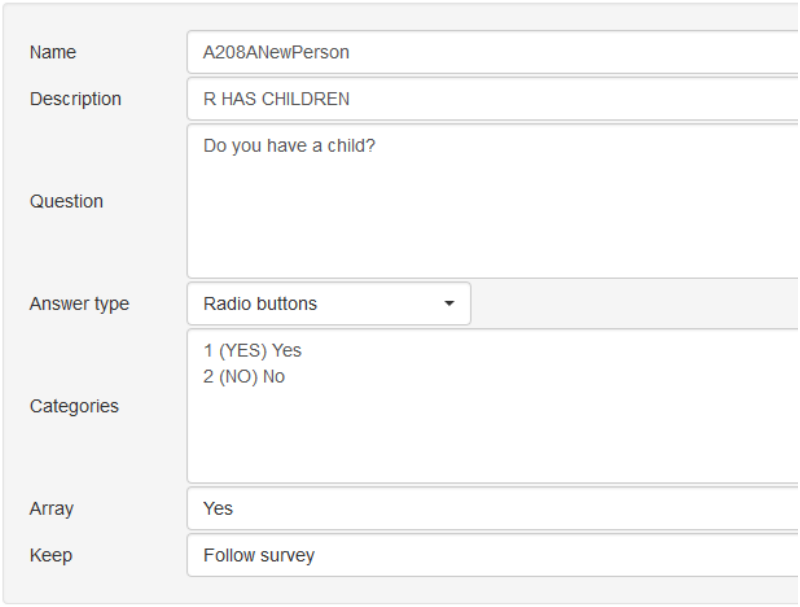
	Name	Questiontext	Description
	numberofchildren	How many children do you have?	
	cnt		
	ChildX060ASex	What is the gender?	SEX OF INDIVIDUAL
	ChildX004AmoBorn	In what month, day and year was the child born?	MONTH OF BIRTH
	ChildX005AdaBorn		DAY OF BIRTH
	ChildX067AYrBorn		YEAR OF BIRTH

Note that we also modified the question texts to make them more neutral.
Switching the routing tab, we adjust the code there to:



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

At this point, in order to restructure the above to follow HRS's setup, we need to introduce a variable that will contain the question of whether the respondent has any children (versus asking how many children the respondent has). For this, let's go to the 'Variables' tab and add a new variable 'A208ANewPerson' as follows:



Observe that we made the variable an array. The reason for this is because we want to set up our routing such that it will enter the loop, ask if the respondent has any children, if yes ask our questions about gender and date of birth, and then ask our new question again. As such, if we did not make 'A208ANewPerson' an array, then once we went into the loop a second time our answer would still be there from the first time (since 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:

```
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    endif
17 enddo
18
```

A few things are different here. Gone is the question asking how many children the respondent has. Instead, we simply start the loop and set a maximum of 50 (i.e. we expect no person to list more than 50 children). Right on entering the loop we instruct NubiS to ask our just added variable (line 2). If the answer is 'Yes', then we ask our questions.

Clearly, the above would accomplish the same thing as our routing before, just in a slightly different manner. We would need to make some refinements of course to make things look more natural, e.g. modifying the question text of 'A208ANewPerson', so that after the first loop we ask 'Do you have another child?' instead of 'Do you have a child?'. Another tweak we would want to make is more structural in nature and has to do with the fact of what should happen if the respondent says 'No' to 'A208ANewPerson'. Currently, NubiS will simply ask the question again until it has completed the loop 50 times. This is obviously not what we want. Instead, if the respondent says 'No', the loop should be ended and NubiS should continue with whatever follows the loop.

Luckily for us NubiS has a routing construct to indicate just that:

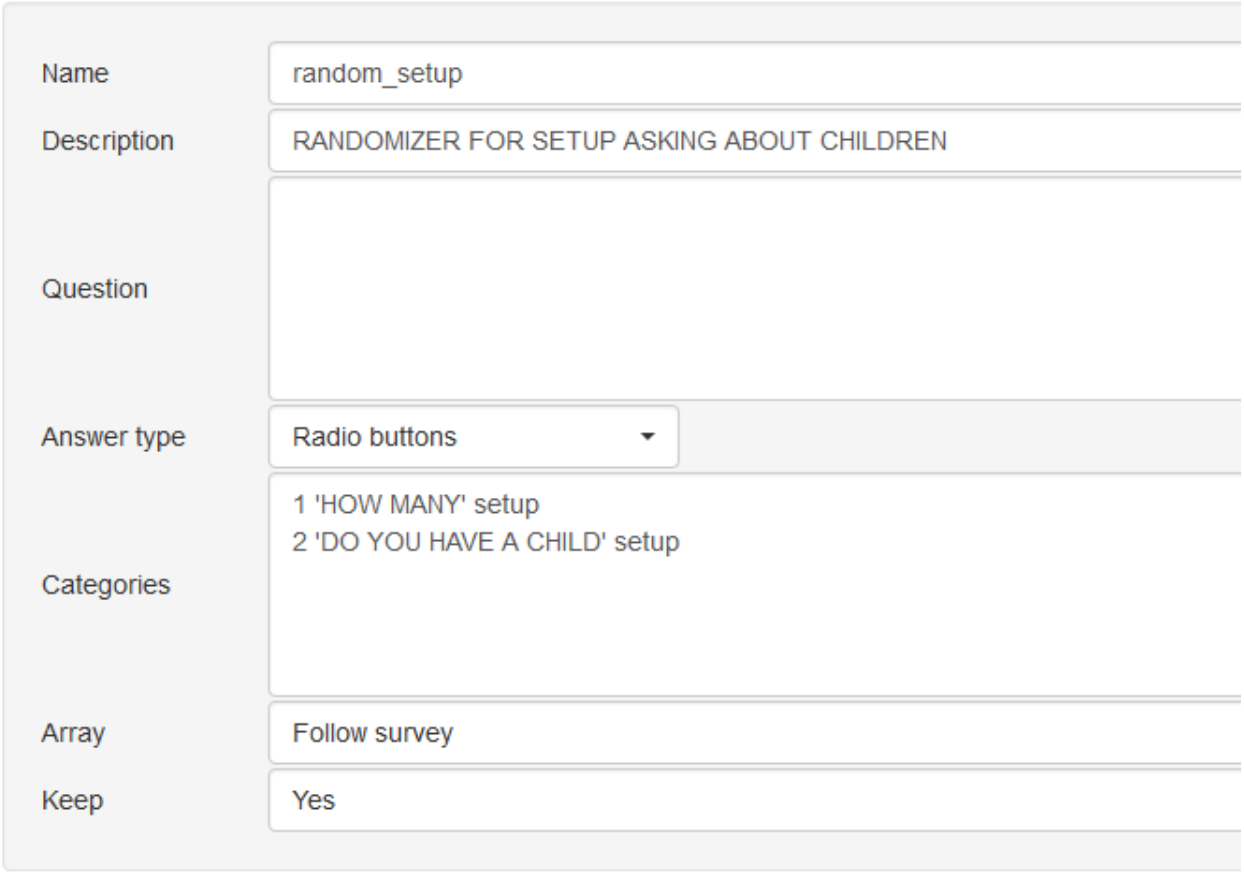
```
1  for cnt := 1 to 50 do
2     A208ANewPerson[cnt]
3     if A208ANewPerson[cnt] = YES then
4        ChildX060ASex[cnt]
5
6        GROUP.TBirthDate
7          ChildX004AmoBorn[cnt]
8          ChildX005AdaBorn[cnt]
9          ChildX067AYrBorn[cnt]
10       ENDGROUP
11
12       X060ASex[cnt] := ChildX060ASex[cnt]
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 we added lines 16-18 comprising an **ELSE** statement with a single action **EXITFOR**. This routing construct instructs NubiS to exit the for loop as soon as it encounters it. In other words, anything following the **EXITFOR** will simply be ignored by NubiS.

12. Randomly asking questions

We have seen a variety of ways in which we can get NubiS to ask questions, perform calculations, set flag variables, and etceteras. Something that we did not touch upon yet is that sometimes we may want to ask questions randomly. For example, let's say we we want to ask a particular question only to a subset of our survey respondents; or we want to use a different question phrasing for different people; or we want to order multiple questions on the same screen in a random order. All these kinds of requests fall under the term 'randomization' in NubiS.

Randomization in NubiS to a large extent consists of the topics we have discussed so far already while adding some bits we will discuss here. To exemplify, let's start with an easy request. Say we want to ask half of the respondents about their children using the 'how many children do you have' set up, and the other half with the other set up (our goal here being to see if we get 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 we need a variable that will be our randomizer:



A randomizer's values are usually (but do not have to be) numeric. In the above we chose though to make our randomizer a radio button answer type rather than just an integer. Why did we do that? The reason has not so much to do with our programming now, but rather with how any collected data gets outputted. By making 'random_setup' a radio button variable we can specify labels for the values our randomizer will take; which in turn can be incorporated into the outputted Stata file making data analysis easier.

The other noteworthy thing is that we set 'Keep' to 'Yes'. We briefly touched on this setting when adding our first question. This setting, if set to 'Yes', 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 let's put our randomizer in 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
```

Here we placed several lines before our for loop. The new line 1 states that if 'random_setup' does not have a value, then assign a value to 'random_setup' that is a number between 1 and 2; 'mt_rand' here is a standard PHP function that we employ to perform the randomization for us. Then, if random_setup = 1, we ask our loop with the 'Do you have a child' set up. Otherwise (not shown here) we use the other setup.

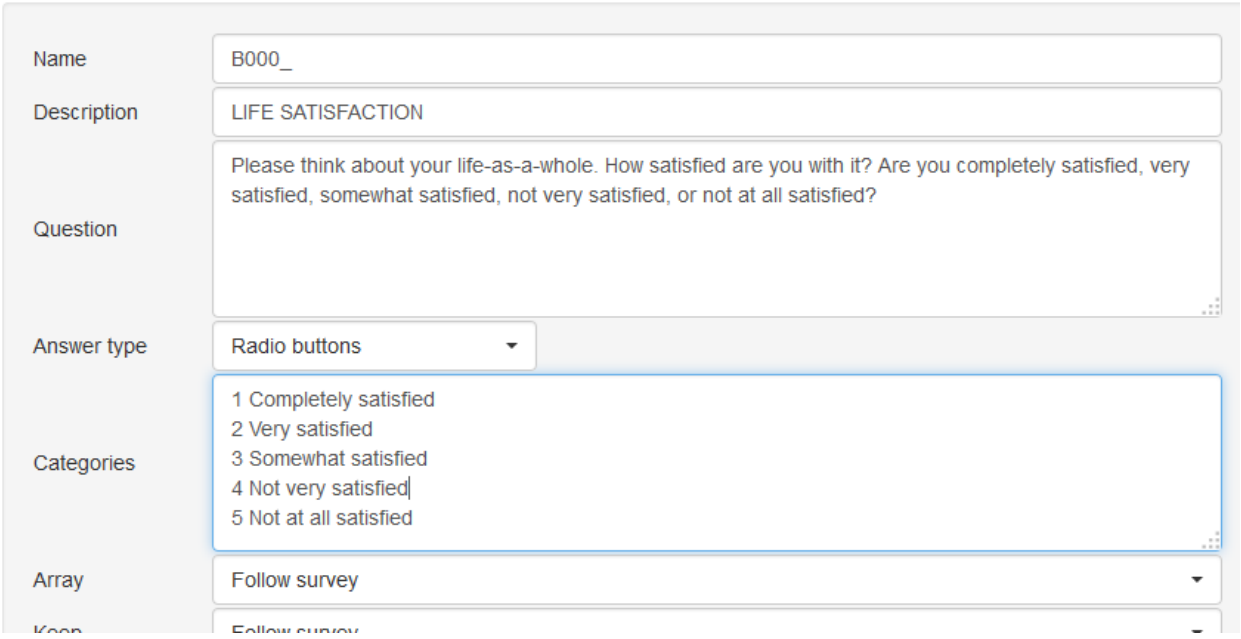
Note that we included asking 'intro' here, a question of 'no input' that we just added so we have a question screen we can go to that is prior to the randomizer being set. This is just to illustrate its value being selected and the role of the 'Keep' setting. Now if we run the survey with the tester, we will get our introduction screen. Then, on clicking 'Next' NubiS will assign a value to 'random_setup' in the background and depending on its value ask either 'A208ANewPerson' or 'numberofchildren'. Suppose that at this point we go back to the first screen by clicking 'Back'. We then click 'Next' again. What would this do to our randomizer?

Well, under normal circumstances NubiS will undo any assignments it made while going from one question screen to another. The reason is that we wish to restore the state of the survey to as it was, that is, we want to pretend assignments never happened and set any variables involved in an assignment back to the value they had prior. This behavior ensures for example

that a for loop counter gets 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. Let's trace the steps. The respondent clicks 'Back' and consequently NubiS undoes assigning a random value to 'random_setup'. Now the respondent clicks 'Next' again. The IF condition on line 2 is met, because NubiS gave 'random_setup' its old value prior to clicking 'Next', which was an empty value. So NubiS then assigns a random value to 'random_setup'. The thing to keep in mind is that this new random value does not have to be the same as the one we had before. As a result, the behavior that follows after might vary.

This is obviously not what we want, so we need something to prevent this. That is where the 'Keep' setting comes in. By setting it to 'Yes' NubiS is made aware of the fact that on going back any assignments made for 'random_setup' should not be undone. In a sense, this has the effect that NubiS treats 'random_setup' as if it were a direct answer given by a respondent. Those answers are always kept by NubiS. Note: if you wish to experiment with this for yourself, just set 'Keep' back to 'Follow survey' or 'No' for 'random_setup', test the survey, and go back and forth a few times. At some point you will notice that NubiS will alternate between following the first or second setup for asking about children.

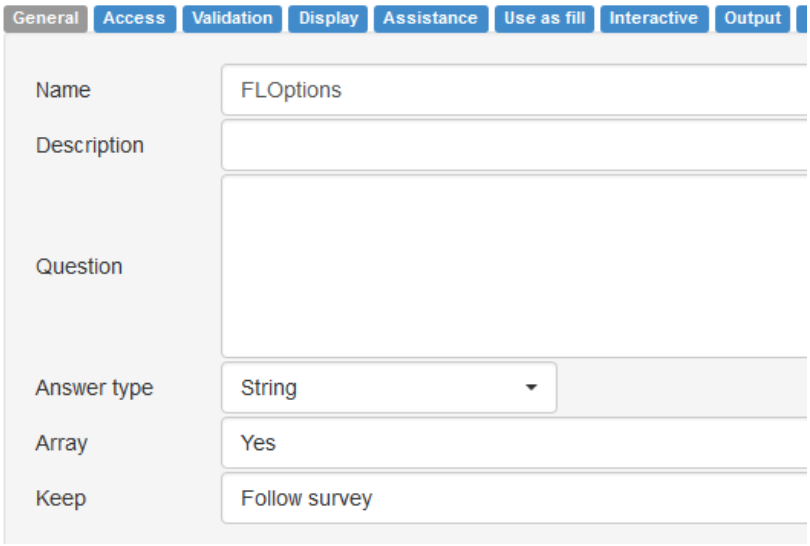
Another way in which a randomizer can be used is to switch between different texts, e.g. question text or answer options. The components used are pretty much the same, the only difference being that the randomizer is now used in the fill code of some fill variable to direct which text gets used when. We'll use the first question of HRS section B to illustrate:



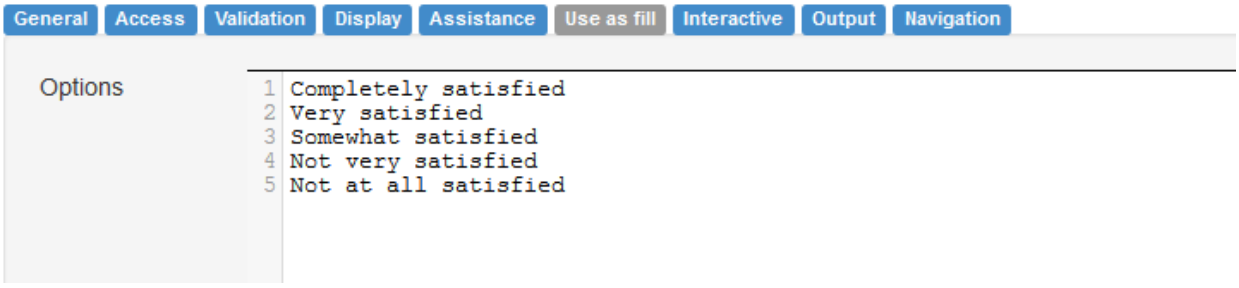
We start by adding the above question (to a new section 'secB'). As you can see it asks about life satisfaction. We then add it to the routing of 'secB' and then put 'secB' on the routing of the 'Base' section instead of 'secA2':

```
1 //secA
2 //secA2
3 secB
4
```

Our answer options right now range from ‘completely satisfied’ to ‘not at all satisfied’. Let’s assume we want to flip the order for half of the respondents. Not surprisingly, we need a randomizer (let’s call it ‘random_order’). We also need a fill variable for the answer options (since they are going to be dynamic now):



We make it an array since it will need to contain all the different answer options. Next, on the ‘Use as fill’ tab we enter:

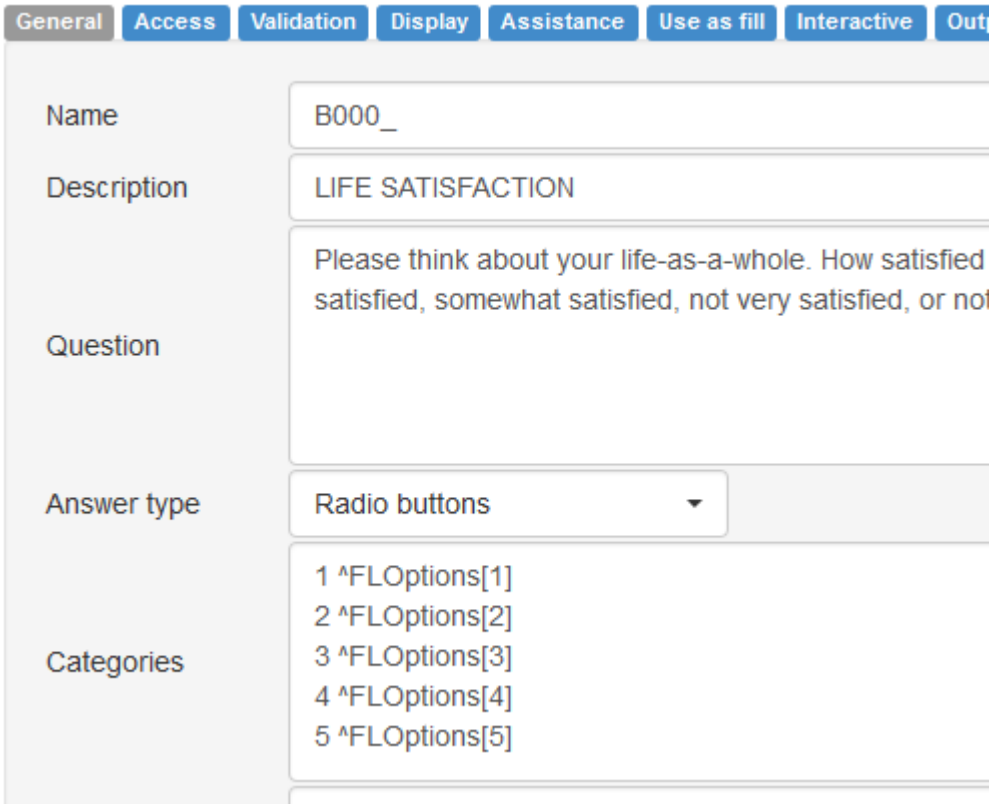


And:

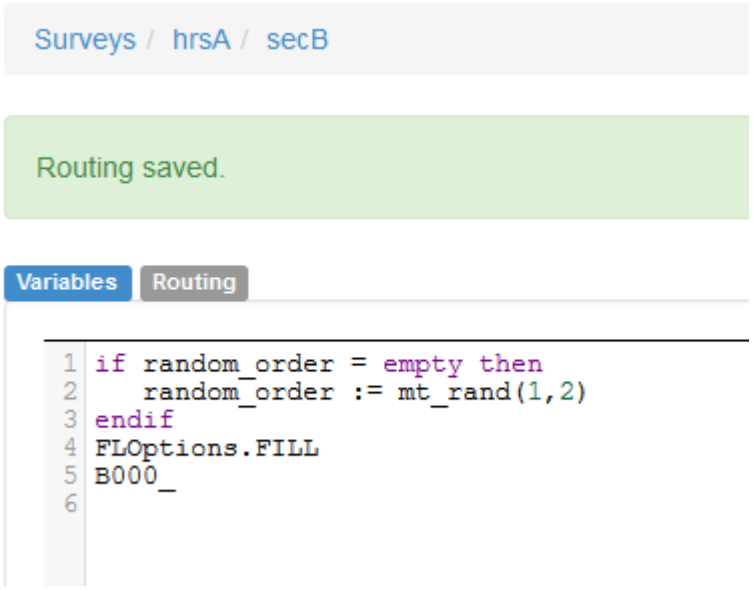
Code

```
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, we update ‘B000_’ to refer to ‘FLOptions’:



Finally, we wrap things up by modifying the routing of ‘secB’:



In a nutshell we have now accomplished that 'random_order' gets a value if not set before, based on that 'FLOptions[1] to [5] get assigned some text', and those are then used in 'B000_' for the answer options; 'completely satisfied' to 'not at all satisfied' or 'not at all satisfied' to 'completely satisfied' depending on the value assigned to 'random_order'.

The types of randomization discussed so far in NubiS are pretty common. There is one other form of randomization that may occur. That form has to do with when a series of questions should be asked in random order. This is more rare, and in fact HRS does not employ it, so we will use an artificial example to illustrate. We will use 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 we would just ask these questions in order:

*B105_*
*B106_*
*B107_*

Suppose that we are worried though that the order in which we 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 recalls his/her childhood better. Whatever be the reason, how would we ask these three questions in random order? One way is to use what we have used so far. We introduce a randomizer and let its value vary between 1 to 6 (to accommodate all six sequences). Then we use that randomizer in the routing to ask the questions in a certain order. In other words, something like this:

```
   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_
```

The above doesn't show all sequences, mainly because it becomes pretty lengthy very quick. One can imagine that if we would add a fourth variable, this method becomes more burdensome still since we would have more combinations to consider. A more efficient way 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))
   language
endif

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

Understandably this looks somewhat daunting at first. Let's break it down starting with the simple things being 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' (it is a randomizer after all), 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, let's turn to the routing:

```
FLQuestions := array(1 => "B105_", 2 => "B106_", 3 => "B107_")
if sizeof(random_sequence) = 0 then
  random_sequence := shuffleArray(array(1 => 1, 2 => 2, 3 => 3))
endif
```

We start by assigning a string array to 'FLQuestions'. The first item with key '1' is 'B105_', the second one is 'B106_' tied to key '2' and the third is 'B107' linked to key '3'. Then, we come up with a random order. This order is contained in 'random_sequence'. Notice how our condition here for checking if the sequence was set before is the same as for non-array questions:

```
if random_sequence = empty then
```

In this regard 'random_sequence' is considered to be empty by NubiS 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 when NubiS encounters this line the condition is met, and 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 it. So for example we might get back the order '3,1,2' in position 1, 2 and 3 of the array respectively. Then follows a for loop responsible for actually asking our questions:

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

The line within the for loop is the one that provides the instruction to NubiS 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]' (let's assume this is 3). Then, it uses the result, 3, to look up the instance of 'FLQuestions[3]'. This 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_'.

Though this method is more convoluted on the face of it, it is also more powerful. Adding a fourth question is as easy as extending the arrays assigned to 'FLQuestions' and 'random_sequence' and increasing the loop maximum to 4. A last note concerns why we used two arrays. The sharp observer will have noticed that we easily could've used one array 'FLQuestions' and randomize its order with 'shuffleArray'. The reason we didn't has to do once more with data output. if we had just used 'FLQuestions', then in the outputted data there would have been three variables 'FLQuestions_1_', 'FLQuestions_2', and 'FLQuestions_3_' with possible values 'B105_' to 'B107_'. By using 'random_sequence' and defining it as a radio button answer type, we could specify labels that subsequently get included in the data set. Looking at those variables would then allow for example to do 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 we defined arrays in the routing we specified them for example as:

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

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

```
for cnt := 0 to 2 do
  FLQuestions[random_sequence[cnt]].INSPECT
enddo
```

This would work here, but the reason we specified it the way we did (to start at '1') is because most often loops in NubiS 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 we are asking for example about the first person, the second person and so on; rather than the zeroeth person, the first person, and so on.
Or we could have a situation in which we are using an array with fill text:

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

Then, if we are using this in a loop question, ideally we would want to do:

for cnt := 1 to 2 do

Q1[cnt]
enddo

And specify Q1's question text for example as "^fills[cnt]". But doing so would then lead to issues since our fills are stored at '0' and '1' and not at '1' and '2'. So we would have to adjust the fill reference to "^fills[cnt-1]". Since this is an extra step to remember it is often 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 our array examples we have used numeric indices for anything we stored in the array, for example in:

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:

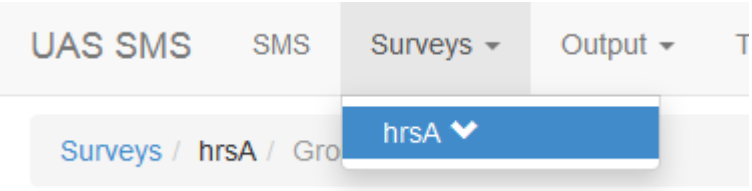
random_sequence := shuffleArray(array('a' => 1, 'b' => 2, 'c' => 3))

Of course this would make it hard to use 'random_sequence' in a loop, but it would not prohibit us from using it for example in a fill reference like '^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"]'. Care should therefore be taken to not use them interchangeably.

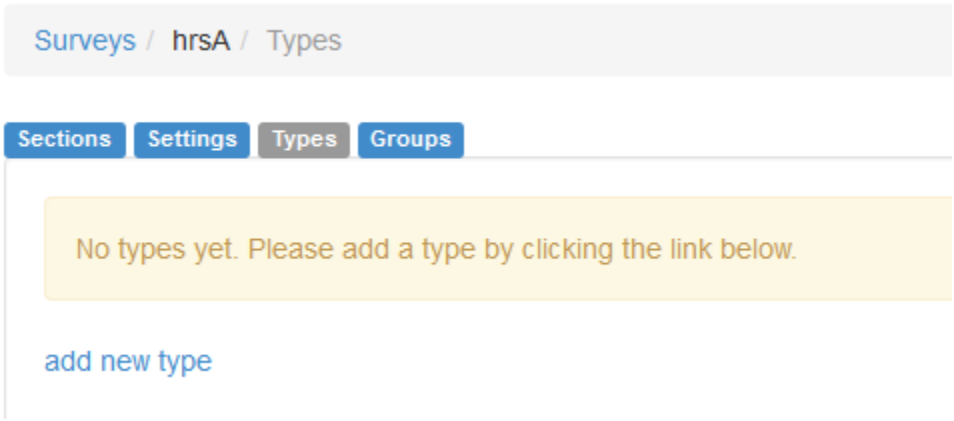
13. Defining and managing similar questions

Over the course of the past few sections we have on multiple occasions added variables whose answer categories were Yes and No. Each time we added those categories, we sometimes alleviated this by having made a copy of a similar variable. While doing that you may have wondered if there is a better way. For example, suppose someone now comes to you and says “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. This can be addressed to some extent by utilizing the batch editor, but it nonetheless requires some effort.

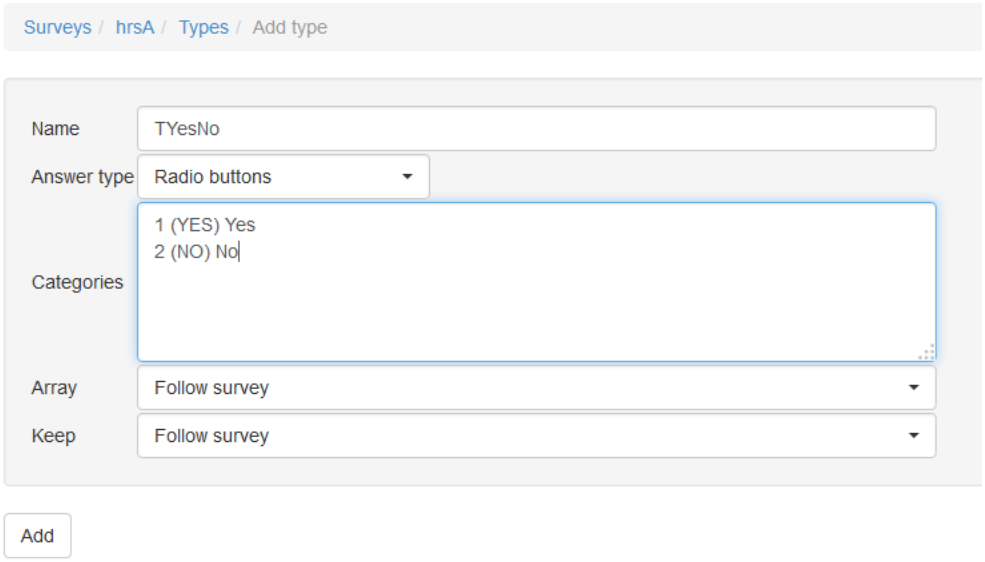
A better way is to use NubiS’ type system. A type can be thought of as the blueprint for a set of variables in which a set of shared characteristics are specified. Those characteristics can then be augmented with variable specific settings such as a name or question text. Furthermore, characteristics can be overridden to customize certain features. To illustrate, let’s add (somewhat belated) a type to define ‘Yes/no’ questions. First we go back to the top level of our survey:



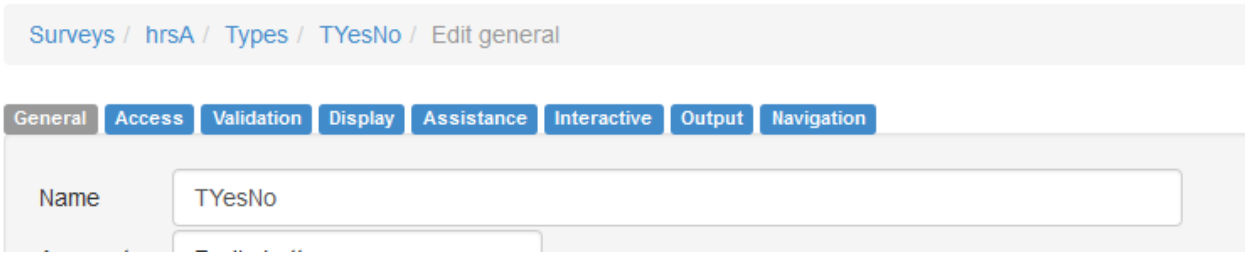
Next we locate and open the ‘Types’ tab:



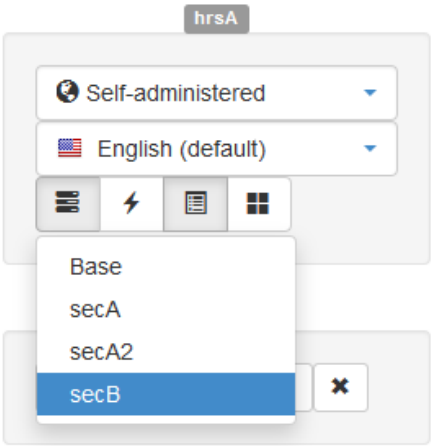
And click ‘Add new type’:



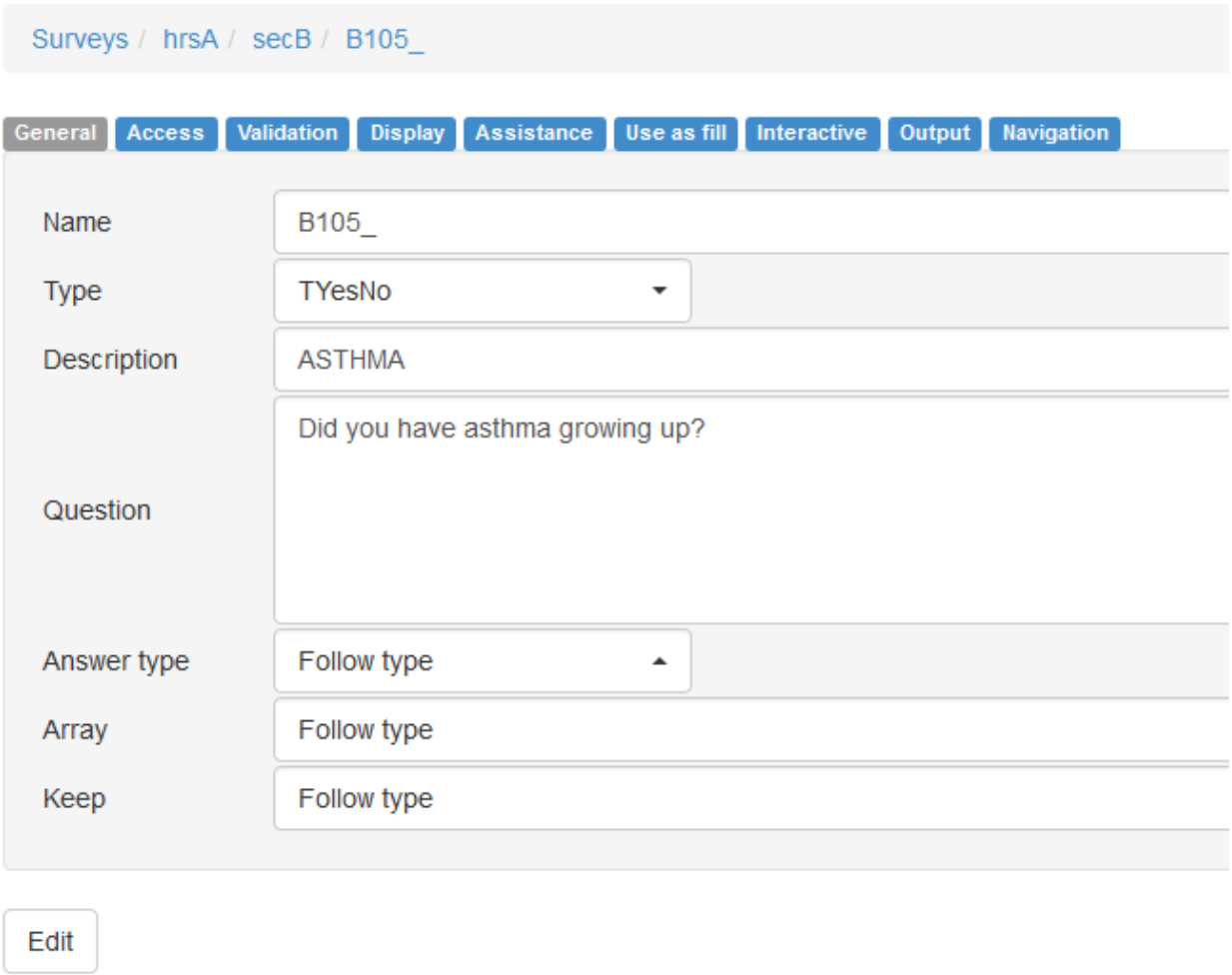
As we can see the basic definition of a type is very similar to that of a variable except it lacks variable-specific items like question text. If we go and edit our new type after adding it, we see that a type can also the other types of setting we discussed for variables:



We are now ready to start using our type. Let's enter 'secB':



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



We see that now we have a 'type' setting that wasn't there before. In it we choose 'TYesNo'. We then set answer type, array and keep to follow whatever the setting of the type is. Note that if we wish to deviate from the type for any of those properties we 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 the programmer who can now update a set of variables of a particular type in one go by modifying the type (rather than each individual variable). As we will see later on, this also is extremely useful when translating a survey into different interview modes and/or languages.

14. Configuring the display of questions

In the previous section we discussed how a set of questions can be grouped on the same screen using **GROUP** statements. We did this for example when asking for the date of birth. We focused there on the mechanics of getting NubiS to ask the questions, but we did not pay too much attention 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; and if needed custom templates can be defined. To illustrate some of the pre-defined templates, let's update our routing for 'secB' slightly:

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

Save

In other words, we want to group our randomly ordered questions on the same screen. Right now if we look at this in the survey we get:

Did you have asthma growing up?

- Yes
- No

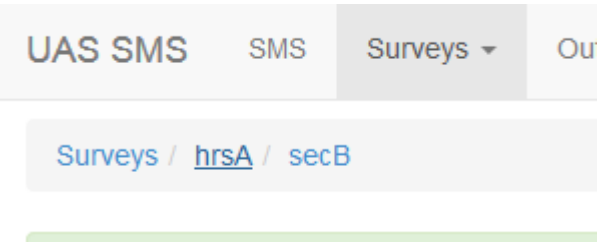
Did you have diabetes growing up?

- Yes
- No

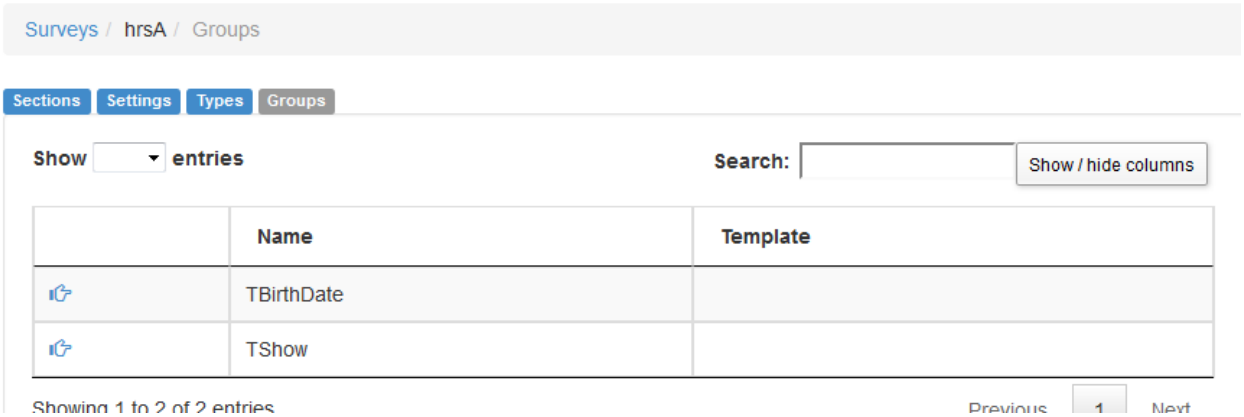
Did you have a respiratory illness such as bronchitis growing up?

- Yes
- No

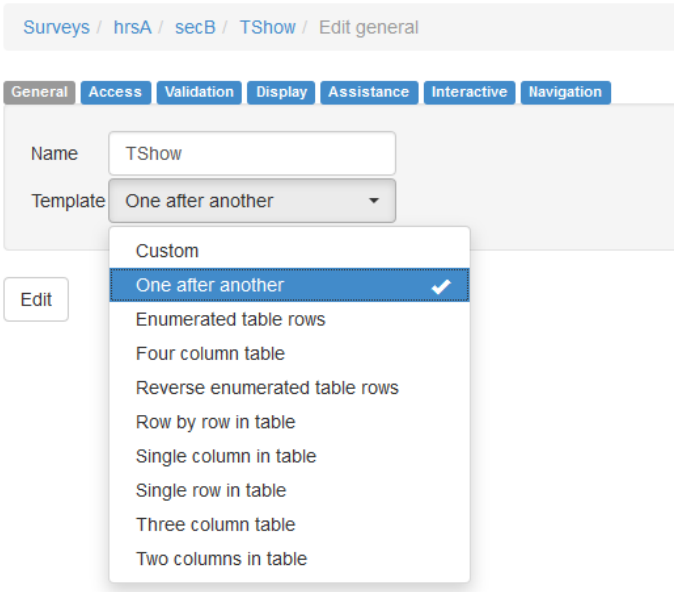
In principle there is nothing wrong with this, but we might want to try another display. Using the 'hrsA' link towards the top:



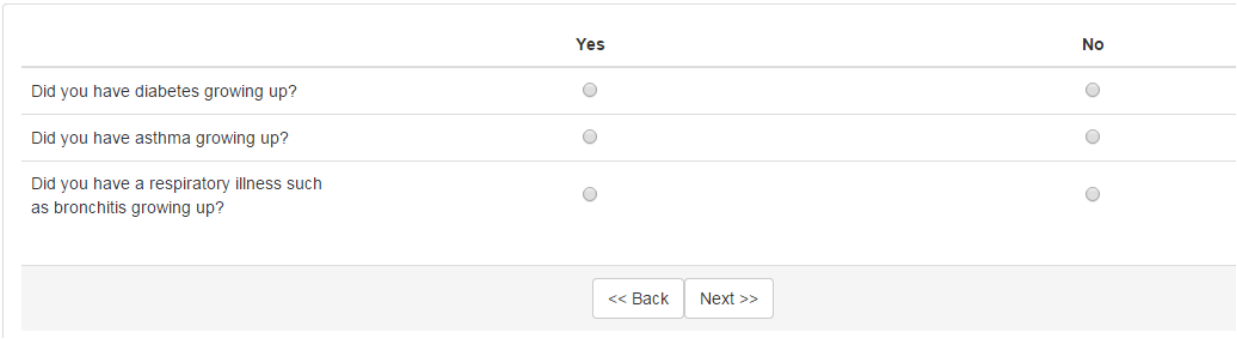
We navigate to the top of the survey and select the ‘Groups’ tab:



The list shows the two groups we have used in the routing so far (NubiS created ‘TShow’ when we saved the routing for ‘secB’). Let’s edit ‘TShow’ and choose a different template:



Let’s choose ‘Enumerated table rows’. If we do that, save, and return to our survey we get the following display instead:



If we would want to we could refine this more by for example adding borders or row highlighting. Or we could choose the ‘Reverse enumerated table rows’ template:

In which the questions now appear as column headers instead of the answer options. Another thing we may want to do is add some introduction text above the table, e.g. saying something like ‘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. We add a ‘no input’ variable to that extent and add it to the routing:

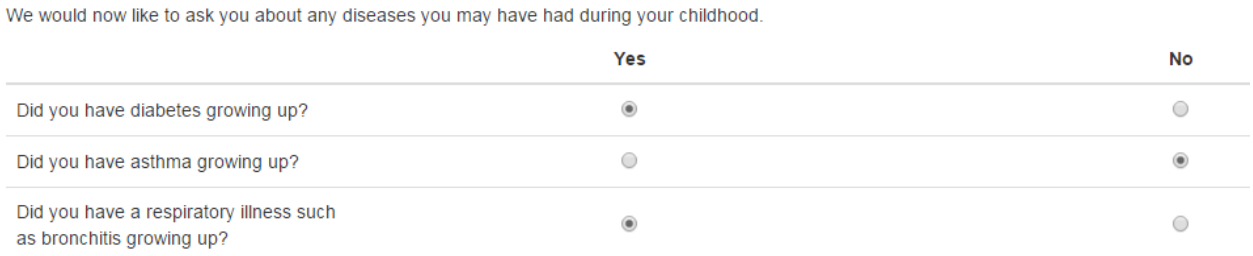
```
group.TShow
b_intro
for cnt := 1 to 3 do
   FLQuestions[random_sequence[cnt]].INSPECT
enddo
endgroup
```

However, the result of this is:

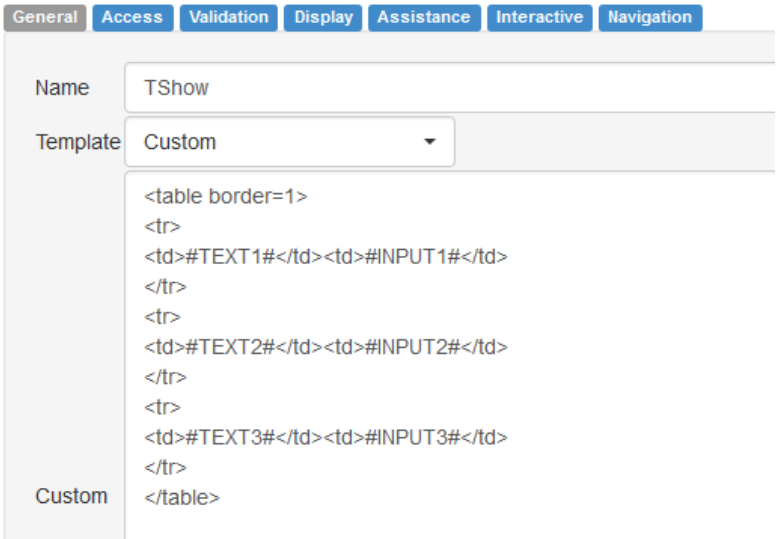
What happened? Well, We told NubiS to use the ‘Enumerated table rows’ template for ‘TShow’. So NubiS also took our new variable ‘B_intro’ and tried to put it into that template. To work around this we need to slightly complicate our GROUP statement to:

