Package ‘FormIOr’

February 19, 2026

Title Download, Clean, and Export 'FormIO' Submissions

Version 1.0.2

Description Tools for downloading 'FormIO' submissions via the 'FormIO API' <https://apidocs.form.io>, flattening nested structures, cleaning and simplifying repeated and multi-select answers, generating diagnostics (codebooks, summaries, plots), maintaining audit logs of dataset processing, and exporting results to 'Excel' or delimited files. Includes an interactive end-to-end workflow for non-technical users.

License MIT + file LICENSE

URL https://github.com/drrodrigosolis-dev/RSS-FormIOr

BugReports https://github.com/drrodrigosolis-dev/RSS-FormIOr/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 3.5)

LazyData true

Imports crayon, dplyr, httr, jsonlite, magrittr, purrr, rlang, stats, tibble, tidyr, utils

Suggests knitr, miniUI, DT, reactable, openxlsx, pkgdown, rmarkdown, rstudioapi, shiny, testthat (>= 3.0.0), wordcloud, writexl

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Rodrigo Solis Sosa [aut, cre]

Maintainer Rodrigo Solis Sosa <dr.rodrigo.solis@gmail.com>

Contents

`FormIOr-package` *FormIOr: Download, Clean, and Export 'FormIO' Submissions*

Description

Tools for downloading 'FormIO' submissions via the 'FormIO API' `https://apidocs.form.io`, flattening nested structures, cleaning and simplifying repeated and multi-select answers, generating diagnostics (codebooks, summaries, plots), maintaining audit logs of dataset processing, and exporting results to 'Excel' or delimited files. Includes an interactive end-to-end workflow for non-technical users.

Author(s)

Maintainer: Rodrigo Solis Sosa <dr.rodrigo.solis@gmail.com>

See Also

Useful links:

- https://github.com/drrodrigosolis-dev/RSS-FormIOr
- Report bugs at https://github.com/drrodrigosolis-dev/RSS-FormIOr/issues

---

`.formior_state` *Internal package state*

Description

Stores ephemeral session state used by interactive helpers.

Usage

```
.formior_state
```

Format

An object of class `environment` of length 1.

`AddSections` *Add Hierarchical Sections to FormIO Response Columns*

Description

This interactive function guides the user through categorizing columns of a flattened FormIO response dataset into hierarchical sections (up to 3 levels deep). It is designed to facilitate easier analysis by adding grouping variables to columns. The function handles input that may be a data frame, a list from `FlattenSubmissions()`, or raw output from `GetResponses()`. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
AddSections(x)
```

Arguments

`x`	A data frame (possibly with nested list-columns), or a list containing `FlatResponses` (a flattened data frame) or `submission_data` (from `GetResponses()` with `content.only = FALSE`).

Details

The function first extracts or flattens the input to obtain a flat data frame of responses. It then prompts the user to:

- Select the depth of sections (1, 2, or 3).
- Provide comma-separated names for sections at each level (no spaces or special characters).
- Assign row numbers (from the displayed column list) to each section at each level. Row numbers can be single values, comma-separated lists, or ranges (e.g., "1:5,8,10:12").

Empty assignments are filled with "General". The process uses console clearing (`\014`) and colored prompts (requires the `crayon` package).

Value

A list with two elements:

`FlatResponses`	The original flattened data frame of responses.
`Sections`	A `tibble` with columns `No` (column number), `Names` (original column names), and `Level-1`, `Level-2`, `Level-3` (section assignments, filled with "General" if empty).

Examples

```
## Not run:
# Assuming FoodTypes is a sample dataset with possible nests
data("FoodTypes")
sectioned <- AddSections(FoodTypes)
print(sectioned$Sections)

## End(Not run)
```

`AdjustSubmissions` *Adjust submissions by ID (delete or edit specific values)*

Description

Sometimes you need to make small, targeted fixes before you export: remove test submissions, correct a value for one submission, or blank out a field for privacy reasons.

Usage

```
AdjustSubmissions(
  x,
  id_col = 1,
  delete_ids = NULL,
  updates = NULL,
  return_flat = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`id_col`	Integer or character. Submission ID column (default 1).
`delete_ids`	Optional character vector of submission IDs to delete.
`updates`	Optional updates to apply. Supported formats: • A data.frame with columns `id`, `column`, `value` • A data.frame with columns `submissionId`/`submission_id`, `column`, `value` • A list of lists, each with elements `id`, `column`, `value` Values are coerced to the target column type when possible. Use `"NA"` (or an actual `NA`) to set a value to missing.
`return_flat`	Logical. If `TRUE` and `x` came from `FlattenSubmissions()`, include the updated list as `flat` in the output.
`quiet`	Logical. If `FALSE`, prints a short summary.

Details

This helper lets you:

- Delete submission(s) by ID (remove rows)
- Update one or more column values for specific submission ID(s)

It works on either a plain data frame or the list produced by `FlattenSubmissions()`. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Value

A list with:

data Updated data frame (the cleaned data)

summary Data frame summarizing deletions/updates

changes Data frame listing each requested update and its status

flat If `return_flat = TRUE`, the updated FlattenSubmissions-style list (for workflows that expect `FlatResponses`)

Examples

```
df <- data.frame(
  submissionId = c("a", "b", "c"),
  status = c("ok", "test", "ok"),
  stringsAsFactors = FALSE
)

# Delete one submission and update a value
out <- AdjustSubmissions(
  df,
  id_col = "submissionId",
  delete_ids = "b",
 updates = data.frame(id = "c", column = "status", value = "review", stringsAsFactors = FALSE),
  quiet = TRUE
)
out$data
```

`alternate_base_url` *Compute the alternate CHEF base URL*

Description

Compute the alternate CHEF base URL

Usage

```
alternate_base_url(base_url)
```

Arguments

`base_url`	Base API URL.

Value

Alternate base URL or `NULL`.

`apply_repeat_strategy` *Apply a repeat-resolution strategy*

Description

Apply a repeat-resolution strategy

Usage

```
apply_repeat_strategy(values, strategy, sep, unique)
```

Arguments

`values`	Vector of values for one submission.
`strategy`	Strategy name.
`sep`	Separator for concatenation.
`unique`	Logical. Remove duplicates before concatenation/first/last.

Value

A single resolved value.

`as_form_schema` *Resolve a form schema from various inputs*

Description

Accepts a schema list, JSON string, file path, or metadata list and returns a schema list with `components`.

Usage

```
as_form_schema(form, version = "latest")
```

Arguments

`form`	Form schema list, JSON string, file path, or metadata list.
`version`	Version identifier used when fetching schema from metadata.

Value

A form schema list.

