Amber Documentation

OBiBa

Mar 24, 2024

CONTENTS

Targeted at individual studies and study consortia, OBiBa software stack (Amber, Opal, Mica etc.) provides a software solution for epidemiological data collection, management, analysis and publication. Amber is the data collection tool.

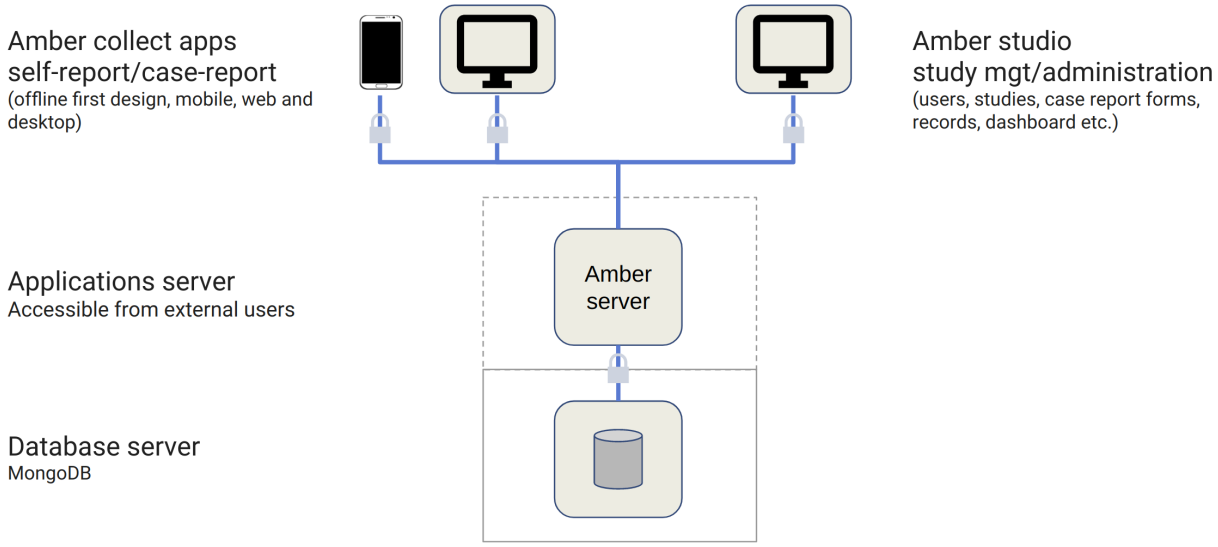
CHAPTER

ONE

INTRODUCTION

Amber is a web-based applications system for collecting case reports data based on forms. This system is composed of the Amber Server application and of several frontend applications with different purposes (content administration or data collection). Amber is designed to ensure that data is collected in an automated, efficient and secure way in order to minimize human errors. Forms can be designed, internationalized and revised without requiring programming skills.

The diagram below illustrates the relationships between the Amber server and the other frontend applications:



1.1 Amber Server

The Amber Server engine is in charge of the core logic, data storage and access controls:

- **Web services API**: all Amber functionalities are available through web services allowing various usage from multiple client applications.
- **Authentication**: signin, signup, password recovery, email validation, two-factor authentication, access token with inactivaty and expiration timeouts.
- **Authorization**: role-based and access control lists.
- **Database encryption**: critical content (individual records) are encrypted in the database.
- **Scalable**: multiple server instances can be deployed with a load balancer to support heavy usage.

1.2 Amber Studio

Amber Studio is a web application for managing Amber system content:

- **Dashboard**: basic activity metrics.
- **User/groups management**: add/delete/validate users, assign roles, grant specific permissions.
- **Studies**: basically a group of forms and records.
- **Form builder**: create new forms with a user-friendly interface and minimal programming. Multi-languages and left-right support. Rich form items: choices, open text entries, dropdowns, image selection, autocomplete, sliders, ratings, geographic features (points, lines, polygons) and more. Complex forms can be handled with skip logic, data validation and computed fields.
- **Form version control**: modify forms without interfering with on-going data collection. Always associate a record with the form revision used. Safely revert to a previous revision.
- **Form import/export**: export the complete form design and import it in another Amber system.
- **Case report form access control**: enable a CRF and restrict its access to specific users/groups.
- **Case report records**: filter records (by date, form and revision), visualize and modify them afterwards. Export in multiple formats (CSV, Excel, JSON), including full data dictionary as converted from the form.
- **Interview designs**: interviews are a sequence of interdependent forms, conducted in the context of a planned data collection campaign. A campaign includes a list of participants who are invited to answer interview questionnaires through the Amber Visit web application.
- **Interview records**: filter records (by date, interview design, campaign, status etc.). Export in multiple formats (CSV, Excel, JSON), including full data dictionary as converted from the form.

1.3 Amber Collect

Amber Collect is a web application for collecting case reports:

- **Secure**: authentication is required and authorizations apply. Two-factor authentication is supported.
- **Efficient**: interface is deliberatly minimal to focus on data collection.
- **Multi-languages**: when the form is designed in several languages, the user can easily switch to another language, with the corresponding character set and left-right layout is automatically detected and applied.
- **Offline-first design**: in case the network in not accessible, collected data are stored encrypted locally. When connection is restored, data are automatically sent to the Amber Server for permanent storage.
- **Progressive web app**: a PWA is an installable web application, facilitating its deployment on mobile devices (no app store required). See What are Progressive Web Apps?.

1.4 Amber Visit

Amber Visit is a web application for collecting interviews:

- **Secure**: authentication is required and authorizations apply. Two-factor authentication is supported.
- **Efficient**: interface is deliberatly minimal to focus on data collection.
- **Multi-languages**: when the interview is designed in several languages, the user can easily switch to another language, with the corresponding character set and left-right layout is automatically detected and applied.
- **Offline-first design**: in case the network in not accessible, collected data are stored encrypted locally. When connection is restored, data are automatically sent to the Amber Server for permanent storage.

CHAPTER

TWO

INSTALLATION

Amber is a system of several applications:

- **Amber** server: Business layer over the database: users, groups, studies, forms, case records etc. are all defined in this application and their access require authentication and authorization. It is only accessible via an API.
- **Amber Studio** web app: Content management web application, using data and services of the Amber server
- **Amber Collect** web app: Case report data collection web app, using data and services of the Amber server
- **Amber Visit** web app: Interview data collection web app, using data and services of the Amber server

These applications are written in JavaScript.

2.1 Requirements

2.1.1 Server Hardware Requirements

Component	Requirement
CPU	Recent server-grade or high-end consumer-grade processor
Disk space	8GB or more.
Memory (RAM)	Minimum: 4GB, Recommended: >4GB

2.1.2 Server Software Requirements

Amber server and apps are pure JavaScript software based on nodejs (Amber) and Quasar (Amber Studio and Amber Collect). Despite it could be possible to set up a native runtime environment, we recommend to use a containerized one (Docker), to facilitate to customization and the deployment of the software applications.

The database that is used by Amber server is MongoDB which can also be deployed from a container.

Then recommended runtime environment is:

Software	Documentation	Usage
Docker Engine	Docker Engine	Docker runtime environment
Docker Compose	Docker Compose	Docker compose plugin

2.2 Install

2.2.1 Docker Images Installation

Because the Amber system is composed of different parts (server and client applications), these need to be integrated with each other for the specific site where they will be deployed.

The set up of the different Docker images of Amber is described in the amber-template project.

The steps to follow are:

1. Download or clone the `amber-template` repository,
2. Define the Amber (server and apps) parts (configuration and/or source files) to customize,
3. Use the `Docker Compose` tool to build and run these images in your specific environment.

The environment variables that are exposed by the `amber` image are:

Environment Variable	Description
CLUSTER_COUNT	Node cluster count, defaults to all available CPU cores
APP_NAME	JWT issuer
APP_SECRET_KEY	Encryption key
APP_SECRET_IV	Secret string for the encryption's initial vector,
APP_URL	JWT audience
APP_API_KEYS	The allowed API keys (comma separated values) for triggering backround tasks execution,
CLIENT_URLS	Comma separated client urls, for the CORS policy
AMBER_STUDIO_URL	Amber Studio app url, to be included in the notification emails
AMBER_COLLECT_URL	Amber Collect app url, to be included in the notification emails
AMBER_VISIT_URL	Amber Visit app url, to be included in the notification emails
MONGODB_URL	The MongoDB connection string
ENCRYPT_DATA	Whether the patient/participant data should be encrypted in the database
OTP_TIMEOUT	Number of minutes during which the onetime password (OTP) sent by email is valid. Default is 5; when
RECAPTCHA_SECRET_KEY	The reCAPTCHA v3 secret key.
SIGNUP_WHITELIST	List of email domains that are allowed to signup. Use '*' for wild card. Default is all.
SIGNUP_BLACKLIST	List of email domains that are NOT allowed to signup. Use '*' for wild card. Default is none.
GMAIL	The Gmail user name for the notification service (optional, only if Gmail transport service is used)
GMAIL_PASSWORD	The Gmail user password for the notification service (optional, only if Gmail transport service is used)
SENDINBLUE_API_KEY	The Sendinblue API key for the notification service (optional, only if Sendinblue transport service is use
SMTP_HOST	The SMTP server host
SMTP_PORT	The SMTP server port
SMTP_NAME	The SMTP server name
SMTP_SECURE	Whether the SMTP connection should use SSL
SMTP_REQUIRE_TLS	Whether the SMTP connection should use TLS (when secure is false)
SMTP_LOGGER	Enable SMTP logging
SMTP_DEBUG	Enable SMTP debug by sending log events
SMTP_USER	The SMTP server user
SMTP_PASSWORD	The SMTP server user's password
FROM_EMAIL	The automated sender email address,
ADMINISTRATOR_EMAIL	User seeding when there is no administrator in the database
ADMINISTRATOR_PWD	User seeding when there is no administrator in the database
LOG_LEVEL	Logger level (*error* ` *warn* ` *info* ` *verbose* ` *debug* ` *silly* ` etc. (see [winstonjs](https://github.com/winston
LOG_FILE	File logger path
LOG_FILE_LEVEL	File logger level when *LOG_FILE* is specified, default is *LOG_LEVEL*

Table

Environment Variable	Description
`NODE_ENV`	Name of the config file to be merged with the default one (e.g. *production*)

The environment variables that are exposed by the `amber-studio`, `amber-collect` and `amber-visit` images are:

Environment Variable	Description
`AMBER_URL`	The URL to the Amber server.
`RECAPTCHA_SITE_KEY`	The reCAPTCHA v3 site key. If not provided, the Sign Up page will not be activated.

2.3 Upgrade

The upgrade procedures are handled by the application itself.

2.3.1 Troubleshooting

If you encounter an issue during the installation and you can't resolve it, please report it in our Amber Issue Tracker.

CHAPTER

THREE

CONFIGURATION

3.1 Amber Server

See the config/default.json settings in the source code for reference.

Some of these settings can be overridden by environment variables (`.env` file supported, see Dotenv). The list of the environment variables is described in the Amber Environment Variables documentation.

3.1.1 Main Settings

Property	Description
`host`	Default is "localhost"
`port`	Default is `3030`.
`encrypt_data`	Whether the collected data are encrypted in the database. Default is `false`.
`encrypt_iv`	Encryption initial vector, to be modified. Overridden by the `APP_SECRET_IV` environment variable.
`mongodb`	MongoDB connection string. Overridden by the `MONGODB_URL` environment variable. Default is "mongodb://localhost:27017/amber"
`client_urls`	Allowed client URLs (comma separated) for CORS check. Wildcard "*" is supported. Overridden by the `CLIENT_URLS` environment variable. Default is "*".
`amber_studio_url`	The Amber Studio URL to be included in notification emails. Default is "http://localhost:3080".
`amber_collect_url`	The Amber Studio URL to be included in notification emails. Default is "http://localhost:3090".
`api_url`	The Amber Server URL. Overridden by the `APP_URL` environment variable. Default is ": "http://localhost:3030/".
`recaptcha_secret_key`	The reCAPTCHA v3 secret key. Overridden by the `RECAPTCHA_SECRET_KEY` environment variable.

3.1.2 Authentication Settings

See Feathers Authentication Service and Feathers Local Authentication documentations for more details.

Property	Description
`authentication.secret`	The authentication token (JWT) signing secret, also used for encrypting individual data (when enabled). Overridden by the `APP_SECRET_KEY` environment variable.
`authentication.jwtOptions.audience`	The JWT audience field. Overridden by the `APP_URL` environment variable. Default is "https://example.com".
`authentication.jwtOptions.issuer`	The JWT issuer field. Overridden by the `APP_NAME` environment variable. Default is "amber".
`authentication.jwtOptions.expiresIn`	Timeout after which the authentication token (JWT) expires. Default is `60d`.
`authentication.totp2faRequired`	Whether the two-factor mechanism "Time based One Time Password" is enabled. Default is `true`.
`authentication.activityTimeout`	Timeout after which the authentication token (JWT) expires when not used. Default is `5d`.

