

---

# Signalbox Documentation

*Release 2*

**Ben Whalley**

September 10, 2014



<b>1</b>	<b>Installation</b>	<b>1</b>
1.1	Prerequisites . . . . .	1
1.2	Hosted installation on Heroku . . . . .	1
1.3	Local install for development . . . . .	3
<b>2</b>	<b>SignalBox’s Roles and Responsibilities</b>	<b>5</b>
2.1	Role and Permissions . . . . .	5
2.2	Role based permissions . . . . .	5
<b>3</b>	<b>Setting up studies</b>	<b>7</b>
3.1	Creating a new study . . . . .	7
3.2	Assessors and blinded studies . . . . .	11
<b>4</b>	<b>Creating questionnaires</b>	<b>13</b>
4.1	Editing via markdown text format . . . . .	13
4.2	A complete example . . . . .	16
4.3	Creating questions . . . . .	17
4.4	Repeating questions within a Questionnaire . . . . .	17
4.5	Approximate completion times for questionnaires . . . . .	17
4.6	Displaying previous answers or summary scores in questions . . . . .	17
<b>5</b>	<b>Recruiting participants and study memberships</b>	<b>19</b>
5.1	Adding participants . . . . .	19
5.2	Randomisation/allocation methods . . . . .	20
<b>6</b>	<b>Collecting data from participants</b>	<b>23</b>
6.1	Marking replies as canonical . . . . .	23
<b>7</b>	<b>Managing data and Exporting Data</b>	<b>25</b>
7.1	Exporting study data . . . . .	25
<b>8</b>	<b>Resources for trial administrators</b>	<b>27</b>
8.1	Creating and editing a public facing website . . . . .	27
8.2	Secured pages . . . . .	28
8.3	Usage Policies . . . . .	29
8.4	Use of Markdown formatting . . . . .	29
<b>9</b>	<b>Reference for database/application structures</b>	<b>31</b>
9.1	Ask application . . . . .	31

9.2	Signalbox application . . . . .	31
<b>10</b>	<b>Glossary</b>	<b>33</b>
<b>11</b>	<b>Overview</b>	<b>35</b>
<b>12</b>	<b>Functional anatomy</b>	<b>37</b>
12.1	Core components . . . . .	37
12.2	Questionnaires and collecting data . . . . .	39
12.3	Additional components which may not always be used . . . . .	41
<b>13</b>	<b>Database schema</b>	<b>43</b>
13.1	Technical overview . . . . .	44
<b>14</b>	<b>Indices and tables</b>	<b>45</b>

---

## Installation

---

To run Signalbox you will need some kind of Unix with python 2.7 available. Ubuntu 12.04 LTS is currently recommended, but OS X is fine too for development.

To get up and running quickly, the easiest way is currently to use [Heroku](#). Heroku have a free plan which is capable enough for even quite large studies, although self-hosting is also straightforward.

### 1.1 Prerequisites

For hosted installations, you just need python, and pip.

For local development, on Ubuntu 12.04, you can install everything you need for a development machine like this:

```
sudo apt-get install -y python-dev postgresql-server-dev-9.1 libjpeg-dev virtualenvwrapper libmagic-  
export WORKON_HOME=~/.Envs  
mkdir -p $WORKON_HOME  
source /usr/local/bin/virtualenvwrapper.sh
```

### 1.2 Hosted installation on Heroku

To get Signalbox running on Heroku's free plan (which is ideal for normal sized studies), you first need to:

1. Sign up for an account with Heroku (<https://devcenter.heroku.com/articles/quickstart>) and install their command line tool. You should upload your keys to the server to avoid having to repeatedly type your password:  

```
heroku keys:add
```
2. Optionally, if you want to upload images or other media for studies or questionnaires, sign up with Amazon for an S3 storage account (uploaded image files cannot be kept on heroku; see <http://aws.amazon.com>).
3. Optionally, if you plan to send email, obtain the details (host, username, password) for an SMTP email server you will use. Amazon's 'simple email service', SES, is good: <http://aws.amazon.com/ses/>
4. Optionally, if you plan on using interactive telephone calls or SMS, sign up with Twilio and make a note of your secret ID and key: <https://www.twilio.com>.

#### 1.2.1 Installation

Install the heroku command line program and authenticate:

```
wget -qO- https://toolbelt.heroku.com/install-ubuntu.sh | sh
ssh-keygen
heroku keys:add
```

Clone the example project:

```
git clone GITHUBREPO newname
cd newname
pip install -r requirements.txt
```

Then run the install script:

```
heroku_install_signalbox
```

At this point, your installation should be up and running on heroku:

```
heroku open
```

But you need to create the first user to login to the admin site:

```
heroku run app/manage.py createsuperuser
```

### 1.2.2 Scheduled tasks

Remember to add a scheduled task to send observations via the heroku control panel. The frequency is up to you - polling more often can cost more in dyno time if it overruns the free quota (but not much), but you'll want to add scheduled tasks for these scripts:

```
app/manage.py send
app/manage.py remind
```

If adding through cron on your own server remember to make sure the python used has signalbox in its path (i.e. activate the virtualenv first).

### 1.2.3 Environment variables

Note that all settings, API keys, and passwords are stored in environment variables (see <http://www.12factor.net/config>).

Environment variables can be set using:

```
heroku config:set VAR=SOMEVALUE
```

The key ones you will need to set are:

```
AWS_STORAGE_BUCKET_NAME
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
```

```
TWILIO_ID
TWILIO_TOKEN
```

```
EMAIL_HOST
EMAIL_HOST_USER
EMAIL_HOST_PASSWORD
```

Others are listed below for reference.

## 1.2.4 Version control

Signalbox can use `django_reversion` to keep track of changes to Answer, Reply and Observation objects to provide an audit trail for a trial. It's not enabled by default, but to turn it on you can set an environment variable:

```
heroku config:set USE_VERSIONING=1
```

## 1.3 Local install for development

Once you have Signalbox installed in a virtualenv and a hosted instance running, it's easy to start hacking on it locally to update templates etc.

First make a database with postgres (for development, allow the local user all permissions).

```
createdb sbox
```

Then update the `DATABASE_URL` environment variable to match your new database. If everything works, open <http://127.0.0.1:8000/admin> to view the admin site on your development machine.

Make changes in the local repo, commit them and then:

```
git push heroku master
```

### 1.3.1 Browser compatibility

The front-end (participant facing pages) should work in almost all browsers, including IE7.

The admin interface works best in a recent webkit browser (Safari or Chrome) but will largely function in IE7 (although the menus are slightly broken, they are usable). Everything will work properly in IE8 onwards.

---

**Note:** It's recommended to use Chrome-FRAME if IE7 is the only available browser. See: <https://developers.google.com/chrome/chrome-frame/>

**Warning:** Check everything works in your target browsers early in the trial setup. The health services and large firms have some weird and wonderful stuff deployed.

### 1.3.2 Custom domain names

You can add your own domain name to the app, but you will need to update the `ALLOWED_HOSTS` environment variable. See <https://docs.djangoproject.com/en/dev/ref/settings/#allowed-hosts> and <https://devcenter.heroku.com/articles/config-vars>.

### 1.3.3 Reference for all user-configurable environment variables

Each of these is loaded from an environment variable by `signalbox.configurable_settings.py`, and some are documented there:

```
DB_URL default: postgres://localhost/sbox
```

```
LOGIN_FROM_OBSERVATION_TOKEN  
SHOW_USER_CURRENT_STUDIES  
DEFAULT_USER_PROFILE_FIELDS
```

DEBUG

AWS\_STORAGE\_BUCKET\_NAME  
COMPRESS\_ENABLED  
AWS\_QUERYSTRING\_AUTH

SECRET\_KEY  
AWS\_ACCESS\_KEY\_ID  
AWS\_SECRET\_ACCESS\_KEY  
TWILIO\_ID  
TWILIO\_TOKEN

ALLOWED\_HOSTS  
SESSION\_COOKIE\_HTTPONLY  
SECURE\_BROWSER\_XSS\_FILTER  
SECURE\_CONTENT\_TYPE\_NOSNIFF  
SECURE\_SSL\_REDIRECT  
SESSION\_COOKIE\_AGE  
SESSION\_SAVE\_EVERY\_REQUEST  
SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE

SESSION\_COOKIE\_SECURE=False

USE\_VERSIONING=False



---

## SignalBox's Roles and Responsibilities

---

### 2.1 Role and Permissions

Signalbox defines 4 roles, each of which have set of permissions.