`ask_keep_rows` *Prompt for rows to keep*

Description

Prompt for rows to keep

Usage

```
ask_keep_rows(n_rows)
```

Arguments

`n_rows`	Number of rows in the duplicate group.

Value

Integer indices to keep, or `"quit"` to stop.

`AskCredentials` *Ask for Form credentials*

Description

Collects the Form ID and API key in an interactive session and caches them for this R session.

Usage

```
AskCredentials()
```

Value

Named character vector with `ID` and `Key`.

Examples

```
## Not run:
AskCredentials()

## End(Not run)
```

`askingDepth` *Prompt for section depth*

Description

Interactively asks the user to choose the number of section levels (1–3).

Usage

```
askingDepth()
```

Value

An integer depth between 1 and 3.

`askingSections` *Prompt for section names at each level*

Description

Collects comma-separated section names for each requested level.

Usage

```
askingSections(DepthAsked)
```

Arguments

`DepthAsked`	Integer depth selected by the user.

Value

A list of character vectors, one per section level.

`assignSections` *Assign section names to columns*

Description

Walks the user through mapping columns to section labels at each level.

Usage

```
assignSections(Sections, df)
```

Arguments

`Sections`	List of section name vectors.
`df`	A data frame of column numbers and names with empty level columns.

Value

The input data frame with section assignments filled in.

`audit_enter` *Increment audit nesting depth*

Description

Tracks nested calls so prompts/logs happen only at top level.

Usage

```
audit_enter()
```

Value

Updated depth.

`audit_exit` *Decrement audit nesting depth*

Description

Ensures nesting depth does not fall below zero.

Usage

```
audit_exit()
```

Value

Updated depth (invisibly).

`auto_strategy` *Pick an automatic repeat strategy*

Description

Pick an automatic repeat strategy

Usage

```
auto_strategy(values)
```

Arguments

`values`	Vector of column values.

Value

A strategy name.

`bind_rows_safe` *Safely bind a list of row lists*

Description

Safely bind a list of row lists

Usage

```
bind_rows_safe(rows)
```

Arguments

`rows`	List of row lists.

Value

A data.frame (possibly empty).

`bucket_time` *Bucket times into intervals*

Description

Converts timestamps into day/week/month/hour buckets.

Usage

```
bucket_time(times, interval, tz = "UTC")
```

Arguments

`times`	POSIXct vector of times.
`interval`	One of "day", "week", "month", or "hour".
`tz`	Time zone for bucketing.

Value

A vector of bucketed times/dates.

`build_codebook` *Build a codebook table*

Description

Build a codebook table

Usage

```
build_codebook(
  data,
  field_info = NULL,
  include_summary = TRUE,
  max_levels = 20
)
```

Arguments

`data`	Data frame of responses.
`field_info`	Optional field dictionary data frame.
`include_summary`	Logical. Include numeric summaries.
`max_levels`	Maximum number of levels to list for categorical data.

Value

A data.frame codebook.

`build_compare_frame_by_id` *Build a comparison table for duplicate groups*

Description

Build a comparison table for duplicate groups

Usage

```
build_compare_frame_by_id(
  df,
  rows_idx,
  ids,
  unique_ids,
  compare_cols,
  highlight_cols = NULL
)
```

Arguments

`df`	Data frame of responses.
`rows_idx`	Row indices for the group.
`ids`	Submission IDs for all rows.
`unique_ids`	Unique IDs in the group.
`compare_cols`	Columns to display.
`highlight_cols`	Columns to highlight when values differ.

Value

A data frame ready for display.

`build_duplicate_keys` *Build duplicate group keys*

Description

Build duplicate group keys

Usage

```
build_duplicate_keys(df, key_cols)
```

Arguments

`df`	Data frame of responses.
`key_cols`	Columns that define a duplicate group.

Value

A list with `key_id` and `group_keys`.

`build_path` *Build a dotted path from parent and key*

Description

Build a dotted path from parent and key

Usage

```
build_path(parent_path, key)
```

Arguments

`parent_path`	Parent path string.
`key`	Current key segment.

Value

Combined path string.

build_path_key *Build a stable path key for a component*

Description

Build a stable path key for a component

Usage

```
build_path_key(key, label, comp_type, index)
```

Arguments

key	Component key.
label	Component label.
comp_type	Component type.
index	Position index used as fallback.

Value

A path-safe key string.

build_survey_question_rows *Build rows for survey question components*

Description

Build rows for survey question components

Usage

```
build_survey_question_rows(
  comp,
  parent_path,
  section,
  parent_key,
  parent_path_full
)
```

Arguments

comp	Survey component definition.
parent_path	Parent path key.
section	Current section label.
parent_key	Parent key name.
parent_path_full	Parent path for display.

Value

A list of row lists for each survey question.

`build_time_sequence` *Build a full time sequence*

Description

Generates a sequence of evenly spaced periods from start to end.

Usage

```
build_time_sequence(start, end, interval, tz = "UTC")
```

Arguments

`start`	Start date/time.
`end`	End date/time.
`interval`	One of "day", "week", "month", or "hour".
`tz`	Time zone for POSIXct sequence.

Value

A sequence of POSIXct or Date values.

`canon_key` *Canonicalize a key for matching*

Description

Canonicalize a key for matching

Usage

```
canon_key(x)
```

Arguments

`x`	Character vector.

Value

Lowercased, alphanumeric-only string.

`coerce_export_sheets` *Coerce input into exportable sheets*

Description

Coerce input into exportable sheets

Usage

```
coerce_export_sheets(data, sheet = "Data")
```

Arguments

`data`	Data frame, list of data frames, or FlattenSubmissions list.
`sheet`	Default sheet name for a single data frame.

Value

A named list of data frames.

`coerce_list_columns` *Coerce list columns to text*

Description

Coerce list columns to text

Usage

```
coerce_list_columns(df)
```

Arguments

`df`	Data frame with possible list columns.

Value

Data frame with list columns converted to text.

`coerce_repeat_values` *Coerce repeated values to stable text*

Description

Coerce repeated values to stable text

Usage

```
coerce_repeat_values(values)
```

Arguments

`values`	Vector or list column values.

Value

A vector with list/data.frame values coerced to text.

`coerce_single_df` *Coerce input to a single data frame*

Description

Coerce input to a single data frame

Usage

```
coerce_single_df(data, arg_name = "data")
```

Arguments

`data`	Data frame or list containing a data frame.
`arg_name`	Argument name for error messages.

Value

A data.frame.

`coerce_time` *Coerce values to POSIXct*

Description

Tries to parse dates/times into POSIXct, falling back to Date parsing.