3.1.3 Notification Email Settings

Mail Service

By default, the mail service tries to connect to a SMTP server. Alternatively, other mail transport providers can be used: the SMTP server settings can be overridden by the the `GMAIL`/`GMAIL_PASSWORD` or the `SENDINBLUE_API_KEY` environment variables if defined. See sendinblue docs for more details on this service.

See Nodemailer SMTP documentation for more details.

Property	Description
`smtp.host`	The SMTP server host name.
`smtp.name`	The SMTP server (optional) name.
`smtp.user`	The SMTP server user name.
`smtp.pw`	The SMTP server user password.
`smtp.secure`	The SMTP server secure flag. Default is `true`.
`smtp.require_tls`	Whether SMTP server requires TLS. Default is `false`.
`smtp.logger`	Whether SMTP server logging is enabled. Default is `false`.
`smtp.debug`	Whether SMTP server debug is enabled. Default is `false`.

Mail Templates

Mail templates are defined per language. They can be specified inline or in separate files.

Example of the `resetPwd` email template specified inline:

```
{
  "en": {
    "subject": "[{app_name}] Successfully Reset Password",
    "html": "<html><p>Dear {firstname} {lastname},</p> <p>The password was reset␣
↪successfully.</p> <p>---<br/>This email was automatically sent, please do not reply.</
↪p></html>"
  }
}
```

Example of the `resetPwd` email template specified in a local file:

```
{
  "en": {
    "subject": "[{app_name}] Successfully Reset Password",
    "file": "/path/to/resetPwd-en-template.html"
  }
}
```

Property	Description
`from_email`	The sender's email address. Default is "no-reply@example.org".
`email_templates.notifySignup`	Informs the administrators that a new user has signed up.
`email_templates.resendVerifySignup`	Sends link to verify email after sign up (resent).
`email_templates.verifySignup`	Sends link to verify email after sign up.
`email_templates.sendResetPwd`	Sends link to reset password email.
`email_templates.resetPwd`	Informs the password was reset successfully.
`email_templates.passwordChange`	Informs the password was updated successfully.

3.1.4 Other Settings

Property	Description
`export.entity_type`	Entity type to be specified in the exported data dictionary. Default is "Participant".
`export.identifier_variable`	Identifier variable name to be used in the exported dataset: Default is "id".

3.2 Amber Studio

Amber Studio is a Single Page Application (SPA) that needs to be built specifically for the deployment environment (Amber server URL and reCAPTCHA site key are to be provided). Then the configuration is used in the build phase. To facilitate the customization of the app, the Amber Studio's settings.json can be amended.

Property	Description
`theme`	Some CSS classes to be applied on different components to minimally alter the style.
`licenses`	Licenses that can be applied to forms.
`i18n`	Translations, per language. New languages can be added.

3.3 Amber Collect

Amber Collect is a Single Page Application (SPA) that needs to be built specifically for the deployment environment (Amber server URL and reCAPTCHA site key are to be provided). Then the configuration is used in the build phase. To facilitate the customization of the app, the Amber Collect's settings.json can be amended.

Property	Description
`theme`	Some CSS classes to be applied on different components to minimally alter the style.
`lock`	Lock feature can be enabled, with an optionally shuffled numeric pad.
`links`	Useful links.
`licenses`	Licenses that were applied to forms.
`i18n`	Translations, per language. New languages can be added.

3.4 Reverse Proxy Configuration

Amber server can be accessed through a reverse proxy server.

Apache

Example of Apache directives that:

- redirects HTTP connection on port 80 to HTTPS connection on port 443,
- specifies acceptable protocols and cipher suites,
- refines organization's specific certificate and private key.

```
<VirtualHost *:80>
    ServerName amber.your-organization.org
    ProxyRequests Off
    ProxyPreserveHost On
    <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>
    RewriteEngine on
    ReWriteCond %{SERVER_PORT} !^443$
    RewriteRule ^/(.*) https://amber.your-organization.org:443/$1 [NC,R,L]
</VirtualHost>
<VirtualHost *:443>
    ServerName amber.your-organization.org
    SSLProxyEngine on
    SSLEngine on
    SSLProtocol All -SSLv2 -SSLv3
    SSLHonorCipherOrder on
    # Prefer PFS, allow TLS, avoid SSL, for IE8 on XP still allow 3DES
    SSLCipherSuite "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384␣
↪EECDH+ECDSA+SHA256 EECDH+aRSA+SHA384 EECDH+aRSA+SHA256 EECDH+AESG CM EECDH EDH+AESGCM␣
↪EDH+aRSA HIGH !MEDIUM !LOW !aNULL !eNULL !LOW !RC4 !MD5 !EXP !PSK !SRP !DSS"
    # Prevent CRIME/BREACH compression attacks
    SSLCompression Off
    SSLCertificateFile /etc/apache2/ssl/cert/your-organization.org.crt
    SSLCertificateKeyFile /etc/apache2/ssl/private/your-organization.org.key
    ProxyRequests Off
    ProxyPreserveHost On
    ProxyPass / http://localhost:3030/
    ProxyPassReverse / http://localhost:3030/
</VirtualHost>
```

For performance, you can also activate Apache's compression module (mod_deflate) with the following settings (note the json content type setting) in file */etc/apache2/mods-available/deflate.conf*:

```
<IfModule mod_deflate.c>
  <IfModule mod_filter.c>
      # these are known to be safe with MSIE 6
      AddOutputFilterByType DEFLATE text/html text/plain text/xml
      # everything else may cause problems with MSIE 6
      AddOutputFilterByType DEFLATE text/css
      AddOutputFilterByType DEFLATE application/x-javascript application/javascript␣
↪application/ecmascript
      AddOutputFilterByType DEFLATE application/rss+xml
      AddOutputFilterByType DEFLATE application/xml
      AddOutputFilterByType DEFLATE application/json
  </IfModule>
</IfModule>
```

Recommended security headers are (to be added to the `apache2.conf` file, requires `headers` module):

```
# Security Headers, see https://securityheaders.com/
Header set Strict-Transport-Security "max-age=63072000"
Header set X-Frame-Options DENY
Header set X-XSS-Protection 1;mode=block
Header set X-Content-Type-Options nosniff
Header set Content-Security-Policy "frame-ancestors 'none'"
Header set Referrer-Policy "same-origin"
Header set Permissions-Policy "fullscreen=(self)"
Header set X-Permitted-Cross-Domain-Policies "none"
Header set Expect-CT: max-age=0
Header onsuccess edit Set-Cookie ^(.+)$ "$1;HttpOnly;Secure;SameSite=Strict"
```

CHAPTER

FOUR

INTRODUCTION

The Amber Studio Application is the administration web interface of the Amber server. It is used for:

- Administrating user access (add/remove users, assign roles, groups membership),
- Managing studies, with their forms and collected data,
- Editing user's personal profile information.

4.1 Requirements

This web interface is a JavaScript application requiring a modern web browser. There is no requirement regarding the operating system.

CHAPTER

FIVE

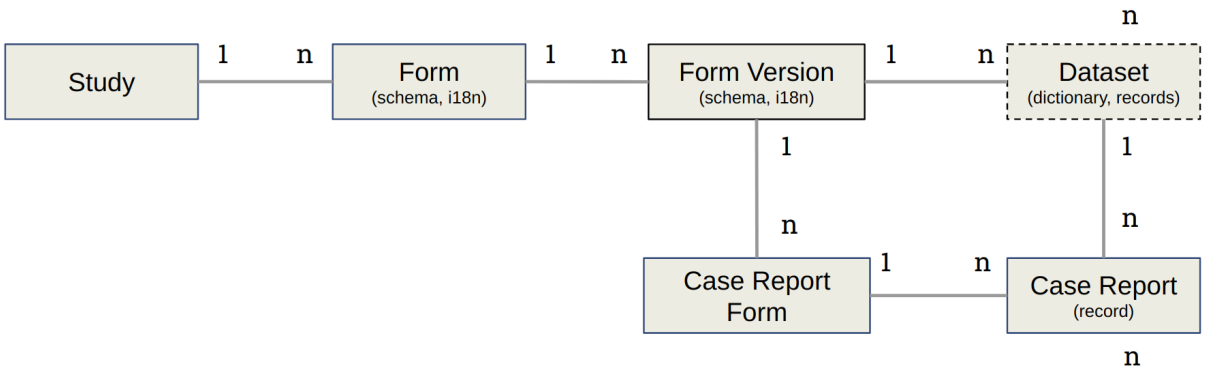
CONTENT

This section is where the studies are designed and from where the data records will be extracted. It is not accessible to `Guest` users, accessible read-only to `Interviewer` users, and editable by `Manager` and `Administrator` users.

The Amber data model is the following:

- A **Study** has **Forms**,
- A **Form** has a schema (the description of fields/items, with multilanguage support) and associated **Form versions**,
- A **Case Report Form** describes the context of a data collection with a **Form version**,
- A **Case Report** is a record of data that were captured using a **Case Report Form**.

An extraction of records constitutes a **Dataset**, i.e. a synthesis of a data dictionary (built from the Form schema, in a specific version) and the data (the records).



5.1 Studies

A study is the workspace for designing forms, triggering case report data collection and extracting records.

5.1.1 List studies

From the list of studies the following operations are available:

Add a study

When creating a study, a name is required, a description is optional.

Delete study(ies)

To delete several studies:

- Select one or more studies in the list,
- Press the Delete selected studies button and confirm.

You can also delete a specific study by pressing the study's Delete action button.

Note: The removal of a study will fail when there are associated case reports. Make sure to extract the recorded data (if necessary) before removing them.

5.1.2 Single study

To navigate to a study page, click on the study's name or the pencil icon. The page that is opened lists the forms, the case report forms and the records of the study.

Edit study

It is possible to edit the study name and description by clicking on the cog icon.

List forms

See *Forms* documentation.

List case report forms

See *Case Report Forms* documentation.

List records

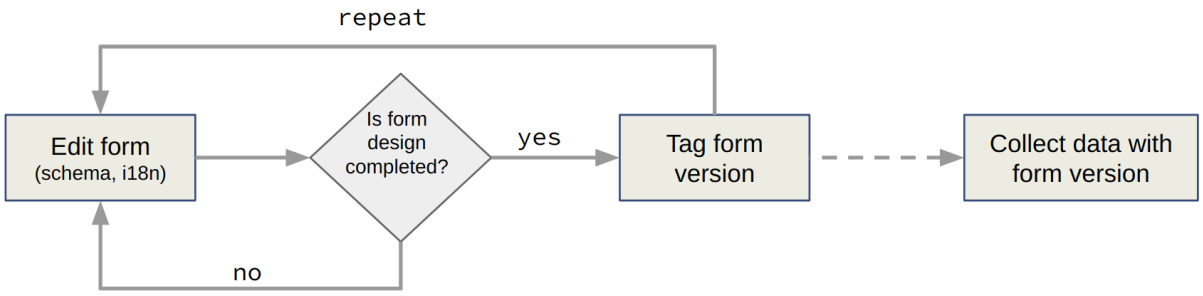
See *Records* documentation.

5.2 Forms

The forms are the heart of Amber. A form describes how the data will be collected, with a built-in support for multi-language. The form's **schema** is made of:

- **Items**, which can be data input fields, structural items (group, section) or computed fields.
- **Translations**, which is a list of translation keys with values for each language.

Different versions of a form can be saved, which allows to safely edit a form while a version is used for data capture. Form versions also ensure that the data collected are properly described by the form that was used to collected them, avoiding variable renaming errors, options addition/removal conflicts, changes in the labels which could affect the question semantic etc.



5.2.1 List of forms

From the list of forms the following operations are available:

Add a form

When creating a form, a name is required, a description is optional. This is purely informative for management purpose and will not appear when the form is administered.

It is possible to pre-populate the form with a schema that was previously exported from another form. This is optional and can also be done later.

Delete form(s)

To delete several forms:

- Select one or more forms in the list,
- Press the Delete selected forms button and confirm.

You can also delete a specific form by pressing the form's Delete action button.

Note: The removal of a form will fail when there are associated case reports. Make sure to extract the recorded data (if necessary) before removing them.

5.2.2 Single form

To navigate to a form page, click on the form's name or the pencil icon. The page that is opened has two tabs:

- The schema tab, with a form item's builder and translations editor,
- The list of the form's versions.

Export

Export the form's schema in text file (JSON format). Advanced users can edit this file, which can be convenient for the translations section for instance.

Import

Import a previously exported form schema. This operation will replace the whole tree of items and the translations.

