Package 'lssdoc'

September 16, 2026

Title 'LimeSurvey' '.lss' Questionnaires to and from Word Documents

Version 0.3.0

Description Turn a 'LimeSurvey' '.lss' survey export into a publication-quality questionnaire document in Word ('.docx') or PDF, with up to four of the survey's own languages side by side. Every label the package adds around that content -- column headers, type names, the audit section -- is written in English, French, German, Spanish or Italian, whatever the survey languages are. A rule-based audit flags missing translations, forward filter references, duplicate codes, array-scale inconsistencies and orphan structural references. Questionnaires travel the other way too: describe one in R, or fill in a Word form, and write a '.lss' file ready to import. Meant for the people who work on questionnaires -- researchers, methodologists, ethics committees, translators and reviewers -- and fully local: the source file is the only input, and no questionnaire content is uploaded to a third-party service.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.1)

Imports cli, lifecycle, rlang, xml2

Suggests flextable, knitr, officer, rmarkdown, testthat (>= 3.0.0), zip

VignetteBuilder knitr

Config/testthat/edition 3

URL https://amaltawfik.github.io/lssdoc/, https://github.com/amaltawfik/lssdoc

BugReports https://github.com/amaltawfik/lssdoc/issues

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Amal Tawfik [aut, cre, cph] (ORCID:
<https://orcid.org/0009-0006-2422-1555>, ROR:
<https://ror.org/04j47fz63>)

Maintainer Amal Tawfik <amal.tawfik@hesav.ch>

Repository CRAN

Date/Publication 2026-09-16 08:40:02 UTC

Contents

as_lss_spec 2
audit_lss 4
check_form_docx 6
lss_spec 7
lss_template_docx 10
read_form_docx 11
read_lss 13
render_audit 14
render_questionnaire 15
write_form_docx 22
write_lss 24

Index **27**

`as_lss_spec` *Turn a parsed LimeSurvey survey into an authoring specification*

Description

[Experimental]

Usage

```
as_lss_spec(lss, strict = TRUE)
```

Arguments

`lss`	An `lss` object from `read_lss()`.
`strict`	Logical. `TRUE` (default) refuses the whole survey with a single classed error (`lssdoc_unconvertible`) listing **every** unconvertible item. `FALSE` drops those items – and the filters and quotas that depend on them – and returns a valid specification, with one warning (`lssdoc_lossy_conversion`) listing what was dropped.

Details

Experimental. Convert an `lss` object – a real LimeSurvey export read by `read_lss()` – into the `lss_spec()` object the authoring side of the package works with. It is the entry point for *modifying an existing questionnaire*: read the `.lss` a colleague sends, render it as the Word authoring form with `write_form_docx()`, let the author edit the form, and hand the result back to `read_form_docx()` and `write_lss()`.

The conversion is deliberately narrower than the file it reads. An `lss` object is whatever LimeSurvey exported; an `lss_spec` is what lssdoc can author and re-emit. Everything in between is reported: refused outright (`strict = TRUE`) or dropped with a warning (`strict = FALSE`). Nothing is ever lost in silence.

Value

An `lss_spec()` object.

What is converted

- **Kinds**: the LimeSurvey type letter maps to the authoring kind through the kind table (`lss_kinds` in the sources). A question theme other than the authorable one is replaced by it and reported.
- **Texts**: survey title, welcome and end text, group titles and descriptions, question wordings and help, option, row and column labels, the "other" label and the quota name and message, each taken per language from the `*_l10ns` tables. HTML is flattened to plain text – paragraphs and `<br>` become line breaks, inline marks are dropped – and every field that really carried markup is named in the lossy warning.
- **Languages**: all the survey's languages, the base language (`surveys.language`) first, as `lss_spec()` requires. A text missing in a language is filled from the base language and reported: real exports do have missing translations – that is what `audit_lss()` flags – and the author fixes them in the form. A text missing in the BASE language but present in another is filled the other way round, and the report names the direction (`"help of question Q1 [fr, from en]"`), so the translations a survey drafted in another language already has are never thrown away.
- **Structure**: options, rows and columns from `answers` and `subquestions`; the native "other" option from `other = "Y"` and `other_replace_text`; exclusive options from `exclude_all_others`; the cap from `max_answers`; the "other" position from `other_position` and `other_position_code`. Every other question attribute passes through unchanged into the question's `attributes`.
- **Filters**: the three equations `write_lss()` emits – `Q.NAOK == "1"`, `(Q.NAOK == "1" or Q.NAOK == "-oth-")` and `count(Q_1.NAOK, ...) >= n` – are translated back into `Q = 1`, `Q in [1, autre]` and `count(Q) >= n`, with or without `.NAOK`.
- **Quotas**: a single-member quota with the terminate action becomes a `code = value` condition on its question – on any kind holding a single coded answer, the implicit scales included, so a quota on the gender question (`M` / `F`) converts like a quota on a declared option code.

What is refused

Each of the following is one row of the unconvertible report (question code, item, reason), and all of them are listed at once:

- a question whose type is not authorable – the eight deferred LimeSurvey types (`P`, `H`, `1`, `;`, `:`, `*`, `|`, `I`) and any unknown one;

- a question code that is not a spec code, or a duplicate of an earlier one;
- a question or option with no wording in the base language, or a shape the `lss_spec()` validator refuses (too few options, an array without rows or columns, a cap larger than the option list);
- a display condition in any other form than the three above – a hand-written equation is never guessed at;
- a quota combining several questions, built on a question the spec cannot target, or carrying an action other than "terminate";
- a group with no title in the base language (its questions go with it), and a question belonging to no group at all;
- a question storing rows on a second answer scale, or in a section its type does not carry: the shape on file is not the shape lssdoc emits, and half an option list is not a smaller option list;
- a structural column carrying something a specification has no field for and `write_lss()` would silently replace by its own constant, where that changes what the respondent meets: a group display equation (`groups.grelevance`) or randomization, a per-subquestion relevance (an array filter), a validation regex (`preg`), per-question JavaScript (`question_l10ns.script`), encrypted storage (`encrypted`). The columns that change nothing a respondent sees – assessment values, `same_default`, `modulename`, `same_script`, a quota URL description – are noted in the lossy warning instead;
- a multilingual survey that does not declare its base language: a specification's first language IS its base language, and `<languages>` lists the base one last.

See Also

`read_lss()`, `lss_spec()`, `write_form_docx()`, `write_lss()`.