```
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, the only difference being that they can only occur within a GROUP and it is not possible to have a subgroup within a subgroup. The main benefit of subgroups is that it allows us to instruct NubiS to apply different templates to different parts (and enforce different validation criteria as we will see shortly). In this case 'Toverall' is a template NubiS made for us on saving the routing, which follows the default of placing one question after another. Then, 'TShow' for our new subgroup will apply the 'Enumerated table rows' template to our three questions. The result is:



Another thing to note is that the pre-defined templates that come with NubiS are intended to cover some frequently used types of display, but are by no means intended to be exhaustive. For this reason it is also possible to define custom templates. If we select 'Custom' as the template for 'TShow' we get an input box in which we can define the template:

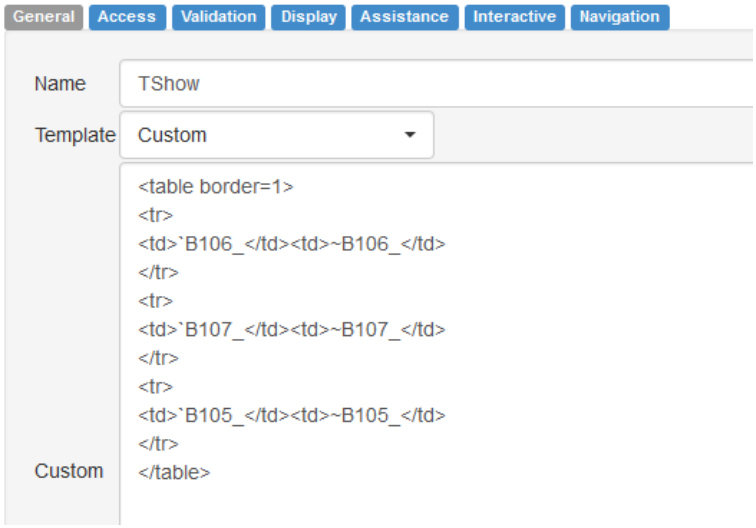


Here we created a simple template using NubiS' template placeholders. '#TEXT#' placeholders represent question text, whereas '#INPUT#' placeholders represent input boxes. In the above we tell NubiS to take our three questions and put them in an HTML table with a border:

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

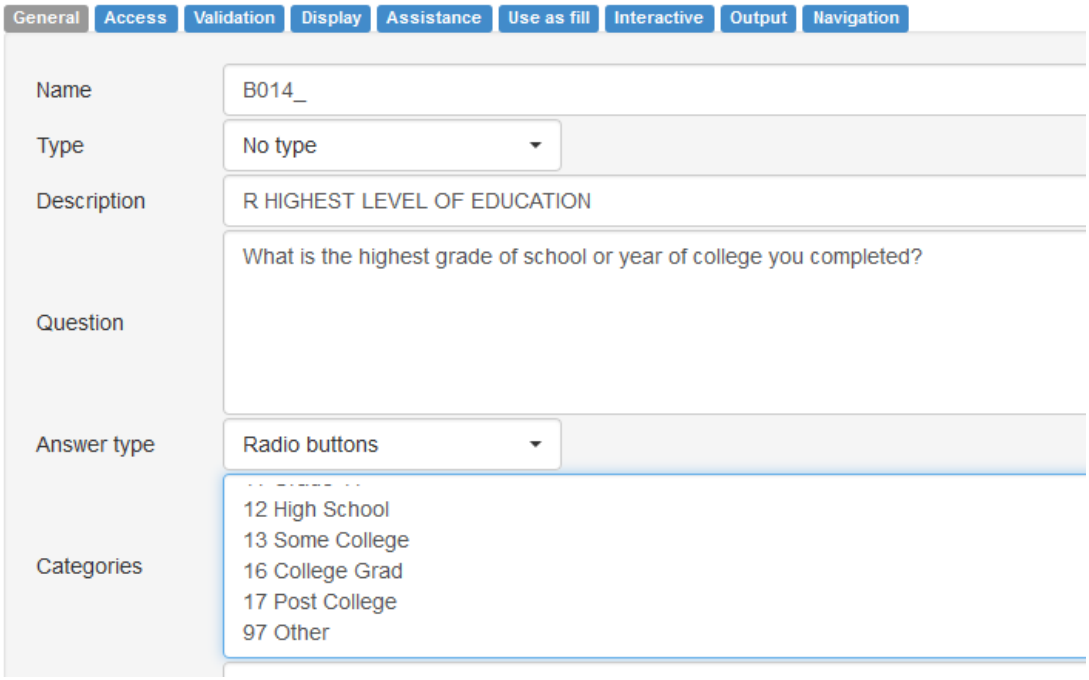
Note that the order in which NubiS inserts our questions is directly related to the order in which they are encountered while processing the routing instructions. That is, supposing the random order of our questions was 'B106_', 'B107_', 'B105_', then 'B106_' would correspond to '#TEXT1' and '#INPUT1#'.

This linkage between question order and display may not always be desirable, e.g. if the display of a certain question should be fixed on screen. For this reason NubiS also supports an alternative manner in which to create 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 'B106_', 'B107_', 'B105_' order:

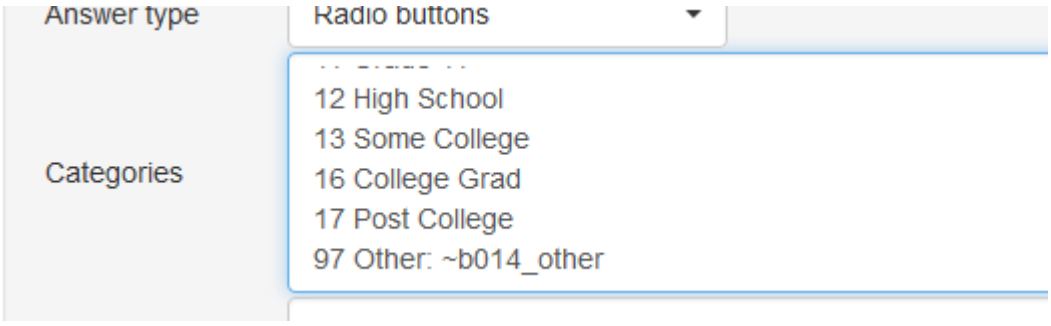


Here "`B106_" instructs NubiS to insert the question text of 'B106_', whereas "~B106_" tells it to insert the input box (here the radio buttons).

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



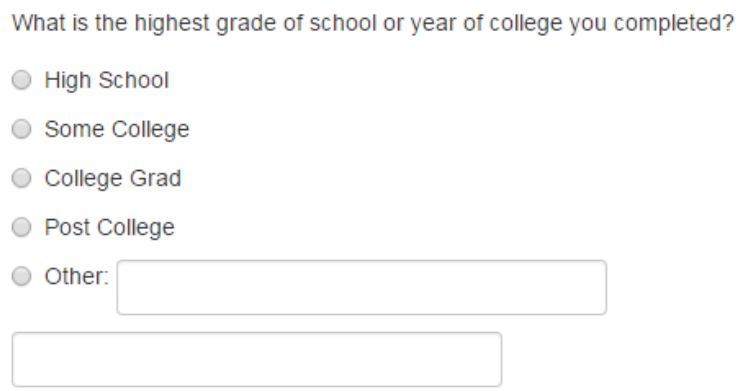
Note how we have an 'Other' category as the last option. Suppose (deviating from HRS for a moment) that we want to give the respondent the opportunity to specify something for 'Other'. To do that we need to do three things. The first is to add a string question 'B014_other'. The second is to revise our answer categories for 'B014_' to include a variable reference:



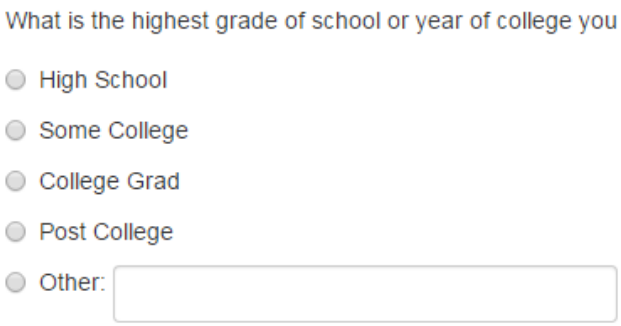
And lastly the third is to write the routing that will instruct NubiS to put 'B014_' and 'B014_other' on the same screen:

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

By and large we use a normal GROUP statement in which ‘TOverall’ is a group that NubiS will create (which will follow the default template showing each question below one another). We need to add one special instruction though and that is ‘.INLINE’ behind ‘B014_other’. The reason for this is that by default NubiS assumes that one variable listed in a GROUP statement must be displayed on the screen independent of whether it was referenced anywhere through a variable reference. As such, leaving out the .’INLINE’ would result in:



Here we see the text box behind the ‘Other’ option, but at the same time we see another text box. This would obviously be somewhat confusing to respondents, so we want to not show this second text box. This is what ‘.INLINE’ accomplishes by telling NubiS that it should not include this variable in the display:

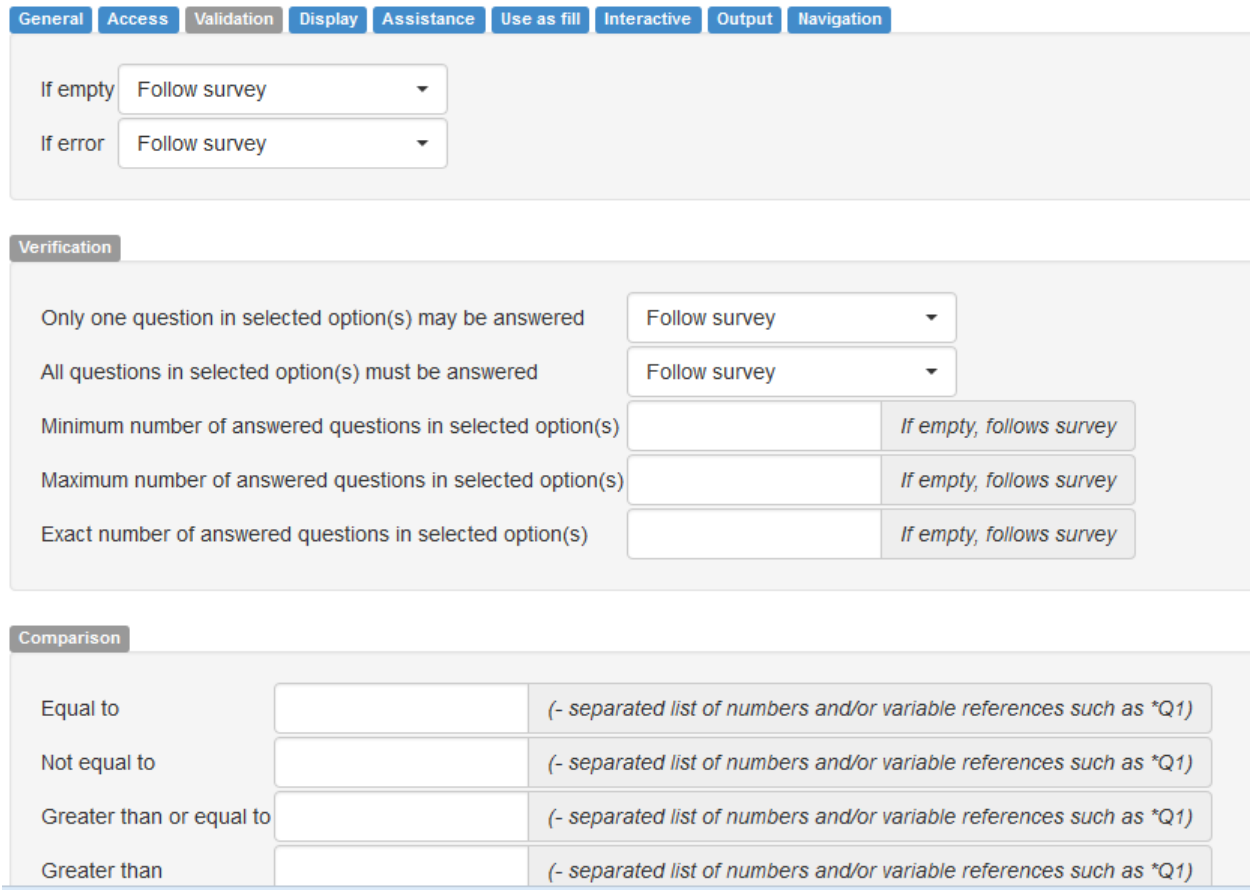


15. Validating respondent answers

In the previous sections we have been talking about how we can add variables and get NubiS to use those variables. The whole idea behind this of course is that in the end we have respondents who provide answers to those questions. We have said little however about how we can ensure that the answers we collect are actually useful. Of course, to a large extent this is dependent on the respondents, and as such beyond our direct control (besides non-technical solutions like incentives). However, there are several mechanisms that can be employed within Nubis to help with this matter.

These mechanisms fall into two main categories: validation mechanisms and input prevention mechanisms. Validation mechanisms within NubiS provide the ability to define if and how to validate the answers given by a respondent. They are accompanied by the capability to specify which assistive error messages should be displayed. Access to these are provided via the 'Validation' and 'Assistance' tabs of a variable, type or group. Variables and types share the same validation and assistance settings, wheres groups complement those with their own.

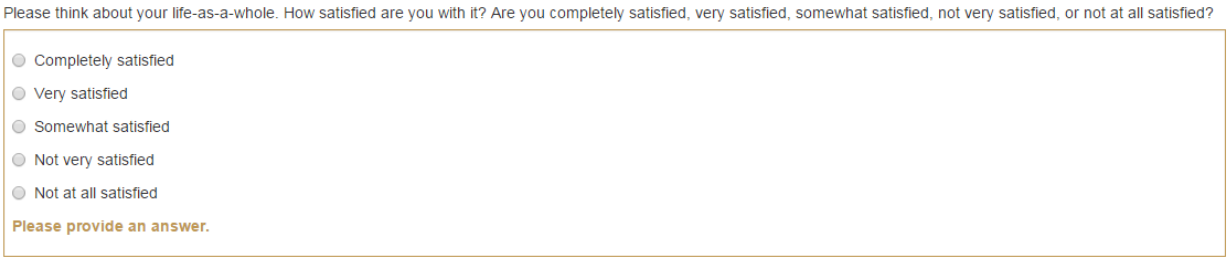
Let's look at the first by opening up 'B000_' and going to the 'Validation' tab:



On the screen we find a variety of validation related settings. The presence of some depends on the answer type (for example we don't see a minimum and maximum input box here, since 'B000_' is not a range variable). Two settings that will always be present though are 'If empty' and 'If error'. These two main settings can be used to define what NubiS should do if a respondent does not provide an answer or provide an erroneous answer. That is, NubiS views not giving an answer as something separate 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 was 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, s/he is 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'. This results in the following for example for 'B000_' if we don't provide an answer:



If we click 'Next' again though, NubiS will continue. If we change 'If empty' to 'Don't allow to continue', NubiS will just keep on showing the same message.

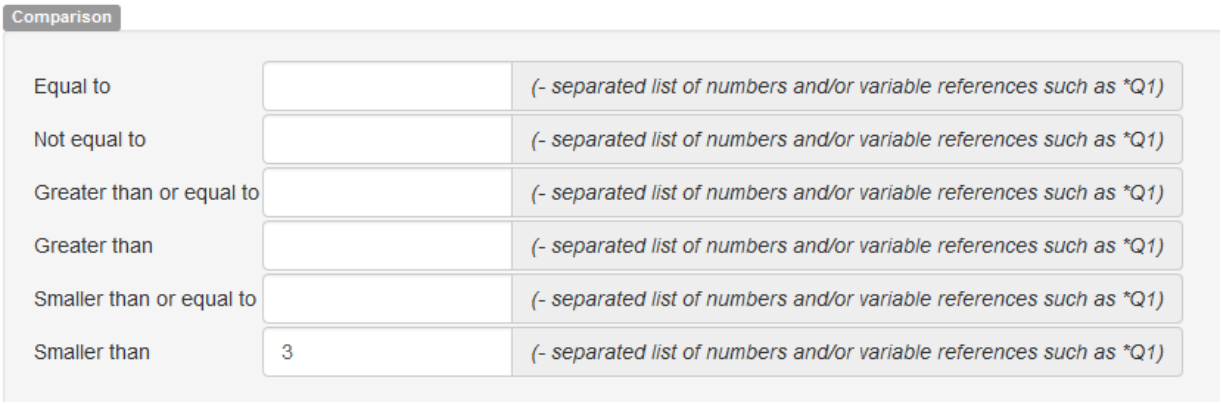
The difference between the two is also sometimes referred to as a soft versus hard check. Soft checks are usually employed more, since not allowing the respondent to continue (via a hard check) can lead to frustration and subsequently survey break-offs (where the respondent just quits the survey altogether). On some occasions though hard checks can be appropriate, for example when asking for the respondent's agreement with the conditions applicable to participating in the survey (consent). Another example is if it involves a question whose answer is crucial for the skip patterns defined in the routing after it.

In contrast, erroneous answers are typically addressed with hard checks. That is, the respondent can't continue until s/he has given a correct answer. Also here of course a balance must be kept between obtaining qualitative good data on the one hand and avoiding excessive respondent frustration on the other.

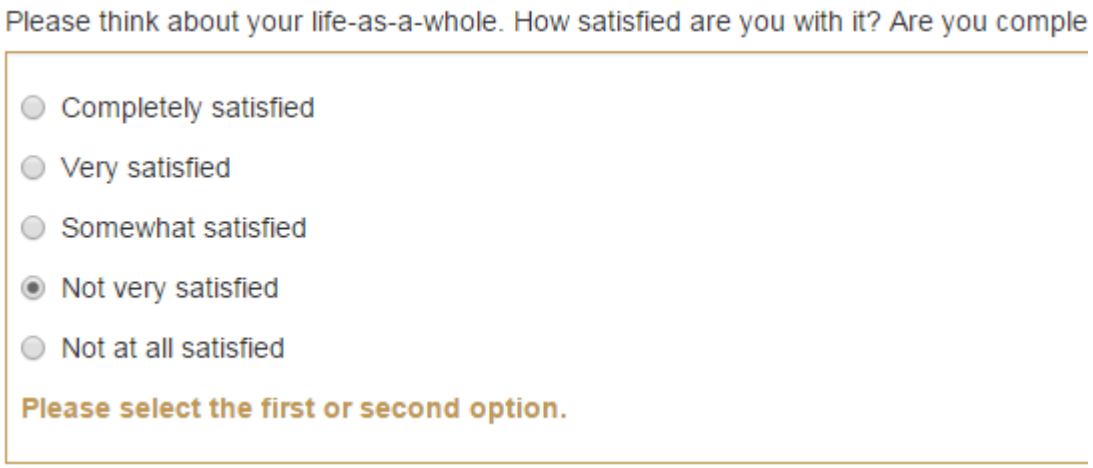
The definition of what constitutes an erroneous answer is to some extent built into NubiS, but to a large extent it must be specified. Automatic validation behavior occurs for the following answer types:

- Integer: NubiS checks automatically if the provided answer is an integer
- Real: NubiS checks automatically if the provided answer is a real number
- Range: NubiS checks automatically 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, we can define a validation criteria for ‘B000_’:

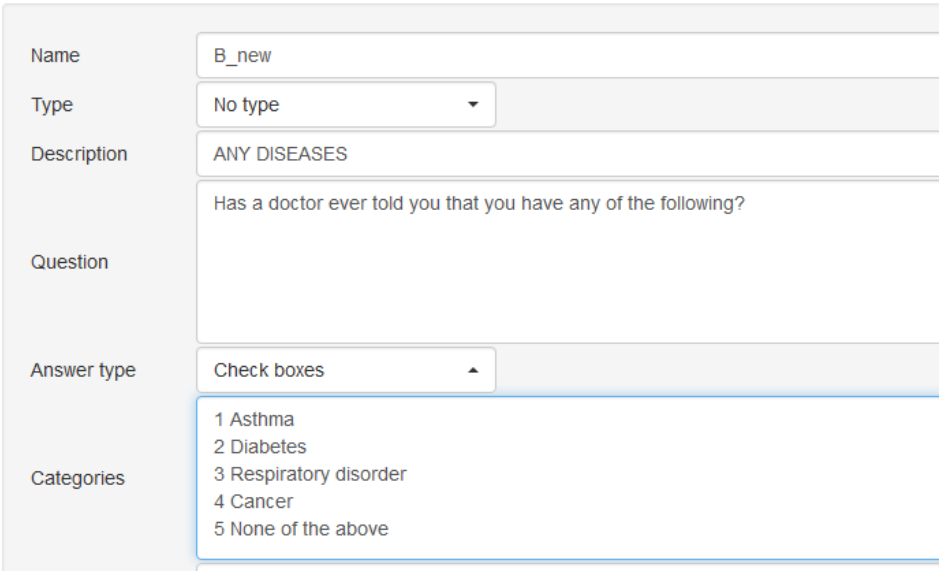


This conveys that the answer to ‘B000_’ must be smaller than 3 (note that we could have also specified multiple values OR referenced the value of other variable(s)). If we look at this question in the survey, we can still select all options but if we select the fourth or fifth option we get an error:

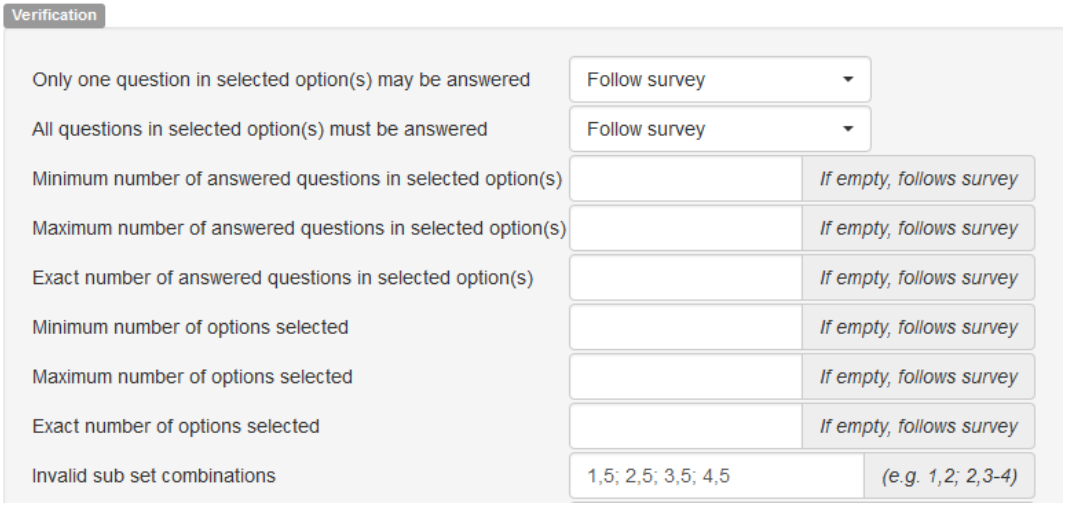


In other words, we can only select ‘Completely or very satisfied’ if we wish to continue to the next screen. Numerical comparisons like these are available for integer, real, range, radio button and drop down variables. String and open variables as well as date/time variables have their own comparative variants.

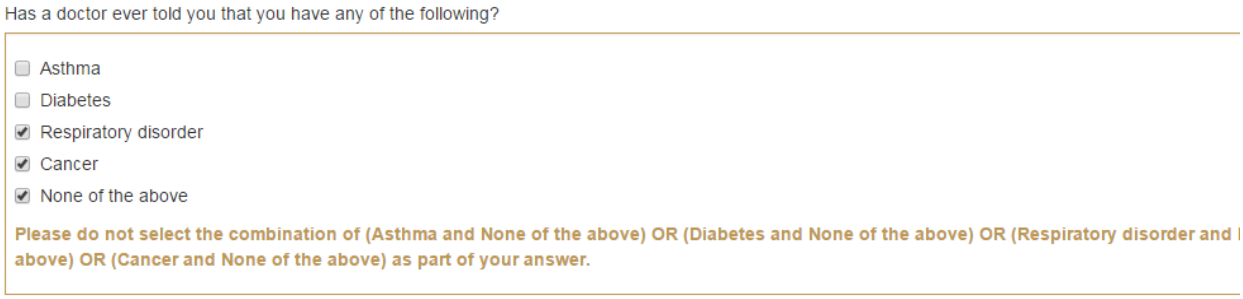
Another form of checks pertains to set of enumerated and multi-dropdown variables. An often occurring pattern with such questions is that they ask the respondent to select one or more applicable options OR select ‘None of the above’. These two are obviously mutually exclusive, and so we would want to enforce this in our surveys to prevent ambiguous answers in which a respondent selected some options but also said none of the options are applicable. To show how we can accomplish that in NubiS let us first add a set of enumerated variable:



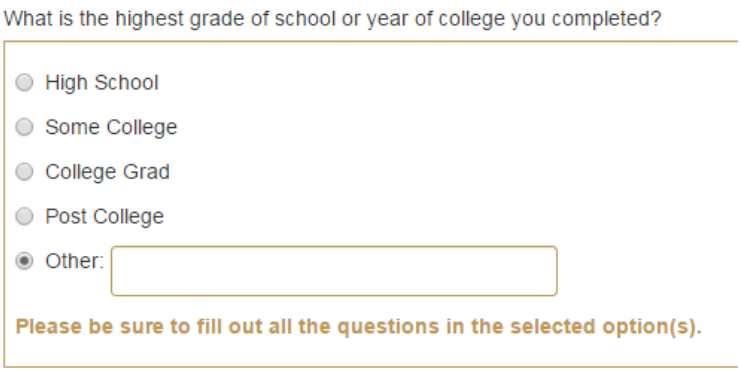
Then we go to the ‘Validation’ tab and enter:



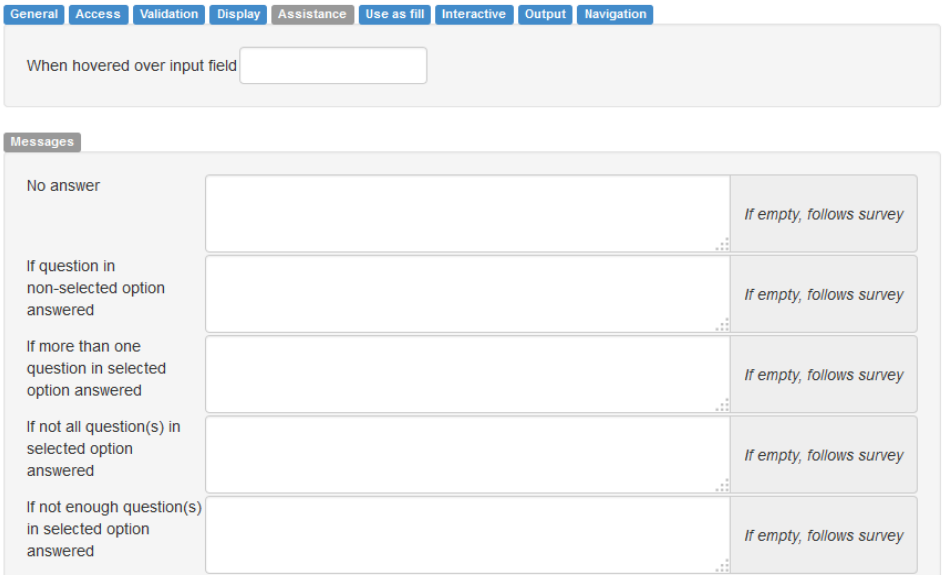
In the survey NubiS then uses this to detect invalid combinations (assuming for a moment of course that we placed '_B_new' on the routing somewhere, e.g. 'secB'):



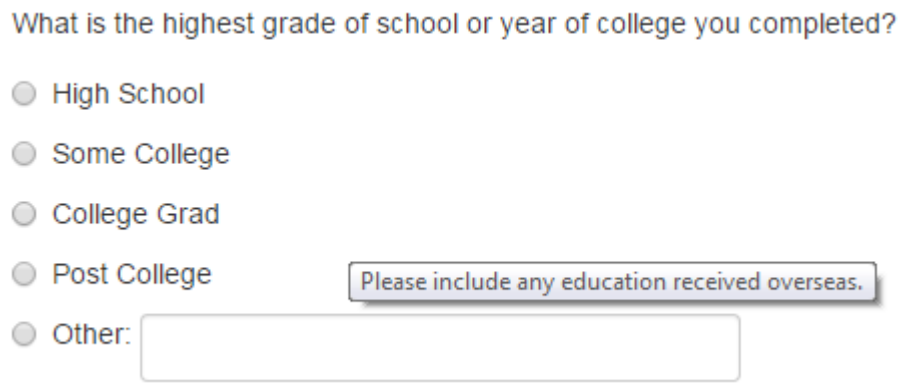
There are also some error checks that are by default automatically enabled. For example, if we look at 'B014_' in the survey we will notice that if we select the 'Other' option but fail to specify anything, this results in an error:



Detecting an error is of course is only part of what is needed. A second part is to help the respondent resolve the error. For this, useful error messages are indispensable. Such messages can be set on the 'Assistance' tab:



One way to provide information is to specify some text shown if the respondent hovers over the input box. For example, as extra information we could state “Please include any education received overseas.”:

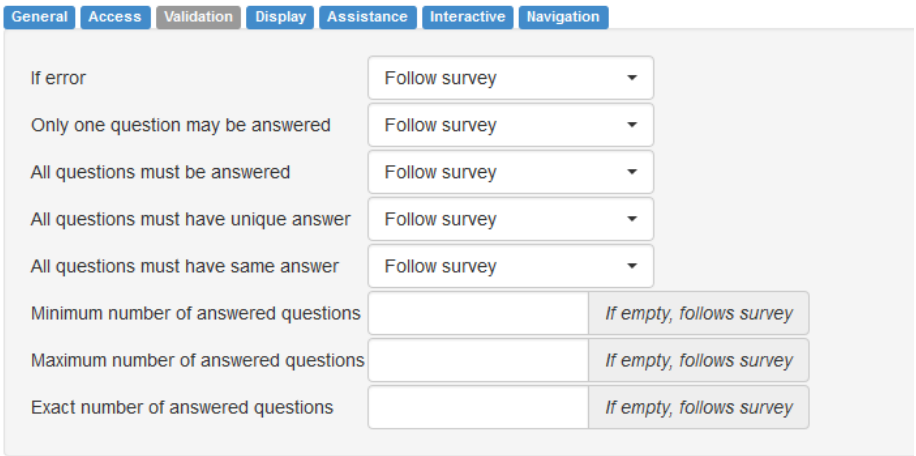


Another is to specify text to appear before or after the input box (applicable for certain answer types only).