Tag

Creates a new form revision. Giving a comment that describes the changes in the form is recommended.

Items

The form items is a tree which root is the form itself.

The following form item types are available:

Type	Category	Description
Auto Complete	Select	Select a single or multiple values by filtering a (long) list of options.
Dropdown	Select	Select a single or multiple values in a menu.
Image Select	Select	Select a single or multiple areas on an image.
Multiple Choices	Select	Checkbox choices.
Single Choice	Select	Radio choices.
Date	Temporal	Date value, in `yyyy-MM-dd` format (ISO 8601).
Date and Time	Temporal	Date and time value, in `yyyy-MM-dd mm:ss` format (ISO 8601).
Time	Temporal	Time value, in `mm:ss` format.
Geographic data	Geographic	Select a single or multiple geographic features (point or polygon).
Number	Number	Input field with numeric value.
Rating	Number	Gradual numeric values, rendered as icons.
Slider	Number	Gradual numeric values, rendered as a slider.
Paragraph	Text	Long text.
Short Answer	Text	Short text.
Group	Structure	Group one or more items.
Section	Structure	Information text.
Toggle	Logical	True or False.
Computed	Other	Not visible, data is computed with the other item's data.

The following definition settings are common to all items:

Setting	Description
`Type`	The type of item (see above).
`Name`	The name of the item is not visible: it will be associated to the data collected.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are there meaning etc.
`Required`	Whether an input is mandatory (does not apply to structural items such as Group and Section).

Some settings are expressed by a script which is evaluated at each data entry.

Setting	Description
`Condition`	The condition makes an item visible or not depending of other data. It is a small script which returns a logical value. True when item is visible.
`Validation`	True when item value is valid. Does not apply to items not capturing data (such as *Group* or *Section*).

The scripting language is Javascript. It allows to express complex logic with a single line. See detailed explanations:

- *How to make fields conditional?*
- *How to validate field values?*

Translations

The translations are a list of tuples:

- A translation key,
- Associated texts for each of the defined languages.

The translation key can be a real text (example: "What is your age?"), but to avoid confusion when the wording of the question changes it is recommended to use a short (yet meaningful enough) text (example: "age_label"). When a language has no translation for a given key, the fallback language is English (`en`). If no translation can be found for the fallback language, the translation key is returned.

Add translation

Add a translation key

A translation key can be added manually. First enter the key in the dialog, then once a new line has been added in the list of translations, click in each language cell to set the translated text. The list of translations will be saved automatically.

Merge with items translations keys

This operation automatically adds translation keys for each item setting representing a text to display (Label, Description, Hint, Close label etc.). If the translation key already exists, it is not modified, only new entries are added. See *How to translate field labels?* for details.

5.3 Form Items

List of the items that can be included in a form.

5.3.1 Auto Complete

The auto complete field captures a text or a list of texts. It has some "options", i.e. possible values that are proposed for selection after typing a few characters (at least two). It is also possible to use these options as suggestions only and allow entering new values.

This field is appropriate when there is a long list of options (place names for instance) and for reducing text variations with an open field.

Preview

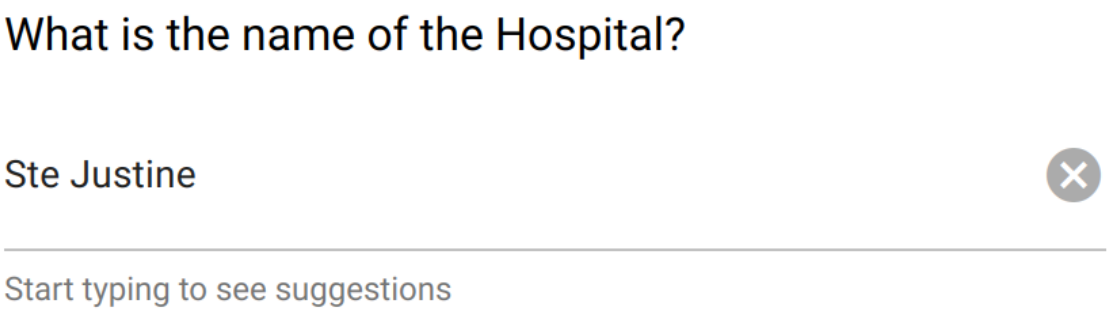


Fig. 1: The auto complete field helps with the formatting of open text entries.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Hint`	Help text below the input field.
`Default`	The default value. Only a single value can be specified, even when `Multiple choices` is set.
`Multiple choices`	When selected, the captured data is a list of texts. Default is `false`, i.e. a single value is captured.
`Options`	The list of predefined values, either to be selected or used as suggestions. For provisioning a lot of options, see *How to make a select with a lot of options?*
`Can enter new value`	When selected, an entry that is not in the list of options can be added. Default is `false`, i.e. a valid entry must be one of the options.

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.2 Dropdown

The dropdown field captures a text or a list of texts. It has some "options", i.e. possible values that are proposed in a dropdown menu.

This is the appropriate field for a compact rending of choices (as opposed to *Single Choice* or *Multiple Choices* fields), and when there are not too much options (see the *Auto Complete* field).

Preview

Fig. 2: The dropdown field proposes a list of choices to select in a menu.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Hint`	Help text below the input field.
`Default`	The default value. Only a single value can be specified, even when `Multiple choices` is set.
`Multiple choices`	When selected, the captured data is a list of texts. Default is `false`, i.e. a single value is captured.
`Options`	The list of predefined values, either to be selected or used as suggestions. For provisioning a lot of options, see *How to make a select with a lot of options?*

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.3 Image Select

The dropdown field captures a text or a list of texts. It has some "options", i.e. possible values that are proposed in a dropdown menu. In addition to these options, an image is to be defined, with some clickable areas. Each image area triggers an option selection. Selecting an option also highlights the corresponding image area.

This field is recommended for having a graphical representation of the options to select. The image can be accessed via an URL or embedded in the form (convenient for offline usage).

Preview

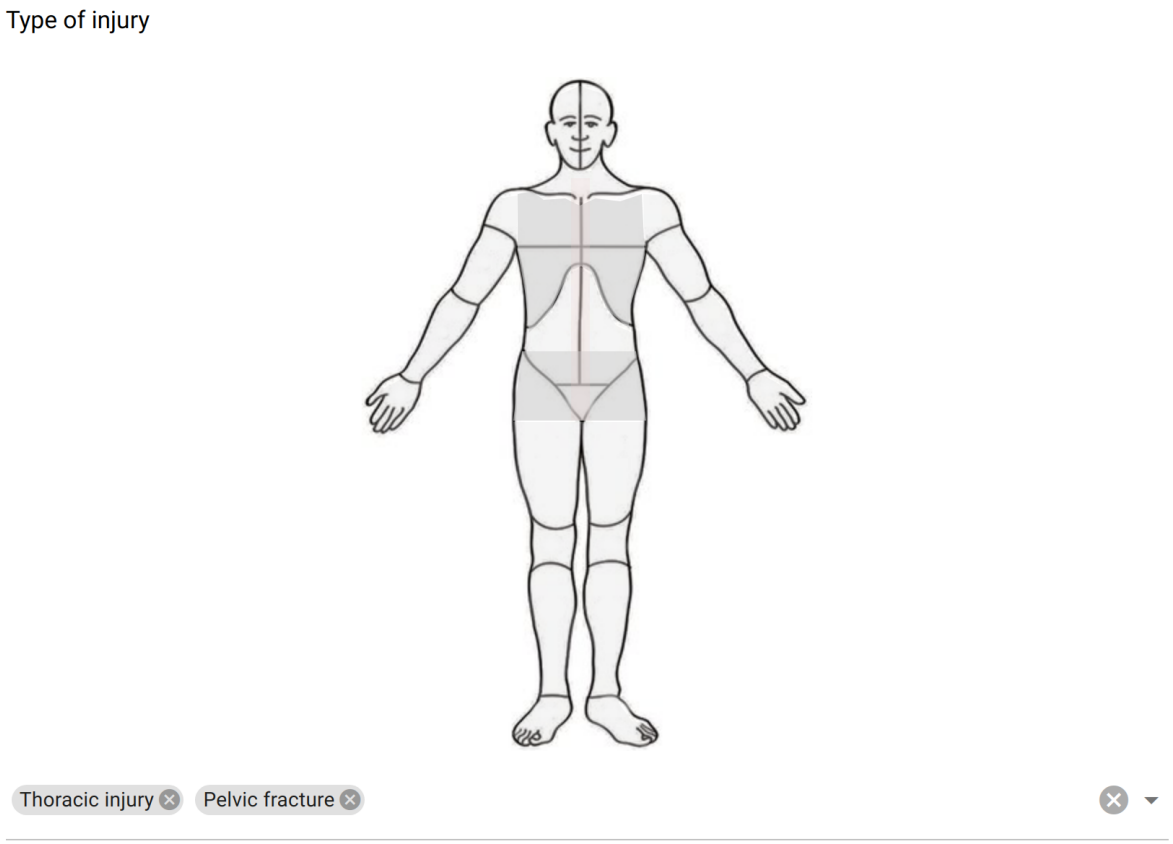


Fig. 3: The image select field proposes a clickable image and (optionally) a list of choices to select in a menu.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Hint`	Help text below the input field.
`Default`	The default value. Only a single value can be specified, even when `Multiple choices` is set.
`Multiple choices`	When selected, the captured data is a list of texts. Default is `false`, i.e. a single value is captured.
`Options`	The list of predefined values, either to be selected or used as suggestions. For provisioning a lot of options, see *How to make a select with a lot of options?*
`Show options selection widget`	The dropdown menu with the options can be hidden. By default it is visible.

Image

Property	Definition
`Image source`	The image source can be a link to an image available on the internet (requires network access). It can also be embedded within the form in base64 format. To help with this operation, the `Upload image` field can be used: select a file (`.jpg` or `.png`), and the image source will be automatically filled in.
`Image areas`	An image area is composed of: a value (the data collected, which must match an option's value), a color (opacity will change when area is selected) and a list of points. Several areas can select the same value. Points are space-separated x,y coordinates and are describing a polygon shape (rectangle and circle shapes are not supported). To help with the area coordinates design, some image map tools are avaliable online (for instance Image Map Generator). These areas can be defined in a CSV (comma separated) or TSV (tab separated) file and uploaded. See details and examples in *How to make an image map?*

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*
`Image class`	CSS class to apply to the image. For instance the image can be centered using `text-center`.

5.3.4 Multiple Choices

The multiple choices item is for selecting a list of texts. It has some "options", i.e. possible values that are proposed with checkboxes.

This is the appropriate field for having the choices immediately visible (as opposed to *Dropdown* or *Auto Complete* fields), and when there are not too much options.

Preview

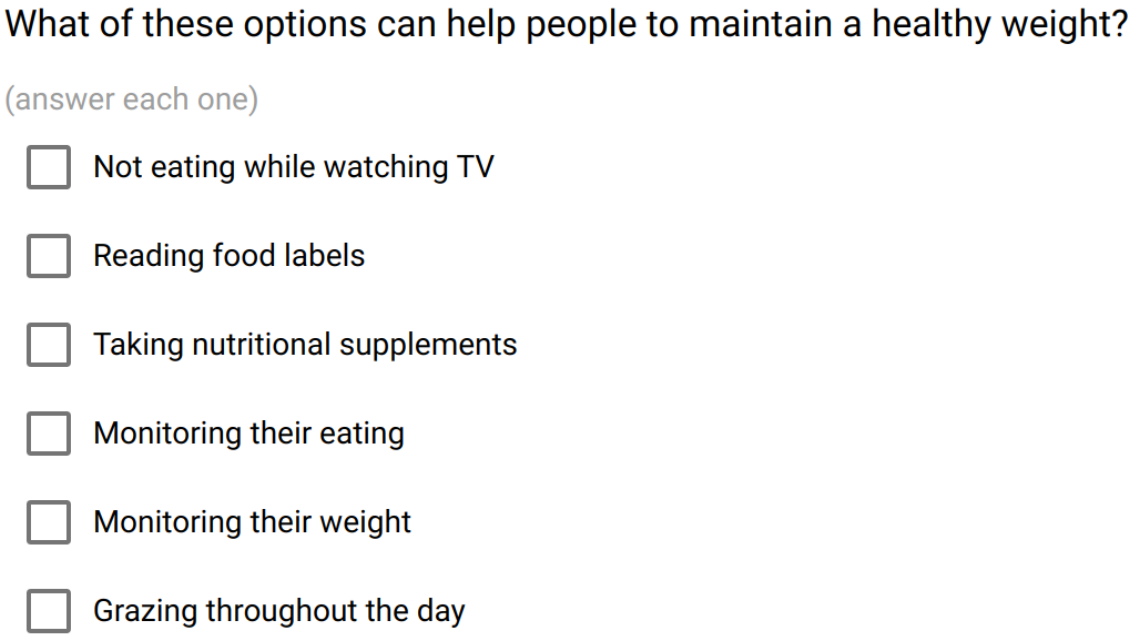


Fig. 4: The multiple choices field proposes a list of choices to select.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Default`	The default value. A single value can be specified, or an array of values (for instance, for selecting 1 and 3, enter default value `["1","3"]`).
`Options`	The list of predefined values to be selected. For provisioning a lot of options, see *How to make a select with a lot of options?*

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.5 Single Choice

The single choice item is for selecting a text. It has some "options", i.e. possible values that are proposed with radios.