Examples

```
# The bundled flawed survey uses LimeSurvey question types lssdoc does
# not author yet, and a filter it cannot express, so it converts only in
# the permissive mode -- which names everything it had to drop.
flawed <- system.file("extdata", "audit_demo.lss", package = "lssdoc")
spec <- suppressWarnings(as_lss_spec(read_lss(flawed), strict = FALSE))
spec
```

`audit_lss` *Audit a LimeSurvey survey for reviewable anomalies*

Description

Inspect a LimeSurvey survey and flag anomalies that can be detected without any AI. The audit guides a human reviewer; it does not silently correct anything. Every finding names a precise location and a severity.

Usage

```
audit_lss(input)
```

Arguments

`input`	Either a path to a `.lss` file (character string) or a pre-parsed `lss` object returned by `read_lss()`. Passing a path parses it on the fly; passing an `lss` object avoids re-parsing when the same survey is also rendered in the same session.

Details

Checks performed:

- **Missing translations** – a question, help, answer, or subquestion text present in at least one language but empty in another.
- **Empty in all languages** – a translatable text empty in every language.
- **Duplicate codes** – a question variable code repeated in the survey, or an answer/subquestion code repeated within one question.
- **Whitespace in codes** – a question, subquestion or answer code containing leading, trailing or interior whitespace (likely a typo; causes subtle bugs in the data export).
- **Missing options for the type** – a question whose type requires answer options or subquestions but has none (per the type taxonomy).
- **Forward filter references** – a relevance expression that names a variable appearing at or after the filtered question (the value is not yet collected when the filter is evaluated).
- **Array-scale inconsistencies** – an array (single or dual) whose subquestions reference a `scale_id` that has no answer options, or vice versa.
- **Orphan references** – a subquestion or answer pointing to a question that does not exist.

Value

An object of class `lss_audit`: a list with `file`, `languages`, summary counts, and a `findings` data frame (`severity`, `check`, `location`, `language`, `message`). It has a `print()` method and an `as.data.frame()` method.

See Also

`render_audit()` to write the same findings to a Word or PDF document.

Examples

```
# A deliberately flawed demo survey ships with the package, seeded
# with every anomaly the audit detects.
demo <- system.file("extdata", "audit_demo.lss", package = "lssdoc")
audit_lss(demo)
```

check_form_docx *Report every problem of a Word authoring form at once*

Description

[Experimental]

Usage

```
check_form_docx(path)

## S3 method for class 'lss_form_check'
print(x, ..., n = 20L)
```

Arguments

path	Character. Path of the .docx form to check.
x	An lss_form_check object.
...	Ignored.
n	Maximum number of problems to print; Inf for all.

Details

Experimental. Run read_form_docx()'s parser in dry-run mode: an author iterates on a form until it is clean, and fixing one problem per round trip is not a workflow. The same rules, the same messages; the first error simply does not stop the read.

A problem in one block does not hide the problems of the next: parsing resumes at the following block. Three problems are still fatal, because nothing can be read past them: an unreadable file, a missing or unsupported contract version, and a document-wide structural refusal (tracked changes, content controls). When one of those fires it is the only row reported.

Value

An object of class lss_form_check: a data frame with one row per problem and the columns severity ("error" or "warning"), class (the condition class), block, code, field and message. Zero rows means the document reads: read_form_docx() will return a specification. A print() method summarizes it.

See Also

read_form_docx(), write_form_docx(), lss_template_docx().

Examples

```
if (requireNamespace("officer", quietly = TRUE) &&
    requireNamespace("flextable", quietly = TRUE)) {
  form <- tempfile(fileext = ".docx")
  lss_template_docx(form, lang = "fr", kinds = c("single", "text"))
  check_form_docx(form)
}
```

`lss_spec` *Build and validate a survey specification*

Description

[Experimental]

Usage

```
lss_spec(
  title,
  groups,
  languages = NULL,
  language = NULL,
  welcome = NULL,
  end_text = NULL,
  quotas = NULL
)
```

Arguments

`title`	Character. Survey title shown to respondents.
`groups`	List of groups. Each group is a list with `title` (character), optionally `description` (character, a localizable introduction shown above the group), and `questions` (list of question specifications, see Details).
`languages`	Character vector of language codes, the primary language first (e.g. `c("fr", "en")`). Defaults to `"fr"`. See the *Languages* section.
`language`	Character. Backward-compatible alias for a single-language survey: `language = "fr"` is `languages = "fr"`. Passing both is allowed only when `language` is `languages[1]`.
`welcome`	Character vector of welcome-text paragraphs, or a single string starting with < used verbatim as HTML. Optional.
`end_text`	Character vector of end-page paragraphs, or a single string starting with < used verbatim as HTML. Optional.

quotas — List of end-of-survey quotas. Each element is a list with `question` (code of a question holding a single coded answer: `"single"`, `"dropdown"`, `"singlecomment"`, `"yesno"`, `"gender"` or `"fivepoint"`), `code` (the answer code that triggers the quota – a declared option code, or one of the kind's implicit codes for the fixed scales: `Y/N`, `M/F`, `1-5`), `message` (text shown to the respondent) and optionally `name` and `limit` (a whole number at or above zero; omitted, it stays the historical zero). A quota emitted by `write_lss()` terminates the survey – the LimeSurvey mechanism for "if the person declines, end here".

Details

Experimental. Assemble a survey specification – the authoring-side counterpart of the `lss` object – that `write_lss()` can turn into an importable LimeSurvey `.lss` file. The specification is validated in depth at construction time, because LimeSurvey itself imports silently: a mistyped attribute, a filter referencing a missing answer code, or a cap larger than the option list are all accepted on import and only surface once respondents hit them.

Each question is a list with fields:

- `code` – stable technical code (letters then letters/digits, at most 20 characters), unique across the survey. Codes become variable names in the data and are pinned once fieldwork starts.
- `kind` – the question type. Choice kinds: `"single"` (radio list), `"dropdown"`, `"singlecomment"` (list with comment), `"multiple"`, `"ranking"`. Array kinds: `"array"` (rows and columns), `"array5"`, `"array10"`, `"arrayyesno"`, `"arraytrend"` (rows only, the scale is implicit). Item batteries: `"multitext"`, `"multinumeric"` (one field per option). Scalar kinds: `"text"`, `"shorttext"`, `"hugetext"`, `"numeric"`, `"date"`, `"yesno"` (implicit Y/N), `"gender"` (implicit M/F), `"fivepoint"` (implicit 1-5). Plus `"display"` (text shown without input). Every kind maps to a LimeSurvey type attested by real exports; eight further LimeSurvey types are deferred, each with its own reason, in `lss_kinds_deferred` in the sources.
- `text` – the question wording. `mandatory` – logical, default `FALSE`. `help` – optional help text shown under the wording.
- `options` – for every kind that takes an option list (`single`, `dropdown`, `singlecomment`, `multiple`, `ranking`, `multitext`, `multinumeric`): list of options, each a list with `text` and optionally `code`, `other = TRUE` (native LimeSurvey "other" with a free-text field; `single`, `dropdown` and `multiple` only) and `exclusive = TRUE` (`multiple` only; unchecks every other box). Options without a `code` are numbered `1..n` in order, skipping the `other` option, which LimeSurvey codes natively. An explicit code is letters and digits, and its length follows the table LimeSurvey stores the list in: **5 characters** for a list emitted as answers (`single`, `dropdown`, `singlecomment`, `ranking` options, and `array` columns – `answers.code` is a `varchar(5)`), **20 characters** for a list emitted as subquestions (`multiple`, `multitext`, `multinumeric` options, and `array` and implicit-scale array rows – `questions.title` is a `varchar(20)`, the same column as a question code).
- `rows` / `columns` – for `array`: the subquestions and the answer scale, same shape as `options`.
- `relevance` – display condition in a minimal syntax: `code = 1`, `code in [1, 2, autre]`, `count(code) >= 2` (at least n boxes ticked in a multiple-choice question). The keyword `autre` designates the native "other" option. Conditions may only reference questions defined earlier in the survey.
- `max_answers` – cap for `multiple` (strictly below the number of options) and `ranking` (at most the number of items).

- `other_position` – where the "other" option is displayed: `"end"` (LimeSurvey default), `"beginning"`, or `"specific"` together with `other_position_code`, the code of the option AFTER which "other" appears. In practice "other" usually belongs before the "none of the above"-type exclusive options, which the default position puts it after.
- `attributes` – optional named list of extra question attributes passed through verbatim (e.g. `display_columns`). A name LimeSurvey stores per language (`prefix`, `suffix`, `choice_title`, `printable_help`, ...) is emitted once per declared language by `write_lss()`, and may be given either as one string for every language or keyed by language code.

Value

An object of class `lss_spec`: the validated specification with normalized questions (auto-numbered option codes filled in).

Languages

`languages` declares the survey languages, the primary one first; `languages[1]` is the base language `write_lss()` emits, and every other declared language is written alongside it. Every localizable text – survey title, welcome and end texts, group titles, question texts and help, option, row and column labels, the "other" label, quota names and messages – accepts either a plain string (read as the primary language) or a named character vector or list keyed by language code:

```
lss_spec(
  title = c(fr = "Enquete", en = "Survey"),
  languages = c("fr", "en"),
  groups = list(list(
    title = c(fr = "Profil", en = "Profile"),
    questions = list(list(
      code = "q1", kind = "yesno",
      text = c(fr = "Etes-vous d'accord ?", en = "Do you agree?")))))
)
```

The spec keeps one canonical form (a named list over the declared languages) and is strict: as soon as several languages are declared, every text must supply every one of them. A missing translation is precisely what `audit_lss()` flags when reading a `.lss`, so the spec refuses to author one. `write_lss()` writes every declared language: `languages[1]` becomes the survey's base language and the others its additional languages, each localized section carrying one row per language.

See Also

`write_lss()` to emit the `.lss` file, `read_lss()` and `audit_lss()` to read it back and check it.

Examples

```
spec <- lss_spec(
  title = "Demo",
  languages = "fr",
  groups = list(list(
    title = "Profil",
```

```
    questions = list(
      list(code = "consent", kind = "single", text = "Participez-vous ?",
           mandatory = TRUE,
           options = list(list(text = "Oui"), list(text = "Non"))),
      list(code = "raisons", kind = "multiple", text = "Pourquoi ?",
           relevance = "consent = 1", max_answers = 2,
           options = list(
             list(text = "Une raison"), list(text = "Une autre"),
             list(text = "Encore une"),
             list(text = "Aucune raison", exclusive = TRUE),
             list(text = "Autre raison", other = TRUE)))
    )
  ))
)
spec$groups[[1]]$questions[[2]]$options[[4]]$code
```

`lss_template_docx` *Write a blank Word template for authoring a questionnaire*

Description

[Experimental]

Usage

```
lss_template_docx(path, lang = "fr", kinds = lss_kinds$kind, languages = lang)
```

Arguments

`path`	Character. Path of the `.docx` file to write.
`lang`	Language of the template's labels and example wording, one of `"en"`, `"fr"` (default), `"de"`, `"es"`, `"it"`.
`kinds`	Character vector of kinds to illustrate, `lss_kinds$kind` (all 21) by default; the order of the kind table is kept.
`languages`	Content languages of the questionnaire to be written, the first one being the primary language. Defaults to `lang`, giving a single-language form. With several, every text row is repeated once per language, keyed `Wording [fr]`, `Wording [en]` and so on, and the reader then requires each of them: a questionnaire with a missing translation is what `audit_lss()` exists to catch. Independent of `lang`, which only sets the language of the form's own labels.

Details

Experimental. Generate the blank authoring form: one example question per requested kind, every default pre-filled, and a muted hint under each key telling the author the syntax the field expects. The template is always **generated** from the kind table – the package ships no static `.docx` – so it cannot describe a field `lss_spec()` would refuse.

Delete the example questions you do not need, edit the others, then hand the file to the reader (0.3.0) to obtain an `lss_spec()` and, through `write_lss()`, a `.lss` file LimeSurvey imports.

The template is the render of a generated example specification, so the same object feeds the template, the tests and the vignette. The value of a Type row is always the spec kind (`single`, `array5`, ...); the localized type label is shown as the hint under the key, because it is many-to-one and could not be read back. Reserved words are localized: an option line reading `Autre` in a French template creates the native other option, and an ordinary option that happens to read "Autre" is written with an explicit code (`9 = Autre`) – which the hint under the Options key spells out.

Requires the suggested packages **officer** and **flextable**.

Value

Invisibly, the path to the written file.

See Also

`write_form_docx()` to render an existing specification, `lss_spec()`, `write_lss()`.

Examples

```
if (requireNamespace("officer", quietly = TRUE) &&
    requireNamespace("flextable", quietly = TRUE)) {
  out <- tempfile(fileext = ".docx")
  lss_template_docx(out, lang = "fr", kinds = c("single", "multiple"))
  file.exists(out)

  # A bilingual questionnaire, with French labels on the form itself.
  both <- tempfile(fileext = ".docx")
  lss_template_docx(both, lang = "fr", languages = c("fr", "en"),
                    kinds = "single")
  file.exists(both)
}
```

---

read_form_docx *Read a Word authoring form back into a survey specification*

Description

[Experimental]

Usage

```
read_form_docx(path)
```

Arguments

`path`	Character. Path of the `.docx` form to read.