In addition to providing information up front, we can also define messages to be shown when an error is detected. Note that the above screenshot only shows 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.

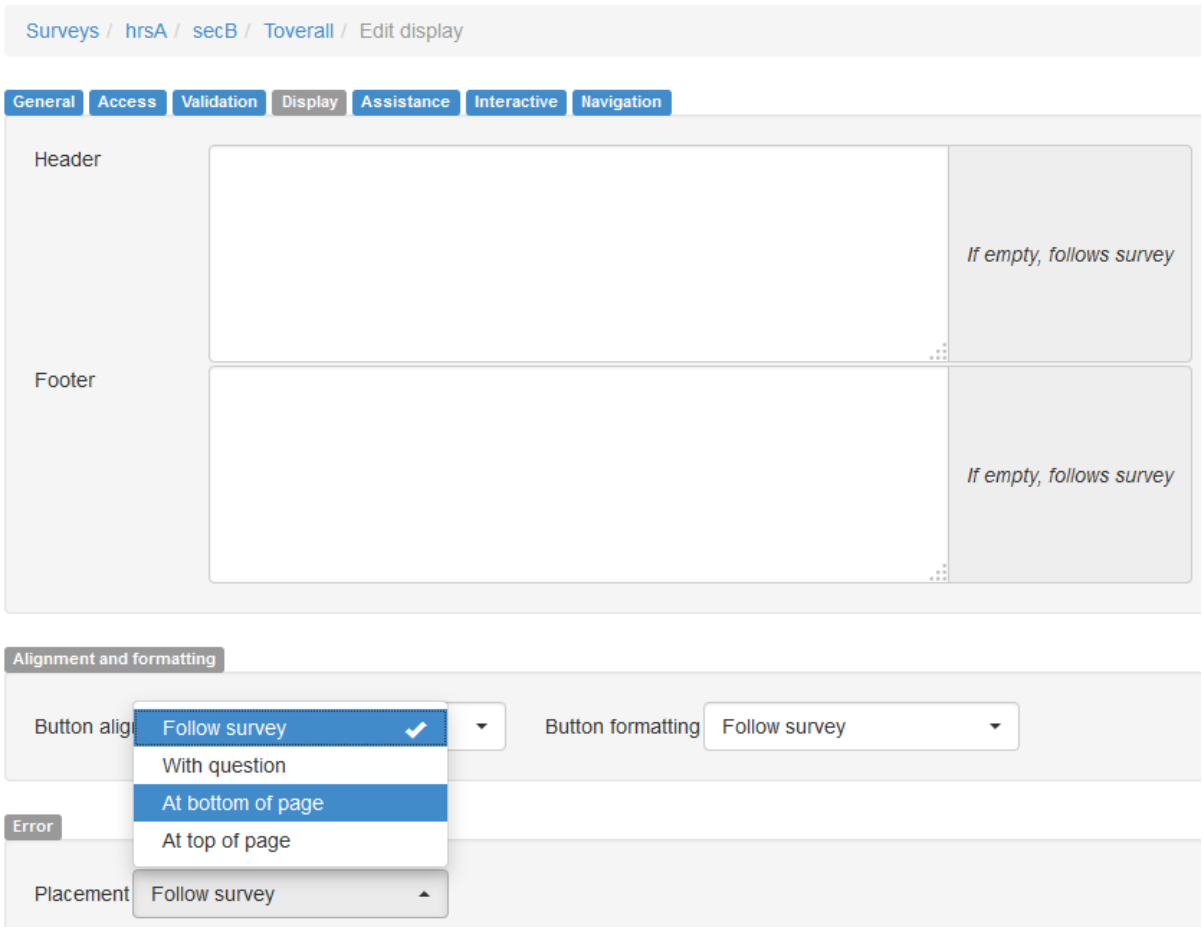
It is also possible to refer to variables in most validation and assistance settings. For example, we could indicate that the value for ‘B000_’ must be lower than that of of another variable.

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 2 questions in the group may be answered. Settings such as these can be found on the ‘Validation’ tab of a group, e.g. for ‘TShow’:



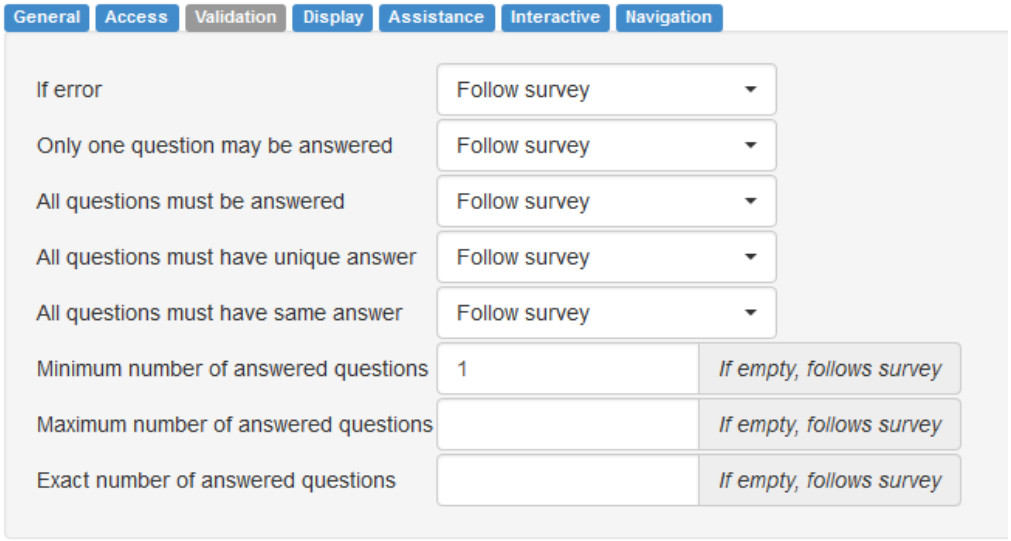
Suppose for example that we state that all questions must have the same answer. Then, if we go to this screen in the survey and try to enter different answers we get:

Note that right now the error message appears as if linked only to the first question, whereas in fact it refers to all questions. To alter this we can change the placement of the error messages in the 'Display' tab of the main group used ('TOverall'):

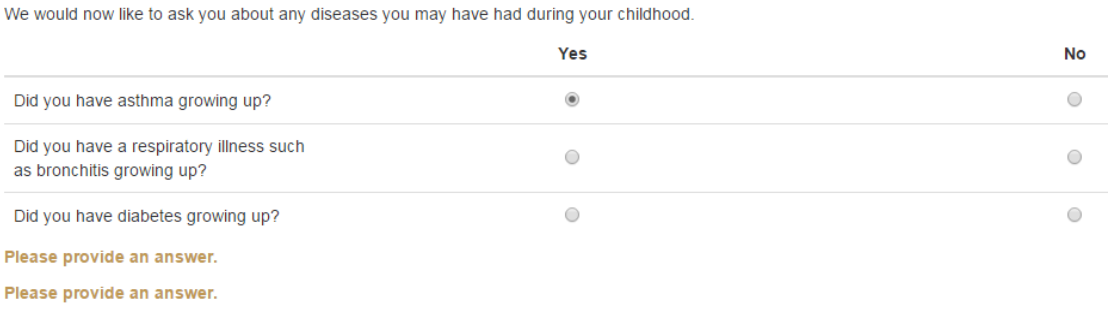


If we set error placement to 'At bottom of page' we get this instead:

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 contradict one another. Suppose for example that we change the validation settings for 'TShow' to:



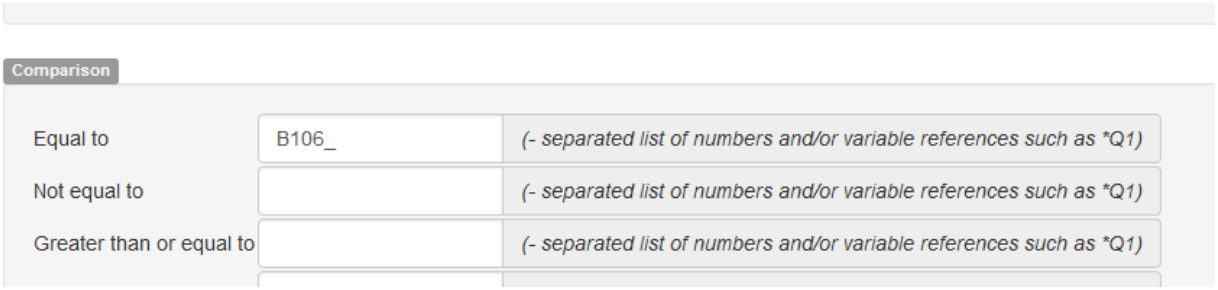
This instructs NubiS to ensure at minimum one question is answered. Now, if we are on the screen displaying our three questions, we might get the following:



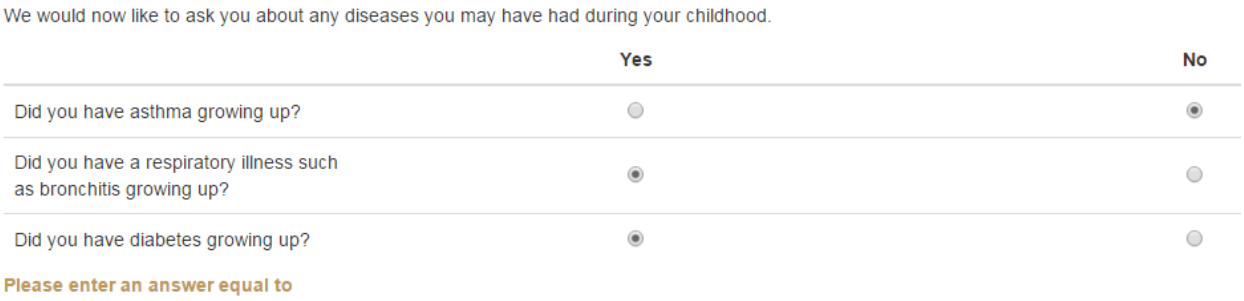
Why is NubiS showing error messages when we met the requirement of answering at least one question? The reason is that although the group validation criteria are met, the individual

question criteria are such that they instruct NubiS to detect any empty answers. In order to change this we would need to set the individual variables' 'If empty' settings to 'Allow empty'. For this reason, typically when group validation settings are employed that mandate the number of questions to be answered, 'If empty' of the variables involved should be set to 'Allow empty'.

An interesting scenario that comes into play when there are multiple questions shown on the same screen is to ask NubiS to enforce validity between the answers. To illustrate, let us first clear out any group level validation criteria we specified for 'TShow'. Then, we go to the 'Validation' tab of 'B105_':

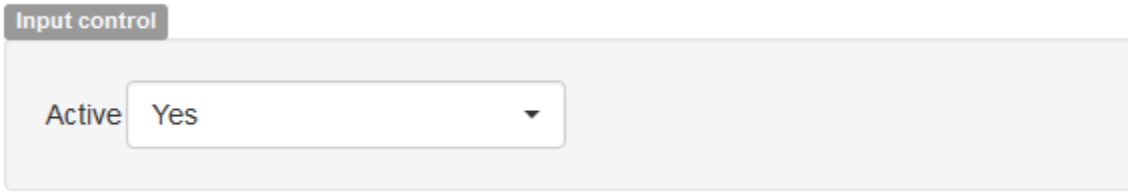


Here we stipulate that whatever answer is chosen for 'B105_', it must be the same one as for 'B106_'. If we try this and not give the same answer we get:



Obviously the standard NubiS message here is not very informative, and so we would want to enter a custom message on the 'Assistance' tab of 'B105_'. But it showcases that we can leverage these criteria also 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_').

Now, the validation mechanisms we have discussed so far are all reactive in nature. That is, a respondent enters answer(s) and on clicking ‘Next’ NubiS verifies whether these are valid. An alternative set of methods that is available focuses on prevention of incorrect answers by providing real-time input control. The nature thereof is dependent on the answer type. For example, let’s go to the ‘Validation’ tab of ‘B_new’ and enable real-time input control:

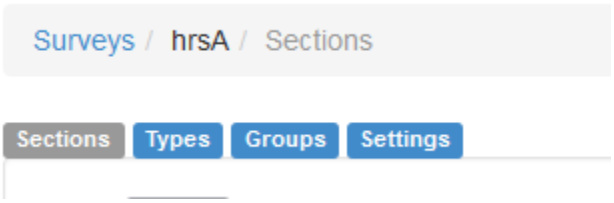


If we go to this question now in the survey, you will notice that if you attempt to select ‘None of the above’, any other selected options will be automatically deselected. In a similar style, if we were to activate real-time input control for our integer variable ‘numberofchildren’, we would find that we can only enter numbers. Any other character is not accepted.

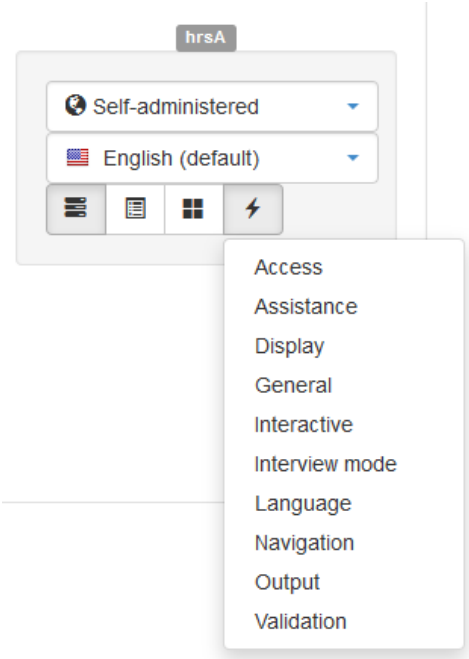
Input control in this fashion is quite powerful, but we should make a caveat on its proper functioning on some platforms. Particularly, devices running older versions of Chrome or Firefox on Android will prevent a respondent from properly entering characters when it comes to integer and real answer types. The same goes for browsers on Kindle devices. Moreover, it may be just plain confusing to respondents that they can’t enter certain characters, so it can be useful to add explanatory instructions on how certain questions can be answered. 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 real concern.

16. Additional interview modes and/or languages

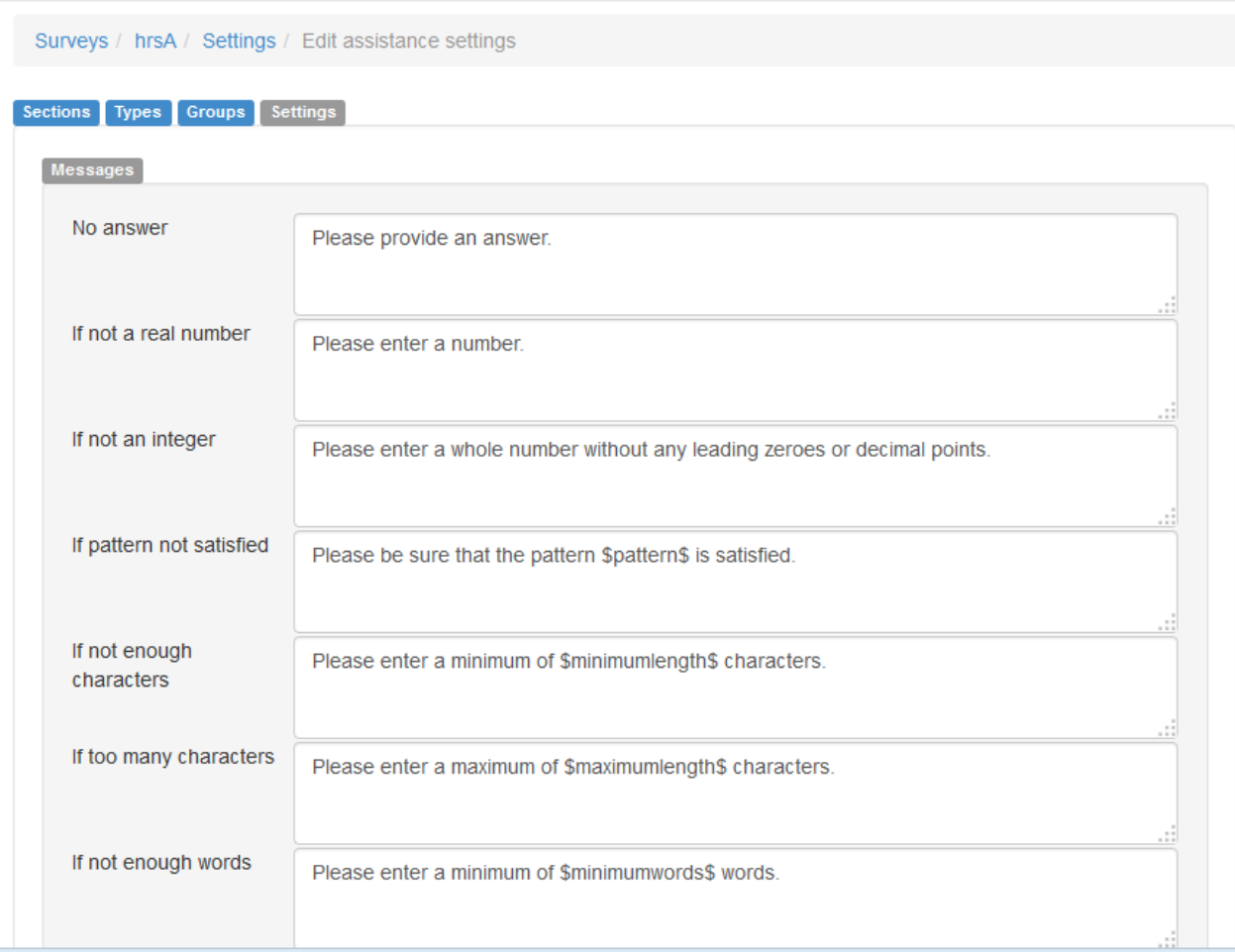
In the previous we have been busily adding sections, variables, types and groups, and tying those together in the routing. While we were doing that we mentioned on several occasions that many of the encountered settings can also be set on a survey level. These survey level settings then permeate downwards,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



They can also be found in the ‘Settings’ dropdown in the right hand side menu:

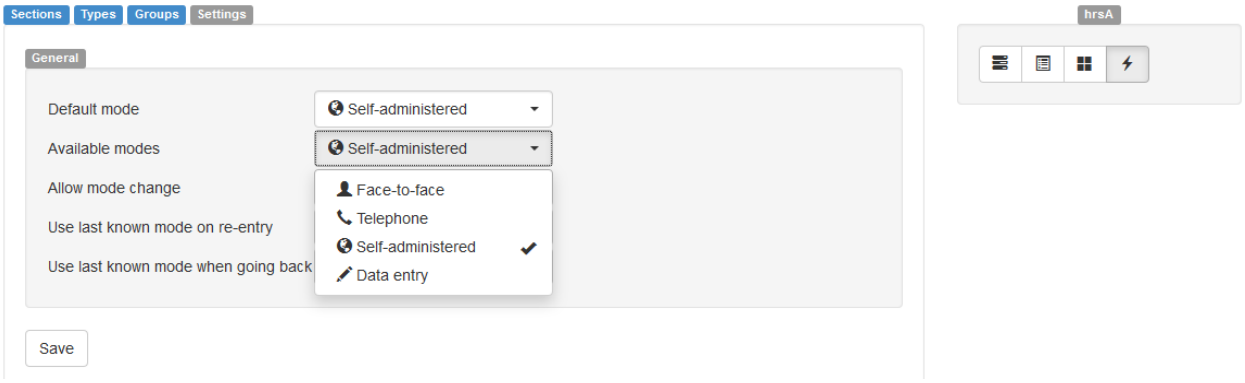


In both cases we find the same types of setting, most of which we have seen one or more examples of already. Each corresponds to a single screen listing all settings available across all answer types in NubiS. For example, in the ‘Assistance’ screen we see a long list of default error messages. Some of these we had seen in our survey already, like the ‘No answer’ message.



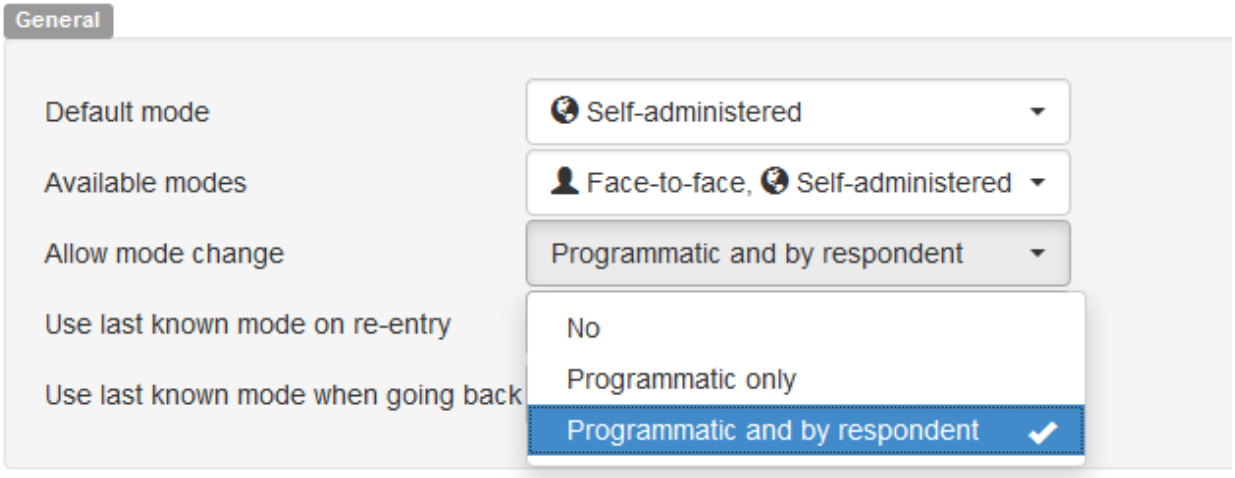
Most of the messages are straightforward enough, but some contain some more cryptic fragments such as **$minimumlength$**. These are NubiS placeholders in which it inserts whatever was specified in a corresponding setting. For example, if we had specified a minimum number of characters of 300 to be used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

Some forms of setting that we have not discussed yet are access, interactive, navigation, and output settings. These we leave for a later discussion. Instead we focus here on mode and language settings. It is those settings that control the manner in which NubiS deals with multiple interview mode(s) and/or interview language(s). The overall dynamic between the two is that each interview mode can have one or more languages. That is, languages are specified per mode instead of modes per language. This becomes evident when we open the interview mode settings screen, since on the right hand side we don't see any options to switch between modes or languages.



There are several mode settings available. First, and foremost, we can set the default mode. The default mode is the one that NubiS will adopt as the mode to be used in 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.

On first setup each NubiS survey has 'Self-administered' as the default mode (it's also the only active mode). To activate more modes we can toggle the 'Available modes'. When we do so, we can then choose another default mode if desired. We can also indicate whether it is allowed to change interview mode. This can be set to the following values:

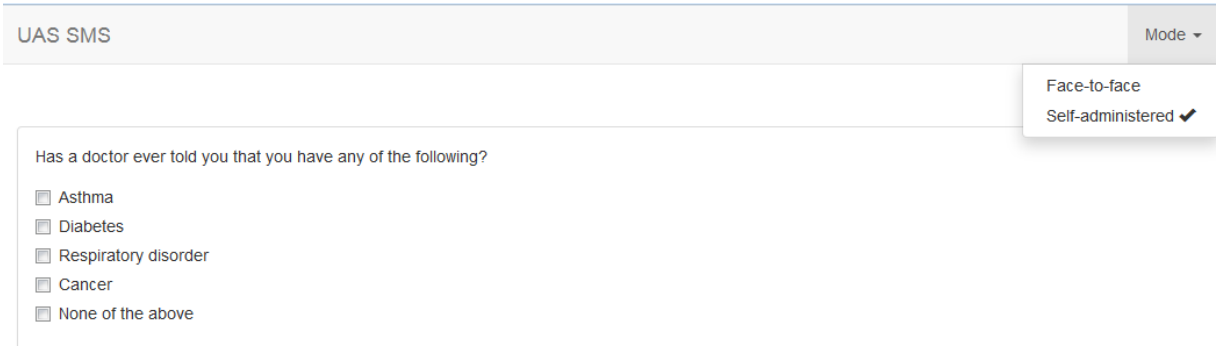


1. Programmatic and by respondent: this means that we can instruct NubiS through programmatic means to start the survey in a particular mode OR the respondent can change the interview mode within the survey.
2. Programmatic only: this means that we can instruct NubiS through programmatic means to start the survey in a particular mode, but the respondent cannot change the interview mode within the survey.

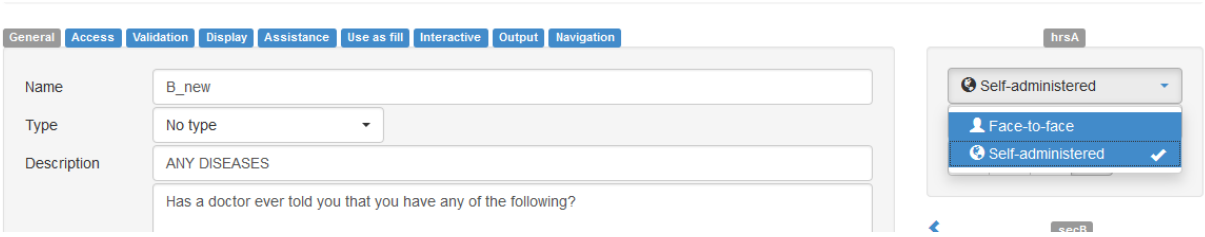
3. No: this means that the interview mode cannot be changed, meaning that the survey will always start in the default mode.

Related, the last two settings concern with how to deal with mode changes within a started survey. For example, we may have that a respondent starts our survey in a self-administered mode, but then call to complete it over the phone. On re-entry in the survey should NubiS then use the last known mode (self-administered) or not? Similarly, if a respondent was allowed to and in fact changed the interview mode and then clicked 'Back', should NubiS act like any previous questions were also answered in the new mode?

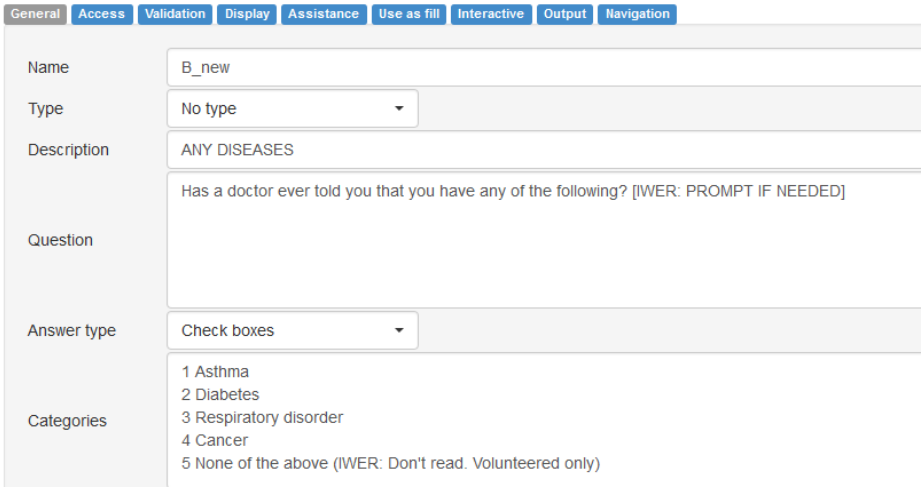
For now, let's activate the 'Face-to-face' interview mode and also set 'Allow mode change' to 'Programmatic and by respondent'. If we now start our survey (assuming that 'secB' is still the section on the 'Base' routing and 'B_new' is the first question in the 'secB' routing), we will see something like this:



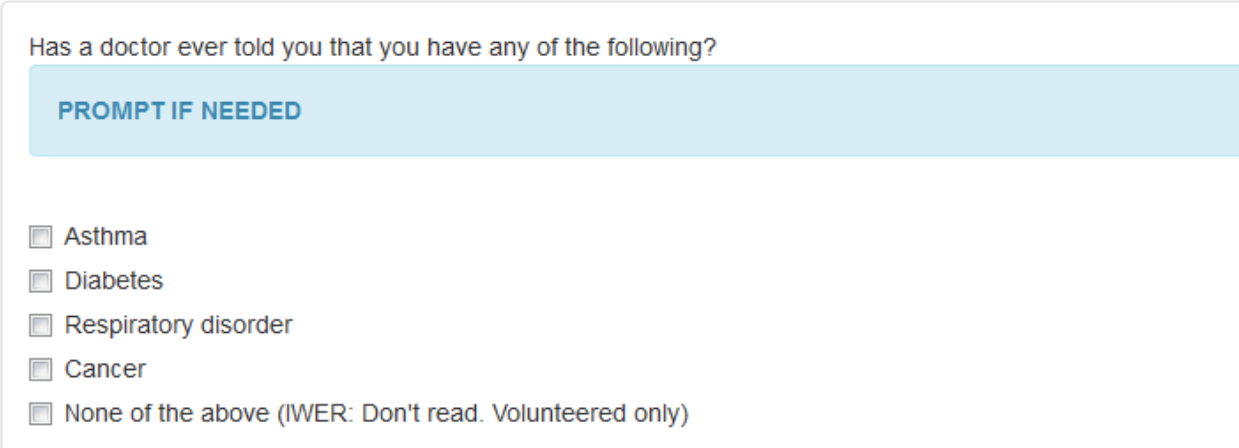
In other words, we now have a dropdown with which we can toggle the interview mode. Let's select the first answer option 'Asthma' and switch to 'Face-to-face'. As a somewhat anti-climactic result we see the exact same! This makes sense of course since we did not tell NubiS that anything of this question should be different between the two modes. And so, NubiS fell back on the default mode to find out what text and so on to use. To have a more exciting result, we exit test mode and go to 'B_new':



We switch to 'Face-to-face' and in the resulting screen change the question text and the answer categories:



Now, if we switch mode in the survey we get:

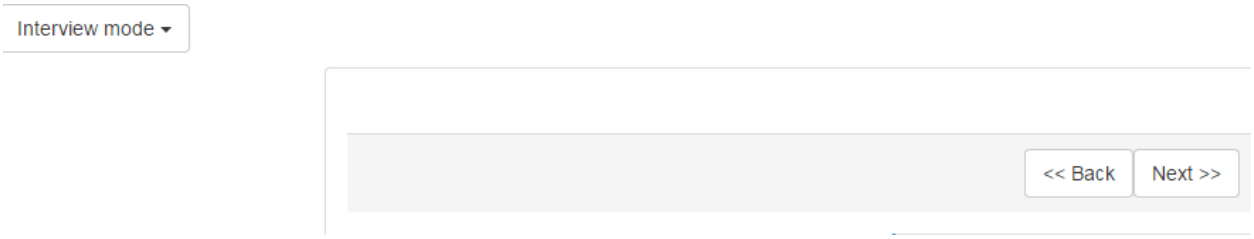


Here we see the text we specified for the ‘Face-to-face’ mode for ‘B_new’. The nice blue box we get courtesy of the fact that NubiS adds styling to “[IWER:] statements (we’ll return to this shortly when discussing advanced usage scenarios for NubiS).

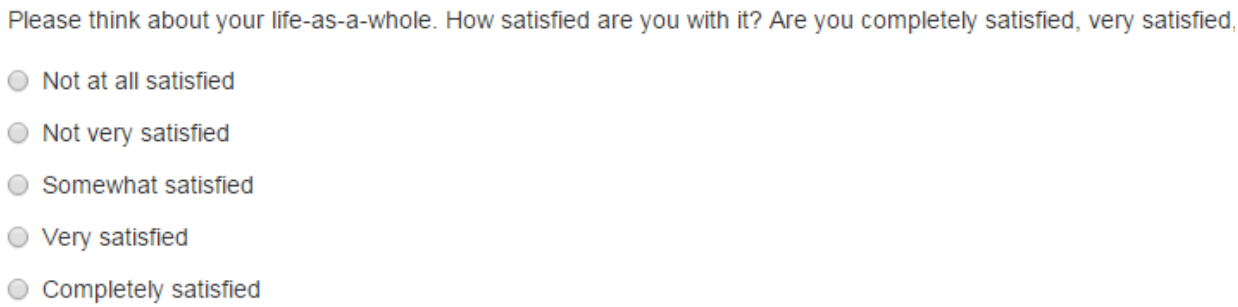
Preparing a survey for one or more additional interview modes is mostly adding settings for defined survey components (like variables) that override those we specified in the default mode. If there is no need to change anything from the default mode, then it is not needed to specify anything in additional modes. This is because of NubiS lookup strategy when handling multiple modes, which is that if it can’t find anything in the mode of the survey it will fall back on the default mode. That is why when we initially switched interview mode we still saw the question and answer options.