Usage

```
coerce_time(values, tz = "UTC")
```

Arguments

`values`	Vector of values to parse.
`tz`	Time zone for parsing.

Value

A POSIXct vector (possibly with `NA` values).

`coerce_update_value` *Coerce an update value to a target column type*

Description

Applies type-aware coercion and handles missing values.

Usage

```
coerce_update_value(value, template)
```

Arguments

`value`	The new value provided by the user.
`template`	The existing column used to infer type.

Value

A value coerced to the column's type.

`collect_components` *Collect components recursively*

Description

Collect components recursively

Usage

```
collect_components(
  components,
  parent_path = NULL,
  section = NULL,
  include = "input",
  expand_surveys = FALSE
)
```

Arguments

`components`	Component list to traverse.
`parent_path`	Current path prefix.
`section`	Current section label.
`include`	Which components to include.
`expand_surveys`	Logical. Expand survey components into question rows.

Value

A list of row lists describing components.

`collect_schema_candidates` *Collect schema-like candidates from nested lists*

Description

Collect schema-like candidates from nested lists

Usage

```
collect_schema_candidates(x, require_schema_like = TRUE, out = NULL)
```

Arguments

`x`	Object to traverse.
`require_schema_like`	Logical. Require strict schema root criteria.
`out`	Accumulator for recursion.

Value

A list of candidate schema objects.

CompactSelections *Compact checkbox/multi-select columns into a single readable column*

Description

When a question generates multiple TRUE/FALSE columns (e.g., select boxes), this function combines them into a single comma-separated column. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
CompactSelections(
  x,
  sep = "-",
  combine_sep = ", ",
  drop = TRUE,
  keep_empty = FALSE,
 yes_values = c(TRUE, "TRUE", "True", "true", "Yes", "YES", "yes", "Y", "y", 1, "1"),
 no_values = c(FALSE, "FALSE", "False", "false", "No", "NO", "no", "N", "n", 0, "0", "",
    " ", "NA"),
  return_flat = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`sep`	Separator used in column names to split prefix and option. Default `"-"` (matches `FlattenSubmissions()` naming).
`combine_sep`	Separator used between selected options.
`drop`	Logical. If `TRUE`, drop the original checkbox columns.
`keep_empty`	Logical. If `TRUE`, keep empty strings instead of `NA` when no options are selected.
`yes_values`	Values that should count as "selected".
`no_values`	Values that should count as "not selected" (defaults include `FALSE`, `"No"`, `0`, and blank strings).
`return_flat`	Logical. If `TRUE` and `x` came from `FlattenSubmissions()`, include the updated list as `flat` in the output.
`quiet`	Logical. If `FALSE`, prints a short summary.

Value

A list with:

data Data frame with compacted selection columns (the cleaned data)

summary Data frame describing which columns were compacted

flat If `return_flat` = `TRUE`, the updated FlattenSubmissions-style list (for workflows that expect `FlatResponses`)

Examples

```
## Not run:
compacted <- CompactSelections(flat)
head(compacted$data)

## End(Not run)
```

CompareFormVersions *Compare two versions of a form*

Description

Shows what changed between two form versions: fields added, removed, or updated. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
CompareFormVersions(
  old,
  new,
  by = c("key", "path"),
  include = c("input", "all"),
  old_version = "latest",
  new_version = "latest",
 compare_cols = c("label", "type", "required", "description", "default", "options",
    "section"),
  include_unchanged = FALSE,
  quiet = FALSE
)
```

Arguments

`old`	Form schema list, JSON string, or path to a JSON file (older version).
`new`	Form schema list, JSON string, or path to a JSON file (newer version).
`by`	How to match fields across versions: `"key"` (default) or `"path"`.
`include`	Which components to include: `"input"` (default) or `"all"`.
`old_version`	Which form version to use when `old` is metadata (from `GetFormMetadata()`). Use `"latest"` (default), a version number, or a version ID.
`new_version`	Which form version to use when `new` is metadata (from `GetFormMetadata()`). Use `"latest"` (default), a version number, or a version ID.
`compare_cols`	Character vector of columns to compare for changes.
`include_unchanged`	Logical. If `TRUE`, include unchanged fields in the output.
`quiet`	Logical. If `FALSE`, prints a short summary.

Details

If you pass the output of GetFormMetadata(), the schema is **not** included in that object. In that case the function will automatically fetch the schema using stored credentials (from AskCredentials() or GetFormMetadata()). If credentials are not available, it will stop with a clear message.

Tip: Use by = "key" for stable field keys, or by = "path" when keys are reused in repeating sections.

Value

A list with:

summary Counts of added, removed, changed, and unchanged fields

added Fields only in the new version

removed Fields only in the old version

changed Fields that changed (with before/after values)

unchanged Unchanged fields (if include_unchanged = TRUE)

Examples

```
old <- list(components = list(
  list(type = "textfield", key = "name", label = "Name", input = TRUE)
))
new <- list(components = list(
  list(type = "textfield", key = "name", label = "Full name", input = TRUE),
  list(type = "number", key = "age", label = "Age", input = TRUE)
))
CompareFormVersions(old, new)

## Not run:
old_meta <- GetFormMetadata(form_id = "123", api_key = "abc")
new_meta <- GetFormMetadata(form_id = "123", api_key = "abc")
CompareFormVersions(old_meta, new_meta)

## End(Not run)
```

---

components_count *Count components in a list or data frame*

Description

Count components in a list or data frame

Usage

```
components_count(components)
```

Arguments

components	Component list or data frame.

Value

Integer count of components.

`CrossTab` *Cross-tabulate two fields*

Description

Builds a simple cross-tabulation between two columns, returning both a wide table and a long table that includes counts and (optional) percents. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
CrossTab(
  x,
  row,
  col,
  include_na = FALSE,
  percent = c("overall", "row", "col", "none"),
  digits = 1,
  quiet = FALSE
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`row`	Column name or number for the rows.
`col`	Column name or number for the columns.
`include_na`	Logical. If `TRUE`, treat missing values as "(Missing)".
`percent`	How to calculate percentages. One of `"overall"` (default), `"row"`, `"col"`, or `"none"`.
`digits`	Number of decimal places for percentages.
`quiet`	Logical. If `FALSE`, prints a short summary.

Value

A list with:

row Resolved row field name

col Resolved column field name

percent Percent calculation mode

table Wide counts table as a data frame

long Long data frame with counts and percents

Examples

```
## Not run:
CrossTab(flat, "region", "program", percent = "row")