- Researchers
- Research assistants
- Assessors
- Clinicians

In addition, the documentation refers to:

- Participants/Patients

However participants are not a distinct group within the system - a participant just needs to be a registered `User` of the site. Note researchers, clinicians etc. may also be participants. Also see *Membership types* .

---

**Note:** All users of the system are stored as `django django.auth.User` objects. We use the standard django authentication mechanisms to log people in and track session state (this uses a cookie for identification, but all the data gets stored in the DB). SignalBox roles are defined using Django's `Group` models (see <https://docs.djangoproject.com/en/1.3/topics/auth/> for more details.)

---

**Note:** for many studies, the only types of users which will be required are Researchers and Participants.

---

### 2.2 Role based permissions

Permissions for many views within the site are determined by group memberships.

#### 2.2.1 Researchers

Researchers have pretty much full access to data and functionality within the site. They don't need to be a superuser within the django auth system though. Examples of things only researchers need to be able to do are resolving duplicate replies for observations (i.e. picking a canonical reply, see *Collecting data from participants*), and exporting data (`exporting_data`).

### 2.2.2 Research assistants

For large trials, Research Assistant's help administer participants and memberships, but do not have full access to participant data. Research assistants can:

- Add a new user to the system (e.g. enter details for a patient who has just been recruited)
- Add a user to a study (create a membership)
- See a list of observations outstanding
- Enter data for a specific observation
- Add notes to a patient (on the patient dashboard page)

### 2.2.3 Assessors

Assessors are responsible for collating data from patients, often in interviews, and are typically blind to the condition a participant has been assigned to. Assessors have only limited access to the site, because much of the data stored from participant self reports would reveal which condition a participant was in.

Assessors can:

- Find a given participant
- See that observations which are due for participants
- See observations which are overdue
- Enter data for an observation

### 2.2.4 Clinicians

Clinicians treating patients may use the system in several ways:

1. By adding treatment records and other clinical data
2. By taking part in sub-studies in which they record data about other participants (see *Membership types*).

In theory clinicians may or may not be blind to the allocation of patients (although in the Reframed study they will not be blind).

Clinicians should have no access to research data (i.e. information about observations due/overdue). They will need to:

- Find a particular participant
- Add a TreatmentRecord for a treatment session
- Complete an ad-hoc questionnaire attached to a TreatmentRecord
- Send messages to patients
- Add notes to patients (patient\_dashboard)

---

## Setting up studies

---

When using Signalbox, it's helpful to distinguish between a Trial or Research Project (i.e. the work you are doing) and a *Study*. In Signalbox, studies have a specific meaning, and a complex trial may be split across several studies for convenience and simplicity.

To get a grip on how things are structured, see the *SignalBox*.

### 3.1 Creating a new study

Start by adding a study.

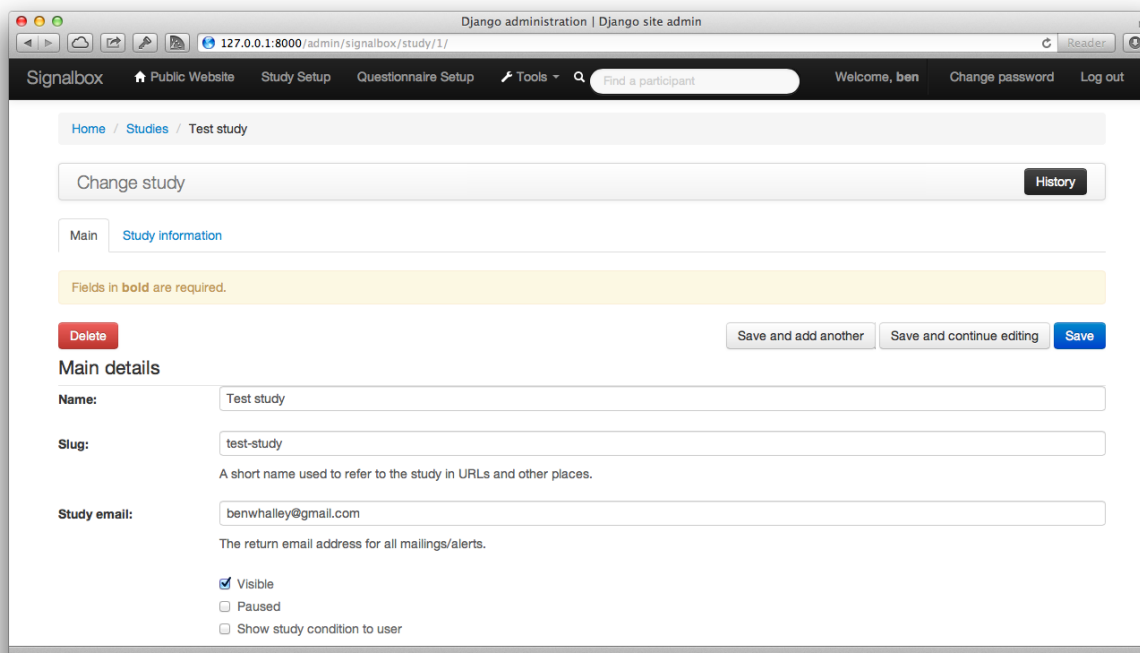


Figure 3.1: The study change view

Complete the necessary details in the *Overview* and *Study Information Fields* sections, and upload an image to repre-

sent the study.

### 3.1.1 Studies with multiple conditions

Studies may contain multiple conditions, each of which may use independent schedules to followup participants. Study conditions can be added at the bottom of the Study change view.

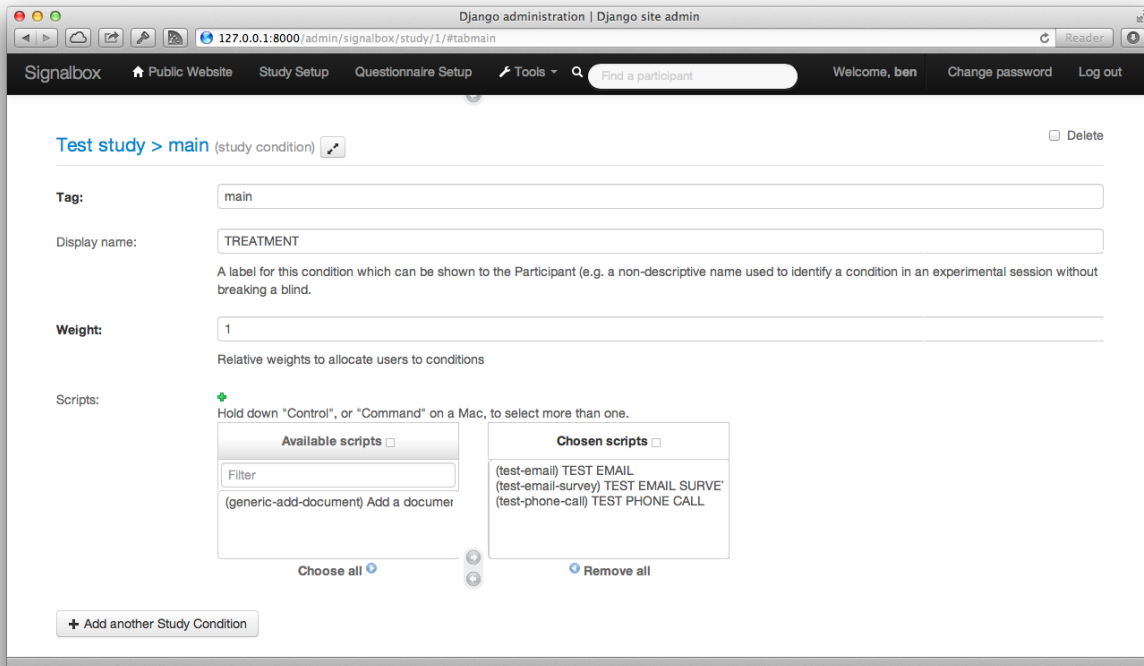


Figure 3.2: Adding study conditions.