This brings up a related point, which is that changing the default mode is best done at the very beginning before any survey components have been added. The reason is that if we start in one default mode (like self-administered) and add variables for example, and then later on change the default mode (e.g. to face-to-face), NubiS will look in this new default mode for things like

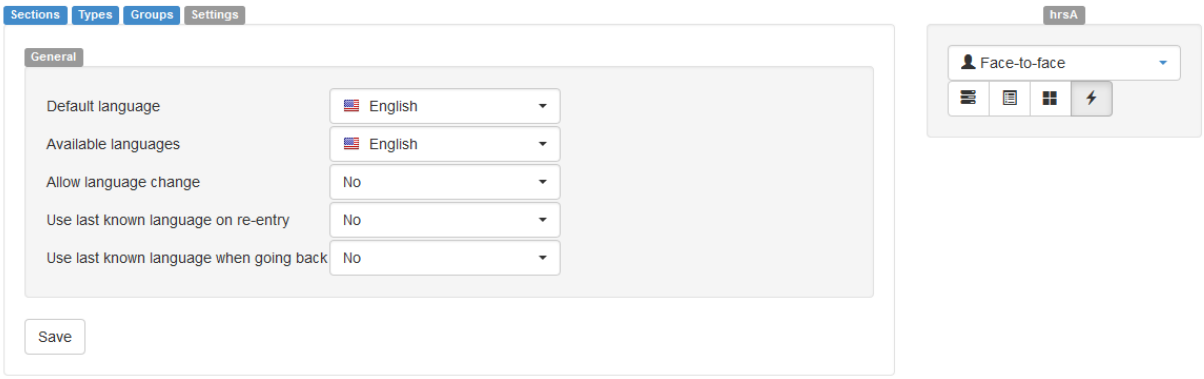
question text. Often case, this will lead to nothing being found because we had already defined everything for the old survey mode. This means that unless we tell NubiS to start in the old mode, we will get nothing. We can illustrate this if we temporarily change the default mode for our survey to ‘Face-to-face’ and then run our survey. Our first question, ‘B_new’, appears fine because we defined a new text and and answer options. But if we click ‘Next’ we get:



If we switch to ‘Self-administered’, all of a sudden the correct screen appears:



We’ll see later how we can get NubiS to start a survey in a mode different from the default mode. For now, we’ll change back the default mode to ‘Self-administered’ and turn our attention to the language settings:



The language settings are virtually the same as the mode settings we discussed, the only difference being that we have the option to set them per interview mode (we can switch between modes using the dropdown in the right hand side menu). NubiS comes with a standard list of available language names. If a language is not listed, it can be added and then used. NubiS does not impose any limitation on the language used, be it character based (like Chinese or Japanese) or not (like English or Spanish).

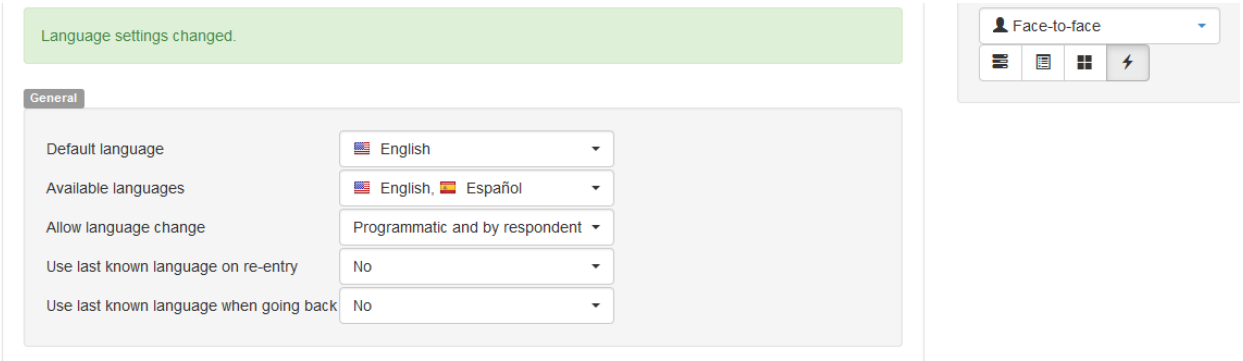
Like with interview modes, all characteristics we discussed for variables, sections, types, and groups can be configured per language as well. This means that if we were to add Spanish as a second survey language, the supported combinations are:

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

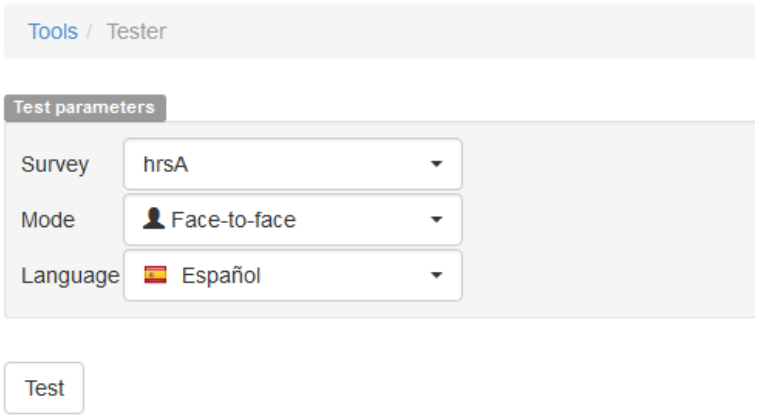
Also, as with interview modes, NubiS will fall back on the default language if nothing has been specified in the language used for the survey. When combined with multiple interview modes, the order is as follows:

1) NubiS first checks the current interview mode and current language of the survey.
2) If nothing could be found, NubiS checks the current mode with the default language of the survey.
3) If nothing could be found, NubiS checks the default mode with the current language of the survey.
4) If nothing could be found, NubiS checks the default mode with the default language of the survey.

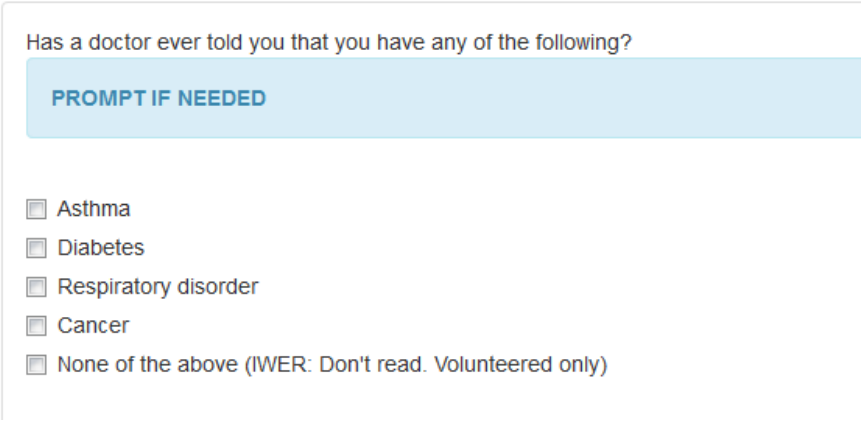
To illustrate all this, let's first add Spanish as another language:



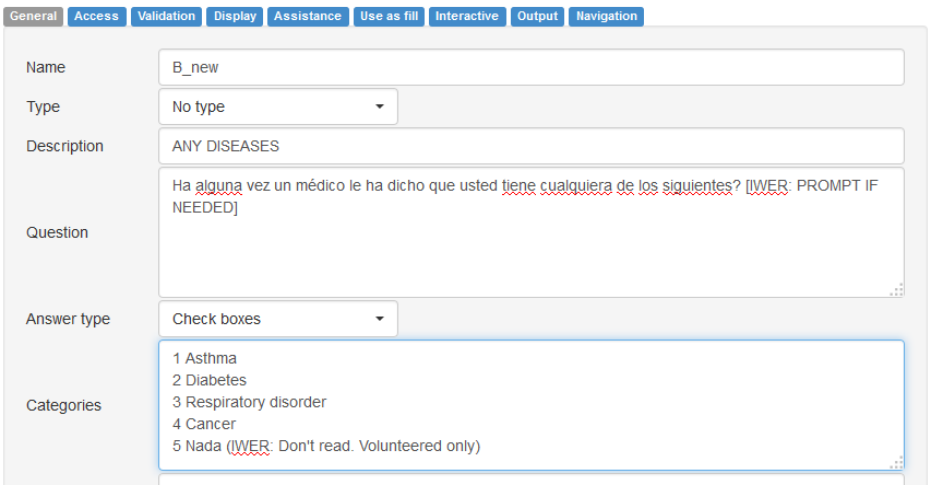
Now, we open the tester:



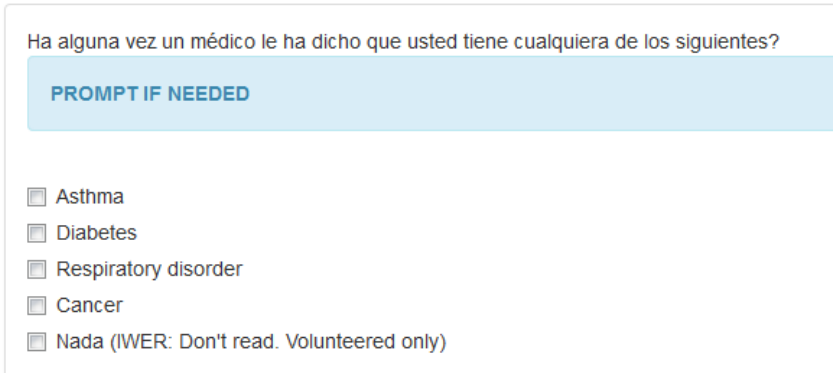
And start the survey in Spanish for face-to-face interviewing. The screen we get is:



This is what we would expect since we did not define a translation yet. If we were to do so, we would get the Spanish equivalent:



Which then shows as:



Note that if we were to switch to self-administered at this point, we get the English self-administered equivalent again, which shows that NubiS is simply ignoring the fact that we were in Spanish at the moment of switching, since no such language exists.

Also, it may of course not always be desirable to have the person programming the survey to also do the translation, particularly if it comes to different languages. For this purpose NubiS allows the creation of translator accounts. These accounts can be configured such that a translator can only enter translations for the selected language(s). More importantly, s/he will only be able to enter translations and nothing else. We will see in section ? on user management on how to add such accounts. For detailed information on NubiS' translator mode please refer to the 'NubiS Translator Manual'.

So far in the above each time we wanted to switch language or interview mode we used the dropdowns available to us. These however are typically only present in test mode. So what if we want to have the respondent be able to choose? Well, luckily that is relatively straightforward because 'language' and 'mode' are variables just like any other; and as such they can be included in the routing as any other variable. For example, on survey start the first question we might ask would be:

language
// the other questions

The respondent can then choose a language and the survey will continue in that language on clicking 'Next'. Note that an underlying assumption here is that 'Allow language change' has been set to 'Programmatic and by respondent'. If not, then NubiS will simply 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, we might wish to ask a question about whether English is a respondent's second language only of Spanish speakers, something we can accomplish quite easily by specifying our routing as:

if language = 2 then

//ask Spanish speakers only (we assume here that '2' represents Spanish for a moment)

endif

17. Advanced features

In addition to the functionality we discussed so far for developing a survey, there are several other options that we want to discuss here (even though their usage is often less frequent).

17.1 Nested loops and variables of type section

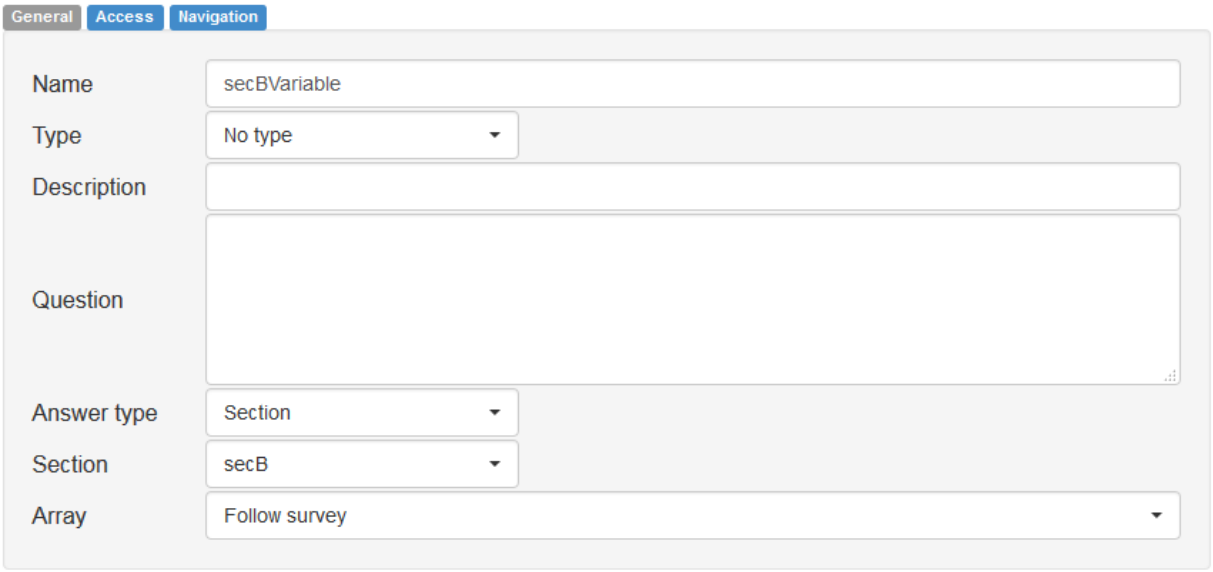
In our discussion of how to instruct NubiS to ask questions we introduced the notion of loops to ask the same question multiple times. In that context we made the simplifying assumption that we only really needed one loop, e.g. to ask the name of each child. Although this covers most cases, there are circumstances in which this might not be sufficient. For example, suppose we wished to ask about the children of multiple families, e.g. all the families living in a neighborhood. One option would be to introduce separate variables for each family, but this would of course in effect be the same as our approach of defining separate variables for each child. A better option would be to introduce a second loop whose routing would look something similar 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 we see that we took our original loop to ask about children and placed it inside another loop. We then modified the '[cnt]' statements to keep track not only of the original loop but also of the new outer loop. There are no limitations to the level of nesting when it comes to loops, although in practice a nesting of three loops inside one another is likely to be sufficient.

Though the above usage of nested loops to accommodate situations like 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 to this is to use variables of type section in combination with loops to let NubiS handle part of that.

Let's start with illustrating the notion of a variable of type section by adding a variable in the 'Base' section of that type called 'secBVariable' (no question text is needed):



In the 'Section' dropdown that appears we choose 'secB' here. Next, we open the routing of 'secB' and instead of stating 'secB' we state 'secBVariable':

Now if we run the survey we get, well, exactly the same as we would have gotten if we had put 'secB' instead of 'secBVariable'. But what if we would want to ask 'secB' multiple times? One way is to add a for loop in the routing of 'secB' and then update all our references, e.g.:

```
for cnt := 1 to 5 do
        B_new[cnt]
        ...rest of section here
enddo
```

Another is to make our 'secBVariable' variable an array and then state the following on the routing of the 'Base' section:

If we run the survey now, we get ‘B_new’ as before. However, in the background NubiS stores this variable now as ‘secBVariable[1].B_new’ instead of ‘B_new’. This consequently has the effect that NubiS can discriminate now between answers for the first loop and any of the subsequent loops. And the nice thing is that we did not have 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 for example ‘secBVariable[1].B_new[1]’.

You may wonder whether in the above we would have to write really long fill references like ‘^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 we can just state ‘^B_new[1]’. However, if we are referencing ‘B_new[1]’ from another section, we would need to include the full reference. The reason is that NubiS has no way of knowing which instance of ‘B_new[1]’ we are talking about (we might have ‘secBVariable[1].B_new[1]’ and ‘secBSecondVariable[1].B_new[1]’ for example).

17.2 Using while loops instead of for loops

Up until now we have discussed how we can use loops to ask a series of questions multiple times. This suffices for cases in which we want to do something a finite amount of times. There may be cases though in which the maximum of the loop is less well defined. A prime example is that of a conditional loop in which we attempt to elicit a response while allowing the respondent the option to correct the response. For example, we might have something like this to elicit information about a payment:

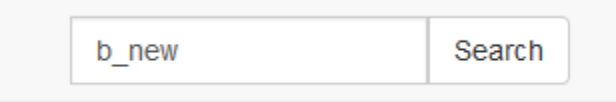
```
FOR cnt := 1 TO 25 DO
        WHILE (correct[cnt] != 2) DO
                // ask details
                GROUP.TStandard
                payment_time[cnt]
                payment_amount[cnt]
                ENDGROUP
                // ask if details are correct
                correct[cnt]
    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

        morepayments[cnt]
        IF morepayments[cnt] = 1 THEN
                EXITFOR
        ENDIF
ENDDO
```

In the above snippet we go through a loop from 1 to 25 times asking about payments. As we enter the loop we encounter the while construct. This construct basically says that we want 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 we enter the while loop and ask the time and amount of the first payment. Upon clicking next we ask the ‘correct’ question, which would e.g. show the answers given and ask ‘Is this correct?’.

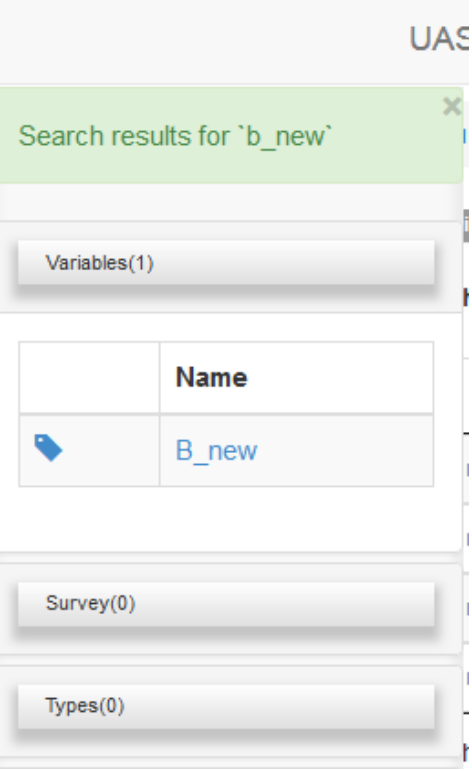
Let’s assume we say ‘No’ because we made a mistake in the amount. On clicking ‘Next’ we reach the end of while (indicated by ENDWHILE) and go back to the beginning WHILE statement. The condition is still true (because correct is not equal to 2), so we show the same payment screen again. We correct the amount, click ‘Next’ and now we say ‘Yes, this is correct’. We get to the ENDWHILE, which takes us back to the WHILE. Now, however, the condition is no longer true. As a result NubiS exits the while loop and moves on to the ‘morepayments’ question. If we say we have more payments, we go back to the while loop. If we don’t, we exit the for loop.

17.3 On screen interactive behavior

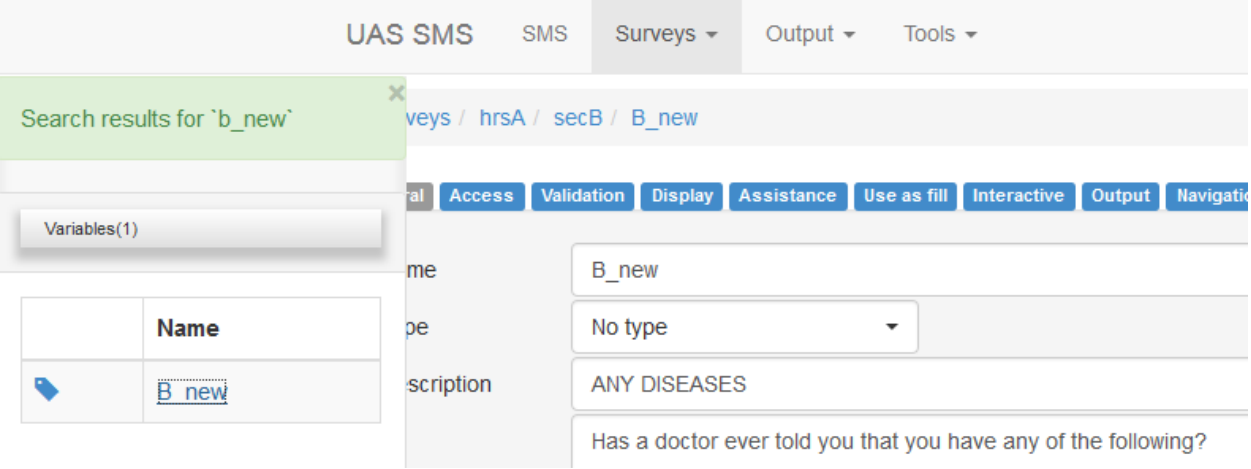
Sometimes we may want to implement some form of interactive behavior on a question screen. We already saw some examples of that, e.g. when looking at the automated deselect of any checked options in a 'Check all that apply' type question when 'None of the above' is checked. Beyond these built-in types of on screen interactive behavior, it is also possible to develop custom behaviors. This can be done on a per question basis by leveraging the settings accessible through the 'Interactive' tab. For an example of this let us return to 'B_new' in section 'secB' using the search function:



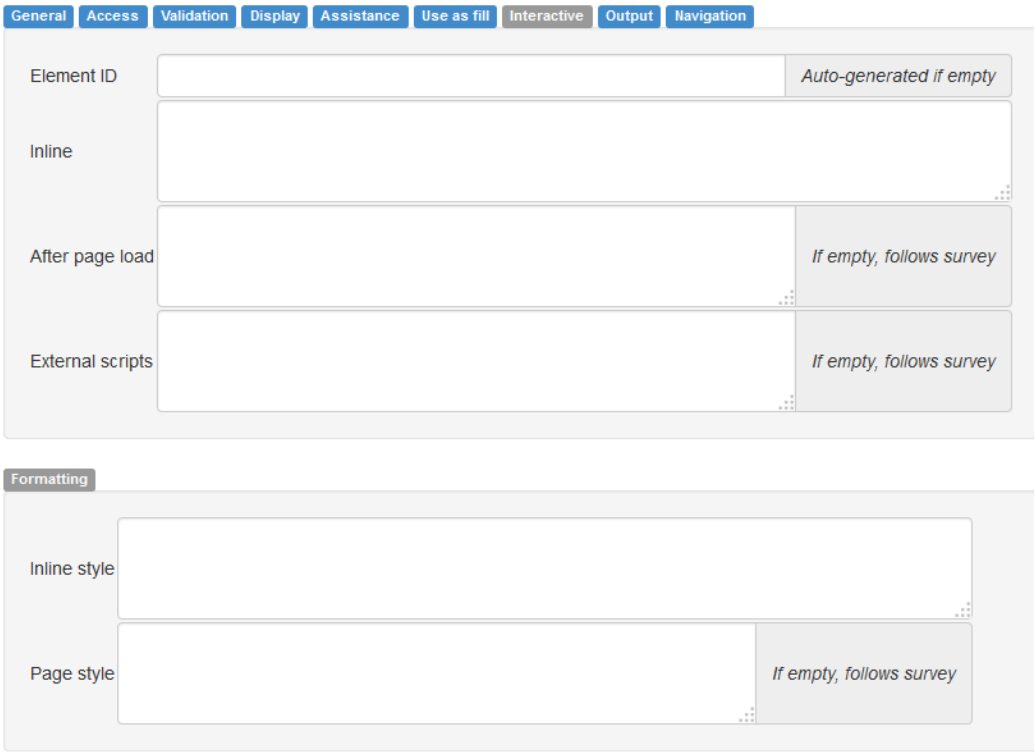
This will give us any results with 'b_new' in it (the search is case insensitive) leading to a result display in the left side:



The search display shows per type of component which ones were a match for the search term. Under 'Variables (1)' we find the one result 'B_new'. A direct link is provided to open this variable (as well as an option to add it to the 'Batch editor'):



We can then close the search display by clicking the 'X', and open the Interactive' tab:

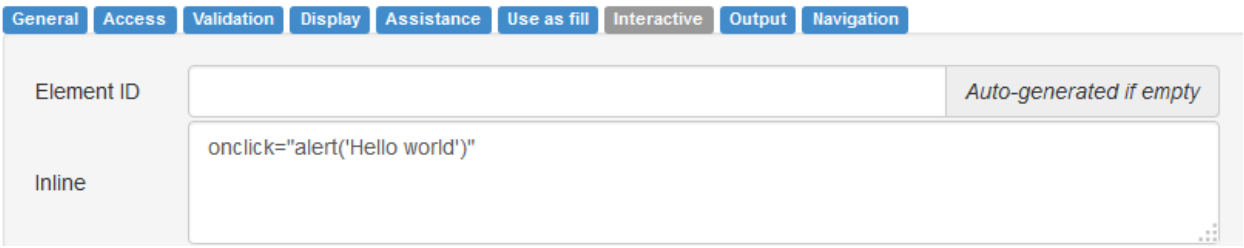


The tab is divided into two parts. The upper part contains settings that can be used to implement interactive behavior, whereas the 'Formatting' part allows custom styling.

In terms of interactive behavior NubiS supports three methods through the inclusion of JavaScript web scripting language statements. The first is by catering for so-called inline statements. They are called this 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">

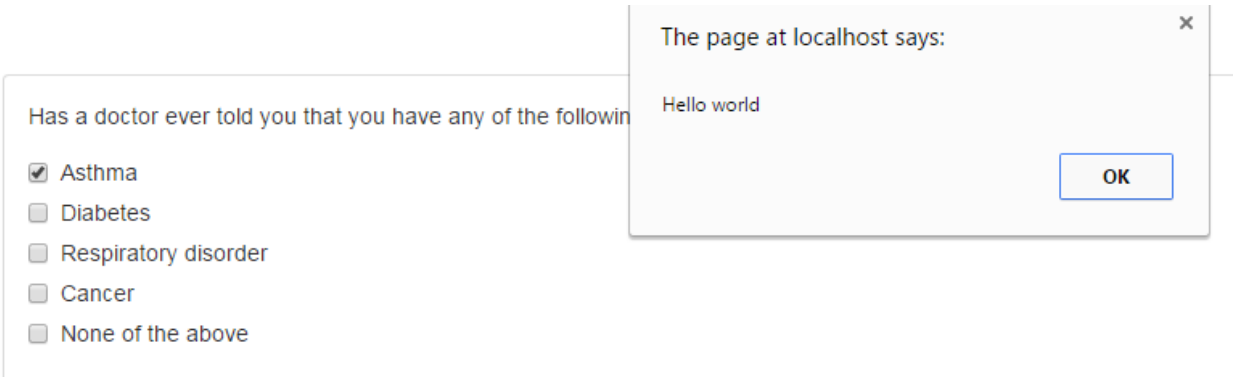
Ignoring for a moment the HTML tag attributes dealing with validation (such as 'data-msg-required'), this is a typical HTML definition of a checkbox. Now let's suppose that we want to show a 'Hello World' prompt if one of the checkboxes is clicked. To do this we can specify:



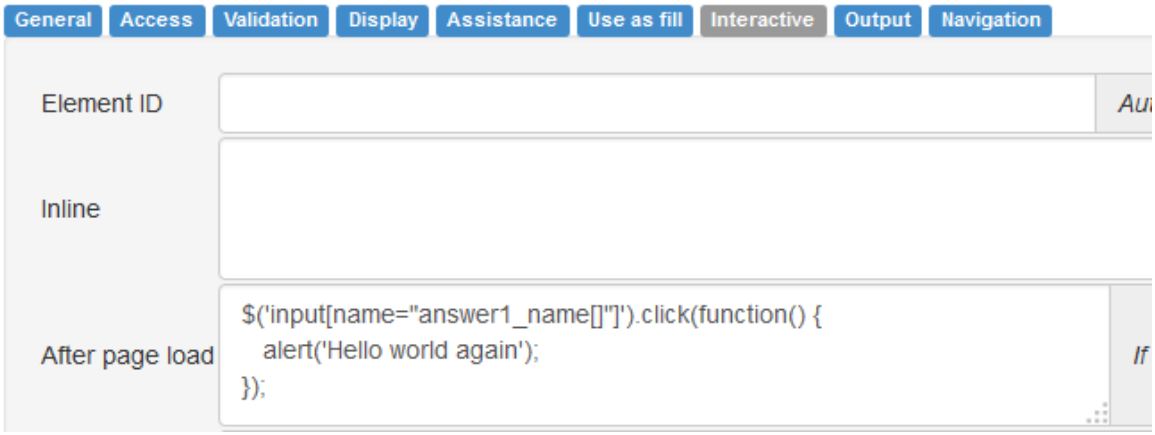
This will lead to:

<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">

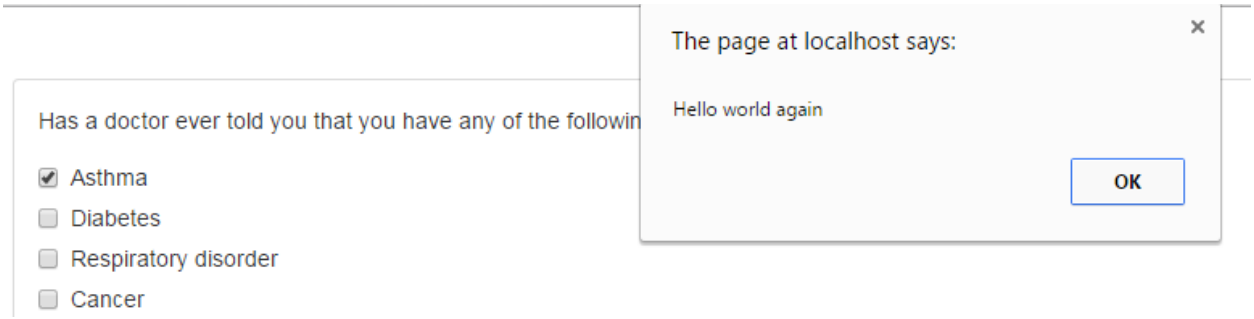
And a prompt if we click a checkbox:



A second manner in which we can accomplish this is by specifying it as an 'after page load' script (which uses a more modern style of programming):



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



If we were to look in the source we would see that our code was included as follows:

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

A close inspection would also reveal that the script is placed AFTER the definition of the checkboxes. The reason is that when the browser interprets the above it will attempt to link our code to the input checkboxes with the name 'answer1_name[ ]'. If none exist yet, this link will not be established. Even with this though there is no guarantee that the browser will always do things in the correct order. To enforce this the usage of JQuery 'on ready' statements is required:

$(document).ready(function() {
$('input[name="answer1_name[ ]"]').click(function() {
alert('Hello world again');
});
});
This will tell the browser to not attempt to link until the entire page has been loaded.

One thing that we might note in the above is that the prompt will appear for any checkbox that is clicked. What if we would only want it for the third checkbox? This is easy to accomplish by:

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

Here we instruct to link to the input checkbox with the identifier '#answer1_3' and no other. How did we know though which identifier to use? One way is to look in the HTML source that 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' plus a number, where the number corresponds to the order in which it was encountered on the routing. In other words, if we were to have a group with two questions, then the second question would have identifiers starting with 'answer2'. In this case NubiS also added '_1' in order to ensure that each checkbox has its own unique identifier (otherwise attempting to lookup an element with JavaScript would lead to ambiguous results). It does this for questions that have radio buttons or checkboxes taking the numerical answer option code and appending it with a '_' to the normal identifier.

Now, obviously it might not always be straightforward to know which identifiers would be assigned. For example, if we were to have a group statement like this:

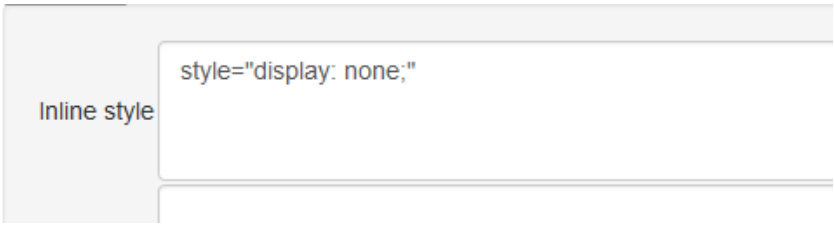
```
group.TExample
        if cnt = 1 then
                Q1
                Q2
        else
                Q2
                Q1
        endif
endgroup
```

Here we would not know which would get 'answer1' and which 'answer2' as this depends on the value of 'cnt'. One solution of course is to make our identifier a fill and define our code like this:

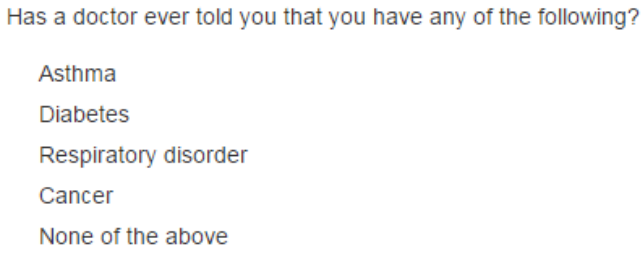
```
$( document ).ready(function() {
        $('#^myfill').click(function() {
           alert('Hello world from me only');
        });
});
```

And this is indeed a viable option. In this case a more straightforward solution is available though, which is to specify an identifier:

The formatting options provided on the ‘Interactive’ tab operate in a similar fashion as the ones for interactive behavior. The inline style setting allows to specify a Cascading Style Sheet compliant instruction into the outputted HTML input element. For example, we might state:



Resulting in no checkboxes:



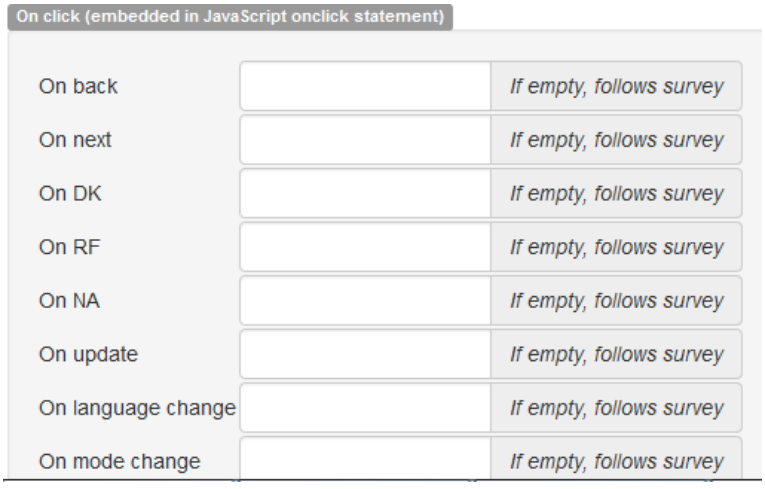
Similarly, we could specify style sheets fragments in ‘Page style’ to do the same thing:

Finally, JavaScript can also be included by referencing external files. For example, in ‘External scripts’ we could state:

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

This script could include function libraries e.g. or some other code we require for whatever we are trying to accomplish.

In addition to the previous, there are is another way in which we can add interaction and that is to the buttons. For this we can use the on click settings:



Any code we enter here must be valid JavaScript and NOT use single quotes. For example, if we would want to call a function ‘mySubmitFunction’ when we click next, we can specify that function in the ‘After page load’ setting and then specify ‘mySubmitFunction();’ as the ‘On next’ setting.

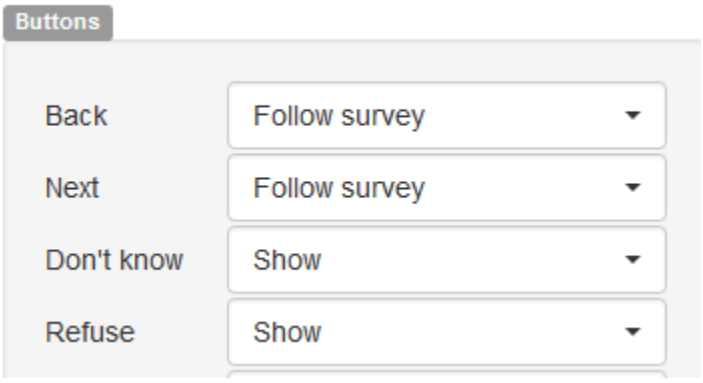
Lastly, it should be noted that a discussion of the above and of JavaScript/JQuery and CSS in general is beyond the scope of this manual. Rather, the examples are intended to be illustrative in nature only of the different options available to include interactive behavior.