Details

Experimental. Parse a `.docx` **authoring form** – the document `write_form_docx()` and `lss_template_docx()` produce, filled in by an author in Word – back into an `lss_spec()`, ready for `write_lss()` and a LimeSurvey import. The round trip is exact: a specification written as a form and read back describes the same survey.

Reading needs no suggested package: only **xml2** and `utils::unzip()`, so an author's file can be turned into a `.lss` on a bare installation. **officer** and **flextable** are needed to WRITE a form, never to read one.

The contract the document must honor – all of it written by `write_form_docx()`, and spelled out in the blank template's hints:

- one top-level two-column table per block (Survey, Group, Question, Quota), the key on the left and the value on the right; the first row of a block names it, and a title row starts a new block even in the middle of a table (pasting a block makes Word merge two tables).
- keys are matched by their TEXT, ignoring case, accents, quotes and a trailing colon, against the labels of all five chrome languages, so a French author can fill an English template. Only the first line of a key cell is read: the muted hints under the key are ignored.
- a value cell holds one value per line, where a line is a paragraph or a soft return (Enter or Shift+Enter). Options, rows and columns are read as `code = label`, or as a bare label that gets auto-numbered; the reserved word (`Other`, `Autre`, `Sonstiges`, `Otro`, `Altro`) alone or on the left of `=` is the native other option.
- the Type cell carries the kind code (`single`, `array5`, ...); a localized type label is accepted only when it names exactly one kind.
- `Mandatory` takes the yes/no words of any language (and `y`/`n`, `true`/`false`, `1`/`0`); blank means no. The `Filter` cell takes the mini-language of `lss_spec()` (`Q1 = 1`, `Q2 in [1, autre]`, `count(Q3) >= 2`); its keywords are matched whatever Word capitalized, and the case of a question or answer code is never changed.
- when the survey declares several languages, every localizable key is suffixed with a language code – `Wording [fr]`, `Wording [en]` – and every declared language must supply every text.
- the file must carry the custom document property `lssdoc-template-version`: it is the reader's proof that the document agreed to this contract. Tracked changes, content controls, merged cells, a third column, a nested table and Word's automatic list numbering are each refused with a classed error naming the block and the field.

Errors carry the class `lssdoc_bad_form` plus one leaf class (`lssdoc_bad_form_file`, `_marker`, `_layout`, `_block`, `_key`, `_value`, `_spec`), and every message names the block, the question code and the field. Use `check_form_docx()` to list every problem of a document at once instead of stopping at the first.

Value

An `lss_spec()` object.

See Also

`write_form_docx()` and `lss_template_docx()` to write the form, `check_form_docx()` for a dry run, `lss_spec()`, `write_lss()`.

Examples

```
if (requireNamespace("officer", quietly = TRUE) &&
    requireNamespace("flextable", quietly = TRUE)) {
  form <- tempfile(fileext = ".docx")
  lss_template_docx(form, lang = "fr", kinds = c("single", "text"))
  spec <- read_form_docx(form)
  spec
}
```

`read_lss` *Read a LimeSurvey* `.lss` *file*

Description

Read a LimeSurvey survey structure export (`.lss`, an XML file) and turn it into a structured `lss` object that the rest of the package can audit (`audit_lss()`) and render (`render_questionnaire()`, `render_audit()`). Parsing is fully local: the file is never uploaded anywhere.

Usage

```
read_lss(file)
```

Arguments

`file`	Character. Path to a `.lss` file. Must be a single string pointing to an existing file, otherwise a classed error is raised (`lssdoc_bad_path`, `lssdoc_file_not_found`).

Details

The `.lss` format is a LimeSurvey XML export. Since DBVersion 4xx/7xx the translatable text lives in dedicated localization sections (`*_l10ns`), keyed by language, while the structural sections hold identifiers and settings. `read_lss()` reads every section into a tidy data frame without mutating any user-facing identifier or text. A field that is present but empty (e.g. `<help/>`) is read as `""`; a field that is absent from a row is read as `NA`.

Failure is graceful by construction: a malformed, truncated, or non-UTF-8 file is refused in R with a classed `lssdoc_invalid_xml` error before or instead of any libxml2 diagnostic, and the parser is never allowed to fetch an external DTD or entity over the network.

Value

An object of class `lss`: a list with the survey languages, metadata, and one data frame per `.lss` section. Structural sections (`surveys`, `groups`, `questions`, `subquestions`, `answers`, `question_attributes`, `conditions`) stay separate from the localized text sections (`survey_language_settings`, `group_l10ns`, `question_l10ns`, `answer_l10ns`), which carry the per-language titles, labels, and help texts. All values are read verbatim as character.

Examples

```
# A synthetic four-language demo survey ships with the package.
demo <- system.file("extdata", "demo_survey.lss", package = "lssdoc")
lss <- read_lss(demo)
lss$languages
```

`render_audit` *Render the audit as a focused Word or PDF document*

Description

Build a short, action-oriented document containing only the audit findings: the same cover page as the full questionnaire document, summary counts, then one table per severity (errors, warnings, notes) listing every finding with its location and message. Use it for QA follow-up or to share issues with a colleague without distributing the full questionnaire.

Usage

```
render_audit(
  input,
  output,
  languages = NULL,
  logo = NULL,
  logo_width = 1.5,
  logo_height = 0.75,
  font = NULL,
  font_code = NULL,
  colors = NULL,
  authors = NULL,
  description = NULL,
  chrome_lang = NULL
)
```

Arguments

`input`	Either a path to a `.lss` file (character string) or a pre-parsed `lss` object returned by `read_lss()`. Passing a path parses it on the fly; passing an `lss` object avoids re-parsing in a workflow that already inspected the audit.
`output`	Character. Path to the file to create. The extension determines the output format: `.docx` writes a Word document directly, `.pdf` writes a Word document into a temporary location and converts it locally via LibreOffice (or Word, on Windows). Any other extension is rejected with `lssdoc_bad_output_ext`.
`languages`	Character vector of language codes used on the cover page. `NULL` (default) keeps all languages of the survey in their declared order.
`logo`	Optional path (character) to a PNG or JPEG image displayed at the top of the cover page. `NULL` (default) keeps the cover logo-free.

`logo_width`, `logo_height`
: Image dimensions in inches. Defaults `1.5` and `0.75`, tuned to a 2:1 logo. Resize or pre-crop your image to fit a different aspect ratio.

`font`
: Optional body font name (character). `NULL` (default) keeps Calibri. See `render_questionnaire()` for guidance on overrides.