By default, participants will be randomised to study conditions in equal proportions. This can be amended by changing the `weight` option of the `StudyCondition`. Weighted randomisation is also possible to minimise group imbalances. These options are available under the *Advanced* tab on the study change view.

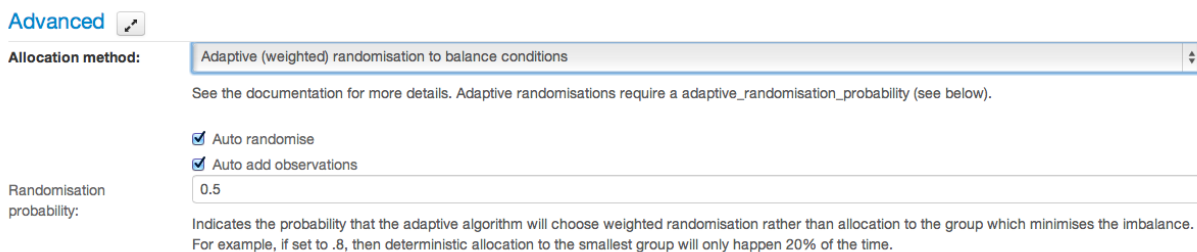


Figure 3.3: Advanced randomisation options.

### 3.1.2 Capturing personal information from users on signup

When users register with the site (normally when they consent to take part in a study) Signalbox can request additional profile information from them, including:

```
:: landline mobile site address_1 address_2 address_3 county postcode
```

By default these fields are optional, and not displayed to users. Each study can specify which of these fields should be (a) displayed and (b) required for participants to complete. This is specified by the *Visible/required profile fields* box on the study change view.

..note:: One current issue with the signup process is that participants are randomised to the study and observations created as soon as the user consents, and these observations can be triggered before they provide necessary profile information (e.g. mobile number). Be aware of this when defining study schedules.

### 3.1.3 User prompted data collection

In addition to observations defined by a Script, data can be collected ad-hoc, as users choose. Script specified in the *Scripts allowed on an ad-hoc basis* field can be executed by users from their profile page.

..note:: Script specified as allowed for user-prompted data collected may generate more than one observation, which may not be what is desired.

### 3.1.4 Using ScoreSheets for conditional or ‘responsive-mode’ data collection

Observations may also be created in a responsive fashion, based on responses users make — for example, if a participant’s questionnaire responses meet some criteria.

When collecting data in responsive mode, the study administrator needs to define the conditions to be met to create and send additional observations. `ObservationCreator`, `ShowIf` and `ScoreSheet` objects are used to do this.

`ObservationCreators` define a Script to be run when a Reply contains answers which meet a certain criteria. `signalbox.models.ScoreSheets` are used to define these criteria. A `ScoreSheet` consists of a name and description, a list of Questions for which responses will be included and a function to be applied to answers to these questions.

For example, a `ScoreSheet` named ‘Beck Depression Inventory Sum Score’ might reference each of the questions in the BDI, and use the `sum` function. The `compute` method of the `scoreSheet` will then apply the function to Answers provided within a given Reply (i.e. on a single occasion for a particular user) and return a single numeric value. At present `sum`, `mean`, `stdev`, `median`, `min` and `max` functions are available.

---

**Note:** Some of the score functions will return a floating point value (including the median function), which means direct equality comparisons with integers will not always work as expected; e.g. if the function returns 2.000000000002 for `median([1,2,3])`, so comparison with the integer 2 will be `False`.

---

`ShowIf` objects define thresholds which scores calculated by `ScoreSheets` must meet before a Script can be run and new observations created. For example, a `ShowIf` might specify that a BDI score in a particular reply must be `> 15` for the script to be executed.

---

**Note:** `ShowIf` objects are also used for conditional display of Questions and Instruments within Askers.

---

Figure 3.4: Creating a ShowIf.

### 3.1.5 Researcher Alerts

In some cases it may be necessary to alert researchers if participants make particular responses: for example, answers to questions relating to suicide which indicate participants may be at risk. To facilitate this, it's possible to attach Alerting rules (`signalbox.models.Alerts`) to studies.

As with responsive-mode data collection, alerts use ShowIf objects to define conditions under which an alert will be sent. Alerts also store an email address and/or mobile phone number to send email or sms-based messages when triggered.

## 3.2 Assessors and blinded studies

Some larger RCTs may require blinded assessments to be made, and employ assessors to interview participants. Although assessors need access to the system to enter (and perhaps double enter) data, it's important that they don't encounter information which might compromise the blind. Such a situation would obviously occur if assessors could see the condition to which a participant was added. However less obvious situations might occur, when assessments differ between study conditions. For example, if assessors could:

- See scripts (or information from scripts) which are only relevant to a particular condition
- See observations or replies which include reference to questionnaires or instruments only shown to a particular condition.

To prevent this, assessors have only limited access to the site, and have a specific view designed to let them safely access observations and update client data. This is available at:

```
/admin/signalbox/observations/outstanding
```

From here, assessors can filter clients by username (which is likely to be a unique alphanumeric code rather than a name) and list Observations which are due to be made. This view automatically filters out observations which:

1. Have a script which is marked *breaks\_blind*
2. Have been added on an ad-hoc basis (determined by checking whether the Observation has an attached script).

From this view, assessors can select an observation and enter data for it.

**Warning:** Blinded Assessors and access to observations-due

Care is needed when creating scripts and questionnaires which blind assessors will access. In particular the following must not include content which could reveal which group a participant is in as the assessor enters data:

- *label* attribute of `:class:'Script's` (this is used when listing observations for assessors)
- The content of questionnaires themselves (clearly no questions should be visible which identify study condition – for example, therapy-relevant information).





---

## Creating questionnaires

---

The `Ask` application deals with creating and displaying questionnaires.

Questionnaires can be spread across multiple pages, and consist of questions. Questions can also be grouped into Instruments, which can be placed in a block into a page. Questions will often refer to ChoiceSets (groups of discreet options), e.g. for likert type responses.

### 4.1 Editing via markdown text format

To enable rapid editing of questionnaires, a text-based format is available in which titles, questions and choice-sets can be specified, and which are converted into Asker, Question and ChoiceSet objects in the database. This text format is based on [markdown](<http://johnmacfarlane.net/pandoc/demo/example9/pandocs-markdown.html>).

The text to edit questionnaires comes in two blocks: the header, which specifies details of the questionnaire as a whole, and the body which contains individual questions and choicsets.

#### 4.1.1 The questionnaire header

The header is formatted as follows:

```
---
name: "Name of the questionnaire here"
slug: "examplequestionnaire"
redirect_url: "http://www.example.com"
show_progress: false
step_navigation: true
steps_are_sequential: true
success_message: "Thanks for completing this questionnaire."
---
```

The *name* and *slug* attributes identify the questionnaire — the *slug* being a short identifier which can be used in url links. The other fields are as follows:

***redirect\_url*** The page the user is sent to when the questionnaire is complete

***success\_message*** If the user is redirected to page within the signalbox site, an extra message which will appear in a banner at the head of the page after completion.

***show\_progress*** Whether participants can see how far through the questionnaire they are

***step\_navigation*** Whether links should be included allowing random access to each page within the questionnaire.

*steps\_are\_sequential* If true, and *step\_navigation* is enabled, then participants can only navigate ‘back’ within the questionnaire, and cannot skip forwards.