This is the appropriate field for having the choices immediately visible (as opposed to *Dropdown* or *Auto Complete* fields), and when there are not too much options.

Preview

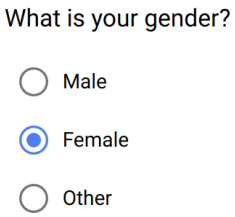


Fig. 5: The single choice field proposes to select an option.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Default`	The default value. A single value can be specified.
`Options`	The list of predefined values to be selected. For provisioning a lot of options, see *How to make a select with a lot of options?*

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.6 Date

This field is for capturing a date value. It is expressed as a string in ISO 8601 format: `yyyy-MM-dd` (4 digits year, 2 digits month, 2 digits day).

Preview

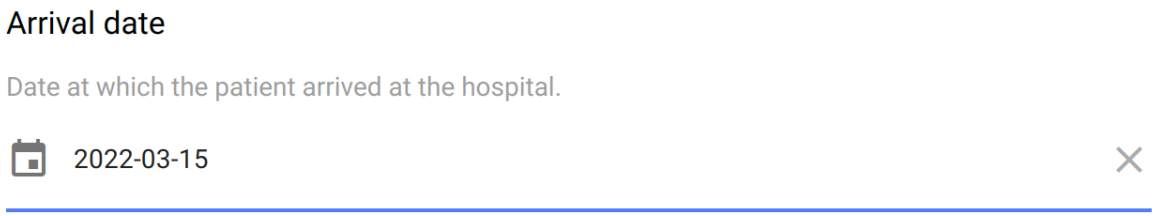


Fig. 6: The date field is in ISO 8601 format.

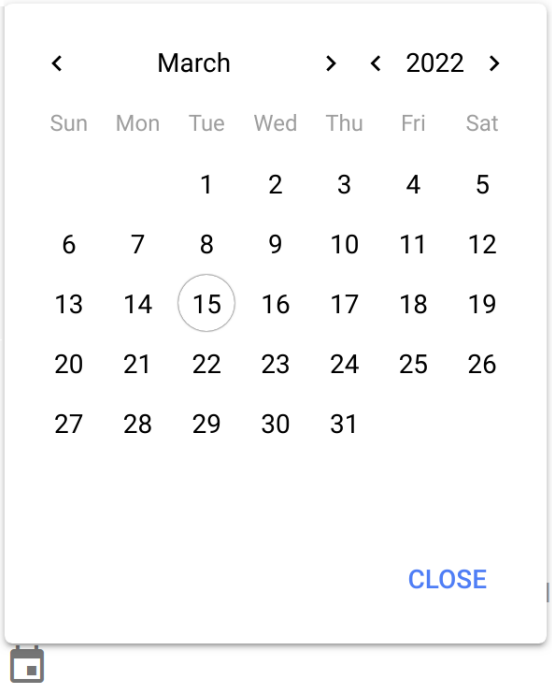


Fig. 7: The date selector allows to navigate in the years/months. The close button can be translated.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Hint`	Help text below the input field.
`Close`	The close button's label. Use a translation key when you want to have it in several languages.
`Default`	The default value. Make sure it is in the right format.

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.7 Date and Time

This field is for capturing a date and time value. It is expressed as a string in ISO 8601 format: `yyyy-MM-dd hh:mm` (4 digits year, 2 digits month, 2 digits day, 2 digits hour, 2 digits minutes).

Preview

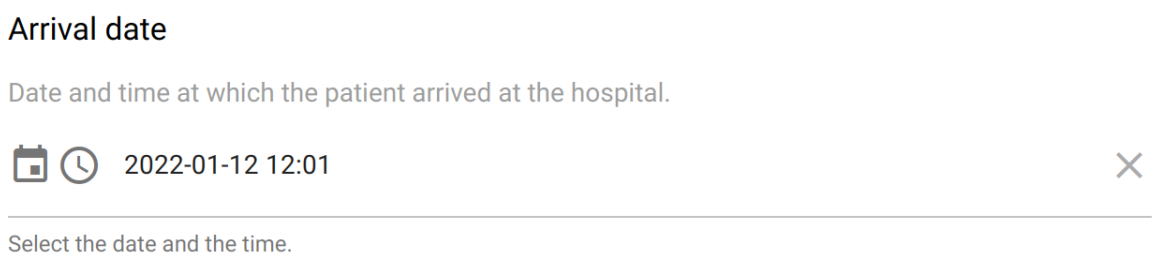


Fig. 8: The date field is in ISO 8601 format.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

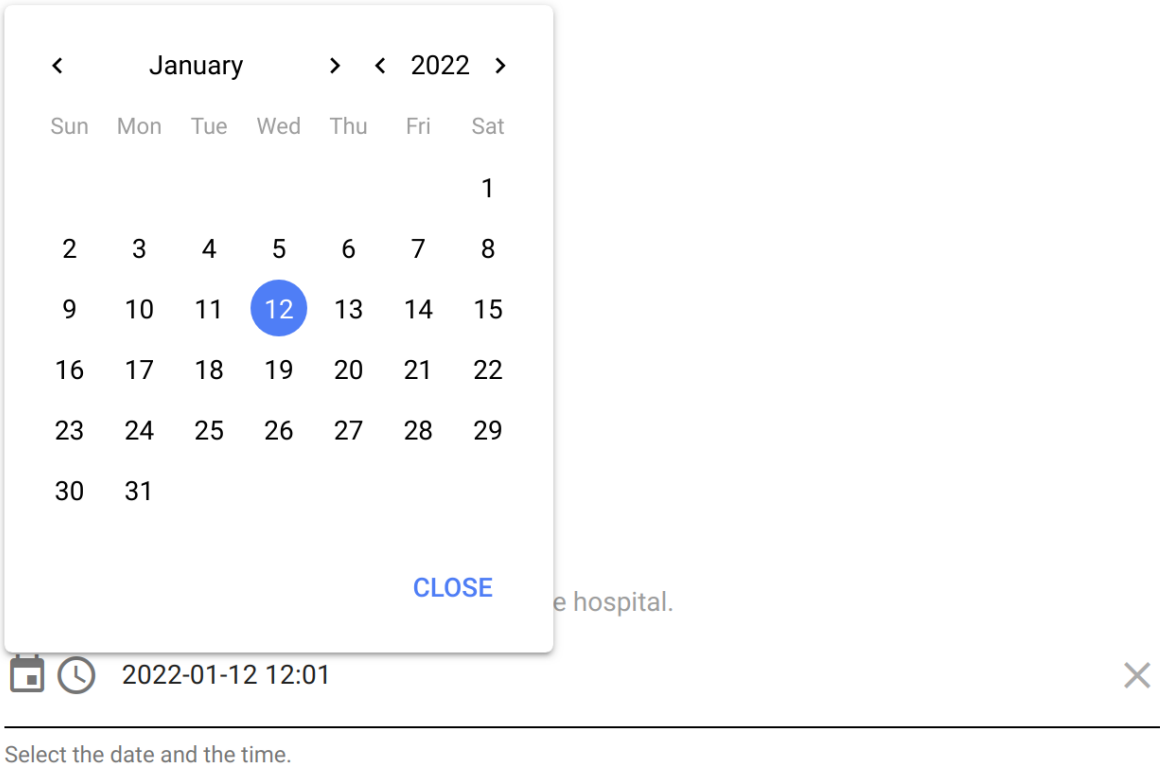


Fig. 9: The date selector allows to navigate in the years/months. The close button can be translated.

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

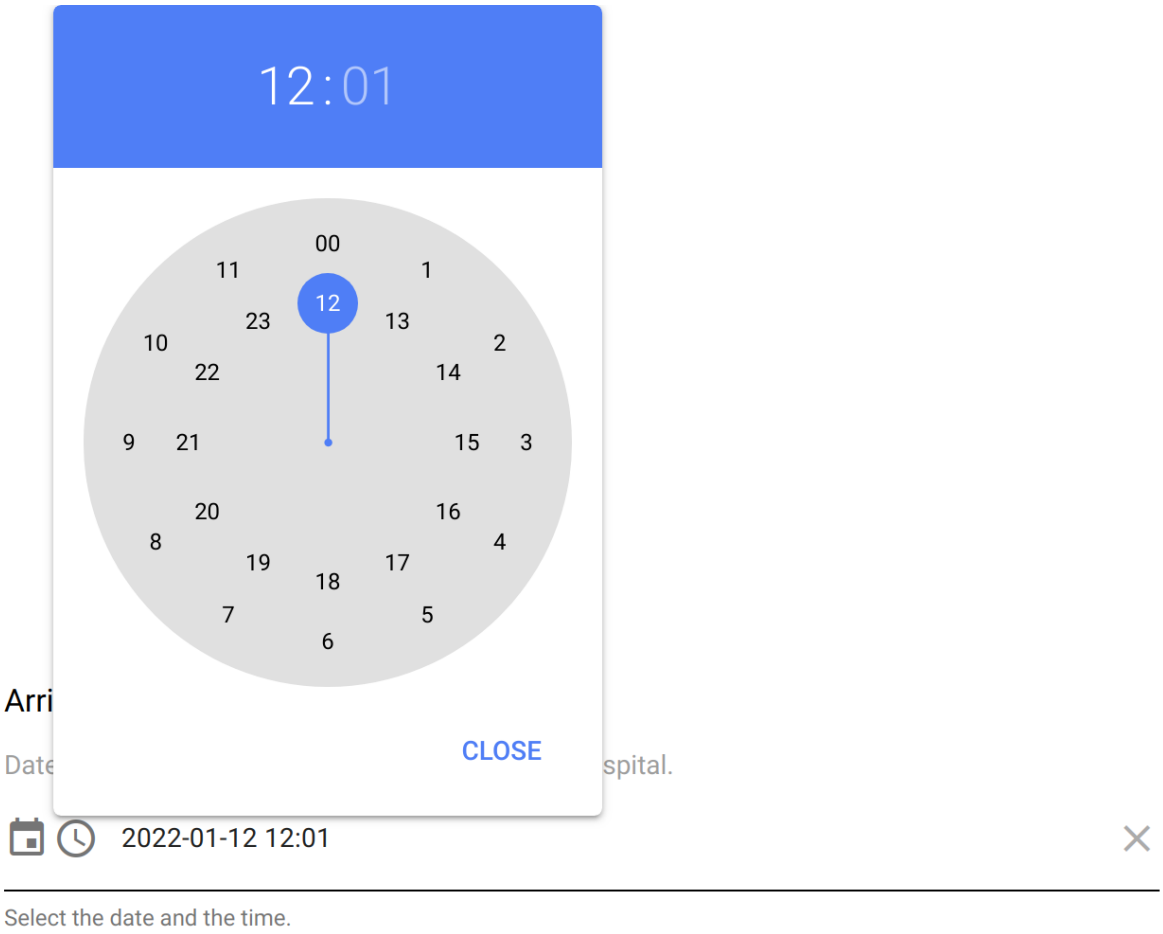


Fig. 10: The time selector allows to select the hour and minutes in the day. The close button can be translated.

Settings

Property	Definition
`Hint`	Help text below the input field.
`Close`	The close button's label. Use a translation key when you want to have it in several languages.
`Default`	The default value. Make sure it is in the right format.

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.8 Time

This field is for capturing a time value. It is expressed as a string in ISO 8601 format: `hh:mm` (2 digits hour, 2 digits minutes).

Preview

Fig. 11: The time field is in ISO 8601 format.