## End(Not run)
```

DeduplicateSubmissions

Deduplicate submissions by submission ID

Description

Keeps one row per submission ID, using a timestamp column when available (for example `created` or `modified`), otherwise keeps first/last row. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
DeduplicateSubmissions(
  x,
  id_col = 1,
  time_col = NULL,
  keep = c("last", "first"),
  return_flat = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`id_col`	Integer or character. Submission ID column (default 1).
`time_col`	Optional column name to use for ordering. If `NULL`, the function tries common timestamp names automatically.
`keep`	One of `"last"` (default) or `"first"`.
`return_flat`	Logical. If `TRUE` and `x` came from `FlattenSubmissions()`, include the updated list as `flat` in the output.
`quiet`	Logical. If `FALSE`, prints a short summary.

Value

A list with:

data Deduplicated data frame (the cleaned data)

summary List with counts and the time column used

flat If `return_flat` = `TRUE`, the updated FlattenSubmissions-style list (for workflows that expect `FlatResponses`)

Examples

```
## Not run:
flat <- FlattenSubmissions(GetResponses(form_id = "123", api_key = "abc"))
dedup <- DeduplicateSubmissions(flat, id_col = "submissionId")
nrow(dedup$data)

## End(Not run)
```

`default_non_form_cols` *Suggest default non-FormIO columns*

Description

Suggest default non-FormIO columns

Usage

```
default_non_form_cols(df_names, id_col_name, n = 3)
```

Arguments

`df_names`	Column names.
`id_col_name`	Submission ID column.
`n`	Number of columns to return.

Value

A character vector of suggested columns.

`default_stopwords` *Default stopword list*

Description

Provides a small set of common English stopwords.

Usage

```
default_stopwords()
```

Value

A character vector of stopwords.

`default_yes_values` *Default values treated as "yes"*

Description

Default values treated as "yes"

Usage

```
default_yes_values()
```

Value

A vector of values considered affirmative.

`derive_prefix` *Derive prefixes from column names*

Description

Derive prefixes from column names

Usage

```
derive_prefix(names_vec, sep)
```

Arguments

`names_vec`	Column names.
`sep`	Separator between prefix and suffix.

Value

Prefixes or `NA` when no separator is present.

`DescribeForm` *Describe a form in plain language*

Description

This is a simple overview of a form: the name, version, and how many fields it contains. It is meant for non-technical users who want a quick summary. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
DescribeForm(
  form,
  include_fields = FALSE,
  version = "latest",
  include = c("input", "all"),
  quiet = FALSE
)
```

Arguments

`form`	Form schema list, JSON string, or path to a JSON file.
`include_fields`	Logical. If `TRUE`, include a field dictionary in the output (from `FieldDictionary()`).
`version`	Which form version to use when `form` is metadata (from `GetFormMetadata()`). Use `"latest"` (default), a version number (e.g., `3`), or a version ID.
`include`	Which components to count for fields. Defaults to input fields only (`"input"`). Use `"all"` to include layout components too.
`quiet`	Logical. If `FALSE`, prints a short summary.

Details

What you can pass to `form`:

- A **schema list** (from a schema API response)
- A **JSON string**
- A **path to a JSON file**
- The output of `GetFormMetadata()`

Important: `GetFormMetadata()` returns **metadata only**, not the full form. If you pass metadata, this function will automatically fetch the form schema using stored credentials (from `AskCredentials()` or `GetFormMetadata()`). If credentials are not available, it will stop with a clear message.

Note (CHEF): Some schema endpoints live at `/api/v1` while other endpoints (like exports) are under `/app/api/v1`. If a schema fetch fails at the provided `base_url`, FormIOr automatically retries the alternate base.

Tip: Use `include = "input"` (default) to count only the fields people fill in, or `include = "all"` to include layout items like panels and tabs.

Value

A list with:

meta Form details like title, name, ID, and version

counts How many components, fields, and sections

fields Field dictionary data frame (only if `include_fields = TRUE`)

Examples

```
form <- list(
  title = "Sample Form",
  name = "sample_form",
  version = 2,
  components = list(
    list(type = "textfield", key = "first_name", label = "First name", input = TRUE),
    list(type = "panel", title = "Details", components = list(
      list(type = "number", key = "age", label = "Age", input = TRUE)
    ))
  )
)
DescribeForm(form)

## Not run:
meta <- GetFormMetadata(form_id = "123", api_key = "abc")
DescribeForm(meta)

## End(Not run)
```

`detect_field_type` *Detect field type*

Description

Classifies a vector as "numeric", "date", or "categorical".

Usage

```
detect_field_type(values)
```

Arguments

`values`	A vector of field values.

Value

A character string describing the type.

`detect_input_flag` *Determine whether a component is an input*

Description

Determine whether a component is an input

Usage

```
detect_input_flag(comp)
```

Arguments

`comp`	Component definition.

Value

`TRUE` if component is an input field.

`display_width` *Compute display width*

Description

Compute display width

Usage

```
display_width(x)
```

Arguments

`x`	Character input.

Value

Display width in characters.

`ExportToExcel` *Export data to Excel (or CSV if needed)*

Description

This is a simple, user-friendly export helper for FormIOr outputs. It accepts a data.frame, a list of data.frames (multiple sheets), or a list returned by `GetResponses()` / `FlattenSubmissions()`. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
ExportToExcel(
  data,
  path = NULL,
  sheet = "Data",
  overwrite = FALSE,
  include_row_names = FALSE,
  quiet = FALSE
)
```

Arguments

`data`	A data.frame, list of data.frames, or list containing `FlatResponses` or `submission_data`.
`path`	File path for the Excel output. Required. If you use a `.csv` or `.tsv` extension, delimited files will be written.
`sheet`	Sheet name to use when exporting a single data.frame.
`overwrite`	Logical. If `FALSE` (default), stop if the file exists.
`include_row_names`	Logical. Include row names in the export.
`quiet`	Logical. If `FALSE`, prints a short message.

Details

If an Excel writer is available (`writexl` or `openxlsx`), it writes `.xlsx`. If not, it falls back to CSV files and prints a clear message.

Value

A list with:

path Output file path(s).

format `"xlsx"`, `"csv"`, or `"tsv"`.

sheets Sheet names used.

rows Row counts for each sheet.

cols Column counts for each sheet.

Examples

```
df <- data.frame(name = c("A", "B"), score = c(1, 2))
## Not run:
  ExportToExcel(df, tempfile(fileext = ".xlsx"), overwrite = TRUE)

## End(Not run)
```

`extract_child_components` *Extract child components from a component*

Description

Extract child components from a component

Usage

```
extract_child_components(comp)
```

Arguments

`comp`	Component definition.

Value

A list of child components (possibly empty).

`extract_flat_df` *Extract a flat data frame*

Description

Extract a flat data frame

Usage