### 4.1.2 The questionnaire body

Questions themselves are defined in what are called ‘fenced code blocks’ in Markdown. For example:

```
~~~{}
This the simplest type of ‘question’ - an instruction. No responses will be collected here.
~~~
```

When this is saved, you’ll notice that the system adds some attributes to the block to enable it to be identified in the database later for editing. So, the example above would get transformed into:

```
~~~{#ecVhsaj7889ij .instruction}
This the simplest type of ‘question’ - an instruction. No responses will be collected here.
~~~
```

Here, a variable name has been added (*ecVhsaj7889ij*) and the *type* of question specified for clarity. To add other types of questions you will need to manually specify the type, and optionally also the variable name for example:

```
~~~{#howoldareyou .integer}
How old are you, in years?
~~~
```

This would create an html question which requires an integer answer to be entered in a small text box. Some questions (including integer questions) have optional attributes. For example:

```
~~~{#howoldareyou .integer min="12" max="120"}
How old are you, in years?
~~~
```

In the example above we add a min and max attribute to validate against some typos. The text of questions can itself include markdown formatting to create headings, emphasis or links within a questionnaire. For example:

```
~~~{#howoldareyou .integer min="12" max="120"}
# Demographic information

How old are you, in years?
~~~
```

In the example above, a level-1 heading (Demographic information) is inserted, and the text ‘in years’ is formatted in bold and italic. For more information on [markdown formatting see the guide here](XXX).

### Defining a list of choices

Some questions require users to select from a restricted range of choices, for example a likert-type scale. To specify the choices, specify a *choiceset* attribute on the question, and define the *choiceset* in a second, separate block:

```
~~~{#howhappyareyou .likert}
How happy are you?
>>>
1=Very happy
2=Miserable
~~~
```

Here the possible options are listed following “>>>” on separate lines, in the form *score=label*. Scores must be integers, and are the values saved when the user provides an answer.

## Default options

To mark one option to be selected by default, insert a star in front of the value:

```
~~~{#rangelto4 .likert}
Question text
>>>
*1=Happy is selected by default
2=2
3=3
4=Unhappy
~~~
```

## Calculating and displaying summary scores from participant responses

For instruction questions, in place of a list of choices, it is possible to specify a score which will be computed from previous participant responses (a ScoreSheet). For example:

```
~~~{#summaryscoreexample .instruction}
Your total score is: {{totalscore}}
>>>
totalscore <- sum(variablename1 variablename2 variablename3 ...)
~~~
```

This question will compute the sum of variable1, 2 and 3, and display it where the `{{totalscore}}` marker is, within the question text. Again markdown formatting can be applied to scores.

Note: because answers must be saved in the database before being available for summary scores, be sure to specify this type of question on a page which comes after the variables to be used.

## Remapping of scores

As notes, scoresheets allow you to specify summary scores from combinations of questions which the participant has already made. Sometimes, you might like to score responses in such a way that several of the options equate to the same value. You can achieve this by adding `[int]` after the score to be stored in the database:

```
~~~{#remappingexample .likert}
Question text
>>>
*1=Stores 1 in the database, and in scoresheets
2=Stores 2 in the database, and in scoresheets
3[2]=Stores 3 in the database, but scores 2 as part of scoresheets
~~~
```

## Using the Django templating language

Signalbox uses the Django template language to render the text of a question as it is presented to the user. Several variables, including summary scores (see above) are available in the render context, and can be included with the `{{varname}}` syntax. Other more advanced features can also be used, for example to conditionally display text based on previous answers. For example:

```
~~~{#djangotemplateexample .instruction}
Your total score was: {{totalscore}}.
{% if totalscore > 10 %}Well done!{% endif %}
>>>
```

```
totalscore <- sum(variablename1 variablename2 variablename3)
~~~
```

This question computes `{{totalscore}}` and then uses it to conditionally display extra text in the question.

Other variables available as the text is rendered are:

- Saved answers, accessed as: `{{answers.variable_name}}`.
- The Reply object (e.g. *You started this reply at: {{reply.started}}*).
- The User object (e.g. *Your name is {{user.first\_name}}*).

## 4.2 A complete example

A complete example can be found in `ask/fixtures/asker_text.md`.

### 4.2.1 Other types of questions available

Different questions types can be specified as attributes on the question, similar to a css class style. Just add a period (.) and the name of the type:

**instruction** No answer required, but ‘question’ text displayed.

**uninterruptible-instruction** Like instructions, but when using IVR systems this type prevents the user continuing until the whole message has played.

**short-text** A small text input box

**long-text** A large <textarea> box.

**likert or likert-list** Discreet options selected via radio-buttons (i.e. options are mutually exclusive). *likert-list* produces a vertical list as opposed to a horizontal scale. Add *.rotate* to rotate the option labels.

**checkboxes** As for *likert*, but options are not mutually exclusive (more than one can be selected).

**integer** The user can only enter an integer. Optional attributes are *min* and *max*.

**decimal** As for integer, but allows only (and validates) decimal numbers.

**pulldown** As for likert, but uses a pulldown selector instead of radio buttons.

**required-checkbox** Displays the question text next to a checkbox which the user must check to progress to the next page.

**slider or range-slider** A slider element which allows users to pick a value between a *min* and a *max* which are specified as additional attributes. E.g.:

```
~~~{#variablename .slider min=0 max=100 value=50}
Slide the slider to a value between 0 and 100 (this slider will default to 50).
~~~
```

Or if you want a slider with two movable points to specify a range of values:

```
~~~{#variablename .range-slider min=0 max=100 values=[10,90]}
Slide the slider to encompass a range of values between 0 and 100 (this slider defaults to the r
~~~
```

Note that for both sliders, a default value will be given and it’s therefore impossible to specify that a response is required (because no response cannot be distinguished from the default response).

**date** A date picker.

**date-time** A date-time picker.

**time** A time-of-day picker.

**hangup** This question will end an IVR call.

**webcam** Experimental support for webcams on user laptops. Allows capturing and sending an image to the server (which is saved in the DB rather than a file).

## 4.3 Creating questions

Questions are created by using django form field elements, and extending them with additional information required by signalbox. The types of questions which can be created are documented here: *Types of questions (Field classes)*

The fields and widgets are as described in the floppyforms documentation: <http://django-floppyforms.readthedocs.org/en/latest/widgets-reference.html>

In addition, for IVR telephone calls, there are:

- Uninterruptible instruction (this speaks the text of the questions, but without allowing the user to ‘barge-in’ and skip the text by pressing a key, as is the case with a normal instruction question.)
- Listen (records audio of the user)
- Hangup (speaks the text of the question and then ends the current call; it is required that the asker ends with a hangup question)

All questions can take an ‘audio’ attribute for use in IVR calls, for example:

```
~~~{#ivrexample .likert audio="http://www.example.com/audio.mp3"}
This text will be shown on the web, but http://www.example.com/audio.mp3 will be played over the tel
>>>
1=1
2=2
...
~~~
```

## 4.4 Repeating questions within a Questionnaire

Each question must have unique variable name which will be used to identify data collected. If a question is to be repeated within a questionnaire, it should either be duplicated and given a second, different, name.

## 4.5 Approximate completion times for questionnaires

These are calculated by a method on the Asker (Questionnaire) model:

## 4.6 Displaying previous answers or summary scores in questions

Read about ScoreSheets first.

Summary scores or previous questionnaire responses can be included on later pages, using the curly brace markers `{{}}`:

~~~

This will include an instruction displaying the users user response to a variable named howoldareyou

```
{{answers.howoldareyou}}
```

~~~

Or to show a summary score:

~~~

```
{{scores.summary_score_name}}
```

~~~

Be sure to enable a particular summary score for your Questionnaire on the main editing page - it won't be available unless you do.