17.4 Don’t know,refuse and not applicable

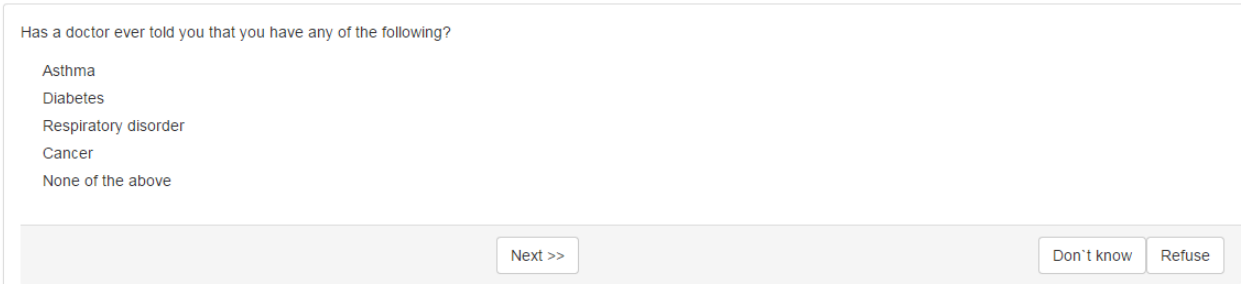
When we looked at the options for validating respondent answers, we spent some time on the settings for what NubiS should do if an answer was empty OR if an answer was erroneous. What we left unexplored though is what it means for an answer to be empty. From a purely programming point of view for NubiS it is simple: empty is when no answer was given. But from an analytical point of view it is not as clear. Does an empty answer mean that the respondent didn’t know the answer or didn’t want to give the answer? For example, suppose we 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 we asked them how large their investment in stocks is saying “Don’t know” can be a genuine interpretation just like “Not applicable”. Or perhaps the respondent does not want to say, i.e. is refusing.

In the context of self-administered mode the loss of this distinction is usefully remedied by adding explicit “Don’t know/refuse/not applicable” options. This is straightforward with radio button or checkbox questions, since it just involves adding an option. For other types of variables it would be necessary though to introduce separate variables that can capture a “Don’t know”, “Refuse” or “Not applicable” answer. A possible way could be to have two checkboxes, but we would want to make sure of course that if we check “Refuse”, that “Don’t know” gets unchecked as well as for example a value we entered in a text box. This is possible with what we discussed so far, particularly also the interactive options we reviewed in the previous section. Luckily, though, we don’t have to because NubiS already provides built-in support for that.

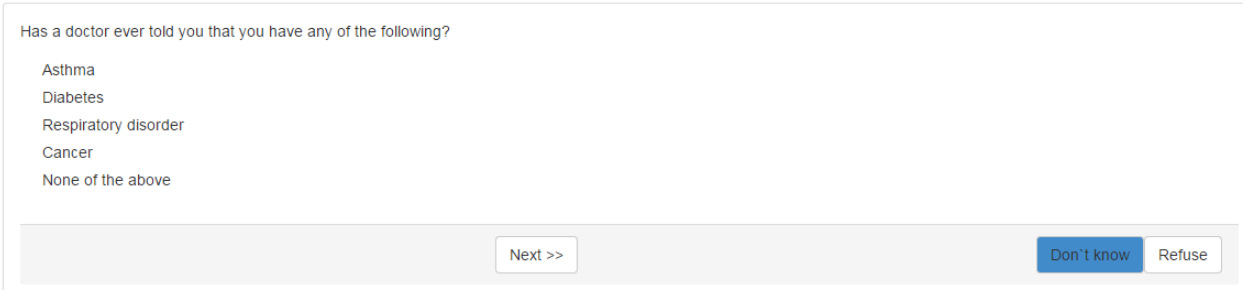
The first option is a per question screen mechanism. To activate this let’s head over to the ‘Display’ tab of ‘B_new’ and do the following:



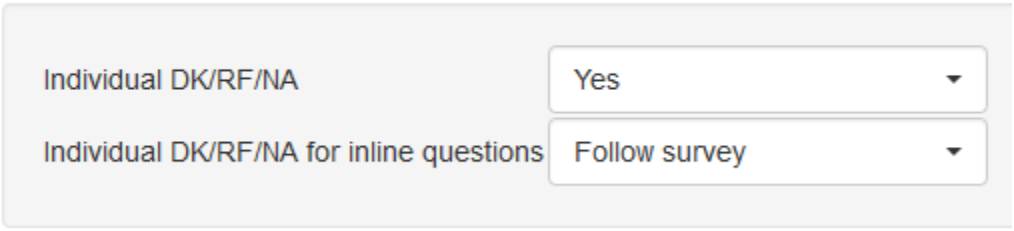
Now, if we go to ‘B_new’ in our survey we get:



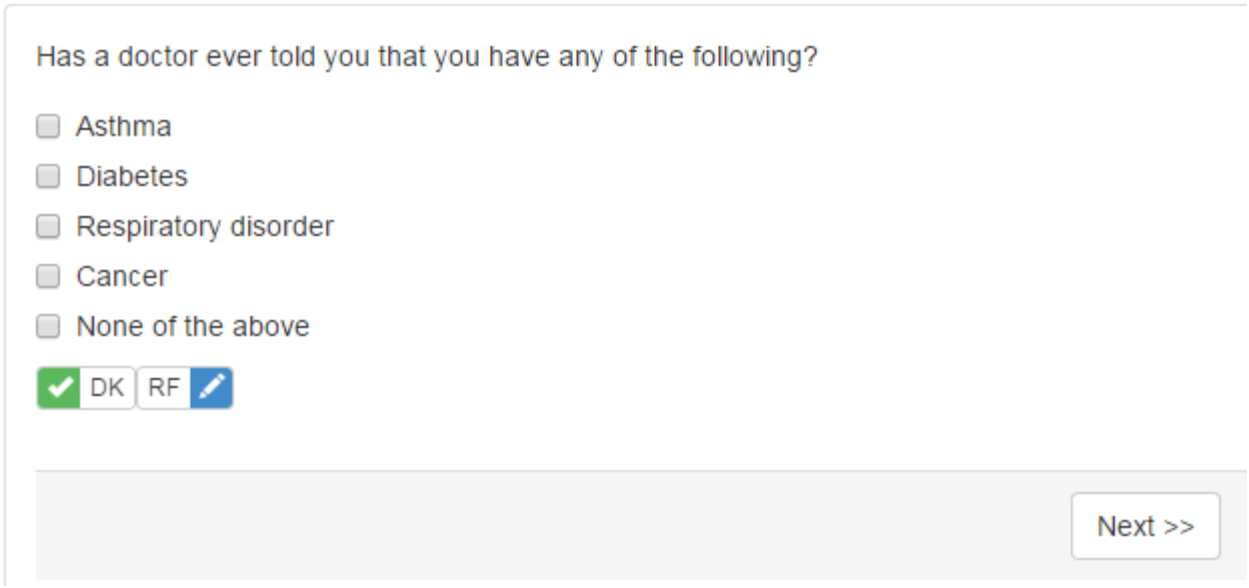
If we click either one of the two new buttons we have, NubiS will proceed to the next screen. Then, if we go back we see our choice was still selected:



The advantage of this approach is that it nicely integrates with the usual buttons that are available, making it easy to use. The downside is that clicking e.g. “Refuse” means refusal to answer any and all questions on the screen. This is typically not an issue, as explicit “Don’t know” and “Refuse” options are often only employed in face-to-face interviewing. And in face-to-face interviewing the custom is to only display one question per screen. However, it might occur though that there is a need for “Don’t know” and “Refuse” options that can be applied on a per question basis. If this is the case, then we can activate the ‘Per question’ mechanism on the ‘Navigation’ tab of ‘B_new’:



If we look at ‘B_new’ now, we see:



The “Don’t know” and “Refuse” options are now placed close to the question. Moreover, clicking one of them does not result in going to the next question screen. Rather, if we were to hit ‘RF’, then we would see that ‘DK’ gets automatically deselected. Similarly, if we were then to check a box, ‘RF’ would be deselected. In other words, this mechanism is akin to the one we just informally described.

Now, once a survey starts to use explicit “Don’t know” and “Refuse” options, one thing to keep in mind is how those are interpreted by NubiS when it comes to routing. For example, suppose we have:

```
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 does “Don’t know” and “Refuse” constitute an answer in this context? The answer (and we can see this if we try out the above) is no. The reason is that an answer is really only considered a response if it is not a non-answer (which is nothing, don’t know, or refuse). Now what about this:

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

If we try this by answering for example ‘Refuse’, we find that NubiS says that this is an answer; i.e. ‘B_new’ is not considered to be empty. This leads to the more general remark that stating:

if B_new = response then

is not the same as stating:

if B_new != empty then

if there are explicit “Don’t know”, “Refuse” and/or “Not applicable” options being used. We can see this by answering ‘Refuse’ and see that the second condition is satisfied (and ‘B000_’ is asked), whereas the first one is not. 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 we did not activate a “Not applicable” button 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 we just described, there is another type of button that we can activate. That button is the ‘Update’ button. If this button is shown on the screen and clicked by the respondent, the current page is updated and shown again. We might wonder in what kind of scenario we would need such functionality. Well, suppose that we are asking about the names of the respondent’s children using the following routing:

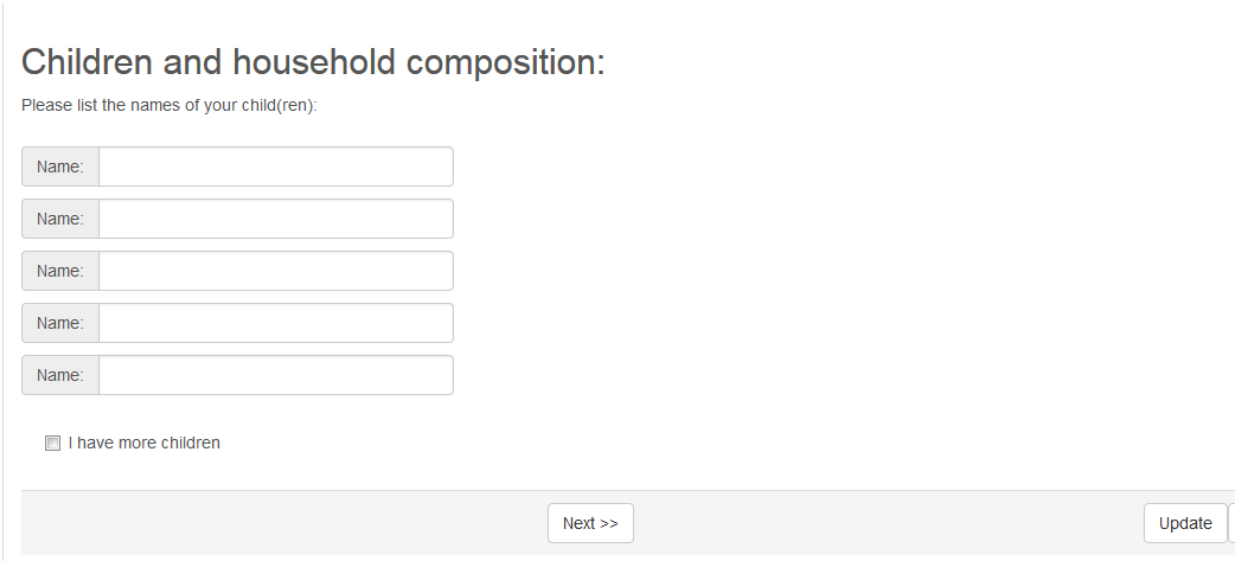
```
group.Toveral
max := 5
introChild // no input variable showing the question text
for cnt := 1 to max do
        ChildX058AFName[cnt] // string variables that asks the name of the child
enddo
endgroup
```

This would show 5 input boxes on the screen in which the respondent can enter names:

Now suppose we want to provide the option to have another 5 input boxes appear for the eventuality that the respondent has more than 5 children. One option would be to do the following:

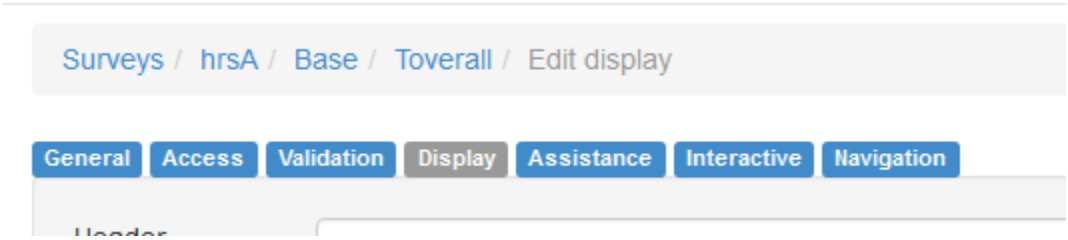
```
group.Toverall
max := 5
introChild // no input variable showing the question text
for cnt := 1 to max do
        ChildX058AFName[cnt] // string variables that asks the name of the child
enddo

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

In the survey this might look something like this:



With this in place there are two ways in which we can proceed to ask the respondent about any more children s/he has if they check the box 'I have more children'. The first is to let them click 'Next' and then follow up with a similar screen. The downside is that the respondent can't see the names entered already (unless we show them of course as fills in the question text of the follow up screen).

Another way is to utilize the 'Update' button. We can activate this button by editing a variable. In this case though we will edit it on the 'Display' tab of group 'TOverall'. The reason is that the display of a button or the display of a progress bar is a setting applicable to the screen as a whole. As such, if we have multiple variables being displayed on the same screen, we need to specify settings related to such matters for the group of variables as a whole (otherwise we would have to do this for each variable in the group statement).

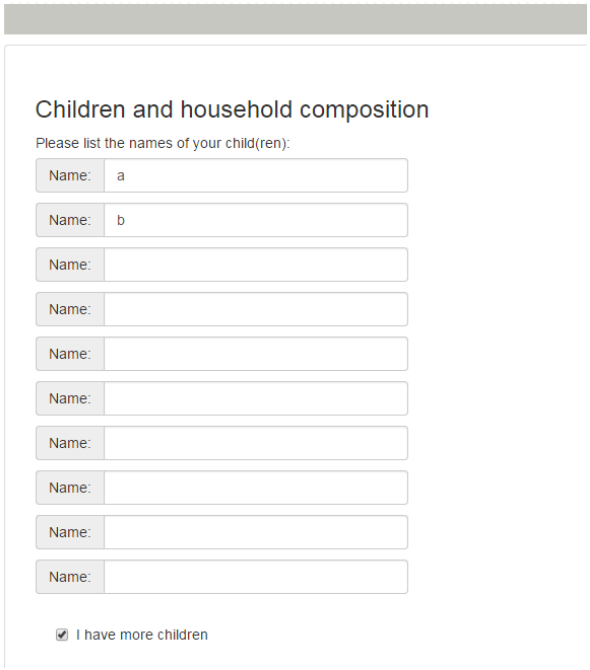


To enable the ‘Update’ button we simply select ‘Show’ instead of ‘Follow survey’ and save our change.

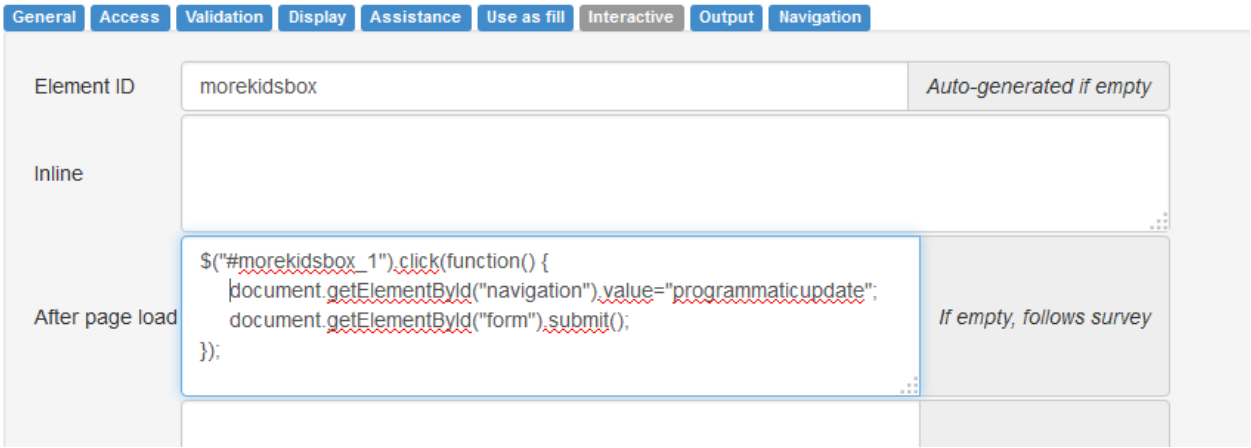
The question now is what will happen when the respondent clicks this button. In a nutshell what happens is that NubiS refreshes the current screen BUT stores any answers given to questions prior to the respondent having clicked ‘Update’. In our example, if we check the box ‘I have more children’ and click ‘Update’, NubiS will store this answer to ‘morekids’ and then show the same screen again. How does that help us? Well, we can modify our 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
```

Remember, if the respondent were to check the box and click ‘Update’ now, NubiS will store the answer to ‘morekids’ (which would be ‘1’ given we checked it), and then run our snippet of routing again to display the same screen. That is, it will once again execute the group statement. Now, however, instead of ‘max’ being set to ‘5’ it would be set to ‘10’ leading to 5 extra instances of ‘ChildX058AFName’ to be included:



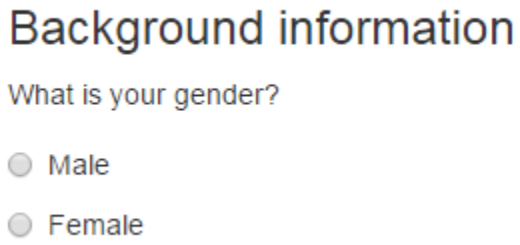
What if we want to reduce the number of actions the respondent needs to perform to refresh the screen? More specifically, could we have the screen refresh when the respondent clicks the checkbox? The short answer is yes! This can be accomplished using NubiS support for a so-called programmatic update of the screen. Setting it up involves a small snippet of JavaScript that is associated with our checkbox. So let's open the 'morekids' variable on the 'Interactive' tab and enter the following:



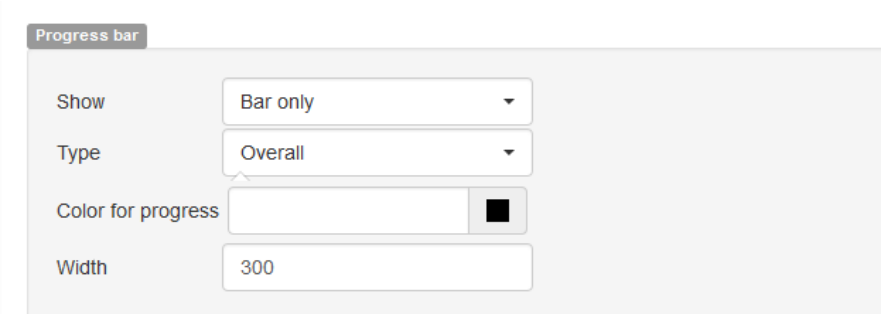
We assign an identifier to 'morekids' just to make it easier to reference it in our JavScript code in the 'After page load' text box. Here we basically state that if the box is clicked, we update the hidden input tracking the action performed. After that we state to submit the form in order to refresh the question screen.

17.6 Progress display

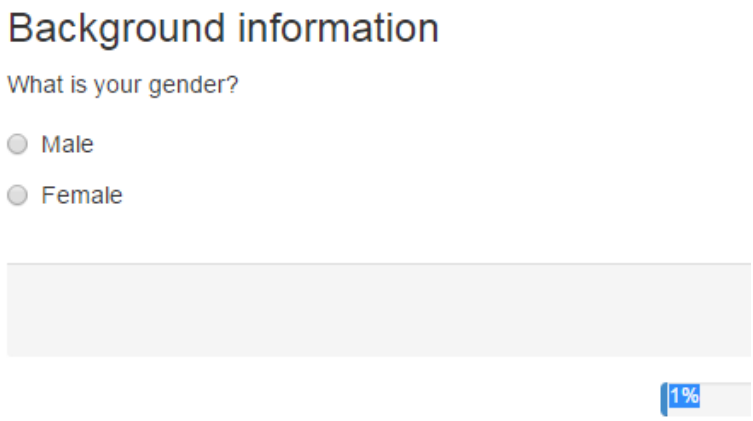
An important feature of a survey from a respondent's point of view is often knowing how much progress s/he has made towards reaching the end. There are a few ways in which such progress can be conveyed in NubiS. One is rudimentary in nature and involves using section headers to indicate in which part of the survey a respondent is. We saw an example of this when we used a header for 'secA' of "Background Information":



Another way is to use a progress measurement. NubiS provides two types of measurement both of which can be activated on the survey level 'Display' screen:

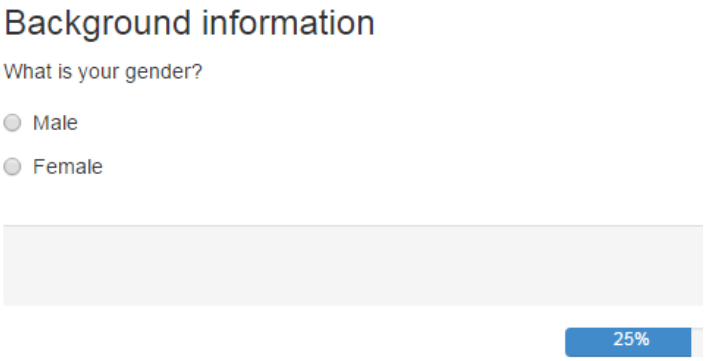


Firstly, the 'Show' setting instructs NubiS on how to display the progress measurement. This can be a bar, a percentage, a bar and a percentage or nothing at all. We have seen the bar already. Adding a percentage would look like:



The color to be used and the width of the bar can then be configured with the appropriate setting.

Secondly, we 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 very end of the survey. In contrast, the ‘Per section’ type measures progress using the end of the section of which the question is part. If we were to switch, then we would get:



We see the progress has ‘jumped’ from 1 to 25 percent, but this is only because we now measure progress until the end of ‘secA’. Once we reach the end of ‘secA’ and move on to another section, the progress bar will reset to start at the beginning again.

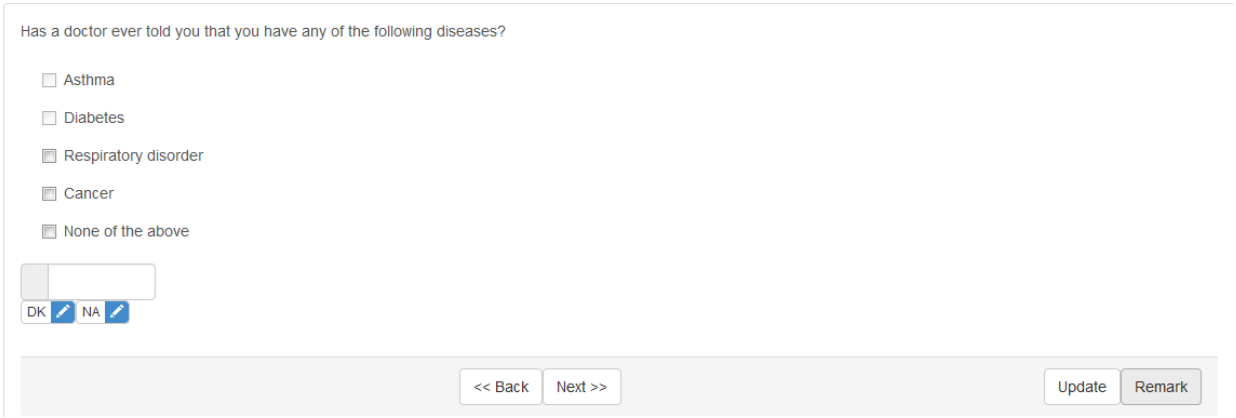
So when would we use one over the other? Well, for relatively simple surveys that are not very long, the ‘Overall’ progress bar would be most suitable. However, if a survey becomes very long, then the behavior of the ‘Overall’ measure can become uninformative. The reason is that if the distance to the end of the survey is very large, then completing one question won’t make much of a dent. As such, it would be better to switch to the ‘Per section’ type in order to avoid that the respondent feels s/he is not making any progress.

Lastly, as a more general remark it should be noted that the progress that NubiS displays is a rough approximation. In any complicated survey the distance until the end of the survey or section would be greatly dependent on the specific answers provided by the respondent. As

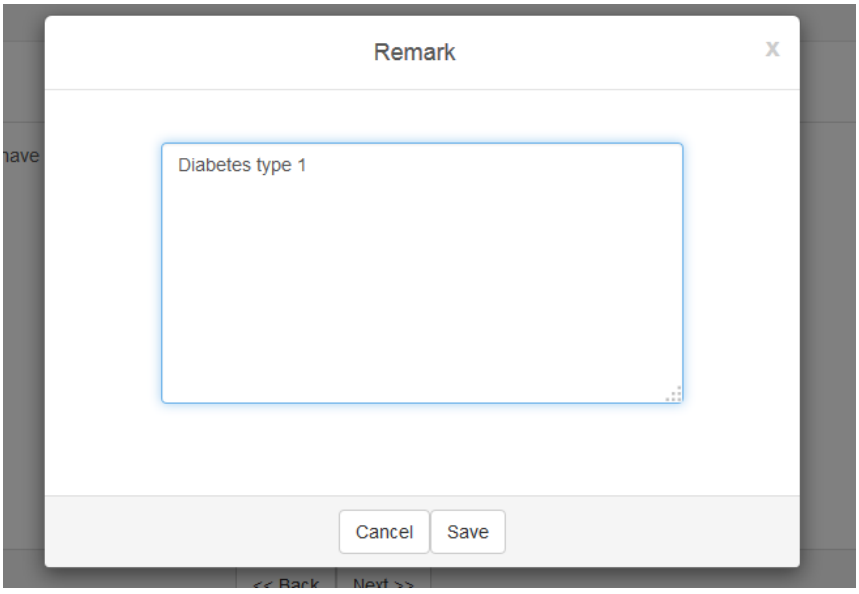
such, progress ‘jumps’ can occur in which a respondent for example skips half of a section due to his/her earlier response.

17.7 Remarks

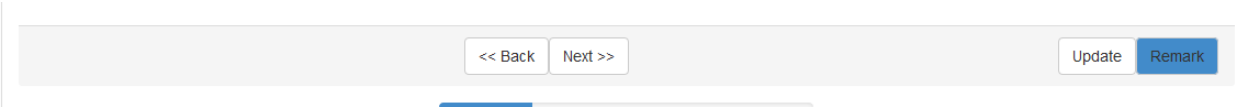
Another type of functionality that may be useful, particularly in a face-to-face or telephone interview mode, is the ability to add in remarks. For example, suppose we ask the following question:



Suppose that 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. That is where the ‘Remark’ button (here positioned at the right next to the ‘Update’ button comes in. Clicking it brings up a popup:

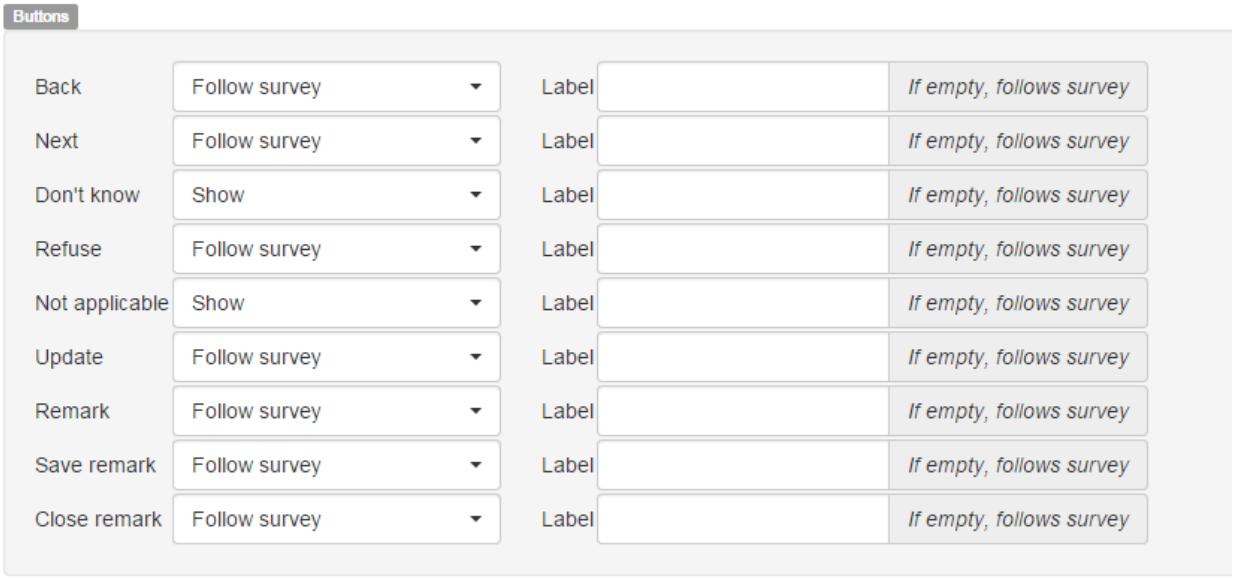


Here the interviewer can enter ‘Diabetes type 1’ and click ‘Save’. The popup will close and the ‘Remark’ button will turn blue to indicate a remark is present for this question screen:



Now, on clicking ‘Next’ the answer to the question will be stored but the remark as well. We’ll see in a bit when we discuss data output that we can extract those remarks, so we can use them to help during data analysis.

Activation of the remark mechanism is straightforward and similar to for example the ‘Update’ button. That is, we can choose whether to show it or not on the ‘Display’ tab of a variable. To illustrate, if we open up ‘B_new’ and select the ‘Display’ tab, we can decide which buttons to show:



Note that it is possible to enable the remark popup without showing the save or close button. The close button for example is not strictly necessary since clicking outside the remark popup will also close it. The save remark button can be disabled, e.g. to have a new interviewer be able to see the remarks but not be able to edit them.

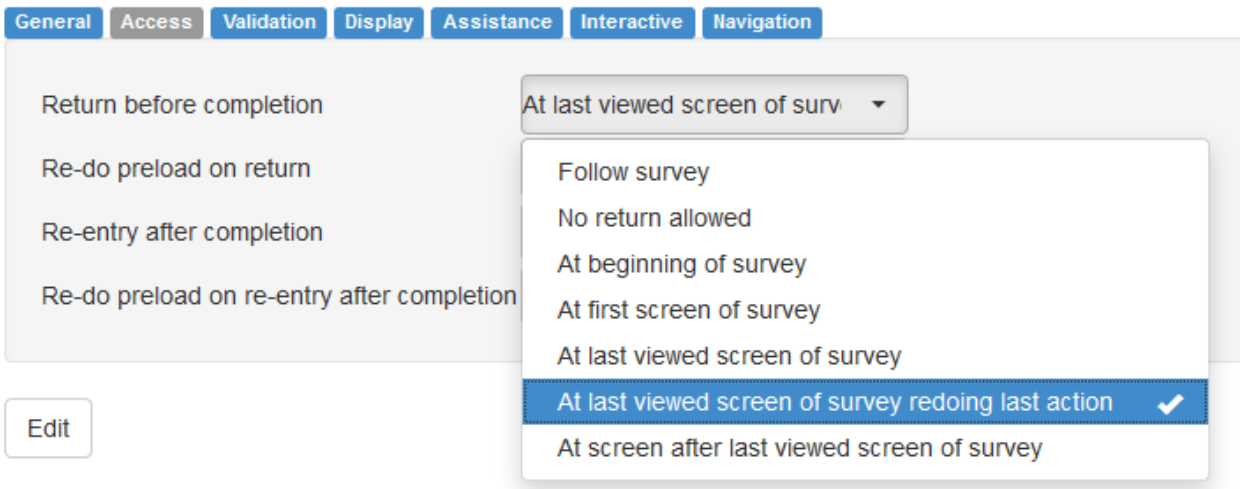
17.8 Showing a different screen on re-entry

On occasion it might be needed for the survey to have something like an eligibility screen informing the respondent whether they can fill out the survey or whether e.g. they have to wait. Imagine for example we want to give access to respondents only AFTER they have completed another survey. How could we accomplish this in NubiS?
At first thought we may come up with something like this:

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

This would have the effect that if the respondent arrives at this part of the survey, the function 'isCompleted' is called, which returns '1' if the other survey was completed (in reality this may be a query in a database for example). Let's assume the first time round the survey was not completed yet. This would lead to 'waitscreen' being shown. Suppose the respondent goes off and completes their other survey. They then re-enter our survey at the last screen that they left off at (NubiS' default behavior as we will see shortly in section 19.4) and...they still get the wait screen. What went wrong?

What went wrong is that by default NubiS re-shows the last screen viewed by the respondent, but it does NOT re-perform the last action. More concretely, NubiS will show 'waitscreen' again without considering the assignment and if condition just before it. To alter this, we need to modify the 'access' settings of our group 'Toverall':



Specifically, we instruct NubiS to redo the last action. This will result in the function 'isCompleted' to be called again and the 'continuescreen' to be shown (assuming the respondent completed the other survey).

18. Custom functionality

The options we discussed so far are often more than sufficient when developing a survey. There may be occasion though sometimes for features that NubiS does not offer by default. In those cases it is typically possible to extend NubiS to incorporate this functionality.

18.1 Using custom functions in the routing

One source of extensibility lies within NubiS' support for integrating custom defined functions. There are several avenues available here depending on the goal to be achieved. A first use case scenario is one in which we want to perform some kind of action in response to a respondent's answers. We have seen examples of this already, for example when we used PHP's built-in functions to calculate the respondent age. The usage of such functions through assignments is not limited though to pre-defined PHP functions. Rather, if required, custom functions can be added to NubiS. This is done by locating the 'customfunctions.php' file in the 'NubiS root/custom' folder and defining the needed PHP functions there. Anything that one could imagine doing in PHP normally can be done here (keeping in mind server imposed restrictions of course). NubiS does not impose any restrictions. Once defined, the function can be called on the routing like:

dummy := executeMyFunction()

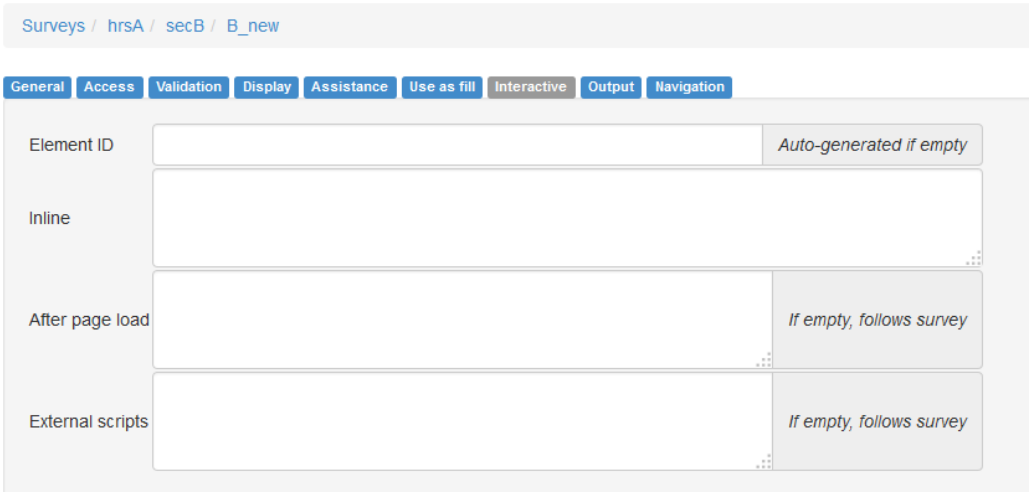
In this regard it is possible to pass along the answers to variables, e.g. by stating:

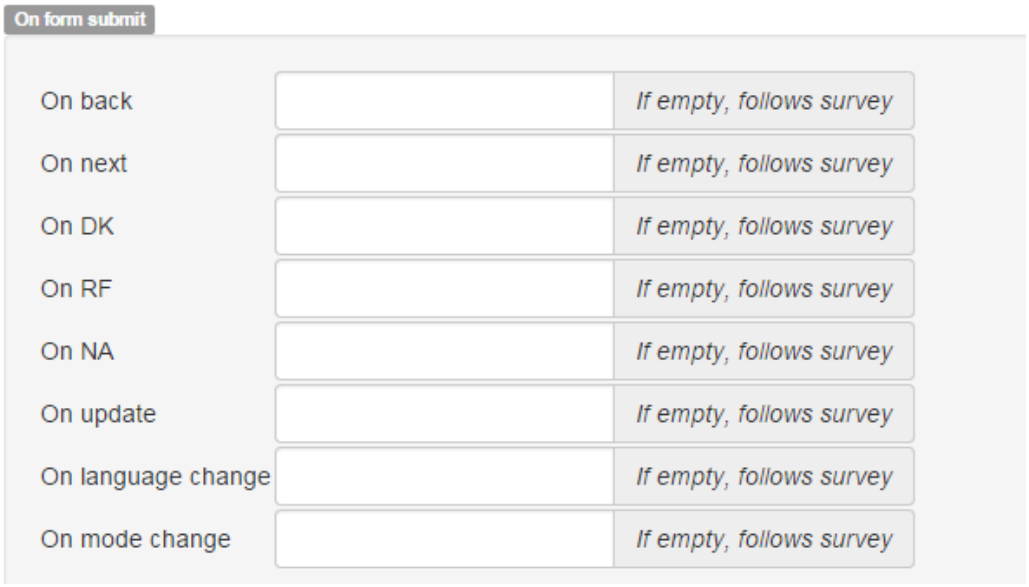
dummy := executeMyFunction(variable1, variable2, variable3)

The 'dummy' variable here is what the name conveys, a dummy variable. This is because NubiS expects to assign the result of a function call to something, in this case 'dummy'. Note that this does not mean that a custom function should return a result. Rather, it is a merely syntactical construct to instruct NubiS to execute the function.

18.2 Using custom functions on button click

Another place where custom functions can be leveraged is on the ‘Interactive’ tab of a variable. We have not inspected this tab before so let’s open it for ‘B_new’:

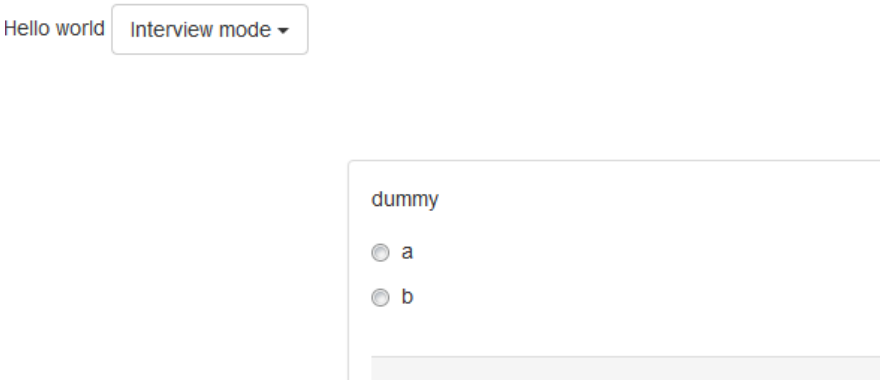




Ignoring the options towards the top, the ones under ‘On form submit’ allow to specify a function to be executed when the corresponding button is clicked. For example, suppose we defined a custom function as follows:

```
function mytest() {
        echo ‘Hello world’;
}
```

and enter ‘mytest’ for the ‘Next button’ value here. Then, if we run the survey and click ‘Next’ on the ‘B_new’ screen, we would see:



Obviously this function is not particularly useful, but one could imagine doing some processing here in response to the respondent’s answer to update an external database. We could also accomplish this of course as well by having a routing that says:

b_new
dummy := mytest()

Now suppose though that we need to execute a function on going back in the survey. NubiS will take care of undoing any assignments done, but this does not extend to whatever was done in a function call. For example, if ‘mytest’ updated a database table, then on going back we might want to rollback that change. 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 things like “bye” or references to variables. For example, we could specify ‘On next’ instead as:

mytest(^prim_key)

And modify our function to:

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

Which would then show as ‘Hello world: ‘ plus the case identifier. Note that the ‘$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 we would add something like the following to our 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;
}
```

As a final comment we should note that an 'On submit' function is executed AFTER NubiS' normal processing. For example, NubiS will first update its database for any answers given by the respondent, undo any assignments if going back in the survey, and so on.