Design

Definition

The standard properties apply:

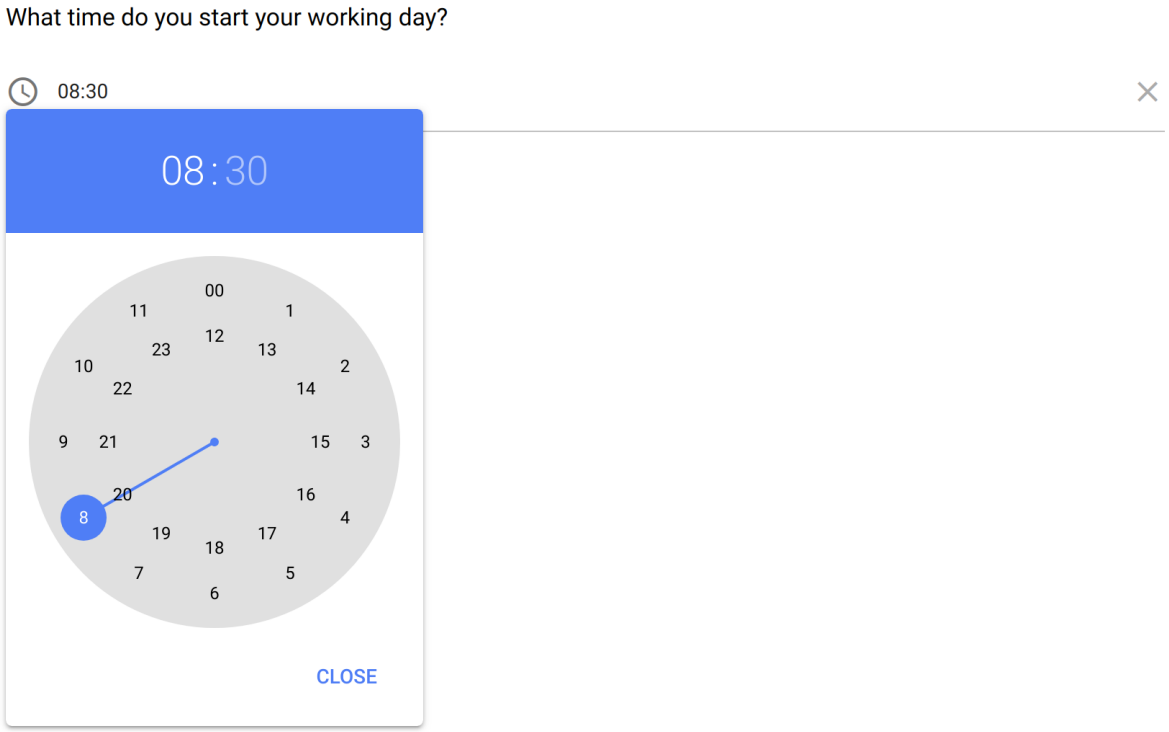


Fig. 12: The time selector allows to select the hour and minutes in the day. The close button can be translated.

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Hint`	Help text below the input field.
`Close`	The close button's label. Use a translation key when you want to have it in several languages.
`Default`	The default value. Make sure it is in the right format.

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.9 Geographic data

Geographic data can be of different type: a point (single location) or a polygon (area). This field captures such geometric features in GeoSON format. It is possible to capture multiple geographic data (of the same type).

Preview

Some controls are available on the map:

- the +/- buttons to zoom in/out,
- the fullscreen button,
- the trash icon to delete all the selections,
- the target icon to center the map.

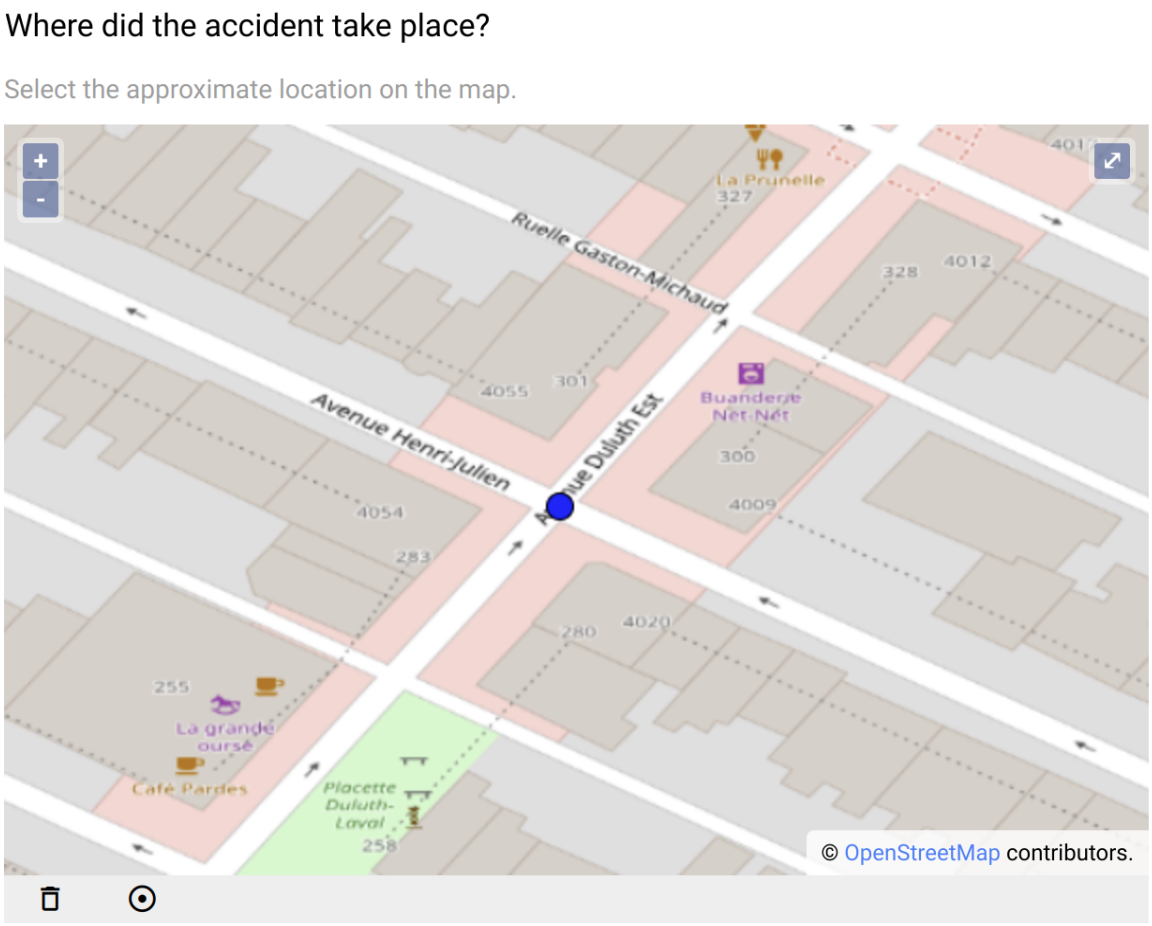


Fig. 13: Single location selection on the map.

Example of geographic data (a point in GeoJSON format):

```
{
   "type": "Feature",
```

(continues on next page)

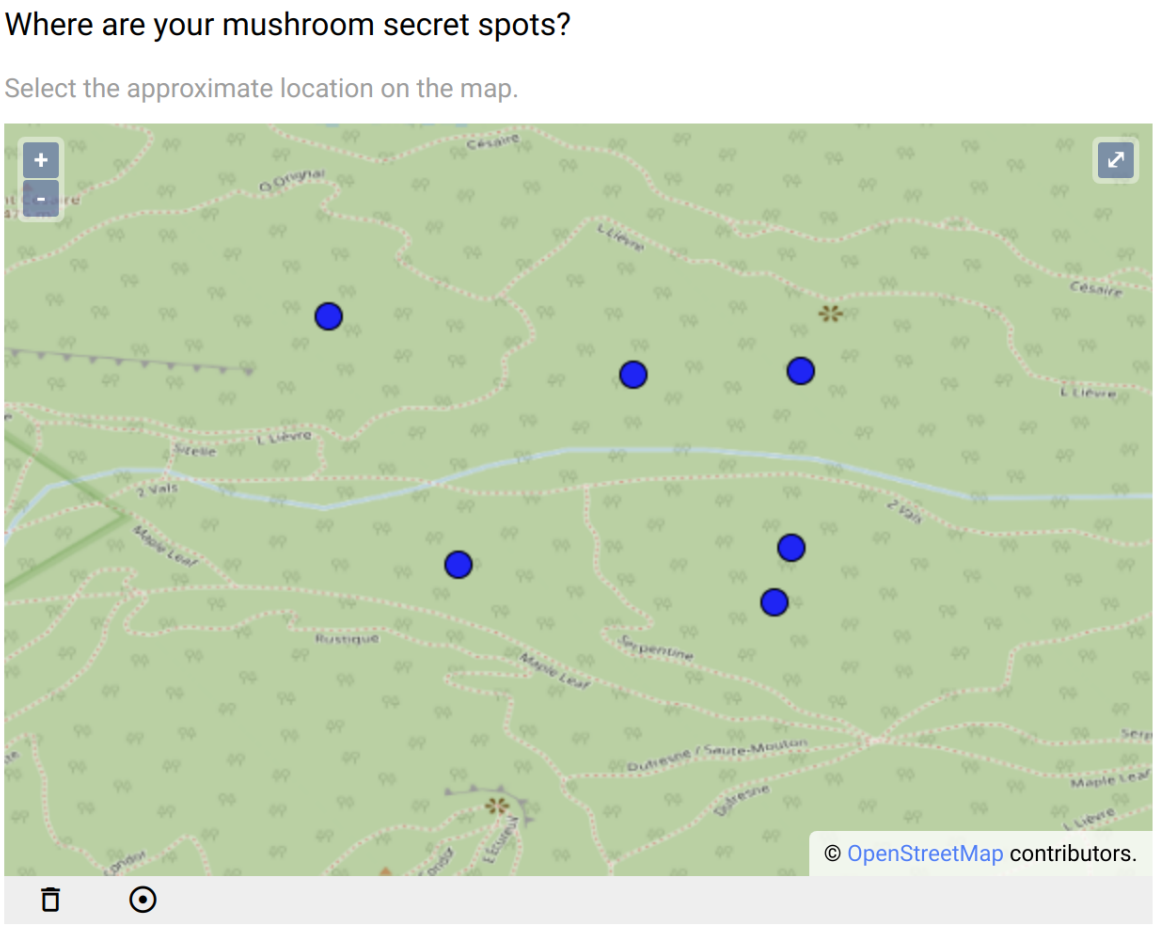


Fig. 14: Multiple locations selection on the map.

What is your preferred neighborhood?

Select an area on the map.

Fig. 15: Selection of an area on the map.

(continued from previous page)

```
  "geometry": {
    "type": "Point",
    "coordinates": [
      -74.18465727715618,
      46.03587230951427
    ]
  },
  "properties": null
}
```

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Default`	The default value. When `Multiple choices` is set, an array of GeoJSON objects is expected, otherwise a single GeoJSON can be provided.
`Geometery type`	As per the GeoJSON standard: *Point* or *Polygon*.
`Map center`	Position of the center of the map, using format: **[longitude,latitude]**. Default is `[0,0]`. When clicking on the target icon, the map will be repositioned at that location. If the geo location is activated, the starting point will be the map center, then the current position as reported by the device will be used.
`Zoom`	Zoom to apply when the map center is defined.
`With geo location`	The device will report the current position (usually a combination of positions from GPS, mobile network, internet entry point).
`Multiple choices`	When selected, the captured data is a list of geometric features. Default is `false`, i.e. a single value is captured.

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.10 Number

This item is for capturing a number value, with a standard input field. There is no restriction regarding the number value that can be entered: one should specify some `Validation` rule in order to ensure the number is in a valid range and/or is an integer vs. float number.

See also the *Slider* or *Rating* items to get more control on the data entry.

Preview

Fig. 16: The number field using standard input.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Hint`	Help text below the input field.
`Default`	The default value. Make sure it is in the right format.

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.11 Rating

This item is for capturing a number value, on a scale starting from 1.

See also the *Slider* item when the range of values is large.

Preview

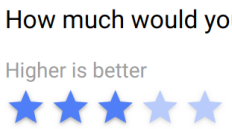


Fig. 17: The rating field consists of selecting icons.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Default`	The default value. Make sure it is in the right format.
`Maximum`	Maximum possible value (starting from 1).

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*
`Icon`	The name of the Material Icon to display. Default is `star`.
`Size`	Size of the icon in CSS units, including unit name or standard size name (xs|sm|md|lg|xl). Examples: `16px`, `2rem`, `xs`, `md`.
`Color`	Color name in the color palette.

5.3.12 Slider

This item is for capturing a number value, with a slider widget.

See also the *Rating* item to get a scale entry.

Preview

Fig. 18: The number field using a slider widget.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Default`	The default value. Make sure it is in the right format.
`Minimum`	Minimum value of the range (inclusive).
`Maximum`	Maximum value of the range (inclusive).

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.13 Paragraph

This item is for capturing a long text, with a standard text area field. See also the *Short Answer*.