---

## Recruiting participants and study memberships

---

XXX TODO XXX

Add overview here

XXX TODO XXX

Update content below A `Membership` links participants – a `User` – with `Study` objects. There are two types of memberships possible within a `Signalbox` study:

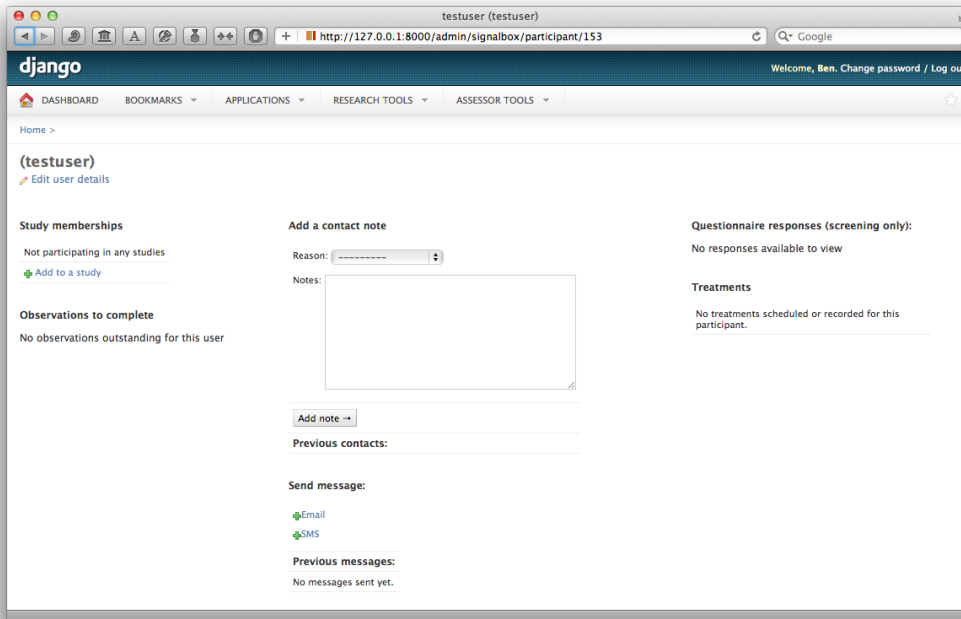
**Regular** This the normal situation, where a participant is simply taking part in a study.

**Related** A second kind, where someone is participating in a study not on their own account, but instead to provide data about another participants (e.g. patients, who are the real object of study). An example of this is where a clinician might take part in a sub-study within a trial to complete assessments of patient progress (or, in fact, report on multiple patients).

### 5.1 Adding participants

In clinical trials, participants may be recruited outside of the web system and added by administrators or researchers (rather than signing up directly online). Where this is the case, participants must be added to studies manually. Participants should be added via the add-participant wizard, available from the *Research Tools* menu, or at `/admin/signalbox/participant/new/`.

Once a participant has been added via the wizard, you are redirected to their dashboard page, from where you can add them to a study.



Where the membership is the *related* type, it's necessary to fill out the additional `relates_to` field on the add membership view. To be in the normal situation:

- The participant is the object of study

- The `relates_to` field is blank

And in the *related* type of membership:

- The participant is someone providing data about the individual being studied.
- The `relates_to` field denotes the individual being studied

## 5.2 Randomisation/allocation methods

When a *user* signs up for a study (or are added by a *researcher*), then the system may automatically allocate them to a `StudyCondition`

At present, allocation is made by weighted, adaptive randomisation.

### 5.2.1 Weighted adaptive randomisation

Participants are allocated to `StudyCondition`'s within a `:class:-signalbox.models.Study` in the proportions specified by the `weight` property of each `StudyCondition`.

For example, if there are 3 `Study` and have weights 2, 2 and 3, then participants will be allocated accordingly.

In addition, the `randomisation_probability` field on studies determines how deterministic this allocation is. Where `randomisation_probability = 1` then all allocations will be made at random (respecting group weights). However, when `randomisation_probability = .5`, half of allocations will be made deterministically to minimise the imbalances between groups.

More complex adaptive randomisation schemes are not currently available.



### 5.2.2 Pausing or deactivating observations for a membership

If a participant decides to leave the study, but is happy for data provided so far to be used, the best method to pause further observations is to deselect the `active` checkbox on their membership page. This will prevent all further observations being sent *for that study*.

.warning:: This will not stop all observations for this *participant* — only those due for this study (i.e. for the membership). If a participant withdraws from a trial which has multiple sub-studies, be sure to deactivate all of their memberships.

### 5.2.3 Randomisation dates and date-shifting observations

In the case that a person has been randomised to a trial prior to being added to the system, it may be necessary to alter the automatically-generated `randomisation_date` field on the membership. This can be done through the admin interface: first find the correct membership at `/admin/signalbox/membership/` and then edit the property directly.

In other cases a participant may be added to the study and observations added automatically. Sometimes, a participant may need to pause participation in the study, or may not have been added to the study early enough and may need to skip some of the observations a script creates. In these cases, a view is provided to *shift* the dates of observations which already exist for a membership, to correct the dates forward or backward. This is available at `/admin/signalbox/membership/dateshift/(membership_number)`, or via link on the edit-membership view.



---

## Collecting data from participants

---

An `signalbox.models.Observation` represents an occasion on which a measurement can be taken.

A `signalbox.models.Reply` collates a set of responses made for an `Observation`, *on a single occasion*.

Multiple replies can therefore be made for a single `Observation`. There may happen when:

1. A participant does not complete a questionnaire in one session, but returns to complete the observation by clicking in a link in an email in a second session.
2. Where researchers are entering paper data, and wish to enter it twice for validation.

Where multiple replies are made for an observation it is important to distinguish which should be used — i.e. exported for later analysis. In exported data the `canonical_reply` variable indicates which row should be preferred for analysis.

---

**Note:** If no `Reply` has been marked as canonical, multiple replies can be exported with none of them marked as canonical.

---

---

**Note:** The `is_canonical_reply` field on the `Reply` determines whether a reply should be considered canonical. The `signalbox.models.Observation.canonical_reply()` method returns the canonical reply for a given `Observation` and is used when exporting data.

---

### 6.1 Marking replies as canonical

Researchers can identify which reply should be considered canonical, using the view at: `/admin/signalbox/resolve/duplicate/replies/(study_id)`. This can be accessed from the admin page for each study.

---

**Note:** It's probably best to keep on top of duplicate replies as they arise — resolving which was the correct response a long time after the event might be difficult, or impossible.

---

#### 6.1.1 Entering data on behalf of someone else

Sometimes participants won't be able to enter data for themselves, and researchers will have to enter it for them.

To enter data for someone else, first find the relevant observation (probably by navigating from the `user_dashboard`), and then click the *Enter data for this observation* link:

You will be presented with the questionnaire that the user would have seen.

Once complete, you'll be redirected back to the `Observation` edit page.

---

Notice that a new Reply will be listed for that observation

**Note:** You may need to set the `originally_collected_on` field of the Reply if the data was collected from the participant sometime before you are entering it into the system. See image below:

XXX TODO replace and update with additional view

The screenshot shows a web interface with three main components:

- Replies Table:** A table with columns 'Asker', 'Complete', and 'Originally collected on'. A row is visible with 'SCIDI screening\_questionnaire' as the asker, a red minus sign for 'Complete', and a date input field for 'Originally collected on' set to 'Today'. There is an 'Edit Reply' button.
- Calendar:** A calendar for June 2012. The 29th is highlighted in yellow. Navigation buttons for 'Yesterday', 'Today', 'Tomorrow', and 'Cancel' are at the bottom.
- Observation Datas:** A section titled 'Observation Data: attempt' with a 'Key: Attempt'.

### 6.1.2 SMS message replies

Because of limitations of the SMS system itself, where participants reply to an SMS it's not possible to reliably reconnect their replies explicitly to a particular observation. For this reason, although SMS replies are stored in the database (using the `signalbox.models.TextMessageCallback` model), the content of SMS replies is not stored as an *Answer*.

Currently Signalbox makes an attempt to connect the inbound SMS messages with Observations in the following way:

- When an SMS is sent via twilio, an *ObservationData* object is stored which records the Twilio *SmsSid* value.
- When a reply is made within a limited period Twilio appears to return a response containing the original *SmsSid* value, which is again stored and used to link the two records (see the `sms_replies()` method of an *Observation*). However this behaviour is undocumented, and is probably not reliable.

When analysing SMS reply data, it's probably better to rely on the sender phone number and match these with users' numbers (e.g. using merge commands in Stata or SPSS).