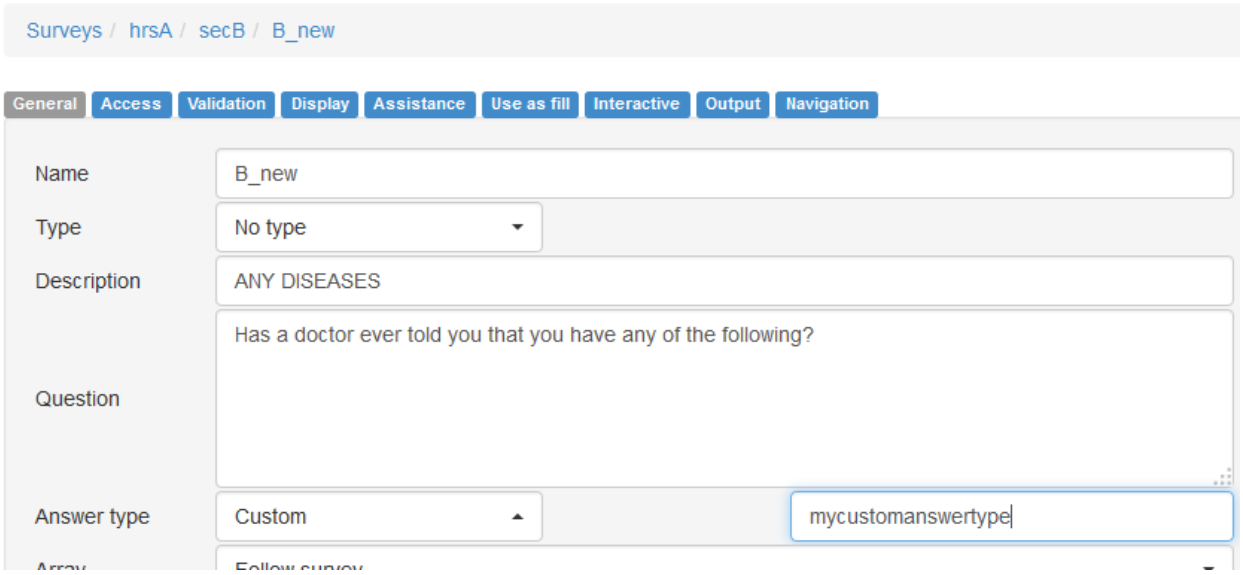
18.3 Modifying the auto-generated question display

Another ability we might need is to alter the outputted question screens in some form. We already saw an example of this when we introduced an [IWER:] syntax that then led to a nice blue box in the outputted 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 to use some form of notation consistently while programming the survey (e.g. for interviewer instructions) and then apply a certain formatting; rather than having to specify such format for each and 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:



Then, when NubiS encounters this question it will look for a function called 'mycustomanswertype' to tell it how to display the manner in which the respondent is to answer the question. We already showed one example of this with the household grid. The code behind

it is extensive, but we wanted to include the key fragment here which 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 that is needed from NubiS' point of view is the one that defines the HTML input element relating to a question on its routing. Here that is:

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

The crux lies in 'name=answer1'. When NubiS builds a question screen, it names all of the input boxes in a numeric fashion ranging e.g. from 'answer1' to 'answer5', where the numbers correspond to the order in which the questions are encountered. So if we have:

```
group.TOverall
Q1
Q2
endgroup
```

Then, on the outputted screen Q1 is demarcated by 'answer1' and Q2 by 'answer2'. When we then click 'Next', NubiS will take the values of 'answer1' and 'answer2' and store them for Q1 and Q2 respectively. As such, in order for NubiS to be able to link the answer to our 'custom answer type' question, the code that creates the custom input mechanism must define this linkage. The remainder of the code in a function for a custom answer type can be anything as long as it defines a proper input mechanism that is returned as a PHP string.

A last manner in which we can alter the display of questions revolves around the notion of display templates. Display templates are defined in the 'root NubiS/display/templates' folder and 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 also included as examples, e.g. 'displayquestion_1.php' which is used in the Understanding America Study or 'displayquestion_2.php' used in the M-Teens data collection project. For example, the function that creates the main body of the screen can be overridden to incorporate things like displaying text from right to left instead of the usual left to right.

18.4 Parallel sections

Another advanced functionality that may be required occasionally has to do with so-called parallel sections. The main idea behind such sections is as follows: suppose that we have a survey in which we want to ask questions about a variety of topics in households. Moreover, we want to ask different topics to different members of that household. For example, we might want to ask all financial questions or all family related questions to the most knowledgeable person. These might not always be the same people.

One option is to simply develop the survey as we did now and ask one section after another, and switch between people when needed. This assumes though that all relevant people are present and moreover that they are available to answer our questions in the order we 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 case, if our routing is such that we first ask financial questions and then family questions, we would not be able to do any part of the interview. If however we could somehow jump to the family questions and return to the financial questions later, we could do part already.

This is what parallel sections are for. They are parallel in the sense that they are considered to be independent from one another with no dependencies between them. If they did have such dependencies, then we could not switch their order since 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 site would be similar in make up to the example test page we discussed earlier, but would specify which section we want to start. An example is provided in ‘root NubiS/parallel.php’, which provides a rudimentary illustration.

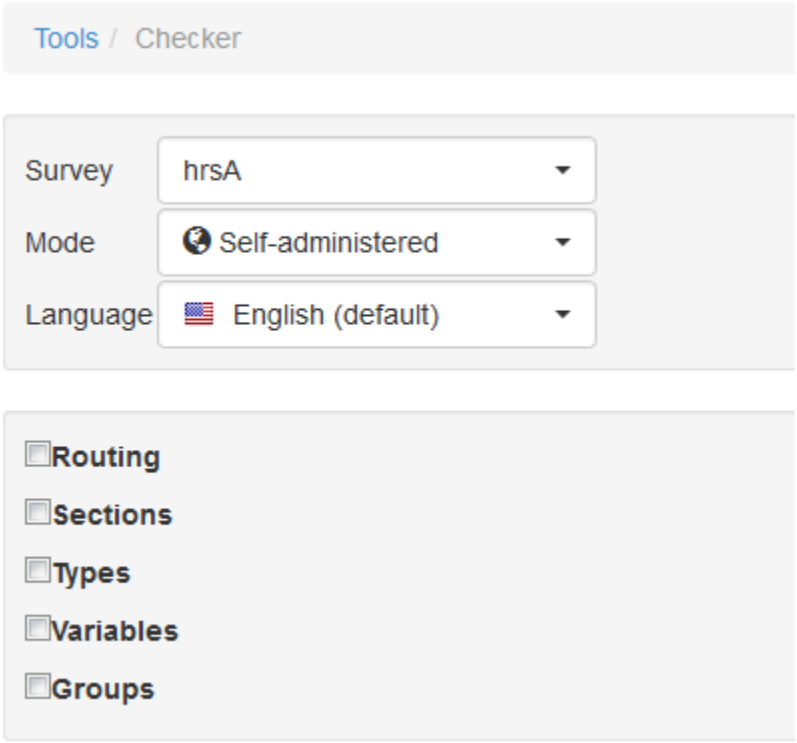
Note: obviously in a real setting we would want to expand this to have different buttons available to launch different sections, build in any dependencies between sections, and so on.

19. Preparing for fielding/sample management

The previous sections have focused almost exclusively at how to develop a survey in NubiS. Occassionaly we mentioned how we could use the tester tool to see how things that we programmed would look like in the survey. Here we want to shift gears and start talking about what actions to undertake in order to move towards actually collecting data, i.e. fielding the survey.

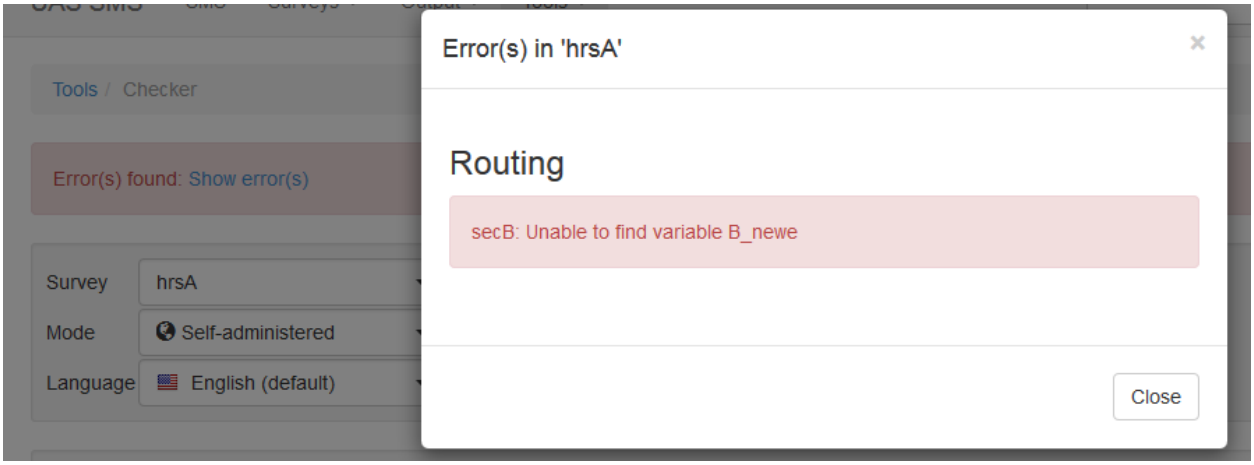
19.1 Checking and testing the survey

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



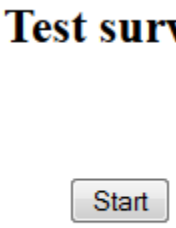
The mechanism operates on a per survey level. After choosing the survey (in case there are multiple surveys in NubiS) we can choose the interview mode and language we want to check. We then select which components we want to check. If there is an error found in any of the selected components, NubiS will provide a message to locate it. For example, suppose that in

the routing of ‘secB’ we had referred to ‘B_new’ as ‘B_newe’ instead. Then on running the checker we would see:



The second step is to run through the survey to ensure that everything works properly from a ‘semantic’ point of view; where we use ‘semantic’ here to mean that the survey displays the intended behavior. One part of that is the programmer him/herself testing the different scenarios, skip patterns, and so on. A second part is to ask other people (for example the researcher(s) involved) to test the survey.

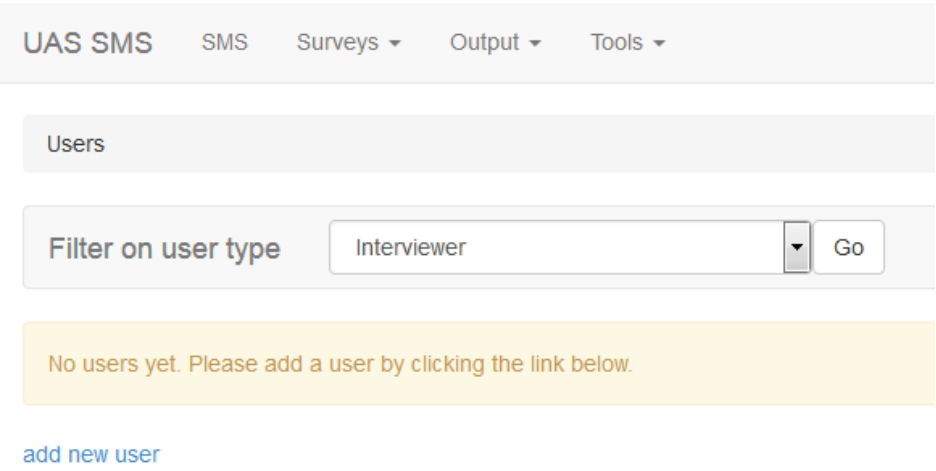
To accommodate for such testing NubiS provides a basic and more advanced mechanism. The basic mechanism utilises a stand-alone test page to which testers can be directed to test the survey. A simple example of such a page is standard included with NubiS and is located in the ‘root NubiS/test’ folder.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



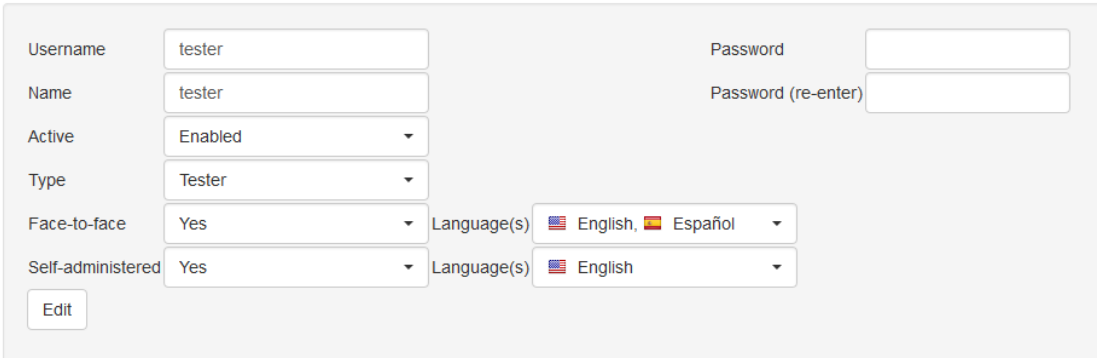
We can of course modify this to include more text/logos and so on, or make a new test page based on the standard one.

Another option is to create a so-called tester account. This is a NubiS account (just like the system administrator one), but whose access is restricted to testing a survey and reporting any found problems. To set up such an account, we access NubiS’ user management:

Opening the ‘Users’ screen shows us the following:

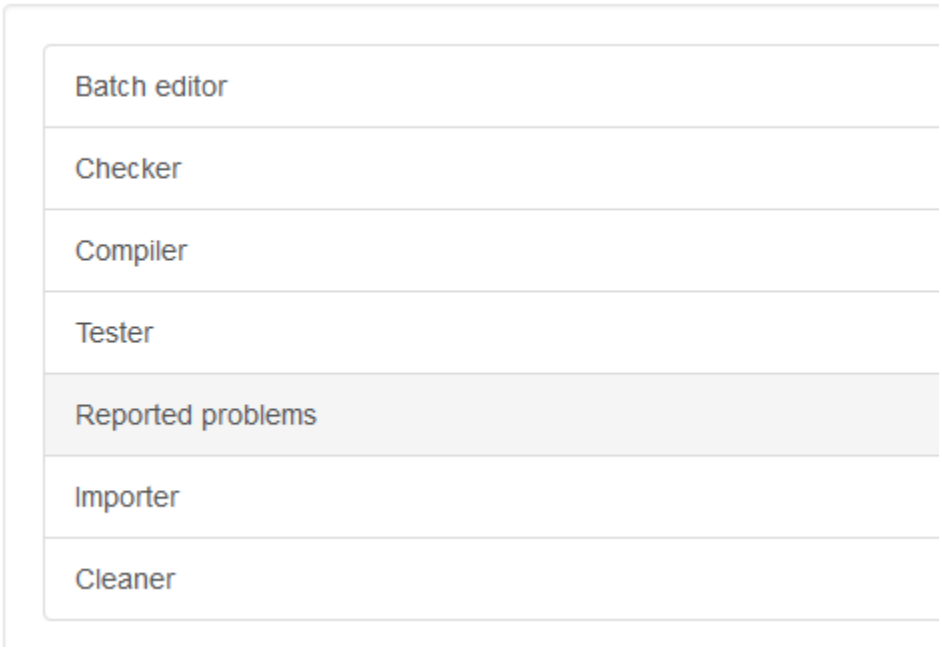


We then click ‘Add new user’:

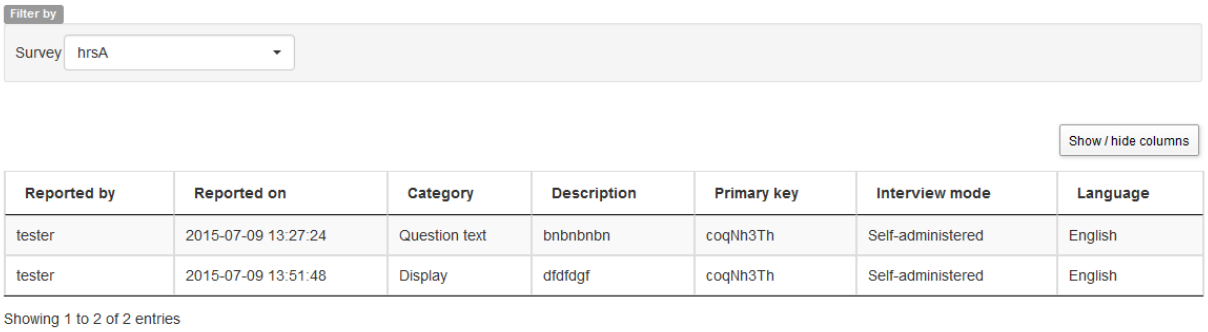


We specify a username/password, indicate that the account is active, and choose ‘Tester’ as the account type. We can also decide which variants of the survey ‘tester’ is allowed to test in terms of interview mode and language. We might for example only make face-to-face available or (like here) every mode with every language. After creating the account, we only need to share the credentials with the person who is to test and s/he can then use them to run tests and report any problems that s/he finds. For a detailed discussion about using NubiS as a tester please refer to the NubiS Tester Manual.

Here let's see where we can find any reported problems. If we click 'Tools' we see our tools overview with the option 'Reported problems':



If we choose this option, we can see an overview of the reported problems:



We can sort these (if needed) and use the information to analyze the problem and if needed, make modifications to the survey.

Another useful feature in NubiS that can facilitate testing more easily is the usage of the 'execution_mode' variable. This variable captures whether the survey is running in normal mode or test mode. Why is this useful? Well, just like we can leverage 'language' and 'mode' on the routing we can do the same with 'execution_mode'. We can then for example include test screens contingent on being in test mode:

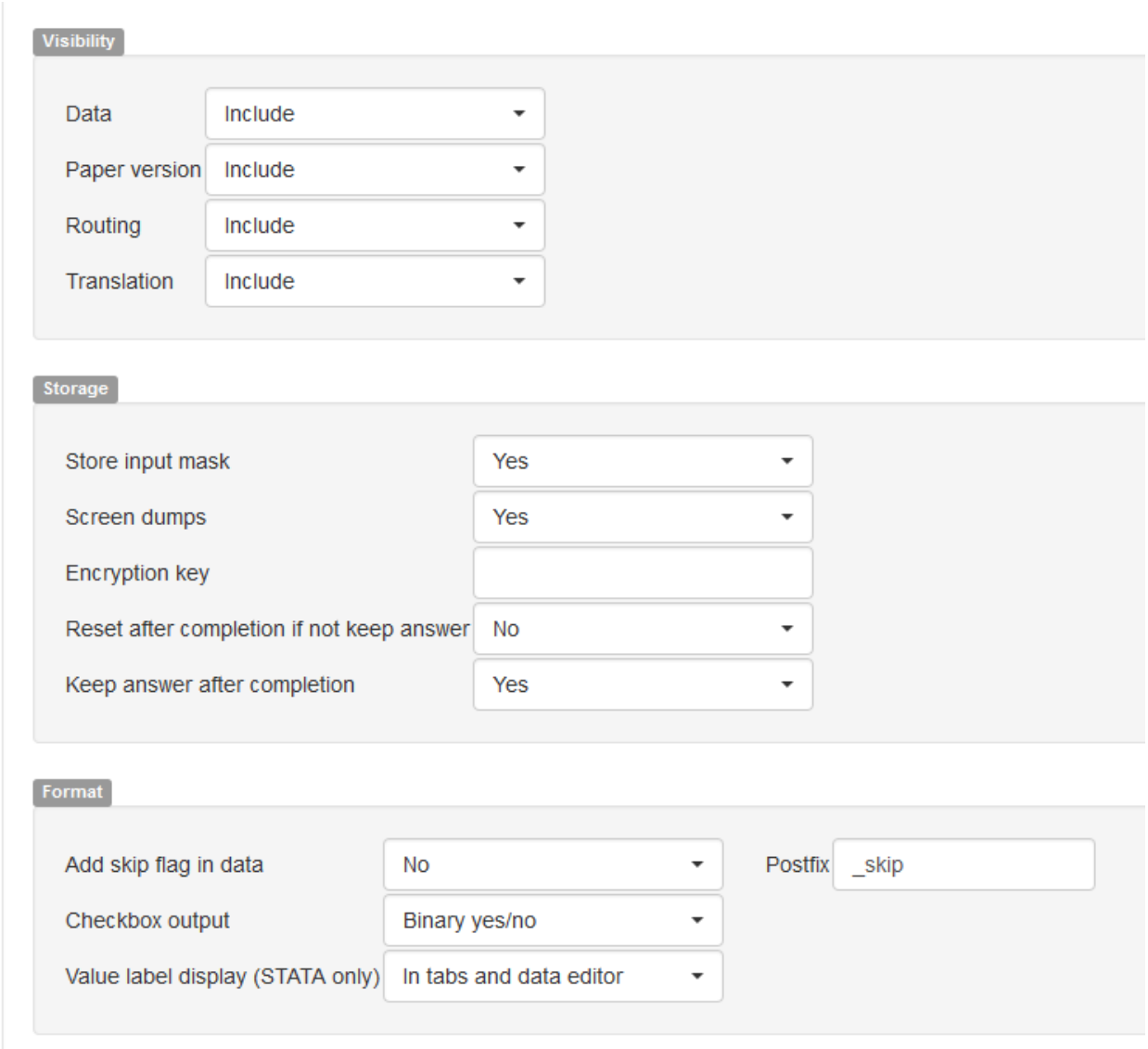
if execution_mode = TEST then
// test screen here, e.g. to pick randomization values to run through a particular scenario

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 we do not need to remind ourselves to take our test screen out once we administer it to real respondents. Moreover, should we wish to run additional tests when the survey is already in the field, we could still use the test screen(s) we had in place prior to fielding.

19.2 Output settings

Another aspect of the survey programming that should be dealt with before fielding is with regard to how NubiS is supposed to capture data, e.g. in terms of what data to collect, whether to encrypt it and so on. All such aspects are governed by NubiS' output settings. Like any other settings these can be set on a survey level as well as (partially) on a variable level. Let's first look at the survey level settings:



The output settings fall into three categories: one dealing with what to include in outputted files (visibility), one with what to capture and how (storage) and the third with how to output data. We saw an example of a visibility setting when we discussed fills, and we said that we typically set their data visibility to hidden. This has the effect that no data collected for them is outputted. The

other three visibility settings work in a similar manner, but have an effect on documentation that NubiS can create. We will revisit them in a little bit.

For now, if we look at the storage settings we see a variety of options. Perhaps foremost is the encryption key. 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 used key (e.g. in terms of length), but needless to say the key should be sufficiently strong to serve its purpose of protecting the stored data.

Three other relevant settings depict whether to store screenshots, whether to capture so-called paradata and whether to store 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 that moves the survey back or next. Such screenshots can be useful to see exactly what a respondent saw in case of a problem or to provide to a researcher for usage in a paper or presentation.

Paradata is data collected by Nubis in addition to the answers given and captures items such 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 get a better insight into how a respondent answered a question rather than what s/he actually answered. Note: depending on the amount of respondent activity on a screen, the size of the collected paradata can be smaller or larger. It may be that the post limit size of the server needs to be increased in order to allow for proper processing of all paradata. Alternatively, one may want to consider to only collect paradata on certain screens if it turns out that collecting this on each and every question screen has an adverse effect of the responsiveness of the server (this can be accomplished by setting the ‘Capture paradata’ option to ‘Yes’ for specific variables and/or groups).

The input mask setting is a bit less intuitive, but has to do with the following: suppose we added an integer variable and applied the ‘US currency’ to it. In the survey this would have the effect that once the respondent enters a number greater than 999, a ‘,’ is automatically added e.g. to show ‘1,215’ instead of ‘1215’. The question is whether in the data that is stored the ‘,’ should be included or not. Including it means that for example in an outputted Stata file the data is represented by a string format variable, whereas excluding it means it is included as a byte format variable; which in turn has consequences for how it can be used in analysis.

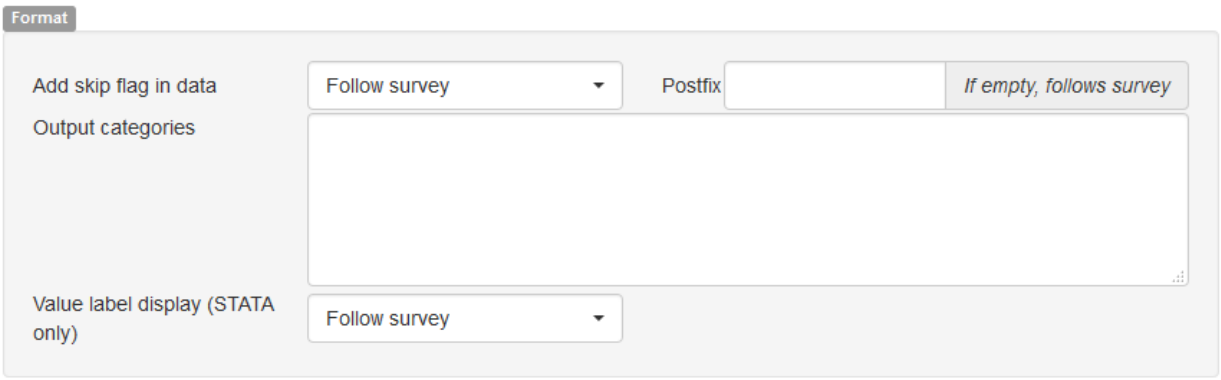
Typically less used are the ‘Reset after completion if not keep answer’ and the ‘Keep answer after completion’ settings. These settings are typically only of use in very specific circumstances. Starting with ‘Keep answer after completion’, this setting determines whether the answer to a question must be kept as the survey is completed OR whether it should be deleted. The second setting, ‘Reset after completion if not keep answer’, then enables/disables this behavior. Now, why would we want to throw away data?

Well, one possible use case is a survey which you intend to ask on multiple occasions to the respondent, e.g. to update their basic background information every quarter. In such a survey

we may use a question like ‘Did anyone else move in since last time we spoke?’. Perhaps the respondent said ‘Yes’ in the first quarter. But now when we ask the respondent to answer the same question in the second quarter, we would not want to show the ‘Yes’ answer from the last time. For such ‘auxiliary’ variables we won’t want to keep the answer, but instead reset them once they are no longer needed. By default all answers are kept for all variables (for obvious reasons, since otherwise we would have respondents answering but end up with no data), but the setting can be specified per variable to override this standard behavior.

The last set of settings control how to output data. One is whether we want NubiS to add so-called skip variables. By default when NubiS outputs data, it marks questions that were skipped by a respondent (e.g. by not satisfying a condition) nor not yet seen (in case of an incomplete interview) as ‘.’, whereas questions that were left empty are marked with ‘.e’. By setting the skip variable parameter to ‘Yes’, we tell NubiS to create special skip variables instead (by default named ‘variable name + _skip’) setting them to ‘1’ for when a respondent skipped that question. Similarly, we can alter NubiS behavior when outputting checkbox variables. By default NubiS creates a binary variable for each answer option, setting it to ‘Yes’ if it was selected and ‘No’ if it was not. We could change this so that NubiS instead creates variables for those options that take as their value the answer option code if selected and no value if not selected.

A third parameter that can be configured deals with radio button, checkbox, dropdown and multi-dropdown variables. NubiS can export those to Stata to only show up in tabulations or also in the data editor. For CSV export this means that in Excel ‘Only in tabs’ results in seeing only the numerical answer codes, whereas the default of also showing in the data editor leads to seeing both the code and value label. Finally, there is a type/variable level setting that can be leveraged for the aforementioned four types of variables, being 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 we would like them to be short. On the ‘Output’ tab of a variable we can define labels that are shorter in the ‘Format’ part:

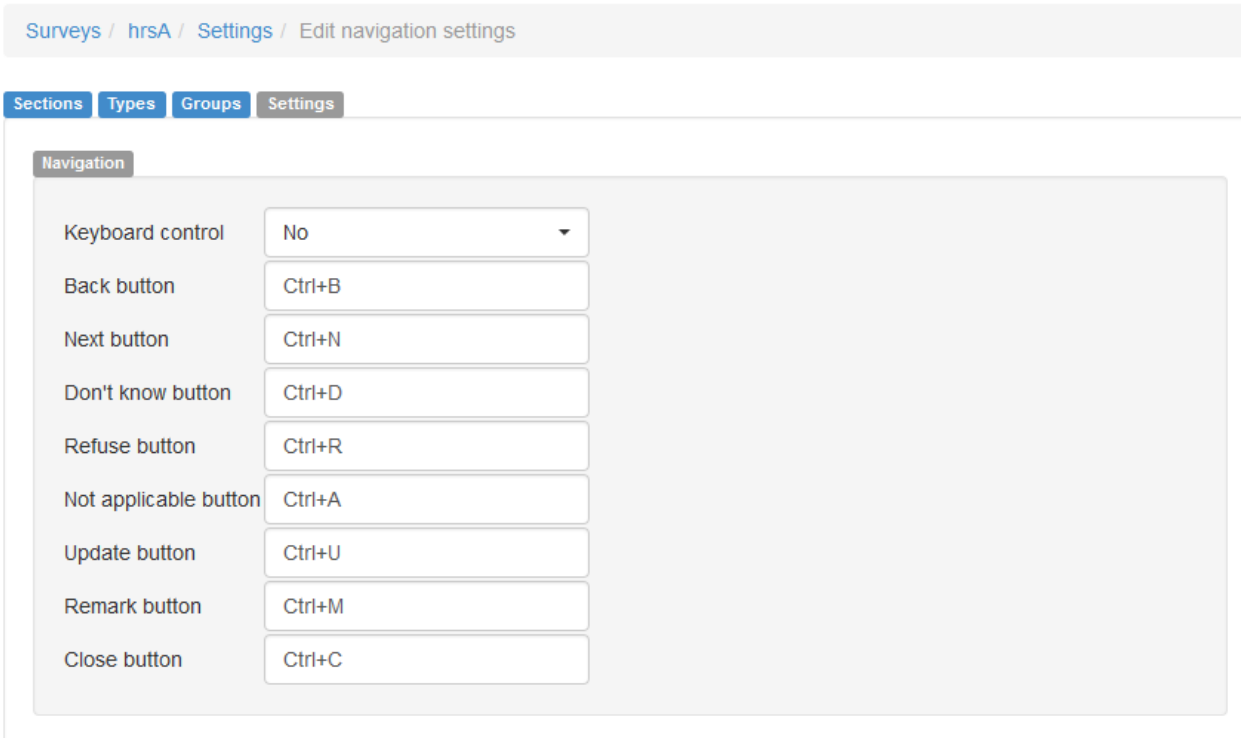


Please note that this setting is not intended for re-coding variables. That is, if 1 meant ‘Yes, I have other people living with me’, then we can define here that the outputted label should be ‘OTHER PEOPLE’ for that answer option. We can not recode ‘1’ to some other value however.

19.3 Configuring the respondent navigation experience

There is another aspect that can be configured with regard to preparing our survey for fielding and that is how the respondent interacts with the survey. Usually such interaction takes place by using 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 such navigational interaction might be too cumbersome. Particularly, in a face to face setting with an interviewer present it can be more expedient to use keyboard based shortcuts instead (or a mouse might simply not be a workable solution if fieldwork is taking place in a remote area).