```
extract_flat_df(x)
```

Arguments

`x`	Data frame or FlattenSubmissions list.

Value

A data frame of responses.

`extract_options` *Extract selectable options from a component*

Description

Extract selectable options from a component

Usage

```
extract_options(comp)
```

Arguments

`comp`	Component definition.

Value

A character string of options, or `NULL` if unavailable.

`fetch_form_schema` *Fetch a form schema from the API*

Description

Fetch a form schema from the API

Usage

```
fetch_form_schema(
  base_url,
  form_id,
  api_key,
  version_id = NULL,
  tried_alt = FALSE
)
```

Arguments

`base_url`	Base API URL.
`form_id`	Form ID.
`api_key`	API key.
`version_id`	Optional version ID.
`tried_alt`	Logical. Whether an alternate base URL has been tried.

Value

A schema list or `NULL`.

`FieldDictionary` *Build a field dictionary for a form*

Description

This creates a clean table that lists each field in your form. It is meant to be easy to read and share with non-technical staff. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
FieldDictionary(
  form,
  include = c("input", "all"),
  version = "latest",
  expand_surveys = FALSE,
  quiet = FALSE
)
```

Arguments

`form`	Form schema list, JSON string, or path to a JSON file.
`include`	Which components to include: `"input"` (default) or `"all"`.
`version`	Which form version to use when `form` is metadata (from `GetFormMetadata()`). Use `"latest"` (default), a version number (e.g., `3`), or a version ID.
`expand_surveys`	Logical. If `TRUE`, expands FormIO survey components (e.g., `simplesurveyadvanced`) into one row per question.
`quiet`	Logical. If `FALSE`, prints a short summary.

Details

If you pass the output of `GetFormMetadata()`, the schema is **not** included in that object. In that case the function will automatically fetch the schema using stored credentials (from `AskCredentials()` or `GetFormMetadata()`). If credentials are not available, it will stop with a clear message.

Note (CHEF): Some schema endpoints live at `/api/v1` while other endpoints (like exports) are under `/app/api/v1`. If a schema fetch fails at the provided `base_url`, FormIOr automatically retries the alternate base.

Tip: Use `include = "all"` to include layout components (panels, tabs, fieldsets). These usually have `input = FALSE` and no field key.

Value

A data.frame with columns such as:

- `key`: field key (may be `NA` for layout components)
- `label`: field label or title
- `type`: component type (text, number, panel, etc.)
- `required`: whether the field is required
- `description`: help text/tooltip
- `default`: default value
- `options`: choice labels (for select/radio)
- `section`: nearest section/panel title
- `path`: location within the form
- `input`: `TRUE` for input fields, `FALSE` for layout

Examples

```
form <- list(
  title = "Sample Form",
  components = list(
    list(type = "textfield", key = "first_name", label = "First name", input = TRUE),
    list(type = "select", key = "color", label = "Favorite color", input = TRUE,
         data = list(values = list(list(label = "Red", value = "red"))))
  )
)
FieldDictionary(form)

## Not run:
meta <- GetFormMetadata(form_id = "123", api_key = "abc")
```

```
FieldDictionary(meta, include = "all")

## End(Not run)
```

`find_schema_root` *Find the best schema root in a nested object*

Description

Find the best schema root in a nested object

Usage

```
find_schema_root(x, require_schema_like = TRUE)
```

Arguments

`x`	Object to traverse.
`require_schema_like`	Logical. Require strict schema root criteria.

Value

The best matching schema list, or `NULL`.

`findMultilines` *Identify columns with multiple distinct values per submission*

Description

Creates a diagnostic data frame with the same shape as the flattened responses, where each original value (except in the ID column) is replaced by the number of **distinct non-NA values** that appear in that column for the corresponding submission. This makes it easy to detect which questions/fields caused row duplication due to repeated sections, multi-select choices, or repeating groups.

Usage

```
findMultilines(x, id_col = 1)
```

Arguments

`x`	A list returned by `FlattenSubmissions()`, containing at minimum `$FlatResponses` (the flattened data frame) and optionally `$ColumnNames`.
`id_col`	Integer. The column number that contains the unique submission identifier (usually `submissionId`). Default = 1 (first column).

Details

This function is typically used before `FixDups()` to help non-technical users identify which columns need special handling (e.g. concatenate, pick first, sum, pivot wider, etc.).

Columns that are constant within each submission will show `1` everywhere (or `NA` if the column is entirely missing for that submission).

Value

A data frame with the **same dimensions and column names** as `x$FlatResponses`, but where (starting from the second column) every cell contains the count of distinct non-NA values in that column for the given submission. Counts = 1 indicate constant/single-value fields; counts > 1 indicate multi-value fields that are likely causing duplicated rows.

A tibble (or data.frame) matching the structure of `x$FlatResponses`.

See Also

`FlattenSubmissions()`, `GetResponses()`, `FixDups()`

Examples

```
## Not run:
flat <- FlattenSubmissions(GetResponses(form_id = "123", api_key = "abc..."))

# See which columns have multiple values per submission
multi_counts <- findMultilines(flat)

# Quick check: which columns ever have more than one distinct value?
multi_counts |>
  summarise(across(-1, ~ max(.x, na.rm = TRUE))) |>
  pivot_longer(everything(), names_to = "column", values_to = "max_distinct") |>
  filter(max_distinct > 1)

## End(Not run)
```

`first_non_empty` *Return the first non-empty value*

Description

Return the first non-empty value

Usage

```
first_non_empty(...)
```

Arguments

`...`	Candidate values.

Value

First non-empty character value or `NA_character_`.

FixDups — *Clean duplicated rows caused by multi-value fields in flattened FormIO data*

Description

Interactive function designed for users with little or no R experience. It helps resolve columns that contain multiple values per submission (e.g. repeating sections, "add another" items, multi-select questions).

Usage

```
FixDups(
  x,
  multi_counts = NULL,
  id_col = 1,
  ask_threshold = 1.05,
  quiet = FALSE,
  dry_run = FALSE,
  strategies = NULL,
  prompt = interactive(),
  default_strategy = "concat_comma"
)
```

Arguments

`x`	List returned by `FlattenSubmissions()`, must contain `$FlatResponses`
`multi_counts`	Optional. Output from `findMultilines()`. If `NULL`, it is computed automatically.
`id_col`	Character or integer. Name or position of the submission ID column (usually `"submissionId"` or column 1). Default: 1.
`ask_threshold`	Numeric. Only prompt about columns where the maximum number of distinct values per submission exceeds this value. Default = 1.05 (catches almost all real multiples while ignoring noise).
`quiet`	Logical. Suppress most messages and summaries? Default `FALSE`.
`dry_run`	Logical. If `TRUE`, shows what would happen without actually modifying the data. Default `FALSE`.
`strategies`	Optional data frame with columns `column` and `strategy` used when `prompt = FALSE`.
`prompt`	Logical. If `TRUE` (default), ask interactively for each column.
`default_strategy`	Strategy used in non-interactive mode when a column is not in `strategies`.