`font_code`
: Optional monospace font (character) used for code-like content (variable codes, raw expressions). `NULL` (default) keeps Consolas.

`colors`
: Optional named list of hex color overrides for the editorial petrol-blue palette. `NULL` (default) keeps the package palette intact. Same shape and accepted names as in `render_questionnaire()`.

`authors`, `description`
: Optional cover-page credit block (`authors`) and free-form note (`description`). `NULL` (default) for both. Same shapes as in `render_questionnaire()`.

`chrome_lang`
: Language used for the document chrome (column headers, row labels, audit section). One of `"en"`, `"fr"`, `"de"`, `"es"`, `"it"`. `NULL` (default) follows `languages[1]` when supported, otherwise falls back to `"en"`.

Value

The `output` path, invisibly.

See Also

`audit_lss()` to inspect the same findings in the console; `render_questionnaire()` for the full questionnaire document.

Examples

```
## Not run:
# One-shot (path -> .docx)
render_audit(
  system.file("extdata", "demo_survey.lss",
              package = "lssdoc"),
  tempfile(fileext = ".docx")
)

# PDF output -- same call, just pass a .pdf path
render_audit("survey.lss", "qa.pdf")

## End(Not run)
```

`render_questionnaire` *Render a LimeSurvey questionnaire to a Word or PDF document*

Description

Build a professional questionnaire document from a LimeSurvey survey, displaying up to four languages side by side. Each question becomes a compact `flextable` with a meta header (variable code, type, mandatory, filter) shown once, language column headers, the question text per language, and the subquestion or answer-option rows underneath – codes on the left, labels per language on the right. Headings, a metadata cover page, an optional table of contents, and an optional audit summary tie the document together. Rendering uses the suggested packages **officer** and **flextable**; both must be installed.

Usage

```
render_questionnaire(
  input,
  output,
  languages = NULL,
  template = c("cards", "table"),
  layout = c("auto", "side-by-side", "stacked"),
  show_audit = TRUE,
  show_help = TRUE,
  show_attrs = c("prefix", "suffix", "other_replace_text", "validation"),
  show_technical_attrs = FALSE,
  page_format = c("auto", "A4-portrait", "A4-landscape", "A3"),
  show_toc = TRUE,
  show_index = TRUE,
  show_quotas = TRUE,
  show_header_title = TRUE,
  show_source = TRUE,
  show_item_heading = FALSE,
  show_raw_filter = FALSE,
  show_groups = TRUE,
  show_welcome = TRUE,
  show_endtext = TRUE,
  show_description = TRUE,
  show_consent = TRUE,
  show_privacy_settings = FALSE,
  show_admin_settings = FALSE,
  title = NULL,
  logo = NULL,
  logo_width = 1.5,
  logo_height = 0.75,
  font = NULL,
  font_code = NULL,
  colors = NULL,
  authors = NULL,
  description = NULL,
  chrome_lang = NULL,
  variable_names = c("brackets", "underscore"),
  base_size = 10L
```

)

Arguments

input	Either a path to a `.lss` file (character string) or a pre-parsed `lss` object returned by `read_lss()`. Passing a path parses it on the fly. Passing an `lss` object avoids re-parsing when the same survey is rendered with different options (e.g. multiple language subsets) in the same session.
output	Character. Path to the file to create. The extension determines the output format: `.docx` writes a Word document directly, `.pdf` writes a Word document into a temporary location and converts it locally via LibreOffice (or Word, on Windows). Any other extension is rejected with `lssdoc_bad_output_ext`.
languages	Character vector of language codes to display, **in the order they will appear as columns**. `NULL` (default) keeps all languages found in the `.lss` file in the order of the `<languages>` section. Acts both as a subset filter and an ordering: e.g. `c("en", "fr")` puts English first, `c("fr", "en")` puts French first. `languages[1]` is treated as the primary language (TOC entries, group fallback). Requesting a language absent from the survey is an error (`lssdoc_unknown_language`).
template	Output style. One of `"cards"` (the default) or `"table"`. • `"cards"` renders one detached pair of tables per item (meta table + item table), stacked vertically, with content languages displayed side-by-side in the item table. • `"table"` renders a single dense table covering the whole document: every variable is one tinted Question row carrying `No | Variable | Type | Mand. | Filter`, followed by one or more white Value rows. Group banners (and a group's description, when the author wrote one) become section rows; the column header repeats on every page. The body font steps down for three or four languages so the columns stay within the portrait content width.
layout	Reserved for future use. Currently `"auto"` only.
show_audit	Logical. If `TRUE` (default), include an audit summary section near the top and inline markers on questions that carry findings. Set to `FALSE` for a clean reading copy.
show_help	Logical. If `TRUE` (default), include question help texts under the question text.
show_attrs	Character vector of question attributes to surface under the question text when present. Default keeps the attributes that change how respondents see the item: `"prefix"`, `"suffix"`, `"other_replace_text"`, `"validation"`. Add `"exclude_all_others"` or `"exclude_all_others_auto"` to also surface the row-level exclusivity flags (debug-style). Pass `character(0)` to hide all.
show_technical_attrs	Logical. If `TRUE`, include technical attributes such as `answer_order` and `location_*`. `FALSE` (the default) hides them.
page_format	Page format. One of `"auto"` (the default), `"A4-portrait"`, `"A4-landscape"`, or `"A3"`. `"auto"` is template-aware and never follows the language count (four languages fit on A4 portrait): the `"cards"` template gets A4 portrait, while the dense `"table"` template gets A4 landscape, which it needs to stay readable. Pass an explicit value to override; selecting a wider page makes every panel (meta

table, item table, audit/quota tables and the dense table) expand automatically to fill the content width, in both templates.

show_toc
: Logical. If TRUE (default), include a table of contents listing the groups (skipped automatically when the survey has fewer than two groups). For per-variable navigation, use show_index.

show_index
: Logical. If TRUE (default), append a variable index at the end of the document listing every item code with its number, sorted alphabetically.

show_quotas
: Logical. If TRUE (default), append a quotas section (after the end text, before the variable index) listing each sampling quota: its localized name, status (active, limit and action when full), the membership condition resolved to question codes and answer labels, and the localized "quota full" message. Skipped when the survey defines no quotas.

show_header_title
: Logical. If TRUE (default), show the survey title at the top right of every page (one line per displayed language, truncated to 80 characters). FALSE keeps only the X/Y page counter at the bottom right.

show_source
: Logical. If TRUE (default), show the **Source file** name and **Survey ID** rows in the cover metadata table. Pass FALSE to hide both (e.g. when sharing without exposing the LimeSurvey internals).