Preview

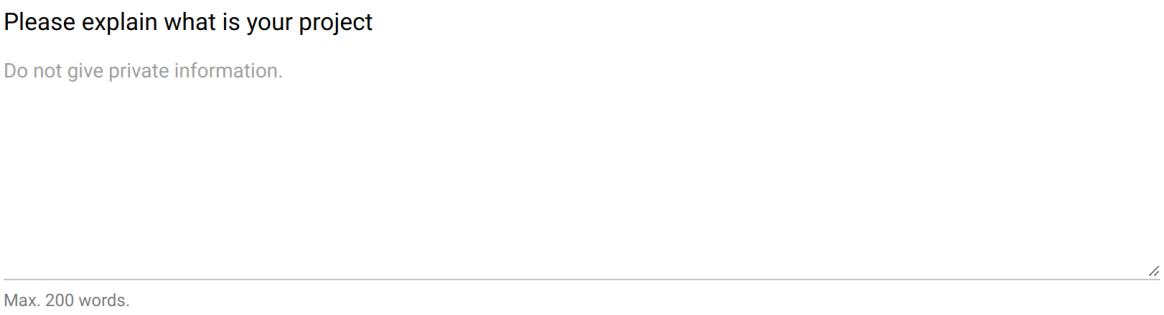


Fig. 19: The long text field using standard text area.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Placeholder`	Example text included in the text area when it is empty.
`Hint`	Help text below the input field.
`Default`	The default value. Make sure it is in the right format.

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.14 Short Answer

This item is for capturing a short text (one line), with a standard input field. See also the *Paragraph*.

Preview

Fig. 20: The short text field using standard input.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Placeholder`	Example text included in the text area when it is empty.
`Hint`	Help text below the input field.
`Default`	The default value. Make sure it is in the right format.
`Mask`	Text formatting using a mask. Below are the mask tokens: `#` Numeric `S` Letter, a to z, case insensitive `N` Alphanumeric, case insensitive for letters `A` Letter, transformed to uppercase `a` Letter, transformed to lowercase `X` Alphanumeric, transformed to uppercase for letters `x` Alphanumeric, transformed to lowercase for letters For instance the mask `(###) ### - ####` will format a phone number entry such as `(514) 286 - 0768`.

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.15 Group

The group item is a structural item: it does not collect any data, it only contains other items. As a consequence, this item will not appear in the exported data dictionary.

This item is appropriate when the form is rendered in multiple pages and several items are to be displayed in the same page. See also *Section*.

Preview

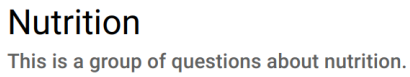


Fig. 21: The group is a title followed by a paragraph and the list of child items.

Design

Definition

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the group is not visible. It is used as a prefix to the child items variable name. For instance if the group name is `NUTRITION` and if it contains an item with name `SUGAR`, the collected data of this child item will be accessible with the full name `NUTRITION.SUGAR`.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Condition`	The condition makes the group and its child items visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the group and its child item are visible. Each child item can have its own condition rule as well. See *How to make fields conditional?*

Style

Property	Definition
`Label class`	CSS class to apply to the group title. Default is `text-h5`. See *How to style a form item?*
`Description class`	CSS class to apply to the group description. Default is `text-subtitle2 text-grey-8`.

5.3.16 Section

The section item is a structural item: it does not collect any data. As a consequence, this item will not appear in the exported data dictionary.

This item is appropriate for separating items in different sections. See also *Group*.

Preview

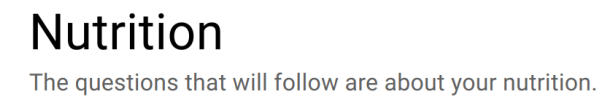


Fig. 22: The section is a title followed by a paragraph.

Design

Definition

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the section is not visible.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Condition`	The condition makes the section visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the section is visible. See *How to make fields conditional?*

Style

Property	Definition
`Label class`	CSS class to apply to the section title. Default is `text-h4`. See *How to style a form item?*
`Description class`	CSS class to apply to the section description. Default is `text-subtitle1 text-grey-8`.

5.3.17 Toggle

The toggle item is for capturing a logical value. The toggle has three states: selected, not selected and undefined. The undefined state is when the user has not made a choice yet. Only selected and not selected states can be chosen.

This field can be an alternative to a yes/no *Single Choice* question.

Preview

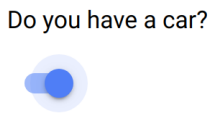


Fig. 23: The toggle field widget.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

The following dynamic properties apply:

Property	Definition
`Condition`	The condition makes an item visible or not, depending of other data. It is a small script which returns a logical value: `true` when item is visible. When no condition is specified (which is the default), the item is visible. See *How to make fields conditional?*
`Validation`	The validation specifies whether the data entry is correct. It is a small script which returns a logical value: `true` when item value is valid. Note that this validation script is not evaluated when the visibility `Condition` is not satisfied. See *How to validate field values?*
`Validation error message`	The validation error message to be displayed when validation fails.

Settings

Property	Definition
`Default`	The default value: either `true` or `false`.

Style

Property	Definition
`Label class`	CSS class to apply to the field label. See *How to style a form item?*

5.3.18 Computed

A computed field has no rendering, nor condition or validation. The value is computed from a script, that will usually aggregate values captured by other fields.

Preview

No preview.

Design

Definition

The standard properties apply:

Property	Definition
`Type`	The type of item (see above).
`Name`	The name of the item is not visible. It is a "variable" name that will be associated to the data collected. You will refer to this variable name when writing condition and/or validation scripts.
`Label`	The label is usually a question or a title.
`Description`	The description gives some guidance about how to enter data, what are their meaning etc.
`Required`	Whether an input is mandatory. Note that this does not apply when the visibility `Condition` is not satisfied.

Settings

Property	Definition
`Compute`	The computation script, which is always evaluated. There is no `Condition` script: instead of that, just return `null` when the computation data is not relevant. Identically, any `Default` value can be returned by this computation script.

5.4 Case Report Forms

The Case report forms drives the way the form will be used by the interviewers. It defines:

- Some descriptive information (name, description) that will be displayed in the Amber Collect app,
- The form and its version to be used,
- The multiple records policy, which is the strategy to adopt when a case report is saved (based on the individual's ID). Multiple records can be registered, or only a single one for the same individual. In this latter case a new record can be rejected or can update the existing one,
- The access permissions,
- The activity (enabled/disabled).

Note: The translations of the Case report form's name and description are looked up in the form's translations configuration.

Note: For now Amber supports only patients data acquisition via an interviewer. In the future versions, it could be possible to capture data about a predefined list of participants (with some selection criteria) or do some self-reporting. The case report form activity could also be scheduled. Please contact us if you have more complex use-cases.

5.4.1 List of case report forms

From the list of case report forms the following operations are available:

Add a case report form

Adding a case report form requires to select a form (in the current study) and its revision. If no revision is defined for the selected form, a new revision will be created. The revision number can be explicitly specified or the `latest` can be used (not recommended, see *How to test a case report form?*).

By default a Case report form can be used by all the interviewers. It is possible to restrict the access to the selected users and/or groups.

Delete case report form(s)

To delete several case report forms:

- Select one or more case report forms in the list,
- Press the Delete selected case report forms button and confirm.

You can also delete a specific case report form by pressing the case report form's Delete action button.

Edit a case report form

A case report form can be edited: only the revision number and/or the access permissions can be amended. To change the form, create a new case report form.

Pause/start a case report form

After being created, a case report form is in *Paused* state: it is not visible by the interviewers. Click on the Play button to activate the case report form. A case report form can be paused at any time: any pending case reports will be rejected at submission time.

5.5 Interviews

Available soon…

5.6 Records

Each record is a Case report and then contains:

- Which form was used to capture data, in which version,
- The captured data, identified by the patient ID,
- The user and the date at which the record was created,
- The list of actions that were performed on the case report.

An action on the case report contains the following information:

- The type of action: *init* when the interview is started, *pause* when it is paused and *complete* when it is finished,
- The user who performed the action,
- The timestamp at which the action was performed.

5.6.1 List of records

Delete record(s)

To delete several records:

- Select one or more record in the list,
- Press the Delete selected records button and confirm.

You can also delete a specific record by pressing the record's Delete action button.

Filter records

The list of records can be filtered by:

- Case report form (and therefore the associated form and its revisions used to collect data), by selecting the case report form in the select box,
- Form (any revisions), by selecting the form in the select box,
- Captured time range, by selecting the **From** and/or **To** dates,
- Patient ID, by entering a string to match in the open text box.

Download records

For consistency, records are always grouped by their form-revision. The downloaded file will contain:

- The data in the selected format,
- The data dictionary, which is a translation of the form schema: each item (capturing data) is a variable (labels are included within the variable, options are the variable's categories etc.).

The download formats proposed are:

- CSV, downloaded as a text file in CSV format. There is one section per "Case report form". The header line of each section are the variable names.
- CSV (zip), downloaded as a ZIP archive containing: one folder per form-revision including the data dictionary in **variables.json** file and the data in **data.csv** file. The CSV files header line are the variable names.
- Excel, downloaded as an Excel file containing several work sheets: **Variables** are the variables for each of the "Case report form" (named by the **table** column), **Categories** are the variable categories and one sheet per "Case report form" holding the data in tabular format (one column per variable).
- JSON, downloaded as a single JSON object dump made of: one entry per form-revision, each of these containing the `data` array and the `variables`.

Note also that:

- When the list of records is filtered, only the filtered records are downloaded,
- Each download is audited: who triggered the download, with which filter criteria, about which patients,
- The meta information about the record (user, history of actions) are not included in the downloaded file.

View record

View the captured raw data of a single record by pressing the Eye button.

CHAPTER

SIX

ADMINISTRATION

This section is only accessible to users with the role `Administrator`.

6.1 Users

Amber has a local registry of users. The username is the email address. These users can be added manually by the administrator, but it is recommended to let the users register themselves using the **Sign up** procedure. The administrators will receive a notification email informing that a user just registered and that its operational role can be set. The user then does not need the administrator for password recovery or to change its preferences (preferred language for instance).

6.1.1 User roles

There are built-in roles in Amber that grant different privileges:

- `Guest`, can just login in the application, this is the landing role after a user has signed up,
- `Interviewer`, has read-only access to Amber studies, forms and case report forms and has the permission to add new case report records,
- `Manager`, is in charge of managing the content, i.e. the study and form design, and the recorded data extractions,
- `Administrator`, is the super-user and then has read/write access on everything, from users registry to forms and recorded data.
- `Inactive`, can't do anything, not even login. This role is convenient when a user is associated to recorded data, to keep track of who did what, even after a collaborator has left the project.

6.1.2 List users

From the list of users the following operations are available:

Add a user

Clicking on the Add user button opens a dialog with the user form to be filled in: first/last name, email, institution etc. Special attention must be taken on the email address, which will be used as the login name and will be the recipient of the automatic notifications (for password recovery for instance).

Note that when a user is added, he is automatically added to a group with the domain name, for instance: `john@example.org` will be added to the group `example.org`.

Delete user(s)

There are different ways of deleting a user:

- Either select one or more users in the list and press the Delete selected users button,
- Or press a specific user's Delete action button.

Group users

Grouping users can be convenient when specific permissions are to be granted, for instance when a Case Report Form has limited access.

Users can be put in a group by selecting them in the list and pressing the Group button. A dialog asks for selecting the destination group. Note that the group must exist. Note also that a user can belong to several groups.

Resend verification email

User email is the key of user management. It is necessary to have it validated, so that a user can manage its password itself. A verification email consists of sending a message to the user's personal mailbox with a special link that will prove the mailbox is accessible.

Reset password

The Reset password procedure requires the email to have been validated. The password recovery email can be triggered manually by the administrator. The same operation is triggered when the user click on the Forgot password link on the login page and then provides its email.

(De)activate

When the user's role is not `Inactive`, pressing the Deactivate icon makes it inactive. Otherwise the Activate icon is visible and the user will be assigned the role `Guest`.

6.1.3 Single user

To view the details of a user, select the user's email address or the pencil icon in the list. The user page proposes to update user's information (except its password, use the Reset password procedure for that).

6.2 Groups

Grouping users can be convenient when specific permissions are to be granted, for instance when a Case Report Form has limited access.

6.2.1 List of groups

From the list of groups the following operations are available:

Add a group

When creating a group, a name is required, a description is optional.

Delete group(s)

There are different ways of deleting a group:

- Either select one or more groups in the list and press the Delete selected groups button,
- Or press a specific group's Delete action button.

Note that the users that are members of the group are not deleted.

6.2.2 Single group

To edit the details of a group, click on the group's name or the pencil icon.

The form allows to:

- Edit the name and/or description of the group,
- Lookup users to be added as group members,
- Remove members from the group.

CHAPTER

SEVEN

COOKBOOK

The following documentation gives some operation guidelines and good practices, from the design of a form to the case reports data extractions.

7.1 Form Builder

The form builder is the user interface for designing and testing a form.