..warning:: TL;DR: If users reply to SMS messages you will have to do extra work to export this data; it won't automatically appear in the main datafile.

---

## Managing data and Exporting Data

---

### 7.1 Exporting study data

Signalbox is able to export data to delimited text, or to Stata (a common statistical package). When using Stata, the system also exports syntax which label data appropriately to help with later analysis. This syntax also computes some useful variables to help with later analyses.

Signalbox understands that within a single clinical trial there may be multiple sub-studies (each set up as a Study object within the system). Because studies may be linked, and require joint analysis, Signalbox allows you to export data for multiple studies at once.

If this is this case, you need to define a *reference study* — this will typically be the study which manages the primary randomisation, e.g. to Treatment/Control. Selecting a reference study is useful, because Signalbox uses the date of randomisation for that study to compute additional variables, including *days\_in\_trial*, which can be contrasted with *days\_in\_study* and indicates the elapsed days since the user was randomised to the reference study.

---

**Note:** Additional computed variables, including *days\_in\_trial* are only computed when the exported syntax file is run using Stata, and are not present in the raw text data, `data.txt`.

---

#### 7.1.1 To export data

1. Select `Tools > Export Data`
2. Select the studies you wish to export data for
3. Select a `reference study` (optional), and press `Submit`.
4. A zip archive will begin downloading, containing three files: `data.txt`, `make.do` and `syntax.do`.
5. Open the zip and, using Stata 11 or later, run `make.do`.
6. This script will generate three additional data files: `data.dta`, `data_values.xlsx`, and `data_labels.xlsx`. If you have StatTransfer installed it will also convert the datafile to SPSS `.sav` format.

---

**Note:** Based on data held by the system about the questions used to collect the data, Signalbox will add meta data to exported files. The `data.dta` file contains multiple value and variable labels and is probably what you want to use; if needed, the StatTransfer program will convert these to an SPSS file including value and variable labels transparently. The excel file `data_values.xlsx` includes the labelled values (i.e. strings for numeric variables); `data_values.xlsx` contains the values themselves.

---

### 7.1.2 Long vs. wide format

Signalbox exports data in a long-ish format. Individual `Answers` are grouped by `Reply` and saved one-reply-per-row in the `txt` file.

However, because different replies will have measured different variables, the resulting file will have many columns (one per variable measured in any of the replies) and lots of blank values (where a reply did not measure a variable, it will be blank).

Many analyses will require you to compute summary scores and restructure the file into a wide format (e.g. the `reshape` command in Stata).

### 7.1.3 Formats and labels

The export syntax attempts to correctly import python date/time objects as dates, and format them correctly. If this isn't happening for you then please file a bug report.

See also `exporting_data`

### 7.1.4 Serialising data

Before deciding to export questionnaires, try using the markdown editing feature (*Creating questionnaires*) and see if that gives you enough of what you want.

If it doesn't, read this first: <https://docs.djangoproject.com/en/dev/topics/serialization/>.

Then see: <http://stackoverflow.com/questions/1499898/django-create-fixtures-without-specifying-a-primary-key>.

So, to export use:

```
app/manage.py dumpdata --indent 2 --natural > data.json
```

Then use a regex like this to replace all pk's with null to be sure they don't clash with others in the db:

```
"pk": (?<path>d+),
```

---

## Resources for trial administrators

---

### 8.1 Creating and editing a public facing website

Alongside Signalbox, we use a content management system to create pages which are visible on the 'front end' or public facing website.

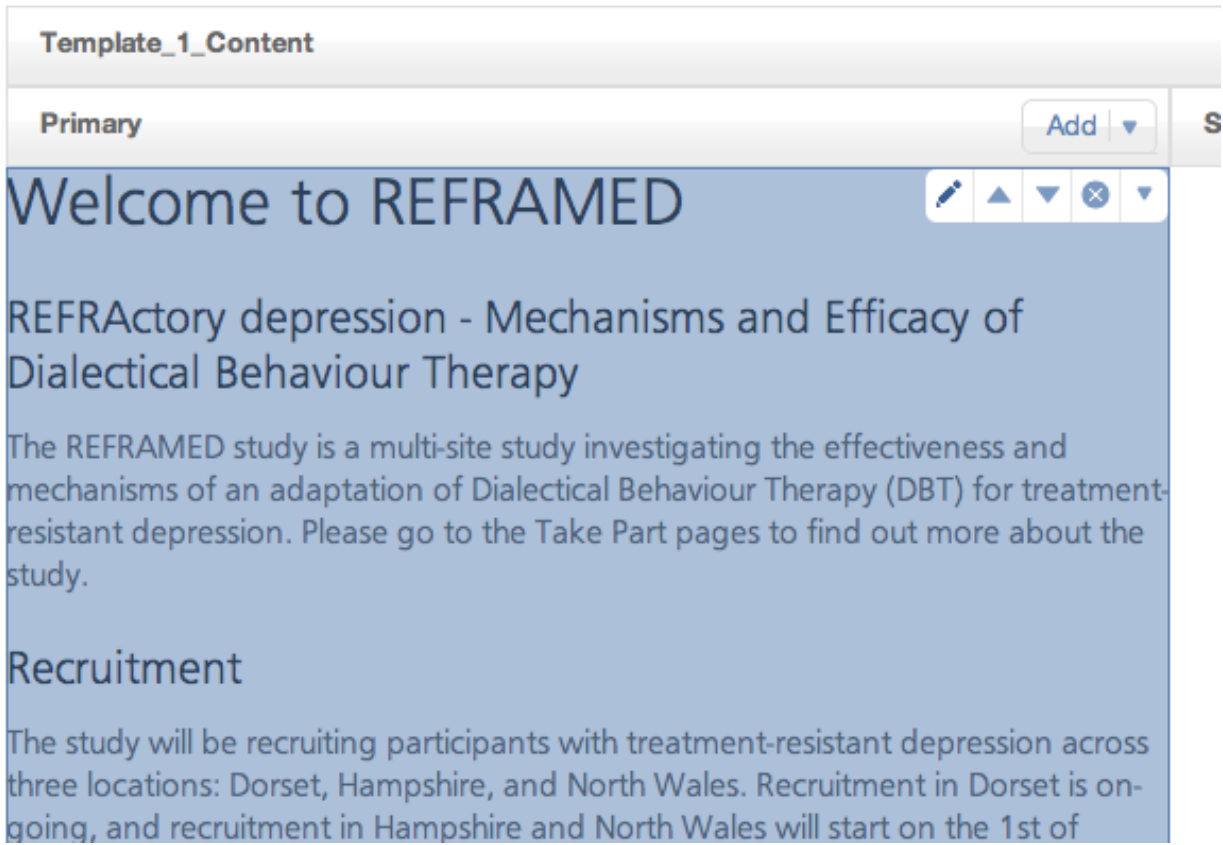
You can see a list of these pages as `/admin/cms/page/`. In addition, once you have logged in, you can press the + icon in the top right hand corner of most pages to edit the content on them. The icon looks like this:



Clicking it brings up an editing toolbar like this:



From which you can turn on editing, and also access the admin site. Once you have turned edit mode on, roll over content and click the edit icon shown in the image below:

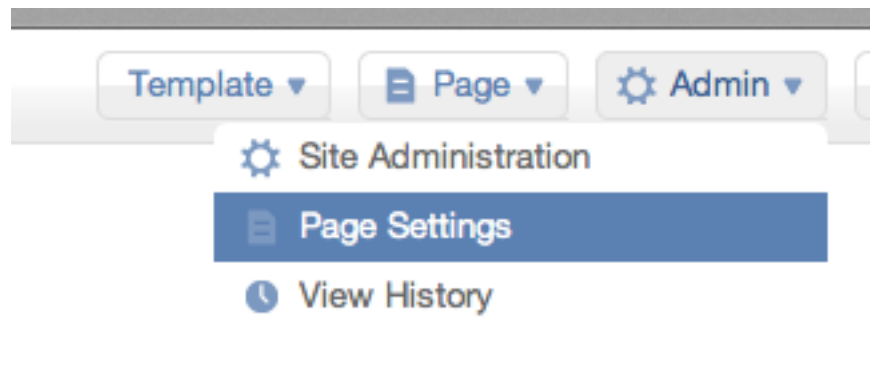


When adding content to the website, you can use markdown syntax to style content (see `markdown` for more details).