show_item_heading
: Logical. If FALSE (the default), the meta table starts each item directly, for a compact layout. If TRUE, a bold "N. variable" heading is added above each item for scroll-time navigation.

show_raw_filter
: Logical. If FALSE (the default), the Filter cell shows only the human-readable form (e.g. Q1 = 1) – editorial style, matching ESS / MOSAiCH / GESIS conventions. Set to TRUE to also surface the raw LimeSurvey relevance expression underneath in small italic gray (e.g. !is_empty(Q1.NAOK) && (Q1.NAOK == 1)), useful for QA cross-checks. The raw form is always shown when the plain form could not be simplified.

show_groups
: Logical. If TRUE (default), show the group banners (cards layout) or group rows (table layout). Pass FALSE to flatten the document into a single sequence of items with no section breaks (useful when groups exist only as internal organization).

show_welcome
: Logical. If TRUE (default), include the survey's multilingual welcome text (surveyls_welcometext) as a side-by-side block (cards) or embedded row (table).

show_endtext
: Logical. If TRUE (default), include the survey's multilingual end text (surveyls_endtext). Same treatment as show_welcome.

show_description
: Logical. If TRUE (default), include the survey's multilingual description (surveyls_description) – the "what this survey is about" intro that LimeSurvey shows above the welcome text on the landing page.

show_consent
: Logical. If TRUE (default), render a data protection and consent block in the front matter (before the welcome text): the survey's privacy policy notice (surveyls_policy_notice) and its consent checkbox label (surveyls_policy_notice_label), side by side across languages, with the checkbox drawn as an empty box. Skipped when the survey turns the policy notice off or carries no notice text.

show_privacy_settings
: Logical. If FALSE (the default), omit the survey-level privacy / tracking flags from the cover. Set to TRUE to surface anonymized, save partial, datestamp, ipaddr, and refurl rows – useful for ethics committee submissions.

show_admin_settings
: Logical. If FALSE (the default), omit the survey-level administrative settings. Set to TRUE to surface alias, end URL with description, and active flag rows.

title
: Optional override of the survey title shown on the cover and the top-right header. NULL (default) uses the per-language titles from the .lss survey settings. Pass a single string to use the same title in every displayed language, or a named character vector keyed by language code (e.g. c(fr = "Mon titre", de = "Mein Titel")) for per-language overrides.

logo
: Optional path (character) to a PNG or JPEG image displayed at the top of the cover page. NULL (default) keeps the cover logo-free, matching the neutral style of survey-methodology references (ESS, MOSAiCH, Panel). The .lss file does not embed a logo, so this image must be supplied by the caller.

logo_width, logo_height
: Image dimensions in inches. Defaults 1.5 and 0.75, tuned to a 2:1 logo. Resize or pre-crop your image to fit a different aspect ratio.

font
: Optional body font name (character). NULL (default) keeps Calibri, which is pre-installed on every recent Windows Office and metric-substituted with Carlito (OFL) on Mac and Linux LibreOffice, so column widths stay stable across platforms. Pass any string to override (e.g. "Source Sans 3", "IBM Plex Sans", or a corporate brand font); install the font on the machine that opens the document, otherwise the reader's application substitutes its own fallback face.

font_code
: Optional monospace font name (character) used for code-like content: the variable column in each meta table, the raw relevance expression under each filter cell, and the variable index entries. NULL (default) keeps Consolas; pass "JetBrains Mono" or "IBM Plex Mono" for sharper code style.

colors
: Optional named list of hex color overrides for the editorial petrol-blue palette. NULL (default) keeps the package palette intact. Accepted names: "primary" (group filets, headers text and item top borders), "accent" (hyperlinks, ORCID iD, URL auto-links, group under-line in cards), "band" (light header backgrounds), "band_dark" (the meta-table dark header in cards), "zebra" (the very-light tint on Question rows in the table template), "grid" (the 0.5 pt border color), "text", "muted". Each value must be a hex string ("#XXXXXX" or "#XXX"). Unknown keys are rejected (lssdoc_bad_colors). Useful for honoring an institutional brand: e.g. colors = list(primary = "#5C9F1A", accent = "#7FA82E") produces a LimeSurvey-green document.

authors
: Optional credit block for the questionnaire's designers, displayed on the cover page below the subtitle. Each author is shown centered on its own line as Name -- Affiliation; when an ORCID iD is provided, a smaller monospace line below shows ORCID 0000-0000-0000-0000 as a hyperlink to https://orcid.org/<id>. Accepts:

  - NULL (default): no authorship block.
  - An **unnamed character vector** (c("Jane Doe", "John Doe")): each entry becomes a line with no affiliation.

- A **named character vector** (`c("Jane Doe" = "HESAV")`): names are authors, values are affiliations. Use `""` to render an author without affiliation.
- A **list of named lists** for the full form, e.g. `list(list(name = "Jane Doe", affiliation = "HESAV", orcid = "0009-0001-2345-6789"), list(name = "John Doe", affiliation = "HESAV"))`. The `name` field is required; `affiliation` and `orcid` are optional.

`description` Optional free-form text (single string) shown on the cover page below the authors block. `NULL` (default) omits the block. Useful for a citation hint, a funding acknowledgement, a methodology note, or a link to a related publication. Line breaks (`\n`) split the block into separate centered lines; `http://` and `https://` tokens are rendered as clickable hyperlinks.

`chrome_lang` Language used for the **chrome** of the document (column headers, row labels, navigation titles, type labels, Value descriptors, audit section). One of `"en"`, `"fr"`, `"de"`, `"es"`, `"it"`. `NULL` (default) follows `languages[1]` when supported, otherwise falls back to `"en"`. Independent from `languages`, which controls the survey's content columns: e.g. `chrome_lang = "en"` with `languages = c("fr", "en")` produces an English-labelled document with French and English content. Spanish and Italian translations should be reviewed by a native speaker before publishing an official document.

`variable_names` How response-variable names are written, so the document matches the data file the reader holds. One of:

- `"brackets"` (default) – the exact column names of the **CSV / Excel data export**, so the variable index reproduces the raw data file column for column: `parent[subq]`, `parent[subq][1]` (dual scale), `parent[CH]` / `parent[other]` (multiple choice), `parent[59842]` (ranking, by answer id), `parent[_Ccomment]` (list-with-comment).
- `"underscore"` – the sanitized code form used by the Expression Manager / relevance equations and the SPSS / Stata / R exports (`parent_subq`, `parent_subq_1`). The two-dimensional arrays (array of numbers / texts with a second axis) and ranking questions are expanded so every produced column appears as its own entry either way.