Details

Shows a summary of problematic columns first, then asks the user one column at a time how to handle each one (concatenate, keep first, sum, count, remove the column entirely, etc.).

Most strategies reduce the data to one row per submission. Choosing "remove" drops the column completely from the output. All decisions are logged so you can review or reproduce the cleaning steps.

Value

A list with two components:

cleaned A tibble containing the cleaned, deduplicated responses

decisions A tibble recording which strategy was applied (or if the column was removed) for each processed column

Examples

```
## Not run:
# Typical workflow
responses <- GetResponses(form_id = "your-form-id", api_key = "your-api-key")
flat <- FlattenSubmissions(responses)

# Run interactive cleaning
result <- FixDups(flat)

# View the cleaned data
View(result$cleaned)

# See what was done to each column
result$decisions

# Dry run to preview changes
FixDups(flat, dry_run = TRUE)

## End(Not run)
```

FlagSuspiciousSubmissions

Flag suspicious submissions and enforce pre/post comparability checks

---

Description

Evaluates whether pre/post time groups are statistically comparable and produces row-level suspiciousness flags. This is designed for workflows where a survey changed mid-collection (for example, from anonymous to identity verified) and pooled analysis may no longer be valid.

Usage

```
FlagSuspiciousSubmissions(
  x,
  id_col = 1,
  time_col = NULL,
  cutoff_time,
  include_cols = NULL,
  group_action = c("split_only", "omit_pre", "omit_post", "none"),
  comparability_rule = c("fdr_effect"),
  alpha = 0.05,
  fdr_method = "BH",
  min_effect_numeric = 0.2,
  min_effect_categorical = 0.1,
  min_flagged_field_share = 0.2,
  min_complete_rate = 0.6,
  risk_weights = NULL,
  risk_threshold = 0.7,
  action = c("flag_only", "exclude_high_risk", "exclude_confirmed_duplicates"),
  return_flat = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`id_col`	Integer or character. Submission ID column (default 1).
`time_col`	Optional timestamp column name. If `NULL`, common timestamp names are guessed.
`cutoff_time`	Required cutoff separating pre and post groups.
`include_cols`	Optional columns to include in comparability tests. If `NULL`, all analyzable non-ID, non-time columns are considered.
`group_action`	One of `"split_only"` (default), `"omit_pre"`, `"omit_post"`, or `"none"`.
`comparability_rule`	Currently only `"fdr_effect"`.
`alpha`	Significance threshold for adjusted p-values.
`fdr_method`	P-adjust method passed to `stats::p.adjust()`.
`min_effect_numeric`	Minimum absolute numeric effect size (SMD).
`min_effect_categorical`	Minimum categorical effect size (Cramer's V).
`min_flagged_field_share`	Minimum share of tested fields marked different to classify groups as non-comparable.
`min_complete_rate`	Minimum non-missing rate required for a field to be tested.
`risk_weights`	Named list or vector of row-level risk weights.
`risk_threshold`	Threshold for high-risk labeling.
`action`	One of `"flag_only"`, `"exclude_high_risk"`, `"exclude_confirmed_duplicates"`.
`return_flat`	Logical. If `TRUE` and `x` came from `FlattenSubmissions()`, include updated list as `flat`.
`quiet`	Logical. If `FALSE`, prints a short summary.

Details

For each analyzable field, the function compares pre (`time` < `cutoff_time`) and post (`time` >= `cutoff_time`) groups using:

- Numeric fields: Wilcoxon rank-sum + standardized mean difference (SMD)
- Categorical fields: Chi-square (or Fisher fallback) + Cramer's V

A field is marked different when both adjusted p-value and effect-size thresholds are met. If enough fields differ, pooled analysis is flagged as non-comparable.

Value

A list with:

data Primary analysis dataset after group action + row action

datasets List with `pre`, `post`, and `full` datasets

comparability List with global comparability metrics and `field_tests`

decision_sentence One-sentence human-readable decision summary

flags Row-level suspiciousness flags and risk scores

summary High-level counts and selected actions

diagnostics Supplementary diagnostic tables

flat If `return_flat = TRUE` and input is a flat list, updated list

When pre/post groups must not be pooled

If `comparability$non_comparable` is `TRUE`, pooled pre+post inference should be treated as invalid by default. Prefer either:

- `group_action = "omit_pre"` (usually preferred when post is identity-verified),
- `group_action = "omit_post"`, or
- `group_action = "split_only"` and analyze groups separately.

Examples

```
## Not run:
data("SuspiciousSurveyDemo", package = "FormIOr")

out <- FlagSuspiciousSubmissions(
  SuspiciousSurveyDemo,
  id_col = "submissionId",
  time_col = "created",
  cutoff_time = "2026-01-06 00:00:00",
  group_action = "split_only"
)

out$comparability$non_comparable
head(out$comparability$field_tests)

# If groups differ and post phase is identity-verified, prefer post-only:
post_only <- FlagSuspiciousSubmissions(
  SuspiciousSurveyDemo,
  id_col = "submissionId",
  time_col = "created",
  cutoff_time = "2026-01-06 00:00:00",
  group_action = "omit_pre"
)

## End(Not run)
```

flatten_components *Flatten components into a row table*

Description

Flatten components into a row table

Usage

```
flatten_components(
  components,
  include = c("input", "all"),
  expand_surveys = FALSE
)
```

Arguments

`components`	Component list from a schema.
`include`	Which components to include ("input" or "all").
`expand_surveys`	Logical. Expand survey components into question rows.

Value

A data.frame of component rows.

FlattenSubmissions *Flatten Submissions*

Description

This function will flatten submissions coming from a FormIO list that has nested elements

Usage

```
FlattenSubmissions(x)
```

Arguments

`x`	list with responses, typically obtained through the GetResponses() function.

Value

list with

- `FlatResponses` which is the entire dataset obtained from GetResponses() but, where all the nested lists are flattened into a bidimensional data frame. nested lists will take their name by using their parent name as a prefix, then a "-" separator, and then their own name.
- `ColumnNames` which is a data frame with column `Number` and column `Names`, listing all the newly created column names in `FlatResponses` and their numerical order. This output can then be passed to other FormIO functions for making easier to subset the dataset

Examples