## 8.2 Secured pages



Additional functionality has been added to the CMS app to enable some pages to be protected from public view. These pages are only accessible by the study team (i.e. researchers, research assistants, assessors, and clinicians).

To make a page on the website protected, use the edit bar to access the page settings:



Scroll to the bottom of this page and click the 'advanced settings link'. Then enable the 'login required' checkbox, and save the page.



Advanced Settings (Hide)	
Id:	<input type="text"/> An unique identifier that is used with the page_url templatetag for linking to this page
Overwrite URL:	<input type="text"/> Keep this field empty if standard path should be used.
<input checked="" type="checkbox"/> Login required	
Menu visibility:	<input type="text" value="-----"/>  limit when this page is visible in the menu
Application:	<input type="text" value="-----"/>  Hook application to this page.

Protecting a page which has sub-pages (i.e. child pages which appear lower than it in the hierarchy of pages) will make all sub-pages protected.

**Warning:** If in doubt CHECK! Logout of your account and make sure the page really is protected.

**Warning:** Although protected from public view, the website is NOT to be used for sensitive content (e.g. correspondence, risk reports etc).

## 8.3 Usage Policies

### 8.3.1 Password policies

This system can contain confidential and often very private data. Pick a strong password, and store it in a password locker.

See <http://xkcd.com/936/>.

Auto generate a password here: <http://rumkin.com/tools/password/diceware.php>.

### 8.3.2 Password lockers

The preferred solution is Keepass <http://keepass.info/download.html>.

..note:: If you download the portable version onto a memory stick you can use it anywhere

## 8.4 Use of Markdown formatting

Markdown syntax is used extensively throughout the application to format text for display. Markdown is a simple format which allows headings, lists and links to be created without knowledge of HTML.

For more information on markdown see: [MarkdownSyntax](#).

Examples of elements which use markdown syntax for formatting:

- Study consent information
- Question text
- Website content (within the CMS)

---

## Reference for database/application structures

---

### 9.1 Ask application

#### 9.1.1 Models

#### 9.1.2 Types of questions (Field classes)

SignalboxField stores the additional information signalbox needs to properly display and process questions.

#### 9.1.3 Notes on all question types

- Question text: this is displayed in syntax (see `../markdown`).

`..warning::` Note that some characters are stripped and used to format the question text in html – for example, text surrounded by two `*` characters will be italicised, and lines starting with a number and a period (e.g. `"1. "`) will be turned into a numbered list.

#### 9.1.4 Types of question

Each of the following question classes extends the standard django form fields to allow for different types of questions:

### 9.2 Signalbox application

#### 9.2.1 Models



---

**Glossary**

---

**user** definition

**researcher** a specific role within the system, researchers have broad rights to configure the system, add/remove users, and other manage data

**twilio** Third party service used to make automated telephone calls. See <https://www.twilio.com>



---

### Overview

---

SignalBox is a web application application designed to make it easy to run clinical and other studies. Signalbox makes it easy to recruit, take consent from, and follow-up large numbers of participants, using a customisable assessment schedule.

Participants can provide self-report data via email, telephone, or SMS message. Study coordinators can login to a secure administration website to manage studies and check participation. The admin interface integrates online and offline elements of a study; researchers can enter additional datapoints collected offline or in the lab, and there is full support for double entry and reconciling of paper-based data, and also for audit trails of changes to participant data.

Signalbox was designed to replace the numerous, ad-hoc systems which have been developed by research groups, providing a flexible, secure, and well-tested system. The software has been independently audited, and used in many numerous studies, including a large, MRC-funded clinical trial (<http://www.reframed.org.uk>).





---

## Functional anatomy

---

Below is a description of the data model used by the system. In essence, all capitalised words are tables in the database; the text below helps describe their structural and functional interrelations.

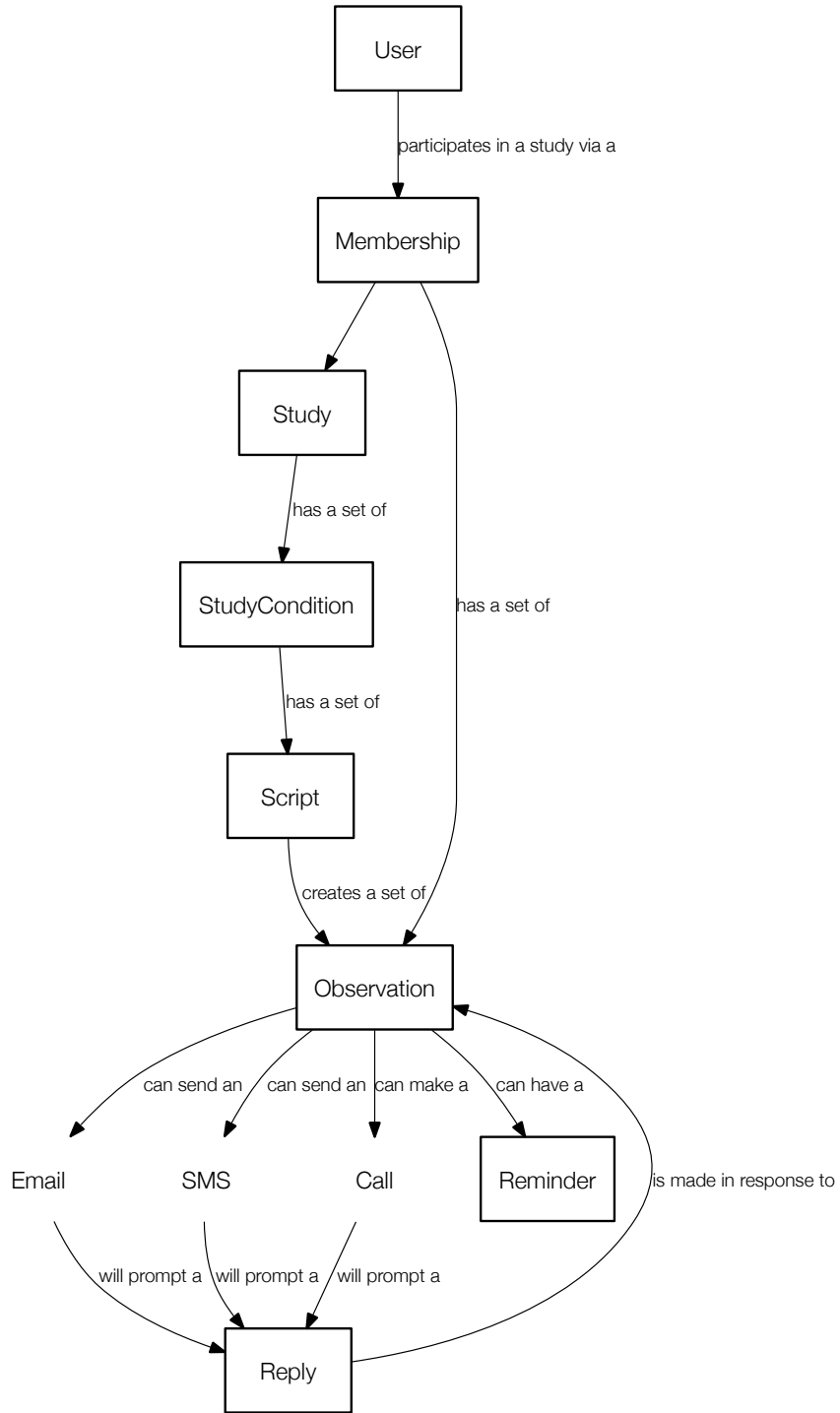
### 12.1 Core components

A `Study` is lined to a number of `StudyConditions` which can each use a number of `Scripts`. A `Script` links to an `Asker` (a questionnaire) which includes a number of `Pages` that contain `Questions`. `Questions` can use a `ChoiceSet` which represents a set of `Choices` which define the range of allowed `Answers` (e.g. as part of a Likert type scale). Alternatively, `Scripts` may define an external url at which participants will enter data (e.g. a bespoke experimental task, or via a third party service like SurveyMonkey).