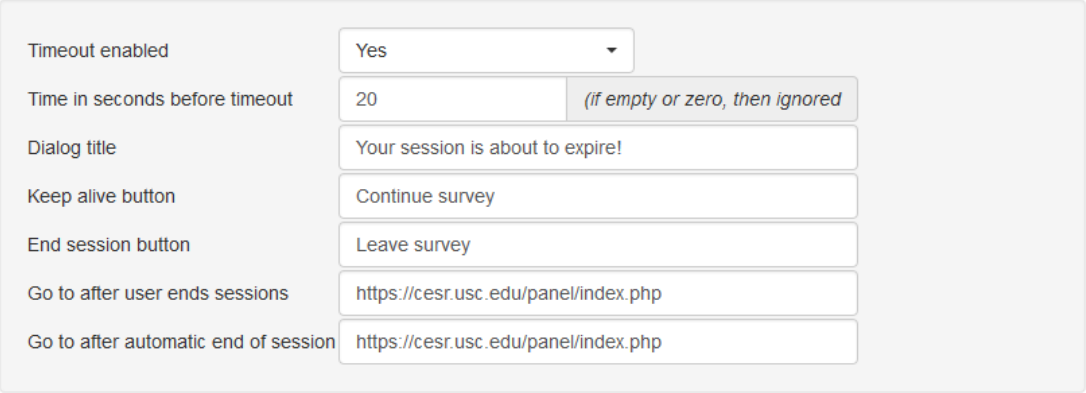
To enable keyboard based interaction we can activate it in the 'Navigation' settings. We saw some of those before when we discussed the "Don't know", "Refuse" and "Not applicable". There is a second set of settings on the same screen:



The above settings define keyboard shortcuts that can be used instead of clicking certain buttons. It should be noted that most browsers enforce their own keyboard shortcuts and do not allow those to be overridden. Specifically, only FireFox allows one 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.

Related, but different, is the third set of settings under 'Navigation' settings. They pertain to whether or not NubiS should end a survey session after a specified amount of time. To activate

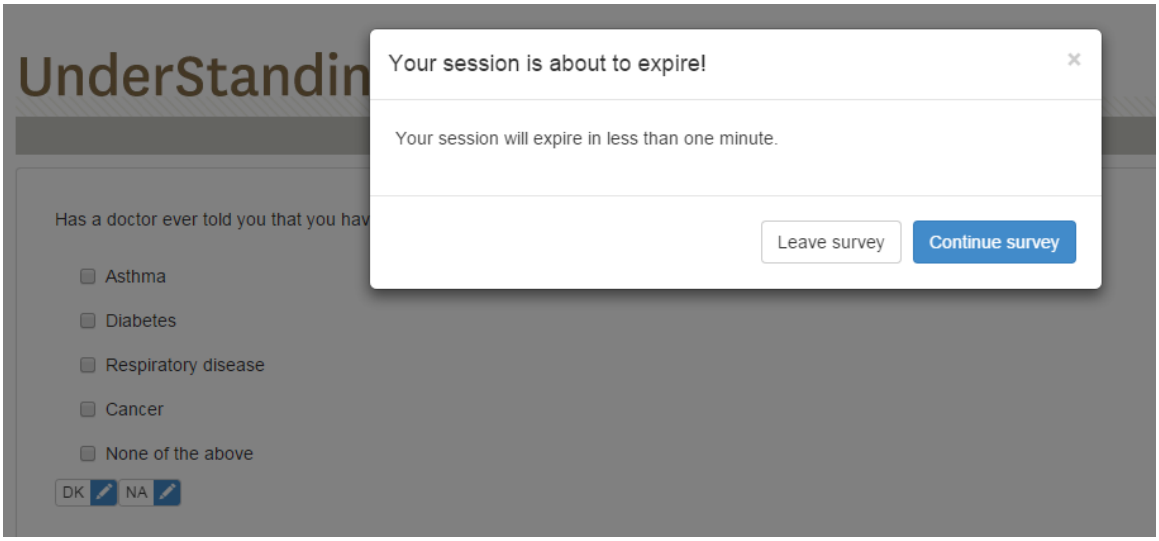
this, we can select ‘Yes’ for ‘Timeout enabled’. We will then see an additional collection of settings to specify how NubiS should enforce it:



As the above shows we can define several parameters:

- 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 we activate this with a timeout period of 20 seconds, then if we open the survey and do not do anything for 20 seconds, the following popup will appear:

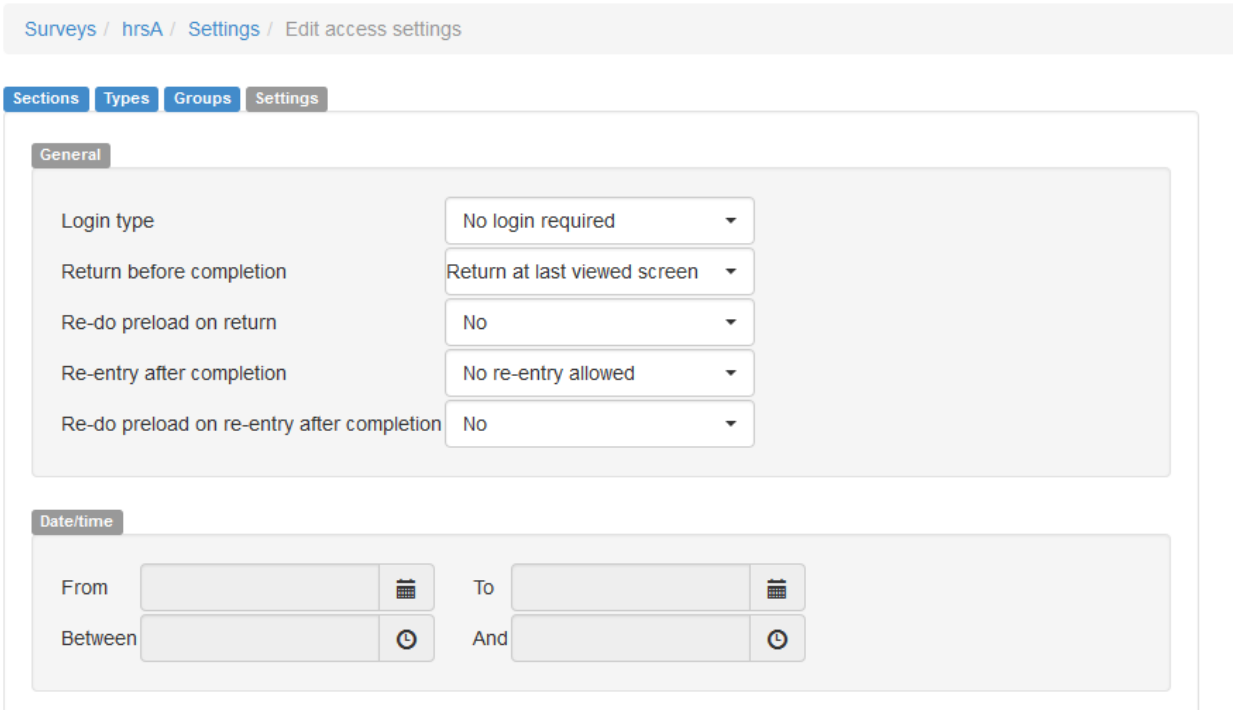


Not doing anything in this regard means that no activity was detected by NubiS in terms of typing in something, selecting an answer, moving the mouse, and so on.

By default NubiS will show the above warning when 60% of the specified timeout period has passed. This can be modified if desired in the 'root NubiS/config.php' file by modifying the 'sessionExpiredWarnPoint' function. Similarly, we could also adjust there how frequently NubiS should check for activity (every 5 seconds) by changing the 'sessionExpiredPingInterval' function.

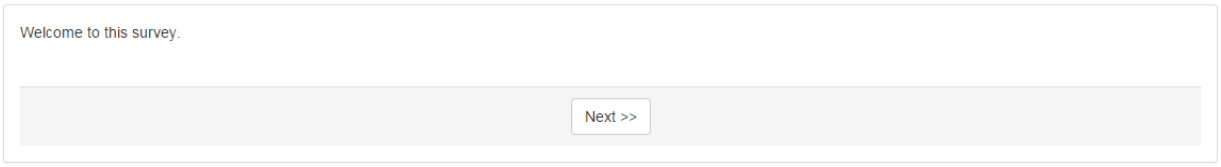
19.4 Setting up survey access

Another important part of preparing the survey for fielding is determining how respondents will be able to access it. The settings that govern this can be found under the 'Access' settings. The following are available:



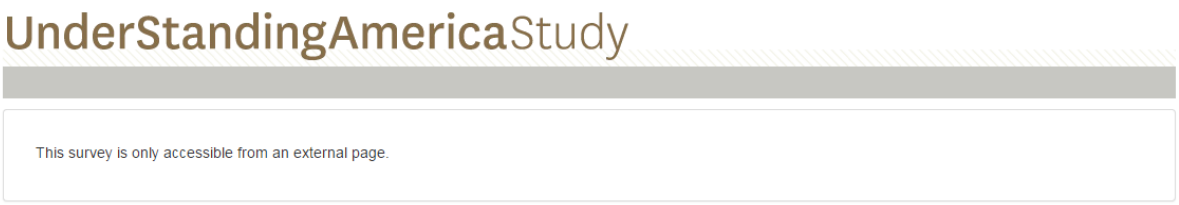
First of is the login type. Three options are available:

- No login: if no login is required, then anyone who opens the web site can access it. This is done by simply navigating to 'root nubis/index.php'. This will open the default starting page of the survey:



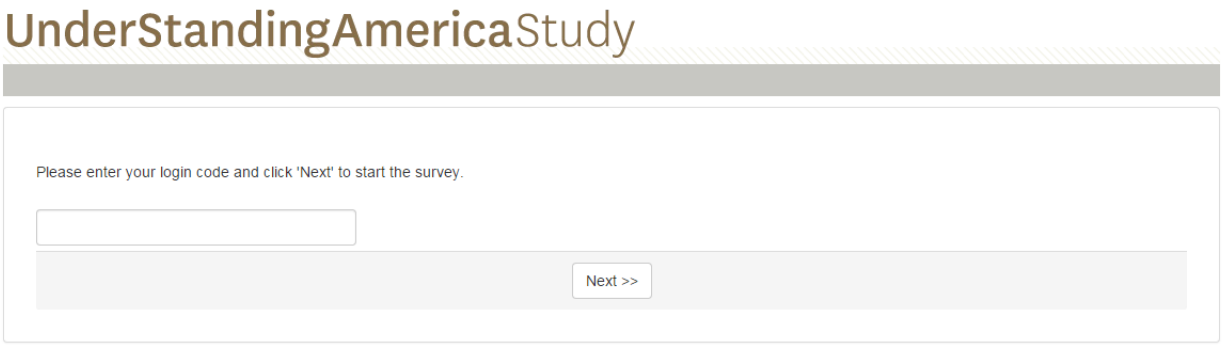
The text displayed on this screen can be customized by changing the question text in the variable ‘Introduction’ in the ‘Base’ section.

- Via external web site only: in this case respondents can only 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: in this case respondents will be able to log in to NubiS with 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 can 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 we will discuss shortly.

Before we do so let’s discuss the other access settings starting with ‘Return before completion’. The default is to allow respondents to re-enter a survey so they can continue. Typically this is on the screen at which they stopped the survey, but it can also be set to one of five other values:

- Return at beginning of survey: upon re-entering the survey will start at the beginning again. It will have retained any answers but start from scratch. Moreover, any

assignments and so on executed prior to the display of the first question screen will be carried out again.

- Return at first screen of survey: upon re-entering the survey will start at the first screen again. It will have retained any answers but start from this screen. 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.
- Return at screen after last viewed screen of survey: upon re-entering the survey will start at the screen that comes AFTER 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. Upon re-entering the respondent should not return to this screen as there is no way forward. Instead we wish to show the next screen.
- Return 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. We saw an example of its usage prior in section 17.7 Typically this is obviously not something we would want to do for each and every question screen in our survey. Rather we would use it for specific groups.
- No re-entry allowed: the respondent cannot leave the survey and return later.

Related to the above is 'Re-do preload on return'. This setting instructs NuBis to redo any preloading that would occur at normal survey start. Such preloading can be done by passing along an encoded string to NubiS when starting the survey from an external web site. For example, we could pass along some 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'.

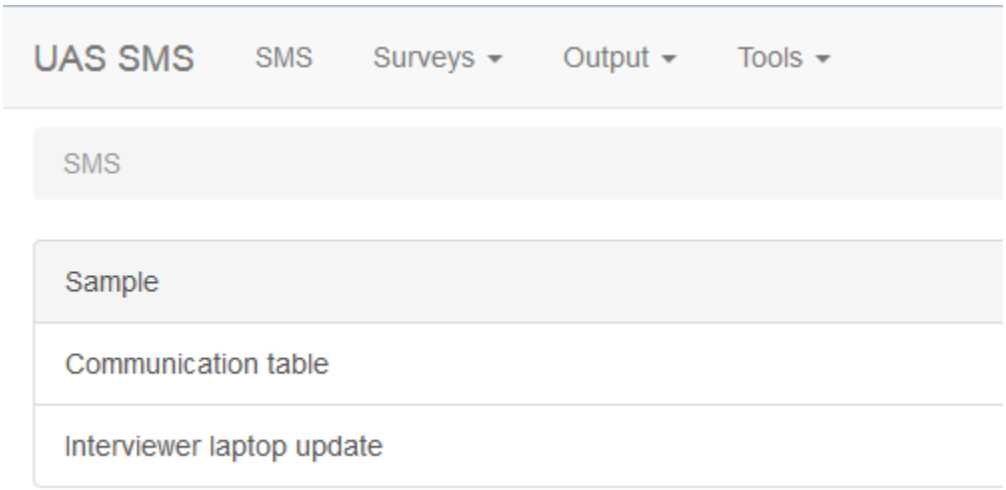
The next two access settings are in the same vein as the previous two, but deal with NubiS' behavior when a respondent returns AFTER they completed the survey. By default such re-entry is not allowed, but it can be set to the same three values as discussed above (with the small distinction that the respondent can return to the last screen they viewed, but not any after it since that would constitute the end of the survey). If re-entry is enabled, then it can be either with or without re-doing any preloads (by configuring the 'Re-do preload on re-entry after completion).

Finally, we can set when the survey can be accessible, i.e. between which begin and end date. In this context NubiS will take midnight as the begin time and end time, unless we stipulate the survey can only be done between certain hours like 9am and 5 pm. 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.

19.5 Adding a sample

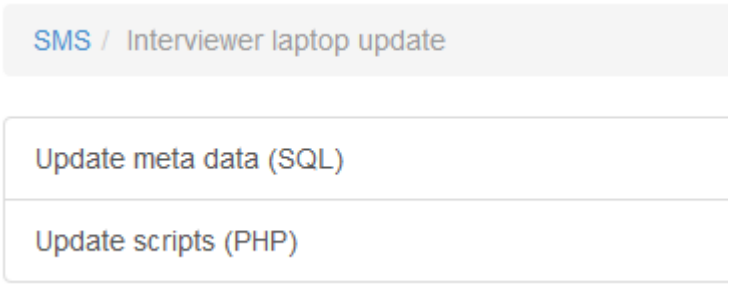
As we just saw there are various ways in which respondents can access a survey in NubiS. With the exception of the anonymous mode the other options require the presence of a sample. To this end NubiS supports uploading a sample into the system by means of pre-formatted CSV files. This functionality is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



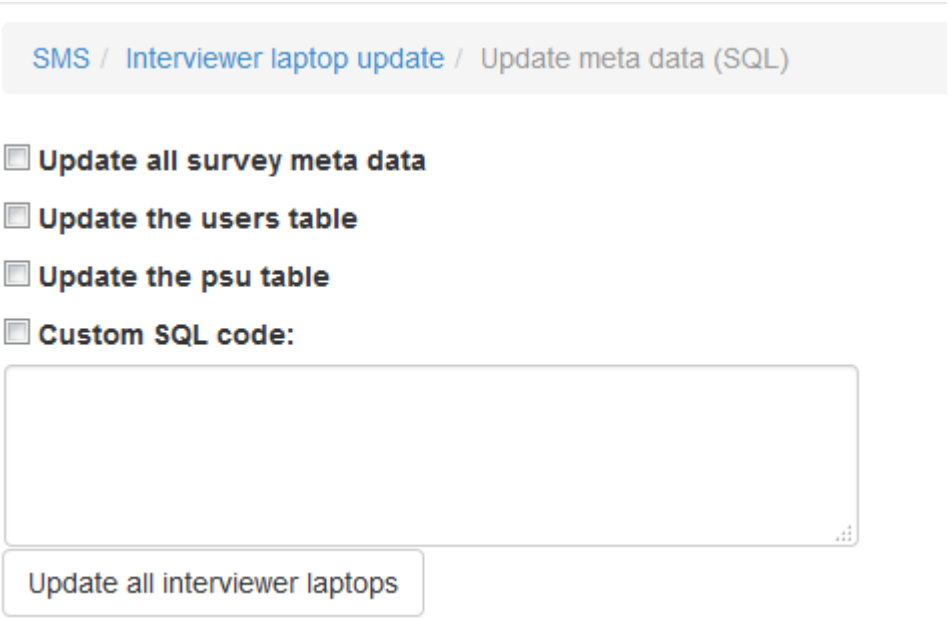
We are interested right now in the first option (the other two cater for managing fieldwork in a face-to-face interviewing setting where interviewers go into the field with a laptop to conduct interviews).

19.6 Managing fieldwork

If you are planning to use NubiS in a face-to-face interview setting, there are several utilities available to you to manage the fieldwork. These are predominantly aimed at 1) interacting with interviewer laptops (e.g. to push out updates) or 2) tracking progress. The ‘Communication table’ and ‘Interviewer laptop update’ options we just saw in the previous section fall in the first category. Starting with the update option:



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

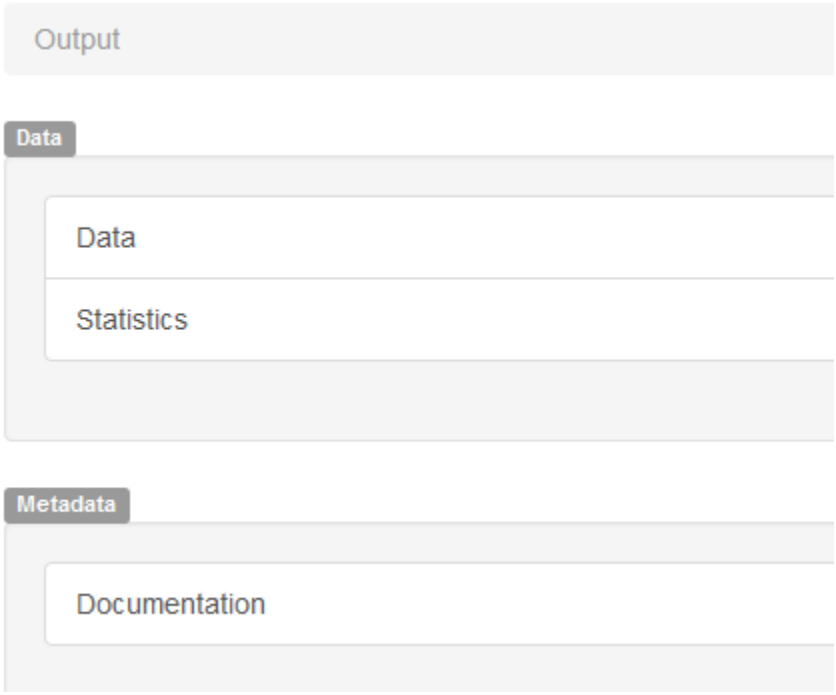


Here we find options to update the survey instrument, the users (for example if a new interviewer laptop was added), the psu table (dividing the sample into sub units), or custom SQL code (e.g. to make other minor changes like correcting erroneous data).

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

20. Exporting data

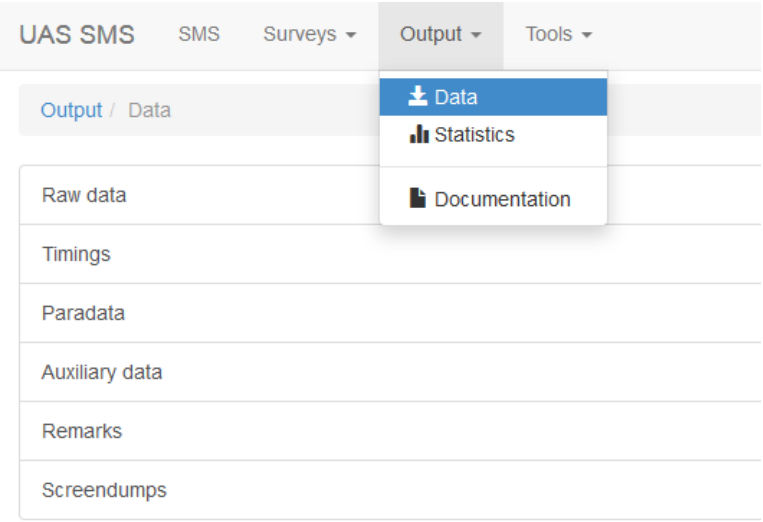
We have spent a lot of time so far on discussing how to program a survey, how to set up a sample and access, and how to prepare a survey for fielding. The whole point of all that is of course to collect data that we can then analyze. In order to support analysis NubiS provides a variety of options to generate real-time data and meta-data output related to the survey (for information on data related to sample management please refer to section 19). These are all accessible through the 'Output' menu:



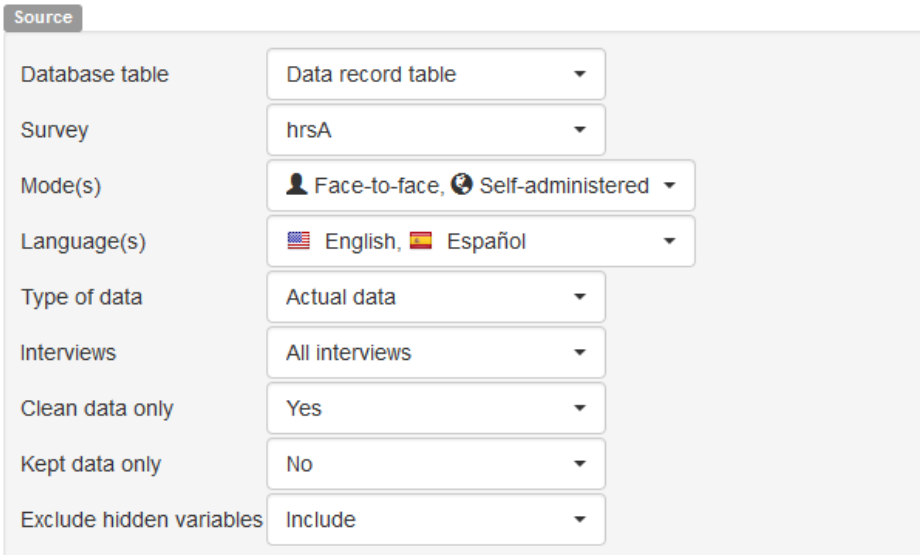
On the data side NubiS provides support to generate data files based on the raw data as well as aggregated statistics. On the meta-data side the options are related to documentation.

20.1 Data

Data output in NubiS comes in the following forms:



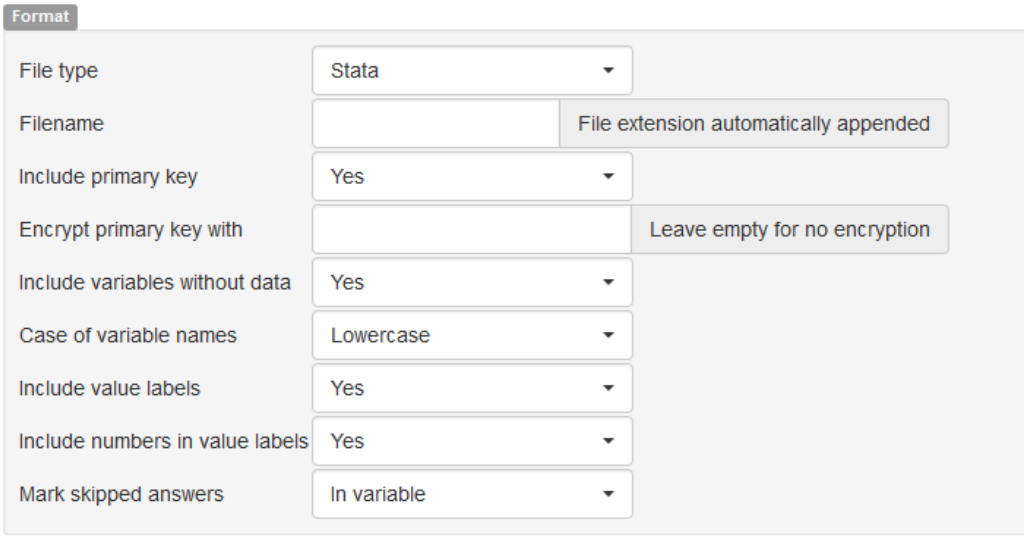
Raw data is the data directly collected through the survey in terms of answers to questions, flag variables being set, and so on. If we open up the raw data screen we see we can specify the output parameters that NubiS will follow when generating a data file:



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

Next, we choose the survey for which we want to download data, from which interview mode(s) and which language(s). By default all available modes and languages are selected. We also choose whether we want to download real or test data, and whether we want to include all interviews or only those that were completed. We can also decide if we want to have 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. If we say we want clean data, NubiS will not include such invalidated data. Lastly, we can say whether we only want to have kept data, that is, data for variables for which we indicated that we want to keep their answers upon survey completion. We can similarly limit the number of outputted variables by excluding variables we said to hide in data output.

Once we have decided on the source settings, we can specify how we want to output the data:



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

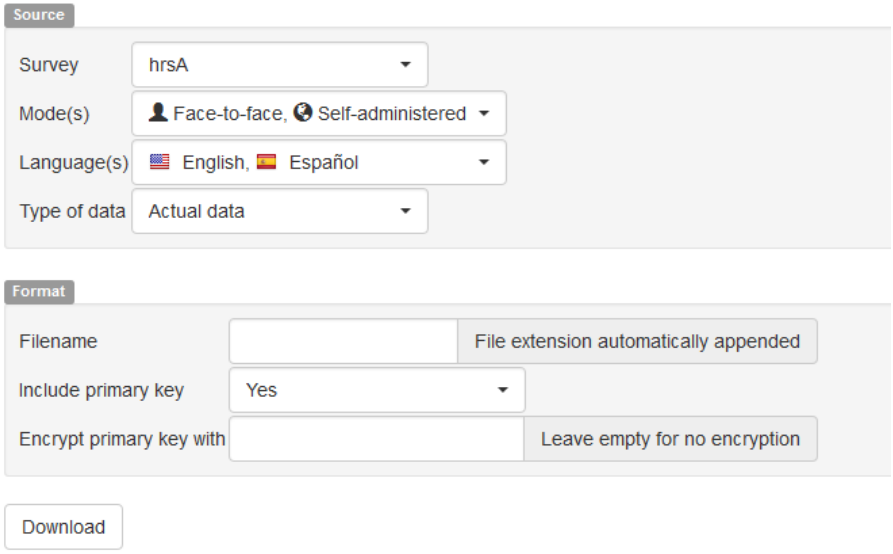
Finally, we can depict if NubiS should mark skipped answers. To recall, a respondent can leave the answer to a question empty. If we set 'Mark skipped answers' to 'No', then in the outputted data file this empty answer will be represented with '.' in Stata and an empty value in CSV. Now, suppose that a respondent never saw a question due to a skip pattern. When NubiS then outputs data for this question, it does not find a value and outputs it as '.' in Stata and an empty value in CSV. As such, there is no distinction between the two and it is not known whether the

respondent did not get the question or chose not to answer the question. This then has consequences for analysis in terms of what we can do with these empty values.

To cater for making 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 as '.e' (both in Stata and CSV). It can also be done by creating skip variables, e.g. 'Q1_skip' for 'Q1' where the skip variable is set to '1' if the corresponding question was skipped and to empty otherwise.

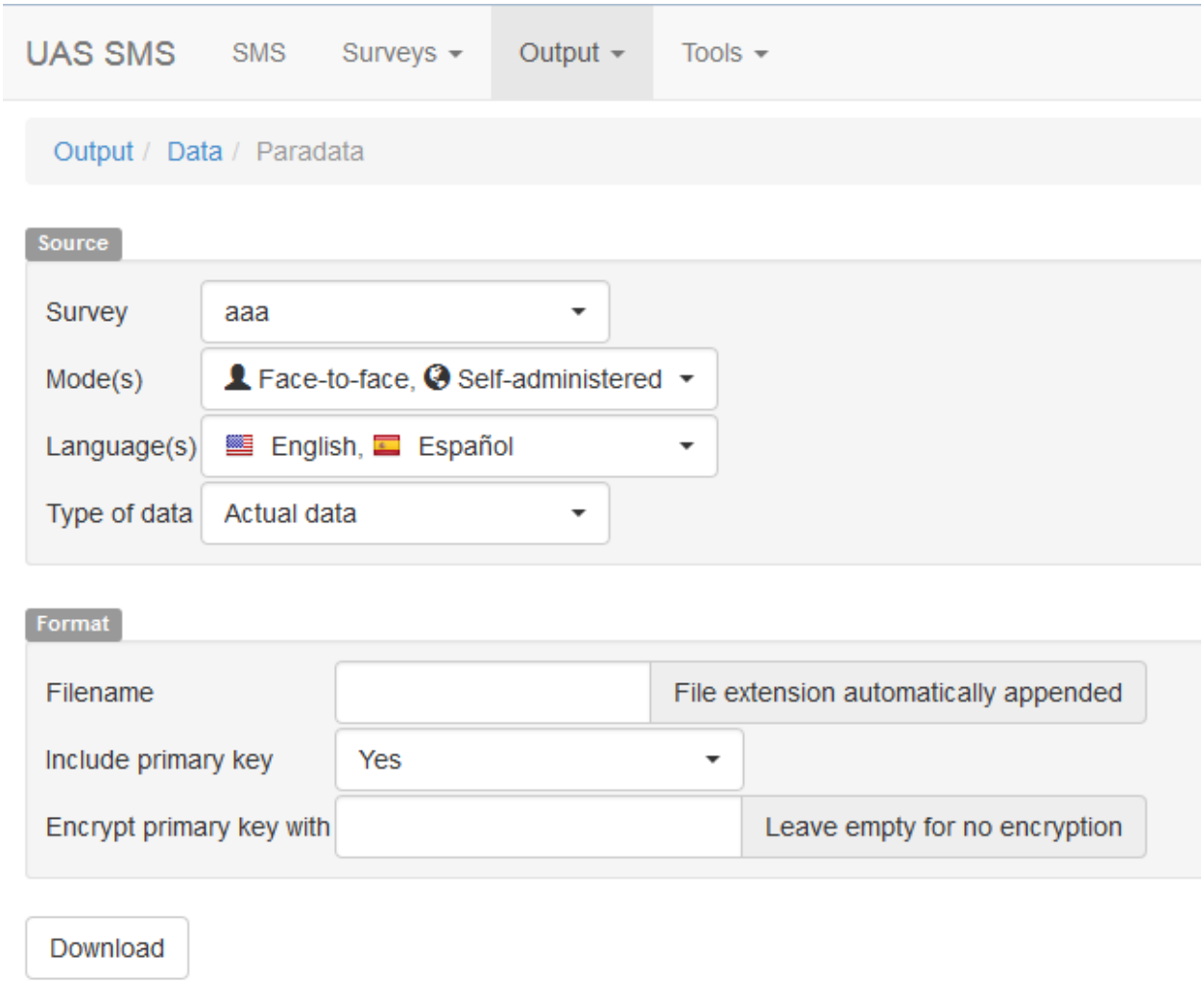
When we are happy with all the settings, we click 'Download'. NubiS will then generate the data file real-time based on the raw data as it is at that point in time. Depending on the size of the survey and the number of interviews this may take less or more time. Once done, NubiS will prompt for the generated file to be downloaded.

We can also download a file that provides us with detailed timings information:



In the main data sets we can create we already have a begin and end time for each interviews, but this might not give us 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 come back ten minutes later. The timings file that NubiS generates for us tells us how long each respondent spent on each individual question screen. Based on that it also informs us of the overall time spent by the respondent on the survey as a whole. We can then use this information for example to inform us on whether a respondent actually read the question. For example if the average time spent by all respondents is 45 seconds and someone answered the question in 3 seconds, then we might reason that s/he did not properly read the question.

Another type of data file that can be downloaded contains so-called paradata and auxiliary data (accessible by clicking 'Paradata' and 'Auxiliary data' respectively). Paradata is data captured by NubiS with regard to mouse clicks and movement, keystrokes, and tab switching. The screen for downloading paradata is as follows:



Paradata strings follow the same format, e.g:

> *||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:

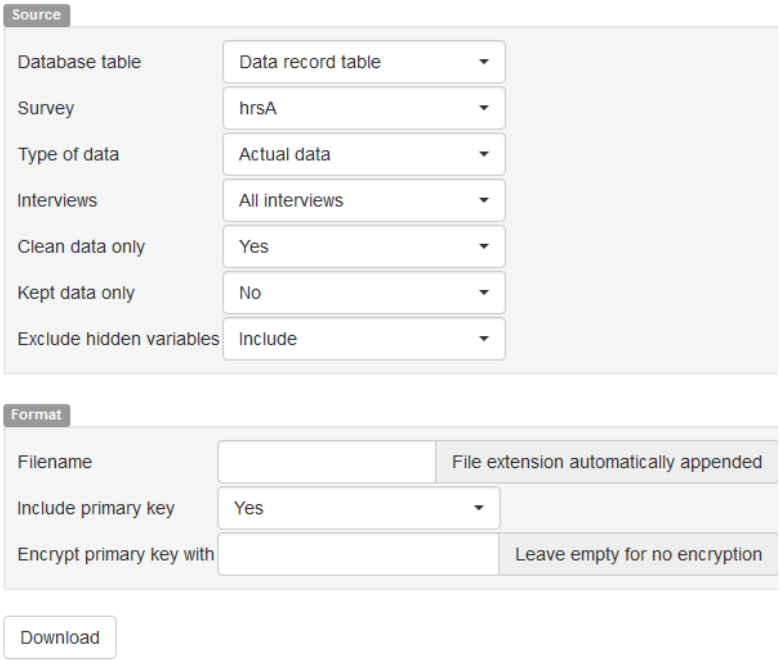
- FI: browser tab displaying the survey received focus. It comes with timestamp in milliseconds passed since midnight of December 31, 1969.
- FO: browser tab displaying the survey lost focus (e.g. the respondent opened another browser tab or another application altogether). It comes with timestamp in milliseconds passed since midnight of December 31, 1969.
- MC: mouse click on the survey browser tab. The first two numbers after 'MC' indicate the X and Y-coordinate respectively. The third number identifies the button pressed (1=left button, 2=middle button, 3=right button). The string after that identifies the component

which was clicked (if the component's HTML element had a name attribute). Lastly, it comes with timestamp in milliseconds passed since midnight of December 31, 1969.

- MM: mouse click on the survey browser tab. The first two numbers after 'MM' indicate the end X and Y-coordinate respectively. The string after that identifies the component which was clicked (if the component's HTML element had a name attribute). Lastly, it comes with a timestamp in milliseconds passed since midnight of December 31, 1969.
- ER: error recorded for a particular input box. Each error action comes with a timestamp in milliseconds passed since midnight of December 31, 1969 and the component's name. 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 are currently 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 an unique answer.
    - ER49: not all questions in the group have the same answer.