```
x<-FoodTypes

# Nested Structure
head(x)

#Flattened Structure
xFlat<-FlattenSubmissions(x)

xFlat$FlatResponses ## Survey Output
xFlat$ColumnNames   ## New Columns Formed
```

`FoodTypes` *Survey Responses Dataset*

Description

A cleaned and structured dataset containing survey mock responses collected from individual respondents. Each row represents a unique respondent, identified by a unique ID, and columns correspond to survey variables, metadata, and derived indicators.

Usage

```
data("FoodTypes")
```

Format

An object of class `data.frame` with 6 rows and 15 columns.

Details

The dataset is intended for quantitative and mixed-methods analysis of respondent-level patterns, including descriptive statistics, modeling, and aggregation of survey-based indicators.

Source

Based on standard structure from BC Public Service FormIO CHEF structure, processed and cleaned using custom R scripts.

`formior_use_color` *Determine whether to use colored output*

Description

Determine whether to use colored output

Usage

```
formior_use_color()
```

Value

`TRUE` if color output is enabled and supported.

`FormIOrAddin` *Launch the FormIOr RStudio addin*

Description

Provides a minimal point-and-click interface for key FormIOr functions.

Usage

```
FormIOrAddin()
```

`FormIOrWorkflow` *Guided end-to-end workflow (download -> clean -> report -> export)*

Description

This interactive helper walks you through a complete FormIOr workflow: downloading responses, flattening and cleaning the data, generating simple diagnostics (like a codebook and basic plots), and creating output files.

Usage

```
FormIOrWorkflow(
  data = NULL,
  base_url = "https://submit.digital.gov.bc.ca/app/api/v1",
  form_id = NULL,
  api_key = NULL,
  output_dir = NULL,
  overwrite = FALSE,
  plan = NULL,
  quiet = FALSE
)
```

Arguments

`data`	Optional. A data.frame of responses to start from. If `NULL` (default), the workflow can download responses using `GetResponses()`.
`base_url`	Character. API base URL. Default: `"https://submit.digital.gov.bc.ca/app/api/v1"`.
`form_id`	Character. Form identifier. If `NULL`, uses stored credentials or prompts via `AskCredentials()`.
`api_key`	Character. API key / secret. If `NULL`, uses stored credentials or prompts via `AskCredentials()`.
`output_dir`	Folder to write output files into. If `NULL`, a new folder is created under `tempdir()` (or you will be prompted to choose). When downloading from FormIO, the default folder name uses the form name (when available) plus a timestamp.
`overwrite`	Logical. If `TRUE`, allow overwriting output files.

`plan`	Optional list. For non-interactive/scripted runs you can pass a simple plan list (created manually). If provided, the workflow runs without prompting and the plan must include `output_dir` (to avoid implicit file writes). (This is separate from the `workflow_plan.rds` file saved by the interactive wizard.)
`quiet`	Logical. If `FALSE`, prints progress messages.

Details

It is designed for non-technical users: you can accept the defaults by pressing Enter at each prompt.

Audit logging:

- If you start an audit log (see `StartAuditLog()`), each step is recorded.
- If you do not start a log, FormIOr will ask once per session whether you want to begin logging.

Output folder naming:

- When downloading from FormIO and you don't supply `output_dir`, the workflow will try to fetch basic form metadata first so it can suggest a folder name based on the form name. This may prompt for credentials a bit earlier than the download step. The default base directory is `tempdir()` unless you choose a different location.

Output files:

- The export workbook includes a `FlattenedRaw` sheet (the flattened data before any cleaning) and, when available, an `AuditLog` sheet.
- Plots are saved as `.png` files under `output_dir/plots` and are listed in a `Plots` sheet.
- The workflow also saves your wizard choices to `workflow_plan.rds` and `workflow_plan.json` inside the output folder, so you can repeat the same steps later.

Repeatable sessions:

- When you run `FormIOrWorkflow()` again, it looks for a previous `workflow_plan.rds/json` and asks whether you want to reuse it.
- You can reuse a previous session in two ways:
  1. Apply automatically (no prompts; uses saved answers where available)
  2. Use as defaults (prompts still appear, but you can press Enter to accept)

Value

A list with:

data_raw The downloaded/raw responses (if downloaded).

flat_raw The fully flattened data *before* any cleaning/wrangling steps.

flat A `FlattenSubmissions()`-style list with `FlatResponses`.

reports Named list of diagnostic tables (codebook, summaries).

files Named list of output file paths.

plan The plan that was executed (useful for reproducibility).

CHEF credentials and base URL

FormIOr is designed for the BC Public Service CHEF FormIO service. The default base URL is `https://submit.digital.gov.bc.ca/app/api/v1`. If you use a different FormIO service, you must override `base_url` and compatibility is not guaranteed. Note (CHEF): Response export endpoints use `/app/api/v1`, while some metadata/schema endpoints live at `/api/v1`. FormIOr automatically retries the alternate base for metadata/schema requests when needed.

For CHEF users, generate your API key from the form's **Manage** page. The Form ID is the final alphanumeric code after the = sign in the Manage page URL.

Examples

```
## Not run:
# Fully guided interactive run
out <- FormIOrWorkflow()

# Start from an existing data.frame (skip download)
out <- FormIOrWorkflow(data = FoodTypes)

## End(Not run)
```

`get_audit_state` *Read audit log state*

Description

Fetches the audit logging settings from R options and fills defaults.

Usage

```
get_audit_state()
```

Value

A list of audit state fields.

`get_base_url` *Get the configured base URL*

Description

Get the configured base URL

Usage

```
get_base_url()
```

Value

Base URL for FormIO API.

`get_numbers` *Parse row number input*

Description

Converts comma-separated values and ranges into a sorted vector of row indices.

Usage

```
get_numbers(prompt = "Enter numbers (comma-separated, ranges with : ok): ")
```

Arguments

`prompt`	Prompt text to display (empty string for none).

Value

An integer vector of selected row numbers.

`GetFormMetadata` *Get metadata for a 'FormIO' form*

Description

Fetches form metadata from the API and retries the alternate base URL when parsing fails (CHEF compatibility behavior).

Usage

```
GetFormMetadata(
  base_url = "https://submit.digital.gov.bc.ca/app/api/v1",
  form_id = NULL,
  api_key = NULL,
  reenter.credentials = FALSE
)
```

Arguments

`base_url`	Character API base URL. Default: `"https://submit.digital.gov.bc.ca/app/api/v1"`.
`form_id`	Optional form ID. If `NULL`, uses cached credentials or prompts interactively via `AskCredentials()`.
`api_key`	Optional API key. If `NULL`, uses cached credentials or prompts interactively via `AskCredentials()`.
`reenter.credentials`	Logical. Force re-entry of credentials.

Details

Returns a hierarchical named list representing the form configuration and metadata, closely matching the JSON response structure.

Value

A named list of form metadata.

Examples