`base_size` Body type size in points (default `10`). One lever scales the whole document: question text, item tables, the meta band, the quotas table, the variable index and the cover metadata all follow it, while headings and answer/help text keep their relative offsets. Useful for a roomier single-language render (e.g. `12`). The cover title and subtitle keep their fixed title-page sizes. Accepted range: 7 to 16.

Value

The `output` path, invisibly.

"LimeSurvey last save" date on the cover

The cover metadata table carries a row labelled *"LimeSurvey last save"* (or its localized equivalent). It is read verbatim from the `surveys.lastmodified` column of the `.lss`, which is the only timestamp LimeSurvey writes into the export – no other table (`questions`, `question_l10ns`, `answer_l10ns`, `groups`, etc.) carries a per-row modification date. The row is named "last save"

rather than "last modified" because LimeSurvey only bumps that field reliably when the user clicks **Save** on a survey-level form (Settings tab); editing a question text, an answer label, or a translation through the Question Editor does **not** consistently update it across LimeSurvey versions. If the date looks stale relative to your most recent edits, the workaround is to open *Survey settings* in LimeSurvey, click **Save** (no other change needed), then re-export the `.lss`. The next render will show the bumped timestamp.

Field-update prompt in Word

Opening the rendered `.docx` in Microsoft Word may surface a security-style prompt: *"This document contains fields that may refer to other files. Do you want to update the fields in this document?"*. This is expected: the package marks the page-number and bookmark-reference fields as needing a refresh so the footer shows the correct page count and the table of contents links resolve to the right pages on first open (this is also what makes headless PDF conversion via LibreOffice produce correctly paginated output without a manual F9). Clicking **Yes** is safe – the document has no `INCLUDETEXT`, `INCLUDEPICTURE`-linked, or DDE fields; the only external links are the ORCID and DOI URLs in the cover credits, which are static `HYPERLINK` targets and not fetched on update.

PDF output

When `output` ends in `.pdf`, the function first renders a `.docx` to a temporary location and then converts it locally via LibreOffice headless (or Word on Windows). LibreOffice (`soffice` executable) must be installed and on `PATH`; otherwise a classed error explains how to install it. Conversion stays on the user's machine: no upload, no network call. LibreOffice headless does not refresh Word field values (TOC, page counts) during conversion, so the table of contents may appear empty in the converted PDF. To obtain a PDF with a populated TOC, render to `.docx` instead, open it in Word (the TOC refreshes automatically) and use `File > Save As > PDF`.

See Also

`render_audit()` for the audit-only document; `audit_lss()` to inspect findings in the console without rendering; `read_lss()` to pre-parse a `.lss` file once and render multiple variants.

Examples

```
## Not run:
file <- system.file("extdata", "demo_survey.lss", package = "lssdoc")

# One-shot: parse + render Word document
render_questionnaire(file, tempfile(fileext = ".docx"))

# Same call, PDF output (format inferred from extension)
render_questionnaire(file, tempfile(fileext = ".pdf"))

# Parse once, render several variants without re-parsing
lss <- read_lss(file)
render_questionnaire(lss, tempfile(fileext = ".docx"),
                     languages = "en")
render_questionnaire(lss, tempfile(fileext = ".docx"),
                     template = "table",
                     languages = c("en", "fr"))
```

```
# Branded cover with authors block and palette override
render_questionnaire(
  lss,
  tempfile(fileext = ".docx"),
  template    = "table",
  chrome_lang = "en",
  colors      = list(primary = "#5C9F1A", accent = "#7FA82E"),
  authors     = list(
    list(name = "Jane Doe", affiliation = "HESAV",
         orcid = "0009-0001-2345-6789"),
    list(name = "John Doe", affiliation = "HESAV",
         orcid = "0009-0002-3456-7890")
  )
)

## End(Not run)
```

`write_form_docx` *Render a survey specification as a Word authoring form*

Description

[Experimental]

Usage

```
write_form_docx(spec, path, lang = NULL, hints = FALSE, strict = TRUE)
```

Arguments

`spec`	An `lss_spec()` object, or a plain list with the same structure (it is then validated through `lss_spec()` first).
`path`	Character. Path of the `.docx` file to write.
`lang`	Language of the form's own labels (its "chrome"), one of `"en"`, `"fr"`, `"de"`, `"es"`, `"it"`. `NULL` (default) follows the survey's primary language when it is one of them, English otherwise. It is independent of the survey's content languages.
`hints`	Logical. Add the muted syntax hint under each key (`"one per line, \"1 = Label\""`, ...). `FALSE` by default; `lss_template_docx()` turns it on for the blank template.
`strict`	Logical, used only when `spec` is an `lss` object read by `read_lss()`: it is converted with `as_lss_spec()`, and `strict` is passed to it. `TRUE` (default) refuses a survey carrying anything the specification cannot express; `FALSE` renders the rest of it and warns.

Details

Experimental. Write an `lss_spec()` as a `.docx` **form**: one two-column table per block (Survey, Group, Question, Quota), keys on the left, values on the right. Unlike the review documents produced by `render_questionnaire()`, this document is meant to be *edited*: an author fills or changes the value cells in Word and the companion reader (0.3.0) turns the file back into a specification.

The rows a Question block carries are decided by the question's kind, so the form shows exactly the fields that kind accepts and never a field the validator would refuse. Defaults are written as real values (Mandatory `No`, an empty Filter, the other option at the End) rather than as placeholders, because anything sitting in a value cell is content.

Conventions the document obeys, and the companion reader relies on:

- Each block is a top-level two-column table whose first row is its title row; two empty paragraphs separate two blocks. No cell is merged.
- Multi-valued fields (Options, Rows, Columns, Exclusive) hold one value per line. A line break inside a single-line field (an option label, a title, a quota name) is refused with a classed error naming the question, the field and the offending lines.
- The Type row carries the spec kind (`single`, `array5`, ...) as its value; the localized type label ("Single choice") is deliberately many-to-one, so it appears only as the key's hint in a blank template, never as content.
- Options are always written with an explicit code, `1 = Label`; the native other option is written with the reserved word of the form language (`Other`, `Autre`, `Sonstiges`, `Otro`, `Altro`), alone when the option has no label and as `Other = Label` as soon as it has one.
- When the spec declares several languages, every localizable key is written once per language and suffixed with the language code – `Question [fr]`, `Question [en]` – primary language first, the primary one suffixed as well.
- The file carries the custom document property `lssdoc-template-version`, the version of this contract.

Requires the suggested packages **officer** and **flextable**.

Value

Invisibly, the path to the written file.