7.1.1 How to format text entries?

Open text fields are convenient for capturing data that are not known in advance (e.g. a phone number, an institutional ID etc.). In order to avoid writing complex validation rules, a mask can be defined to format the data when they are entered.

Step 1 - Select Short Answer field

Select the **Short answer** field type: the **Mask** setting will match the format of the data to capture. The mask can also be combined with a validation rule (to verify that all characters have been entered for instance).

Note that it is not recommended to use this open text field with a mask to capture date or time values. Use the **Date**, **Date and time** or **Time** specific field types instead.

Step - 2 Set the Mask

A mask is made of tokens, designating what type of character is expected at each position. The possible mask tokens are:

Token	Description
`#`	Numeric
`S`	Letter, a to z, case insensitive
`N`	Alphanumeric, case insensitive for letters
`A`	Letter, transformed to uppercase
`a`	Letter, transformed to lowercase
`X`	Alphanumeric, transformed to uppercase for letters
`x`	Alphanumeric, transformed to lowercase for letters

Examples

The phone number `(514) 286-0768` can be captured by the mask `(###) ###-####`.

The institutional ID `CA32 4455 6E5Z` can be captured by the mask `AA## #### XXXX`.

7.1.2 How to make a select with a lot of options?

When there are a lot of possible options it can be tricky to design a user friendly selection field.

Step 1 - Select Auto complete field

There several type of fields for selecting among a list of options:

- Single/Multiple choice
- Dropdown
- Auto complete

The **Auto complete** is the most appropriate because there is not the problem of rendering a long list of proposed options. The field works as a filter: user enters at least two letters to trigger the display of the ten first items matching the entered text.

Step 2 - Prepare the options

The options will be defined in file. The file format is CSV (comma separated values, with file suffix `.csv` or `.txt`) or TSV (tab separated values, with file suffix `.tsv`). No header is expected. The first value if the option value (the data collected), the second one is the option label (can be a translation key for a multilanguage form).

Example of options file (CSV):

```
J0Y,Abitibi-Témiscamingue-Est (Radisson)
J0Z,Abitibi-Témiscamingue-Ouest (Guigues)
H2C,Ahuntsic Central
H2M,Ahuntsic East
H2B,Ahuntsic North
H2N,Ahuntsic Southeast
H3L,Ahuntsic Southwest
H0M,Akwesasne Region (Akwesasne)
G8M,Albanel
G8E,Alma North
G8B,Alma Southeast
G8C,Alma Southwest
J9T,Amos
G5J,Amqui
H1K,Anjou East
H1J,Anjou West
G0R,Appalaches (La Pocatière)
```

When the value and the label are the same, it is possible to provide a simple list:

```
Hôpital chinois de Montréal
Hôpital géneral juif
Hôpital Santa Cabrini
Hôpital Mont-Sinaï - Montréal
```

(continues on next page)

(continued from previous page)

```
Hôpital de Lachine
Hôpital Richardson
Hôpital Marie-Clarac
Institut de réadaptation Gingras-Lindsay de Montréal
Hôpital neurologique de Montréal
Centre gériatrique Maimonides Donald Berman
Hôpital Sainte-Anne
```

Step 3 - Upload options

The **Options** setting proposes to **Upload options**: select the options file and the options values and labels will be automatically added.

7.1.3 How to make fields conditional?

A conditional field is a field that depends of data collected by one or more fields. The condition is logical value:

- `true`, the field is visible and data can be captured,
- `false`, the field is not visible and has no value (i.e. value will be removed when switching from `true` to `false` condition).

The condition is a short Javascript code. In the context of the execution of this code, the other field values can be accessed with the following syntax:

```
$('VARIABLE')
```

where `VARIABLE` is the name of the field from which the value is to retrieved. When the field is in a group, use a `.` for building the full name of the field. For instance, if the field with name `VARIABLE` is in a group with name `GROUP`, the value of the field will be:

```
$('GROUP.VARIABLE')
```

Examples

If the field is a numeric value:

```
$('VARIABLE') > 12
```

If the field is an array value (case of multiple choices), to check that there is a value and that 2 choices have been made exactly:

```
$('VARIABLE') && $('VARIABLE').length === 2
```

If the field is a string value representing a date, to check that there is a value and that the date is not in the future:

```
$('VARIABLE') && Date.parse($('VARIABLE')) <= Date.now()
```

7.1.4 How to validate field values?

An open field needs to be validated.

The validation is a short Javascript code. In the context of the execution of this code, the `val` Javascript variable represents the value captured by the field to validate. The other field values can be accessed with the following syntax:

```
$('VARIABLE')
```

where `VARIABLE` is the name of the field from which the value is to retrieved. When the field is in a group, use a `.` for building the full name of the field. For instance, if the field with name `VARIABLE` is in a group with name `GROUP`, the value of the field will be:

```
$('GROUP.VARIABLE')
```

Validation message

When field validation fails, a validation message should be prompted to help user to correct the answer.

Examples

The text entered must have at least 10 characters:

```
val && val.length >= 10
```

The number entered must be lower than another field value:

```
val && val < $('VARIABLE')
```

The date can be empty or not be in the future:

```
val === null || Date.parse(val) <= Date.now()
```

Working with date and time can be tricky, as the supported text format of the date can be different from one browser to another. For instance `2022-03-07` (ISO 8601 format, this is how a date value is captured) is not supported by Safari. For Safari `2022/03/07` is a valid date format, which works for the other browsers as well. Then we will use the `replace()` Javascript function to switch from one format to the other. The following validates that the arrival date at the hospital is either empty or is not in the future and is after the trauma date time:

```
val == null ||
  (Date.parse($('ARRIVAL.DATE').replace(/-/g, '/') + ' ' + val) <= Date.now()
    && Date.parse($('ARRIVAL.DATE').replace(/-/g, '/') + ' ' + val) >= Date.parse($(
↪'TRAUMA.DATE').replace(/-/g, '/') + ' ' + $('TRAUMA.TIME')))
```

7.1.5 How to make a computed field?

A computed field is a field that do not require the user to enter a value, it is computed with other fields values.

The computation rule is a short Javascript code. Other field values can be accessed with the following syntax:

```
$('VARIABLE')
```

where `VARIABLE` is the name of the field from which the value is to retrieved. When the field is in a group, use a `.` for building the full name of the field. For instance, if the field with name `VARIABLE` is in a group with name `GROUP`, the value of the field will be:

```
$('GROUP.VARIABLE')
```

Examples

The computed value is the sum of other numeric values:

```
$('VAR1') + $('VAR2')
```

7.1.6 How to group fields?

Amber has limited capabilities in terms of page layout. It is still possible to have several fields on the same page, even if they depend on each other.

Step 1 - Add a group item

Add an item and select the **Group** type. A name is required as it will be used a prefix to name included fields. It is also possible to define a label and a description.

A condition defined at a group level will apply to all its children (see *How to make fields conditional?*).

Step 2 - Add child items

There are different ways of adding a item inside of a group:

- Select the group item and add a new item: it will be placed at the last position.
- Select a child item of the group and add a new item: it will be placed at the following position.
- Select an existing item and move it up/down until it is included to the group item.
- Select an existing item, cut it, select the group item and paste the item as a new child.

The position of the added item in the group can be modified by clicking on the move up or move down buttons of the item.

Note: A group inside another group is currently not supported.

Step 3 - Use the children values in conditions/validations

The name of the group constitutes a namespace for the included items and therefore these item names must be prefixed with the group name.

Examples

If the child field `VARIABLE` of the group `GROUP` is a numeric value, a validation script will look like:

```
$('GROUP.VARIABLE') > 12
```

Because of the group namespacing, it is possible to define items with same name in different groups:

```
$('GROUP1.VARIABLE') > $('GROUP2.VARIABLE')
```

7.1.7 How to make an image map?

A selection field can be coupled with an image. When selected, areas of this image will be highlighted.

Step 1 - Add an Image Select Item

Add an item and select the **Image select** type.

The supported image formats are the ones that are supported by web browsers, mainly PNG, JPEG and GIF. The image can be integrated to the form in different ways:

- **Using a web link**, which must be publicly accessible (i.e. no authentication/authorization required). This makes the form unfunctional when offline: network access is required to download the image when the form is administered.
- **Embeded in the form**. The binary data of the image are directly encoded in the form, which makes the image available while in offline mode. The encoding process is done by Amber Studio by selecting a local image file to upload.

Step 2 - Add Image Areas

Once an image is defined, it is required to define areas on this image that will be used for option selection. An image area is defined by:

- **A value**, which is the stored value when the area is selected. It must match the value of one of the options.
- **A color**, in any of the supported CSS color formats.
- **A serie of points** that forms a polygon (circle and rectangle shapes are not supported). Each point is defined by a coordinate `x,y`. The serie of point coordinates are to be separated by commas "," or space characters: for instance `x1,y1,x2,y2,x3,y3` or `x1,y1 x2,y2 x3,y3`. There are some free online tools that can help with drawing these polygons, see for instance image-map.net.

The image areas can be entered manually or uploaded from a file. The file format is CSV (comma separated values, with file suffix `.csv` or `.txt`) or TSV (tab separated values, with file suffix `.tsv`). No header is expected.

Example of image areas file (CSV format), with space-separated coordinates:

```
1,#ccc,0,0 150,0 150,750 0,750
2,#f8bbd0,150,0 300,0 300,750 150,750
3,#c8e6c9,345,0 495,0 495,750 345,750
4,#ffe0b2,495,0 645,0 645,750 495,750
```

Example of image areas file (TSV format), with comma-separated coordinates:

```
1 #ccc    0,0,150,0,150,750,0,750
2 #f8bbd0 150,0,300,0,300,750,150,750
3 #c8e6c9 345,0,495,0,495,750,345,750
4 #ffe0b2 495,0,645,0,645,750,495,750
```

7.1.8 How to translate field labels?

When the form is proposed in several languages, here are the good practices for designing the form translations with a minimum of efforts.

Step 1: Use Translation Keys

Whenever the label or the description (or other text to be translated) is set on an item, it is recommended to use a translation key for a better long term maintenability: typos, rewording are easier to implement. For instance use `q1_label` or `nb_cigars_label` instead of `How many cigars do you smoke?`.

Step 2: Generate Translation Entries

Once the form is ready, select the **Translations** menu on the left of the form builder and then add translation entries by selecting the **Merge with items translation keys** option. This operation will go through the designed form to retrieve the translation keys defined at Step 1.

By default all the web application languages are enabled for the form. You can restrict which form languages are to be used. Once done, there are several options for entering the translated texts:

Using the web interface

For each entry and language, select the cell in the table and type the translated text. The form is automatically saved along with the translations.

Using text file

When there are many entries, it can be more convenient to edit a file with translations. Use the form **Export** button to download the form in JSON format along with its translations. Look for the section `i18n` (stands for *internationalization*). Translation entries are group by languages.

```
{
  "fr": {
    "nb_cigars_label": "Combien de cigares fumez-vous ?",
    "nb_cigars_description": "Entrer 0 si vous ne fumez pas de cigares."
  },
  "en": {
    "nb_cigars_label": "How many cigars do you smoke?",
    "nb_cigars_description": "Enter 0 if you do not smoke cigars."
  }
}
```

Then **Import** the form along with its translations by uploading the file.

7.1.9 How to backup/restore/share a form?

In order to be adapted to the different populations, a form can have several variants. A form can also be reproduced from one study to another before being modified. Finally, a form can be backed up for further reference. This can be achieved by the following operations (from the form page):

Using Export/Import

Export

Press the **Export** button to download the form in JSON format. This file contains the list of form items (`items` key) and the translations (`i18n` key, stands for *internationalization*).

```
{
  "label": "Some form",
  "description": "Some form description",
  "items": [ ... ],
  "i18n": {
    "fr": { ... },
    "en": { ... }
  }
}
```

Import

The form import process consists of uploading a local file which expected format (JSON) and structure is the one you get when exporting the form. The form described in this file is self-sufficient and has no reference to the original study.

- When creating a new form: press the **Add a new form** button and select the form file to upload.
- When updating an existing: press the **Import** button and select the form file to upload. **Important!** the current form items and translations will be overridden by the imported ones.

Using Form Revisions

When designing a form, it is possible to "tag" the current form state into a new form revision. From the form page press the **Tag** button and provide a short comment describing the current form state. There is no limit on the number of form revisions that can be created.

From the list of form revisions, it is possible to **view** and test each form revision. Export in JSON format is also available.

Each form revision can be **reinstated**: it is similar to the import process described above (i.e. the current state of the form will be overridden).

7.1.10 How to style a form item?

In each form item design form, there is a **Rendering** section where you can specify some CSS classes to apply. These classes can either be built-in or custom ones.

Built-in Style

Amber front-end applications (Studio or Collect) are based on the Quasar web framework. See the "Style & Identity" section in their documentation.