When a `User` joins a `Study` then a `Membership` is created, which stores the randomisation times etc. When a `Membership` is randomised to a `StudyCondition` then the `Script` is used to create `Observations` (scripts use a special syntax to describe the offset at which each `Observation` will be created, by default counting from when the user is randomised).

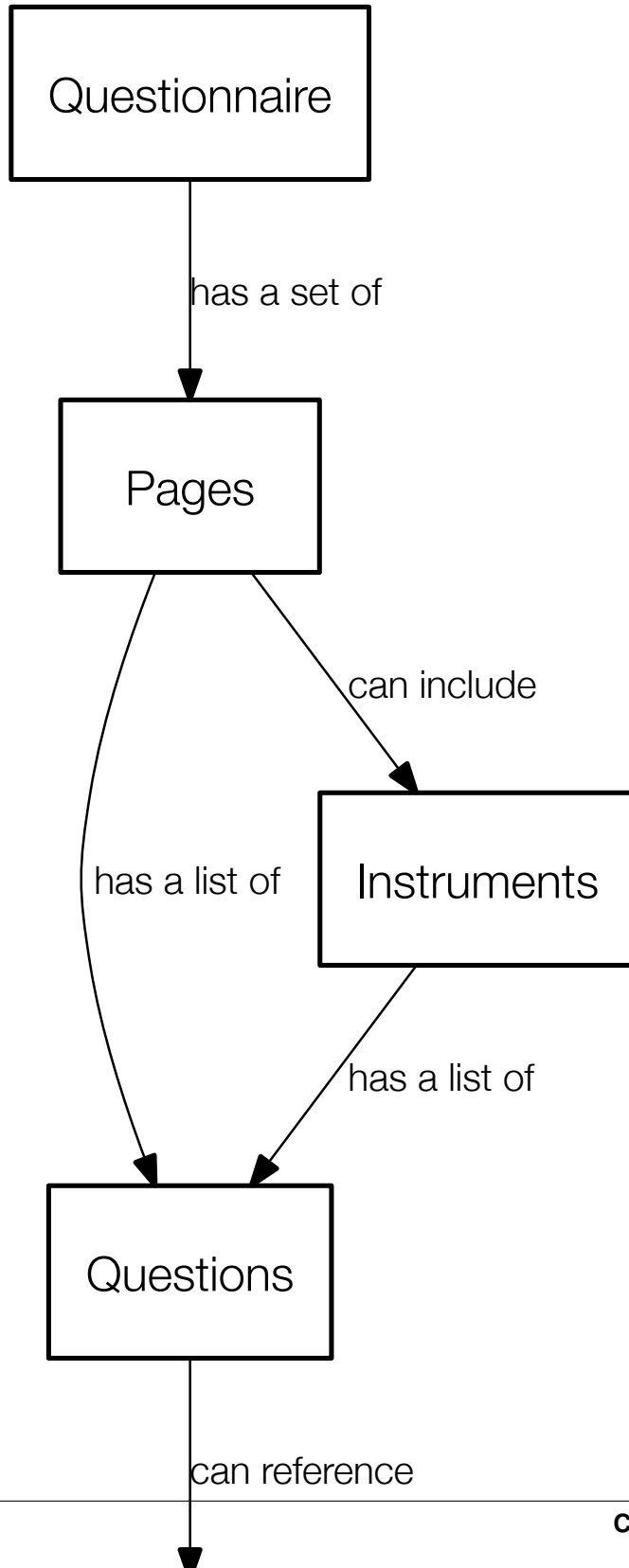
When `Observations` are due then they are executed by a background task, causing an email, SMS to be sent, or a phone call to be made. In responding, users create a `Reply` to the `Observation`: A `Reply` consists of multiple `Answers`, which represent responses to individual questions. Because a `Reply` might be interrupted or left incomplete, multiple replies can be made to each `Observation` (which the administrator needs to remember when the data are exported; they can chose a particular `Reply` to use by marking it as canonical).

The diagram below represents the core constructs within Signalbox (but is not complete... see below).



## 12.2 Questionnaires and collecting data

As noted above, Askers contain Pages which in turn contain Questions. Some questions may also refer to Instruments, which represent a bundle of Questions which are commonly shown together (e.g. a psychometric scale). When a questionnaire is displayed, 'instrument' questions automatically include this group of questions on a single page. See the diagram below:



## 12.3 Additional components which may not always be used

Script`s generate Observations, but can also generate Reminders for those Observations, and these send an additional messages to users at intervals after the Observation falls due.

Scripts can also have ScoreSheets attached to them, which are sets of rules which describe how a set of Answers in response to the Script's Asker can be reduced to a single number (e.g. the mean or total for a set of Questions). ScoreSheets create scores which can be viewed for a particular User (e.g. to check whether a user meets a criteria for study entry from a screening questionnaire). ShowIfs are rules which evaluate to a boolean (yes/no) based on Replies Users have previously made (or Replies that are in progress). ShowIfs can be used to determine whether a particular Question should be shown based on previous responses. They can also be used to create new Observations based on a Reply, using an ObservationCreator. For example, if a User completes a depression questionnaire (Asker) and a ScoreSheet computes they have scored above a certain threshold, then an additional Observation could be created and a followup email sent.

Users are linked to a UserProfile which can contain additional fields like a telephone number, address etc. A Study can specify the subset of these fields which are required when a user signs up. ContactRecords are made when the administration interface is used to send a User a message. In some studies, its necessary to collect data from some Users (e.g. therapists) about other Users (e.g. clients); where this is the case a Membership can store an additional field indicating which User the data is about, as well as who it has been collected from

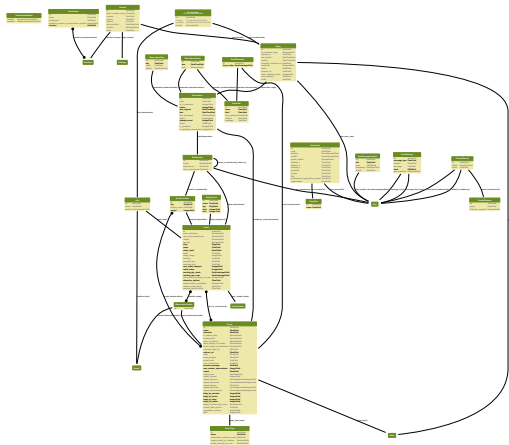
StudySites represent different locations in a multi-site trial, can can be used to filter Memberships when viewing date.



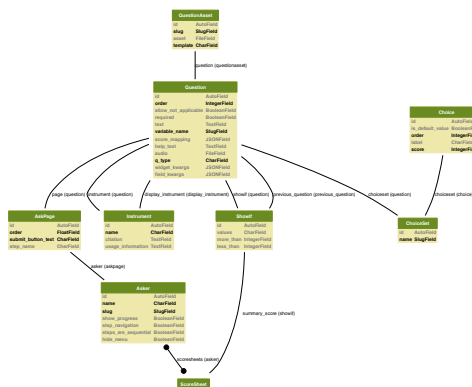
Database schema

In addition to the simplified diagrams above, the database schema for the Signalbox and Ask apps may also help clarify the structure of the application:

Download Signalbox app schema .pdf



Download Ask app schema .pdf



**Note:** Required fields in the database are displayed in **bold**

## 13.1 Technical overview

Signalbox is based on a number of excellent open source software projects. It was written using the Django web framework (i.e. in Python), and uses either an sqlite or a postgresql database (postgresql is strongly recommended). Configuration is provided to quickly host an instance using Heroku (a cloud-based application host, see <http://en.wikipedia.org/wiki/Heroku>), although self hosting is also possible.



---

**Indices and tables**

---

- *genindex*
- *modindex*
- *search*