See Also

`lss_template_docx()` for a blank template, `lss_spec()`, `write_lss()`, `render_questionnaire()`.

Examples

```
if (requireNamespace("officer", quietly = TRUE) &&
    requireNamespace("flextable", quietly = TRUE)) {
  spec <- lss_spec(
    title = "Demo",
    groups = list(list(title = "G", questions = list(
      list(code = "q1", kind = "single", text = "Oui ou non ?",
           options = list(list(text = "Oui"), list(text = "Non")))
    )))
```

```
  )
  out <- tempfile(fileext = ".docx")
  write_form_docx(spec, out, lang = "fr")
  file.exists(out)
}
```

`write_lss` *Write a survey specification to an importable* `.lss` *file*

Description

[Experimental]

Usage

```
write_lss(spec, file, sid = 100001L, settings = list())
```

Arguments

`spec`	An `lss_spec()` object, or a plain list with the same structure (it is then validated through `lss_spec()` first).
`file`	Character. Path of the `.lss` file to write.
`sid`	Integer. Survey id embedded in the file. LimeSurvey assigns a fresh id on import when this one is taken, so the value rarely matters.
`settings`	Named list of survey fields overriding the built-in defaults (e.g. `list(anonymized = "Y", showprogress = "Y")`). A name belonging to the `surveys` table takes a single string; a name belonging to the per-language `surveys_languagesettings` table (`surveyls_dateformat`, `surveyls_numberformat`, `surveyls_description`, the e-mail templates, ...) takes either a single string applied to every language or a list or vector keyed by language code – see the *Languages* section. The defaults ship with the package and come from a real LimeSurvey 6 export, scrubbed – see `lss_default_surveys_fields` in the sources for the rationale.

Details

Experimental. Turn an `lss_spec()` specification into a LimeSurvey structure file (`.lss`) that imports directly through *Create survey* -> *Import*. The output targets LimeSurvey 6 (DBVersion 700). The emitted file can be read back with `read_lss()`, checked with `audit_lss()` and rendered with `render_questionnaire()` – so the document reviewers read is produced from the very file LimeSurvey receives.

Mapping choices, each validated against real LimeSurvey 6 imports:

- Each kind maps to a LimeSurvey type and theme attested by a corpus of real exports (see `lss_kinds` in the sources). Options of single-choice lists, rankings and array columns are emitted as `answers`; options of multiple-choice questions, item batteries and array rows as `subquestions`; scalar kinds and implicit scales (yes/no, gender, five-point, 5/10-point arrays) emit none.

- The native `other` option is emitted as `other = "Y"` plus the localized attribute `other_replace_text`. Localized attributes MUST carry the language code: emitted without one, LimeSurvey silently ignores them and shows its default wording. Global attributes (`exclude_all_others`, `max_answers`, ...) stay language-less.
- `other_position` / `other_position_code` control where the other option is displayed; `exclude_all_others` accepts several codes separated by `;`.
- Relevance equations are translated from the minimal syntax of `lss_spec()` into ExpressionScript (`code.NAOK == "1"`).
- Quotas are emitted with the terminate action and the quota's `limit` (zero unless the spec gives one).
- A group's optional `description` is emitted into `group_l10ns.description` (empty when the spec gives none).
- A mandatory or capped ranking also receives `min_answers = 1`, overridable through the question's `attributes`.

Value

Invisibly, the path to the written file.

Languages

Every language the spec declares is written. The `surveys` row carries the base language – `languages[1]` – as `language` and the others, space-separated, as `additional_languages`. Every localized section (`surveys_languagesettings`, `group_l10ns`, `question_l10ns` for questions and subquestions alike, `answer_l10ns`, `quota_languagesettings`) receives one row per language, grouped by entity as a real LimeSurvey export groups them. `lss_spec()` already requires every declared language for every text, so no translation can go missing at emission.

The `<languages>` element lists the additional languages first and the base language last, the order LimeSurvey itself writes. It is only a membership set: read the base language from `read_lss()$base_language`, never from `read_lss()$languages[1]`.

Localized question attributes – `other_replace_text`, `prefix`, `suffix`, `choice_title`, `rank_title`, `printable_help`, ... – are emitted once per language, each row carrying its language code; without it LimeSurvey silently ignores the attribute and shows its own default wording. An attribute passed through `question$attributes` under one of those names is treated the same way: a plain string is repeated in every language, a value keyed by language code is resolved language by language (`attributes = list(prefix = "CHF")` or `list(choice_title = c(fr = "Choix", en = "Choice"))`). Every other attribute stays language-less, as LimeSurvey stores it.

Per-language survey settings work the same way: `settings` accepts a single value repeated in every `surveys_languagesettings` row, or a list keyed by language code for the fields LimeSurvey genuinely varies – `list(surveyls_dateformat = c(fr = "5", en = "2"))` gives the French respondent a `dd.mm.yyyy` date picker and the English one `mm/dd/yyyy`. A declared language missing from such a value is an error, not a silent fallback.

A quota is localized through `quotals_name` and `quotals_message`, one row per language. The administration-side label `quota.name` has a single column in LimeSurvey and therefore keeps the primary-language wording.

See Also

lss_spec(), read_lss(), audit_lss(), render_questionnaire().

Examples

```
spec <- lss_spec(
  title = "Demo",
  groups = list(list(title = "G", questions = list(
    list(code = "q1", kind = "single", text = "Oui ou non ?",
         options = list(list(text = "Oui"), list(text = "Non")))
  )))
)
out <- tempfile(fileext = ".lss")
write_lss(spec, out)
audit_lss(out)
```

Index

as_lss_spec, 2
as_lss_spec(), *22*
audit_lss, 4
audit_lss(), *3*, *9, 10*, *13*, *15*, *21*, *24*, *26*

check_form_docx, 6
check_form_docx(), *12*

lss_spec, 7
lss_spec(), *3, 4*, *10–12*, *22–26*
lss_template_docx, 10
lss_template_docx(), *6*, *12*, *22, 23*

print.lss_form_check *(*check_form_docx*)*, 6

read_form_docx, 11
read_form_docx(), *3*, *6*
read_lss, 13
read_lss(), *2–5*, *9*, *14*, *17*, *21, 22*, *24*, *26*
render_audit, 14
render_audit(), *5*, *13*, *21*
render_questionnaire, 15
render_questionnaire(), *13*, *15*, *23, 24*, *26*

utils::unzip(), *12*

write_form_docx, 22
write_form_docx(), *3, 4*, *6*, *11, 12*
write_lss, 24
write_lss(), *3, 4*, *8, 9*, *11, 12*, *23*