Custom Style

A custom style can be defined in the file `src/css/custom.scss`. The format of this style specification is SCSS.

See the amber-template project to integrate the custom style file into the web app built process.

7.1.11 How to extend to new type of fields?

There is currently no dynamic way to add a new form item component. Please contact the OBiBa developers for contribution or collaboration: info@obiba.org or use the Amber Studio issue tracker.

7.2 Case Reports

7.2.1 How to test a case report form?

After a form has been designed, it is time to test it in the real world, without interfering with the normal data collection flow. We will use the following Amber's case report form (CRF) features:

- A CRF can be linked to a specific form version,
- Permission to use the CRF can be explicitly granted to some users or group of users.

Step 1 - Tag a new version of the form

After the design of the form is completed, with basic testing in the form builder preview, a new version of the form needs to be tagged. In the form builder page, select the **Tag** button and enter a comment describing what changes have been made.

You can verify that a new version has been added to the list of form revisions.

Step 2 - Create a CRF for this form version with restricted access

To create a new CRF, select the **Design > Case report forms** menu in the left drawer and press the button to add a new CRF. The dialog that appears allows you to:

- Select the **Form** name to be tested,
- Select the **Revision** number that you just created,
- Select the **Restricted access** toggle button to open permissions options,
- Select the **Permitted users** and/or **Permitted groups** who will be allowed to use this CRF,

- Click on **Add**.

Step 3 - Start the CRF

A CRF is always created in the **Paused** state. Because of this state, the CRF is not visible by the Amber Collect app. Click on the play icon to switch it to the **Active** state. Your CRF is now operational and only permitted users can see it.

Step 4 - Test in Amber Collect

Login in the Amber Collect web app. In the home page, the CRF should appear in the list (verify the version number displayed is the one you expect). Press on **Start** to collect new Case Report data with this CRF. Test different data entries to make sure conditions and validations are correct.

You do not necessarily need to save the Case Report to see what the data collected look like. After pausing the Case Report, select on the **Case Reports** menu (in the left drawer). The list of the Case Reports will appear. Click on the eye icon to see the data of this individual (in the standard JSON format).

In case you have saved the Case Report data, the data cannot be seen in the Amber Collect any more. Then go to the Amber Studio app, select the study and the **Data collection > Case reports** menu in the left drawer. Identify which case report you just saved (with the individual ID, form name and version) and press the eye icon to see the data (in JSON format). This test case report must then be removed to not pollute the real collected data.

7.2.2 How to extract case report records data and dictionary?

The records of a study's case reports can be downloaded in a file from the study's **Data collection > Case reports** page.

Step 1 - Filter the records of interest

When the list of records is filtered, only the filtered records are downloaded. The list of records can be filtered by:

- Case report form, by selecting the case report form in the select box,
- Form, by selecting the form in the select box,
- Capture time range, by entering From and/or To dates,
- Patient ID, by entering a string to match in the open text box.

Step 2 - Select the download format

The download formats proposed in different formats.

CSV

The file downloaded is a text file in CSV format. There is no data dictionary included, only data (with one column per variable). The CSV file is divided in sections: one per form and its revision.

CSV (zip)

The file downloaded is a ZIP archive containing one folder per form-revision. Each of these folders includes:

- the data dictionary in a **variables.json** file,
- the data in **data.csv** file. The CSV files header line are the variable names.

Excel

The file downloaded is an Excel file, containing both the data dictionaries and the data. This Excel file contains several work sheets:

- **Variables** is a work sheet describing each variable for each form revision (identified by the **table** column),
- **Categories** is a work sheet describing each variable categories (i.e. select options),
- one work sheet for each form revision's collected data.

JSON

The file downloaded is a single JSON object dump. It is made of one entry per form-revision, each of these containg:

- the `data` array of collected data objects (one per case report),
- the `variables` array of variable objects (one per form field).

7.3 Interviews

7.3.1 How to test an interview design?

After some forms have been designed, it is time to test them in the real world, without interfering with the normal data collection flow. We will use the following Amber's interview design features:

- Each step of an interview design can be linked to a specific form version,
- Permission to use the interview design can be explicitly granted to some users or a group of users.

Step 1 - Tag a new version for each form

After the design of a form is completed, with basic testing in the form builder preview, a new version of the form needs to be tagged. In the form builder page, select the **Tag** button and enter a comment describing what changes have been made.

You can verify that a new version has been added to the list of form revisions.

Step 2 - Create an interview design with restricted access

To create a new interview design, select the **Design > Interview designs** menu in the left drawer and press the button to add a new interview design. The dialog that appears allows you to:

- Enter a name and a title for this design,
- Select the **Restricted access** toggle button to open permissions options,
- Select the **Permitted users** and/or **Permitted groups** who will be allowed to use this design,
- Click on **Add**.

Then in this interview design page, add for each interview step:

- Select the **Form** name to be tested,
- Select the **Revision** number that you just created.

Add a test campaign with participants. When adding participants, make sure they have any attributes that could be interpreted by the form's logical rules.

Step 3 - Start the interview design

An interview design is always created in the **Paused** state. Because of this state, no participant could be identified by the Amber Visit app. Click on the play icon to switch it to the **Active** state. Your interview design is now operational and only permitted users can see it.

Step 4 - Test in Amber Visit

As one of the participant, login in the Amber Visit web app. In the home page, the interview steps should appear. Start answering interview steps and test different data entries to make sure conditions and validations within each form and between steps are correct.

Once the interview is saved, go to the Amber Studio app, select the study and the **Data collection > Interviews** menu in the left drawer. Identify which interview you just saved (with the participant ID) and press the eye icon to see the data (in JSON format). This test interview must then be removed to not pollute the real collected data.

7.3.2 How to extract interview records data and dictionary?

The records of a study's case reports can be downloaded in a file from the study's **Data collection > Interviews** page.

Step 1 - Filter the records of interest

When the list of records is filtered, only the filtered records are downloaded. The list of records can be filtered by:

- Interview design, by selecting the interview design in the select box,
- Campaign, by selecting the campaign in the select box,
- State, by selecting the interview state in the select box,
- Eligibility, by selecting the associated participant eligibility in the select box,
- Capture time range, by entering From and/or To dates,
- Participant ID, by entering a string to match in the open text box.

Step 2 - Select the download format

The download formats proposed in different formats.

CSV

The file downloaded is a text file in CSV format. There is no data dictionary included, only data (with one column per variable). The CSV file is divided in sections: one per form and its revision.

CSV (zip)

The file downloaded is a ZIP archive containing one folder per form-revision. Each of these folders includes:

- the data dictionary in a **variables.json** file,
- the data in **data.csv** file. The CSV files header line are the variable names.

Excel

The file downloaded is an Excel file, containing both the data dictionaries and the data. This Excel file contains several work sheets:

- **Variables** is a work sheet describing each variable for each form revision (identified by the **table** column),

- **Categories** is a work sheet describing each variable categories (i.e. select options),
- one work sheet for each form revision's collected data.

JSON

The file downloaded is a single JSON object dump. It is made of a data section with one entry per form-revision, each of these containg:

- the `data` array of collected data objects (one per case report),
- the `variables` array of variable objects (one per form field).

CHAPTER

EIGHT

INTRODUCTION

The Amber Collect application is the case report data collection web interface of the Amber server. The Case Report Forms that are enabled can be used to report cases.

Using Amber Collect requires authentication. Depending on the user role and authorizations, some Case Report Forms will be visible or not.

No personal information is stored in the app after the case report has been completed: private data are automatically saved on the Amber server (as soon as the network is accessible) and cannot be read back.

8.1 Requirements

This web interface is a JavaScript application requiring a modern web browser. There is no requirement regarding the operating system: it can be either a desktop or a mobile one.

8.2 Installation

Amber Collect is a Progressive Web App which means that it is a web app (i.e. accessible from a browser) that can be installed on the user's desktop (computer) or home screen (mobile), without the need of deploying it in a proprietary App Store.

8.2.1 Step 1 - Go to Amber Collect web site

In your favorite browser, enter the Amber Collect app address, e.g. https://amber-collect-demo.obiba.org. From this point, the app is fully functional. The page can be bookmarked or "installed" (see next Step).

8.2.2 Step 2 - Install the app (optional)

Depending on the browser's vendor, the procedure to install a shortcut icon to your home screen might differ a bit.

Chrome

Firefox/Ghostery

Phoenix

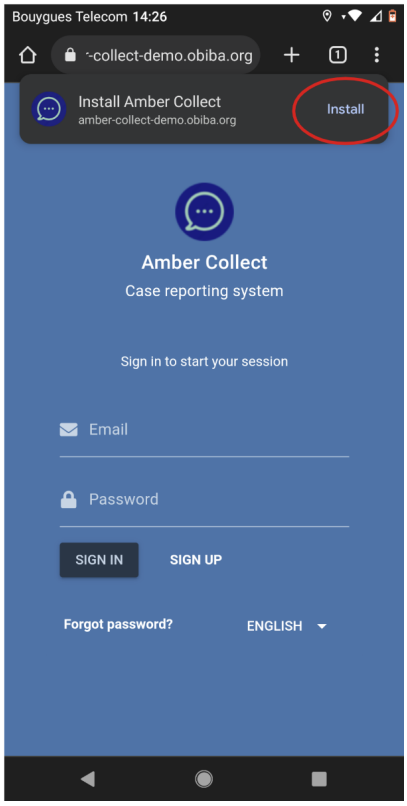


Fig. 1: Install Amber Collect app from Chrome browser: a popup message proposes to install the app.

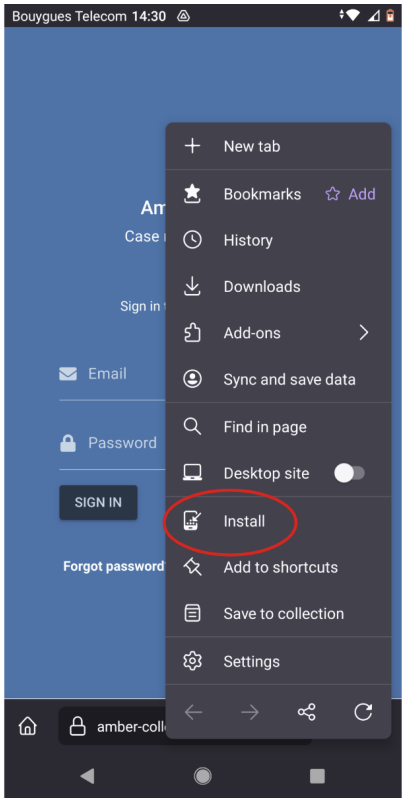


Fig. 2: Install Amber Collect app from Firefox browser: the “Install” entry is proposed in the page’s menu.

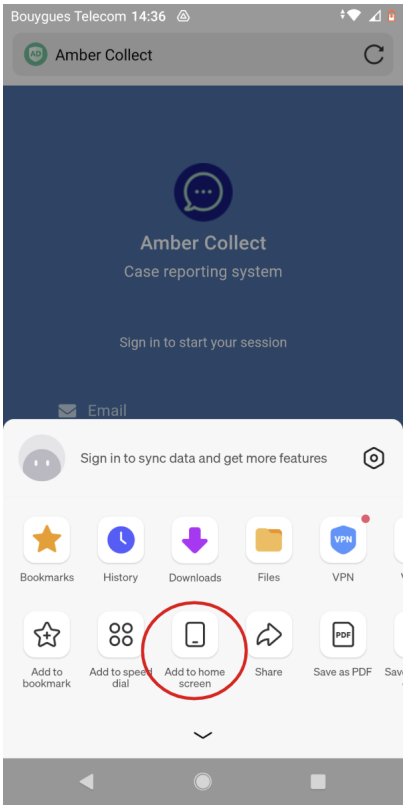


Fig. 3: Install Amber Collect app from Phoenix browser: the “Add to home screen” action is proposed in the page’s menu.

CHAPTER

NINE

INTRODUCTION

The Amber Visit application is the interview data collection web interface of the Amber server. The Interview Designs that are enabled can be used to conduct interviews, either self-administered by the participant or with the help of a dedicated interviewer.

Using Amber Visit requires authentication. A participant will provide its personal code (and optionally a password), an interviewer will use its Amber credentials (user name, password and 2FA code).

No personal information is stored in the app after the interview has been completed: private data are automatically saved on the Amber server (as soon as the network is accessible).

9.1 Requirements

This web interface is a JavaScript application requiring a modern web browser. There is no requirement regarding the operating system: it can be either a desktop or a mobile one.

CHAPTER

TEN

PARTNERS AND FUNDERS

The development of this application was made possible thanks to the support of our partners and funders:

CHAPTER

ELEVEN

SUPPORT

Please visit OBiBa support page.