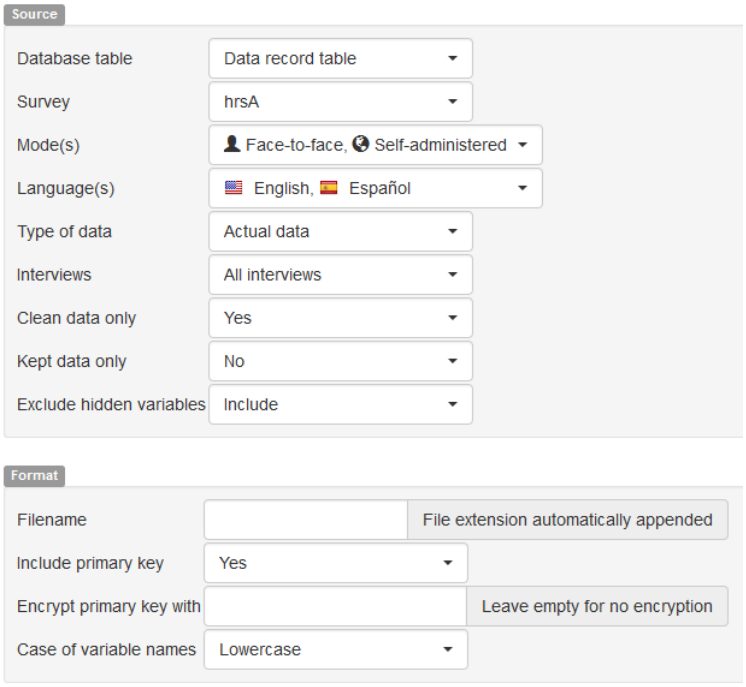
- ER50: invalid answer option code entered in enumerated text box.
- ER51: invalid answer option code entered in set of enumerated text box.
- ER52: question answer not equal to specified value.
- ER53: question answer equal to specified value
- ER54: question answer smaller than specified value.
- ER55: 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 the ones we have seen so far:



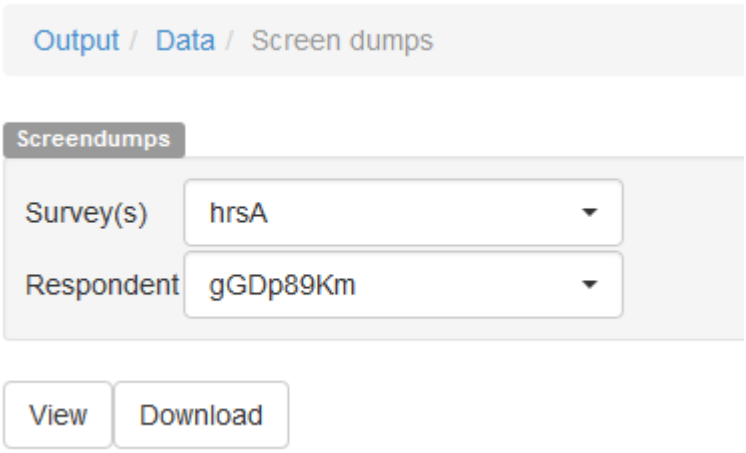
Like for raw data files, we can define the conditions for the source data to use as well as the output conditions. NubiS will generate a CSV file that we can download, which informs us per variable per respondent in what interview mode and language it was collected.

A different form of data is remarks provided by respondents:

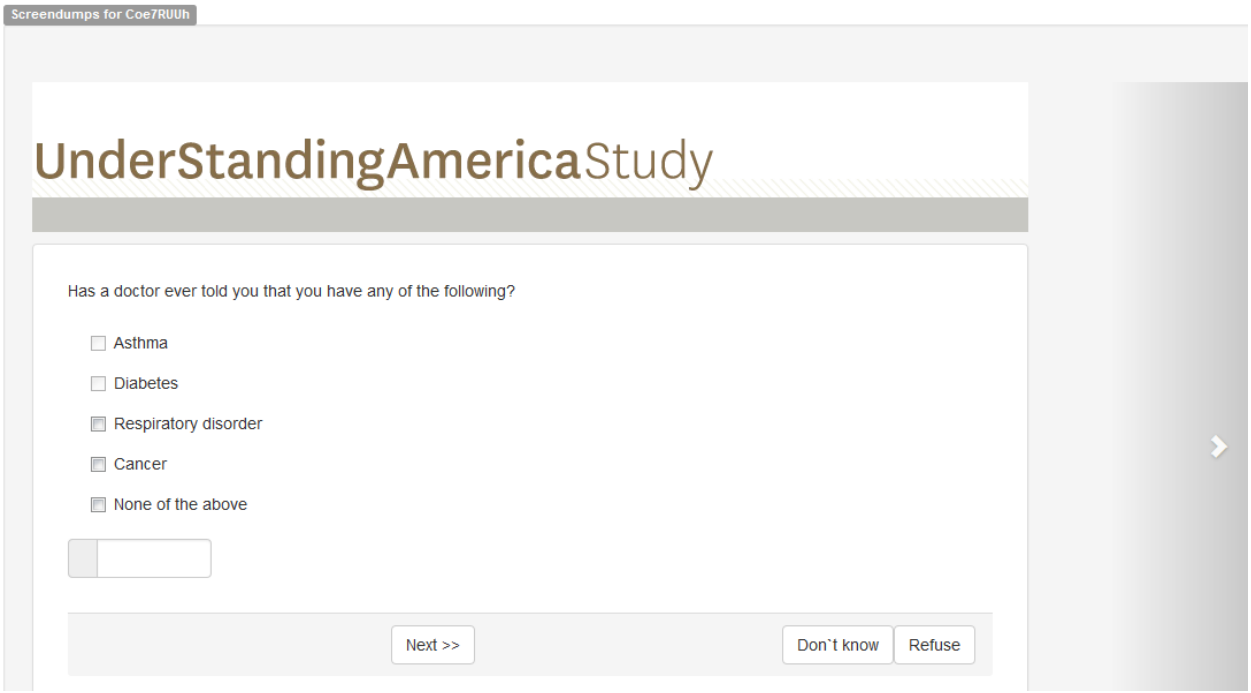


These are comments that were provided through the Remark mechanism we discussed in Section 18.5. We can choose, as before, the interview modes, languages, and so on.

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



We can choose to view the screenshots for an interview or download them in one single HTML file. If we select a case and choose view, we see for example:



Beyond these standard options in NubiS to extract data, it is always possible of course as well to pull data directly from the database. The following tables in the database for NubiS 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 language.
    - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched mode.
    - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched version.
    - 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 language.
    - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched mode.
    - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched version.
    - 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.

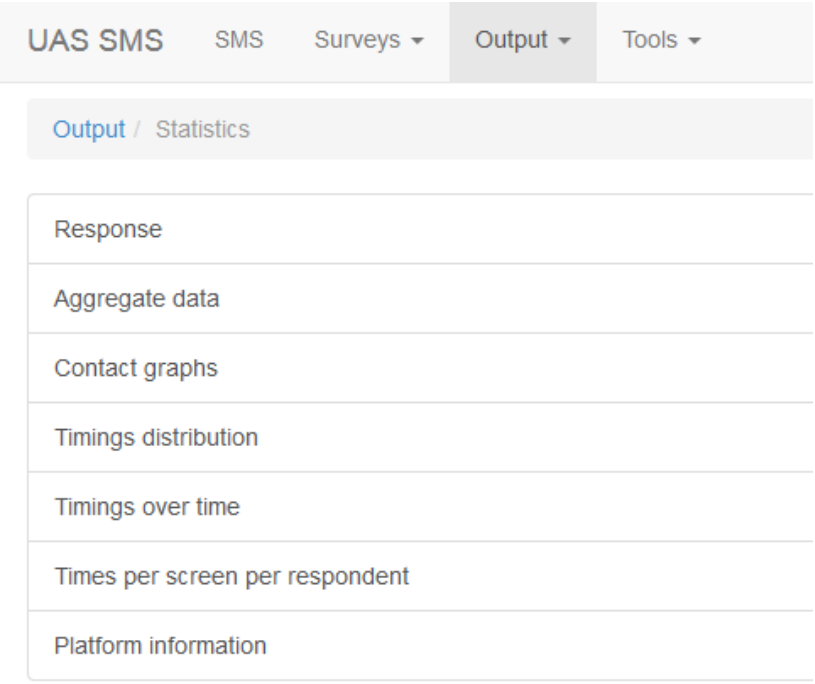
    - language: indicates the language of the survey for the stored value. May differ within the same interview if the respondent switched language.
    - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched mode.
    - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched version.
    - 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 language.
    - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched mode.
    - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched version.
    - 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.
    - stateid: 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 language.
    - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched mode.
    - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched version.
    - 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 language.
    - mode: indicates the interview mode of the survey for the stored value. May differ within the same interview if the respondent switched mode.
    - version: indicates the version of the survey for the stored value. May differ within the same interview if the respondent switched version.
    - 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' a 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? TODO
        - 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.

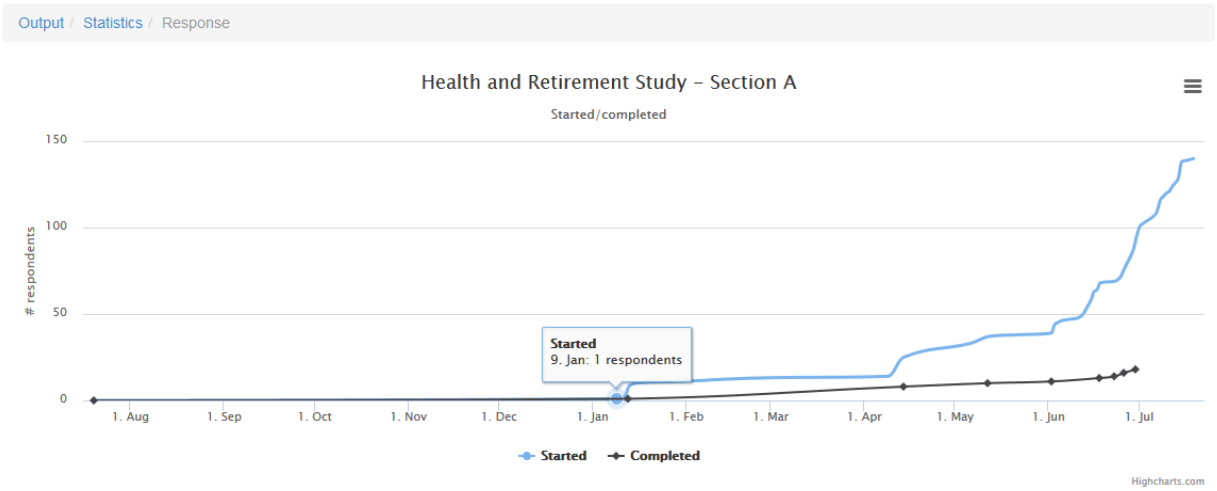
- action: identifies the specific action. In the case of survey actions this identifies the routing number to which the action corresponds.
- params: provides a serialized representation of the $_POST array as submitted with the action.
- language: indicates the language of the survey for the stored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

20.2 Statistics

We can also ask NubiS to provide us with aggregated data to get a quick sense of the data. Located under ‘Statistics’ we have the following options:

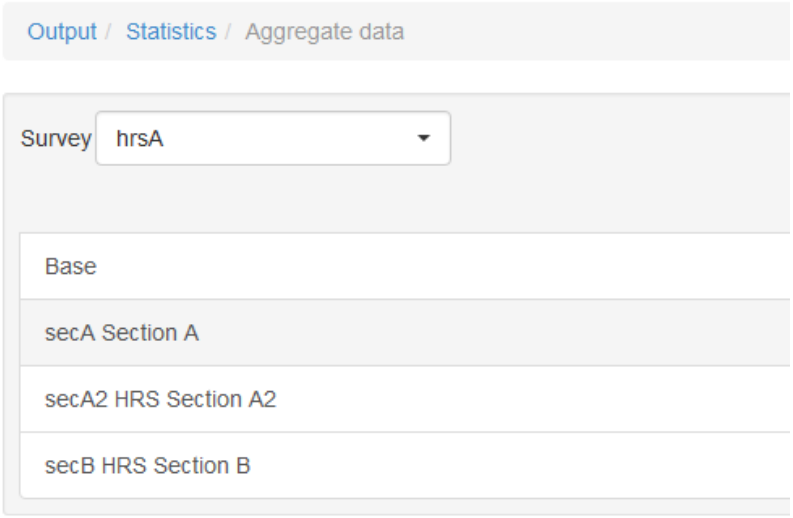


Under ‘Response’ we find the response rate for the survey:



Similarly, in ‘Contact graphs’ we can see how many contacts were made during telephone interviewing.

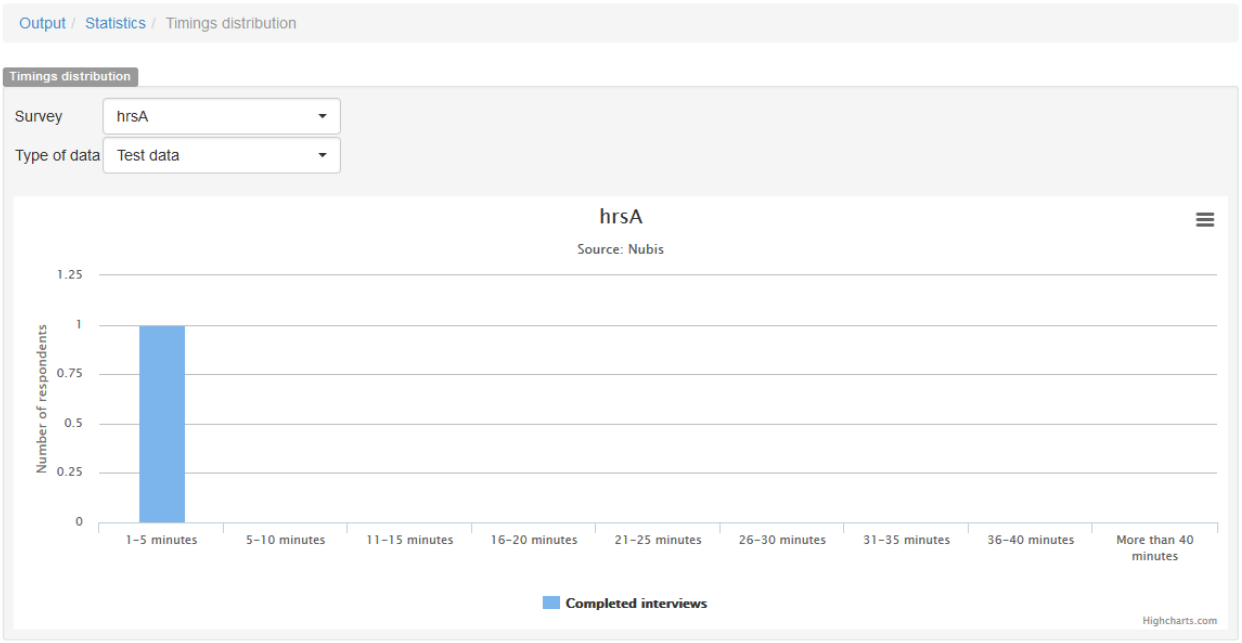
We can also get quick aggregate statistics for variables in our survey by navigating to a specific question:



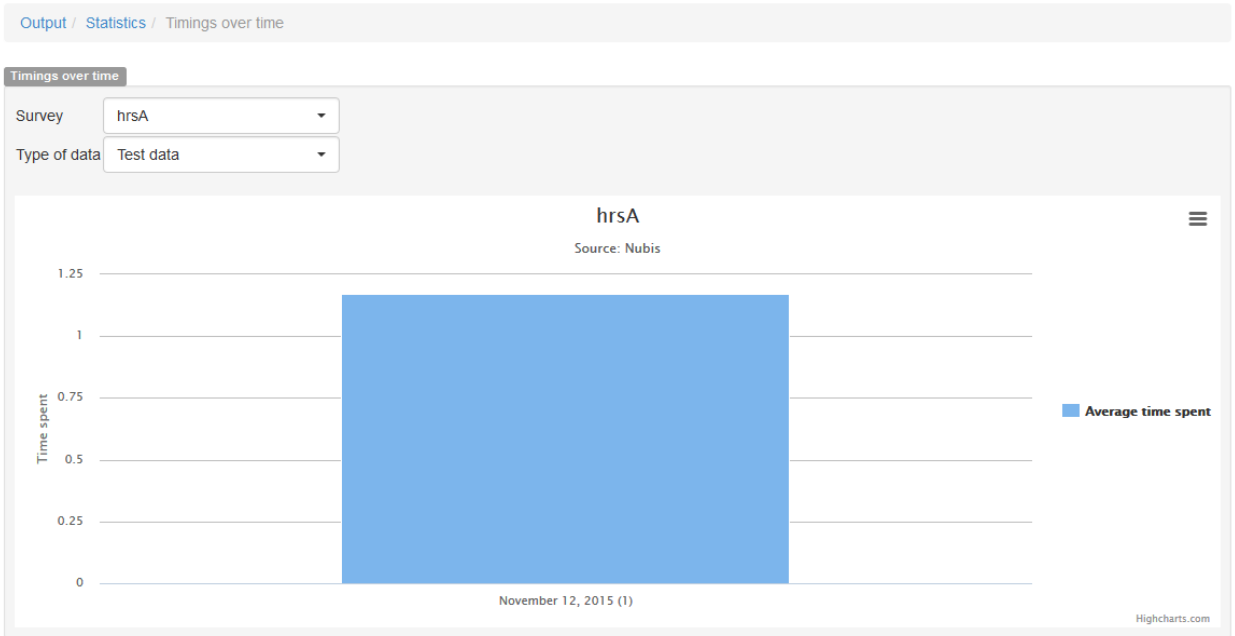
Selecting ‘secB’ and ‘B_new’ then would get us this for example (depending on the data present of course):

Though such graphs don’t replace detailed analysis of course, they can give 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 specific instance NubiS should show aggregated data.

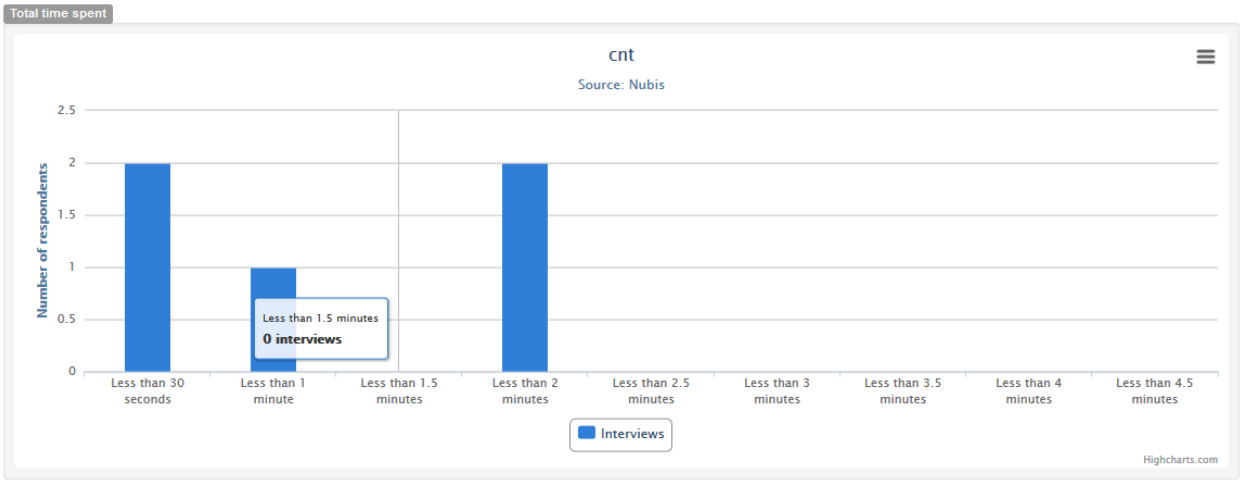
Another available option is to look at the time spent by respondents. The overall timings distribution can be accessed under ‘Timings distribution’ (note: only completed interviews are included in order to have a reliable time estimate):



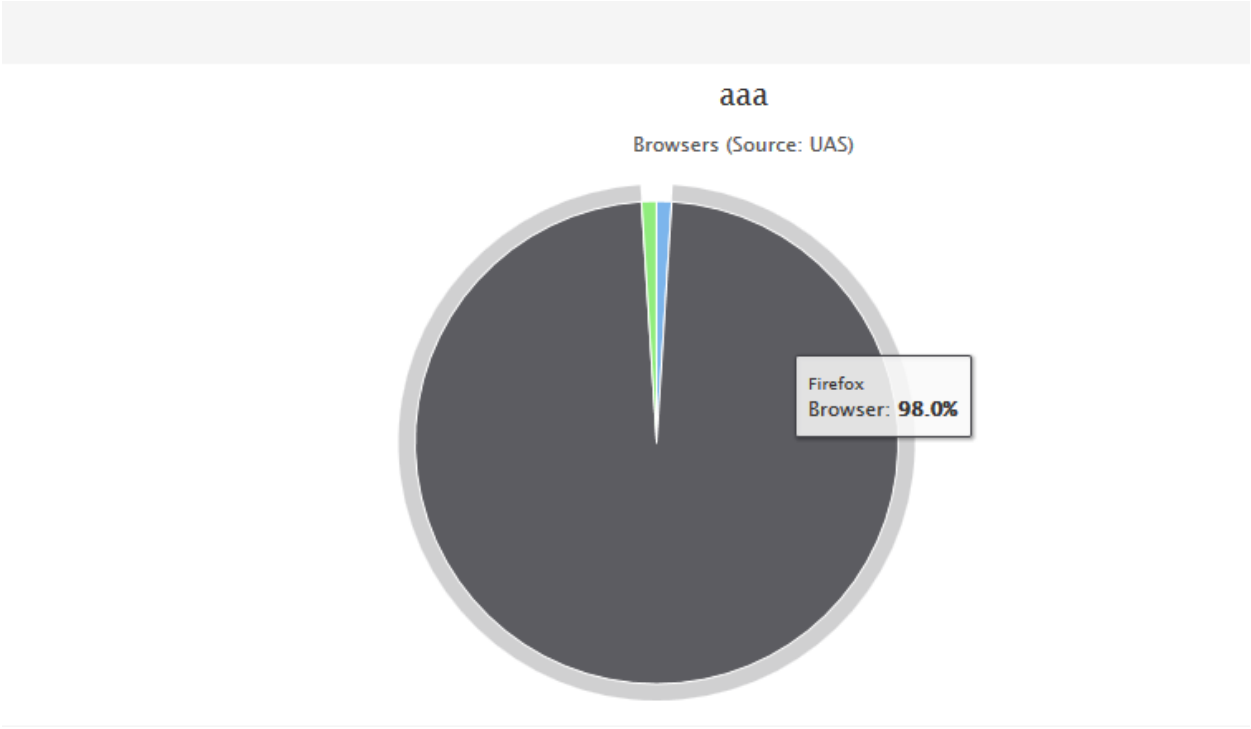
A different perspective on the same is to look at the timings over time. This shows the average time spent by respondents per day. This allows to see for example if the time spent goes down as the fielding period of the survey increases:



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

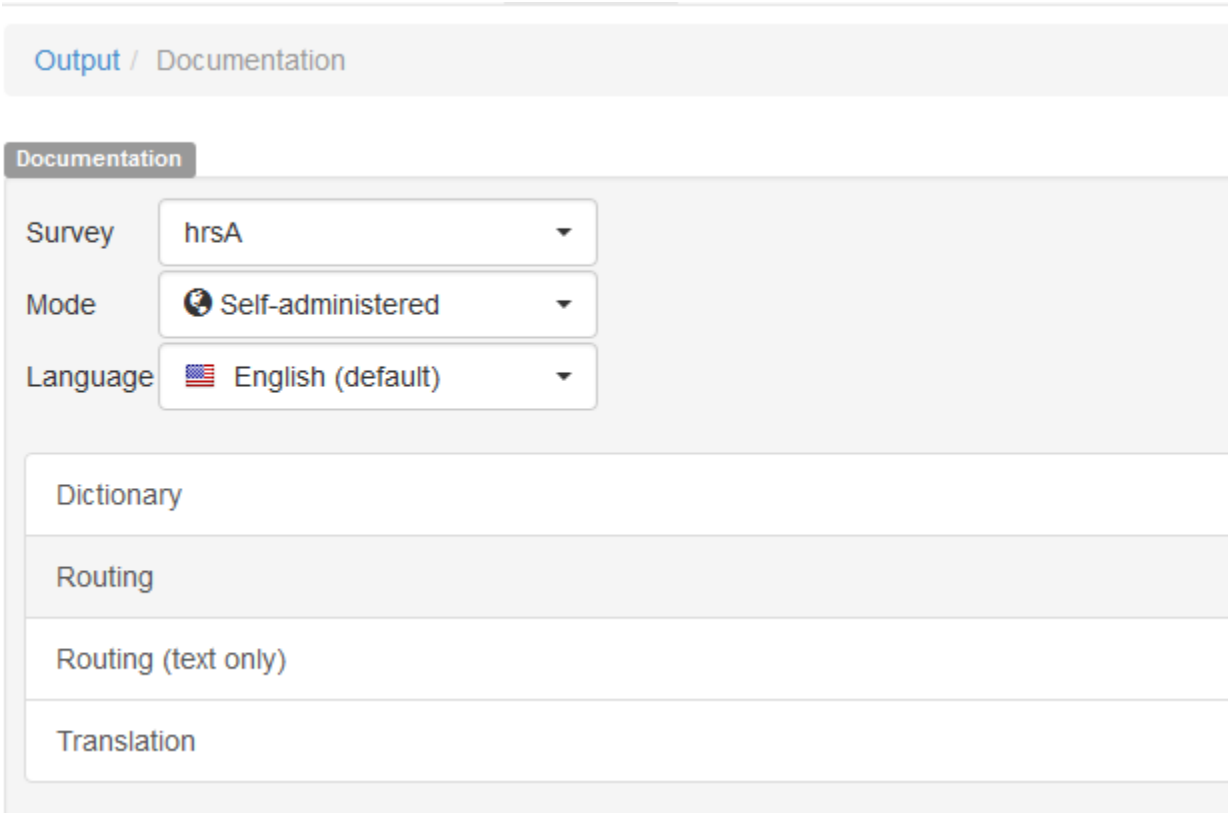


A last interesting series of graphs is to look at ‘Platform information’. Based on the user agent strings provided by the respondents’ browsers these graphs give insight into the type of device, browser and operating system these survey respondents use (Note: the platform information graphs include both incompleted and completed interviews).



20.3 Meta-data

Besides data we can also ask NubiS to provide us with so-called meta-data. This is data about the survey rather than data collected by the survey. The following types of meta-data are standard available:



The data dictionary provides us with 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 document can also assist. It comes in two flavors: one with graphical markup and one with 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.

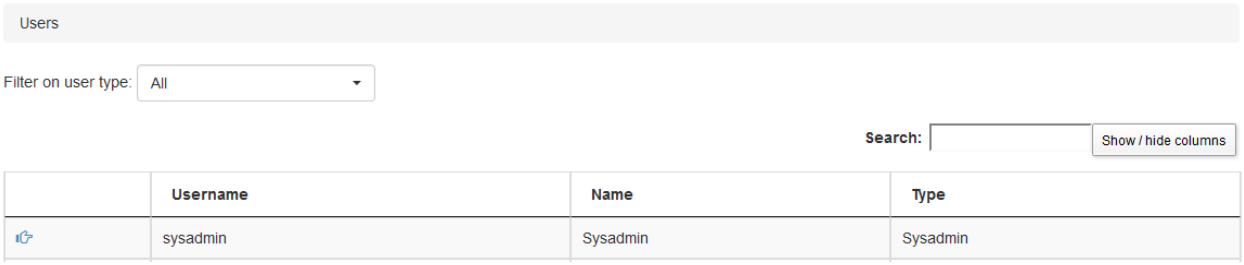
Lastly, the translation document gives a detailed overview of all the text in the survey that might require translation. It can be used for example as the basis for a first translation or to compare between the original language and the translation. For the latter we can indicate for which interview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

21. User management

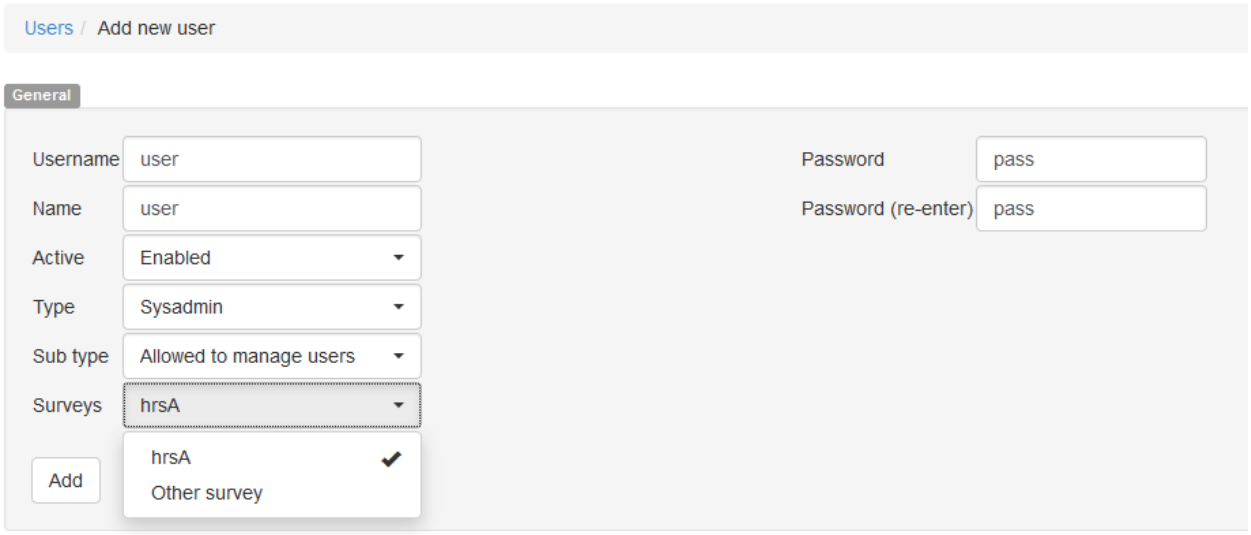
Throughout the previous sections we already made mention on several occasions of various ways of interacting with NubiS. For example, this manual itself focuses mainly on activities related to programming a survey, but we also made mention of other activities such as translation and testing. These different activities have been logically grouped and when bound together constitute user types. NubiS currently provides the following roles:

- System administrator: is the focus of the **NubiS System Administrator** manual (this 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: is 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: is 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: is the focus of the **NubiS Interviewer manual**. An interviewer has access to all functionality needed for managing interviews assigned to him/her, e.g. in terms of adding contacts or conducting interviews.
- Supervisor: is the focus of the **NubiS Supervisor manual**. A supervisor has access to all functionality needed for assigning interviews, monitor fieldwork progress, and so on.
- Nurse: is 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: is 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 administrators only and then only with those 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:



Since we already saw in section 19.1 how to add a user, we will just briefly repeat the procedure here. A user can be added using the ‘Add new user’ link:



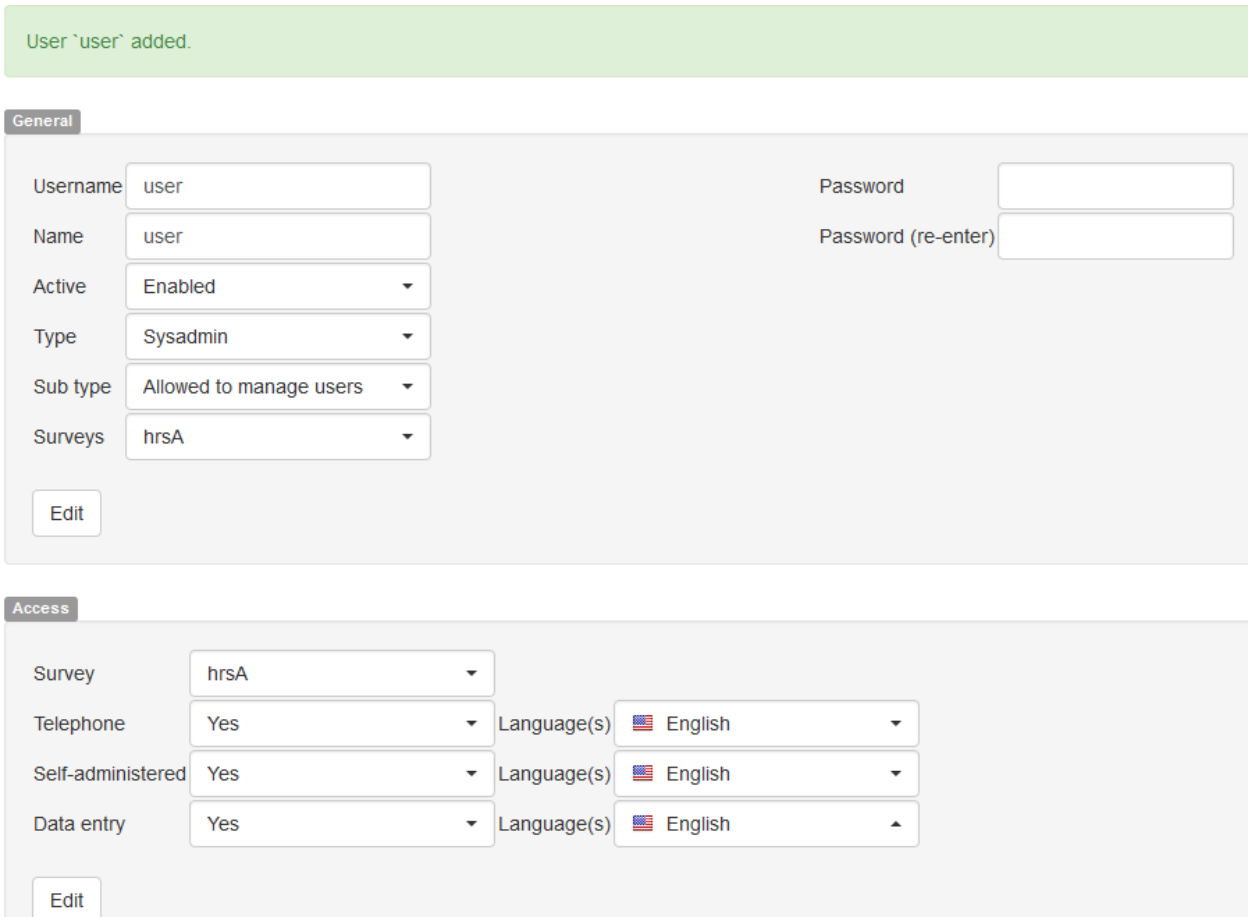
Adding a new user involves specifying a username/password combination. We can also specify a name in order to make it easier to see who the account belongs to. Next, we can indicate whether the account is active or not. For example, over time an account may become obsolete. One option is to remove it altogether, but another is to simply disable it so it can be re-activated if needed.

Next up is the choice of user account type. If the type chosen is that of system administrator, translator or tester, we also need to specify the level of access in terms of interview modes and languages. They take on slightly different meaning depending on the account type, but all of them enforce access limited to the selected interview modes and languages. For example, a translator for whom only Spanish was selected, will be able to see English during translation but not be able to edit it. Similarly, a tester for whom only face-to-face was selected, s/he won't be able to test the survey in the self-administered interview mode.

If instead we choose a nurse account, we can specify if it is for example a head nurse. Head nurses have access to more functionality than normal nurses. Please refer to the NubiS Nurse manual for more information. Similarly, for a sysadmin account we can indicate whether the account will be allowed to access user management. This allows to create sysadmin accounts while at the same restricting user management.

Lastly, we can indicate the surveys to which the account will have access. This enables an account to have access to for example our hrs survey, but not the other survey:

Once we've added the account, we get the following confirmation screen:



Notice in the above that we have an additional ‘Access’ section below. This section appears for sysadmin, researcher, tester and translator accounts, and gives us 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 we can use the ‘Survey’ dropdown to switch to another survey (assuming for a moment the account has access to multiple surveys, as specified in the ‘General’ section.