```
## Not run:
GetFormMetadata(form_id = "your_form_id", api_key = "your_api_key")

## End(Not run)
```

`GetResponses` *Get responses submitted to a 'FormIO' form*

Description

Downloads response export JSON from a form endpoint and returns parsed submissions by default.

Usage

```
GetResponses(
  base_url = "https://submit.digital.gov.bc.ca/app/api/v1",
  form_id = NULL,
  api_key = NULL,
  drafts = FALSE,
  deleted = FALSE,
  content.only = TRUE,
  reenter.credentials = FALSE
)
```

Arguments

`base_url`	Character API base URL. Default: `"https://submit.digital.gov.bc.ca/app/api/v1"`.
`form_id`	Optional form ID. If `NULL`, uses cached credentials or prompts interactively via `AskCredentials()`.
`api_key`	Optional API key. If `NULL`, uses cached credentials or prompts interactively via `AskCredentials()`.
`drafts`	Logical. Include drafts. Default `FALSE`.
`deleted`	Logical. Include deleted submissions. Default `FALSE`.
`content.only`	Logical or character. • `TRUE` (default): return parsed submissions • `FALSE`: return full response list (status, headers, parsed data) • `"raw"`: return raw JSON text
`reenter.credentials`	Logical. Force re-entry of credentials.

Value

Parsed submissions, full response list, or raw JSON depending on `content.only`.

Examples

```
## Not run:
GetResponses(form_id = "your_form_id", api_key = "your_api_key")

## End(Not run)
```

`GetSubmissions` *Retrieve submission metadata for a form*

Description

Fetches submission-level metadata (not full response payloads) from a 'FormIO' endpoint.

Usage

```
GetSubmissions(
  base_url = "https://submit.digital.gov.bc.ca/app/api/v1",
  form_id = NULL,
  api_key = NULL,
  content.only = TRUE,
  reenter.credentials = FALSE,
  AdditionalCols = character()
)
```

Arguments

`base_url`	Character API base URL. Default: `"https://submit.digital.gov.bc.ca/app/api/v1"`.
`form_id`	Optional form ID. If `NULL`, uses cached credentials or prompts interactively via `AskCredentials()`.
`api_key`	Optional API key. If `NULL`, uses cached credentials or prompts interactively via `AskCredentials()`.
`content.only`	Logical or character. • `TRUE` (default): return cleaned metadata data frame • `FALSE`: return full response list (status, headers, parsed data) • `"raw"`: return raw JSON text
`reenter.credentials`	Logical. Force re-entry of credentials.
`AdditionalCols`	Character vector of extra fields requested via the `fields` query parameter.

Value

Submission metadata data frame, full response list, or raw JSON depending on `content.only`.

`guess_time_col` *Guess a timestamp column name*

Description

Guess a timestamp column name

Usage

```
guess_time_col(names_vec)
```

Arguments

`names_vec`	Column names.

Value

The first matching timestamp column name, or `NULL`.

`has_components_field` *Check for a components field*

Description

Check for a components field

Usage

```
has_components_field(x)
```

Arguments

`x`	Object to inspect.

Value

`TRUE` if `x$components` exists and is list-like.

`infer_data_dimensions` *Infer row/column counts from data*

Description

Infer row/column counts from data

Usage

```
infer_data_dimensions(data)
```

Arguments

`data`	Data frame or list containing one.

Value

Named integer vector with rows and cols.

`is_checkbox_like` *Determine if values resemble checkbox selections*

Description

Determine if values resemble checkbox selections

Usage

```
is_checkbox_like(
  values,
  values_df = NULL,
  yes_values = default_yes_values(),
  no_values = NULL
)
```

Arguments

`values`	Vector of values.
`values_df`	Optional data frame (unused, for compatibility).
`yes_values`	Values treated as selected.
`no_values`	Values treated as not selected.

Value

`TRUE` if values look like checkbox inputs.

`is_flat_list` *Check whether input is a FlattenSubmissions list*

Description

Check whether input is a FlattenSubmissions list

Usage

```
is_flat_list(x)
```

Arguments

`x`	Object to test.

Value

`TRUE` if `x` is a list with `FlatResponses`, otherwise `FALSE`.

`is_layout_type` *Check if a component type is layout-only*

Description

Check if a component type is layout-only

Usage

```
is_layout_type(type)
```

Arguments

`type`	Component type string.

Value

`TRUE` if the type is layout-only.

`is_numeric_like` *Check if values are numeric-like*

Description

Check if values are numeric-like

Usage

```
is_numeric_like(values)
```

Arguments

`values`	Vector of values.

Value

`TRUE` if values appear numeric.

`is_schema_root_candidate` *Check if an object looks like a schema root*

Description

Check if an object looks like a schema root

Usage

```
is_schema_root_candidate(x)
```

Arguments

`x`	Object to inspect.

Value

`TRUE` when the object is a plausible schema root.

`is_section_type` *Check if a type represents a section*

Description

Check if a type represents a section

Usage

```
is_section_type(type)
```

Arguments

`type`	Component type string.

Value

`TRUE` if the type is a section-like layout.

`is_survey_type` *Check if component type is a survey*

Description

Check if component type is a survey

Usage

```
is_survey_type(type)
```

Arguments

`type`	Component type string.

Value

`TRUE` if the type is a survey component.

`IsAuditLogActive` *Check whether audit logging is active*

Description

This is most useful in scripts when you want to conditionally add a manual note using `WriteAuditLog()` only when logging is enabled.

Usage

```
IsAuditLogActive()
```

Value

`TRUE` if an audit log is active, otherwise `FALSE`.

Examples

```
IsAuditLogActive()
```

`last_segment` *Extract the last segment of a key*

Description

Extract the last segment of a key

Usage

```
last_segment(x)
```

Arguments

`x`	Character vector.

Value

Last segment after delimiters.

`list_element_to_text` *Convert nested list elements to text*

Description

Convert nested list elements to text

Usage

```
list_element_to_text(x)
```

Arguments

`x`	List, data.frame, or atomic value.

Value

A character representation (or `NA_character_`).

`looks_like_metadata` *Check whether an object looks like metadata*

Description

Check whether an object looks like metadata

Usage

```
looks_like_metadata(x)
```

Arguments

`x`	Object to inspect.

Value

`TRUE` if it looks like a metadata response.

MakeCodebook *Create a codebook for a dataset*

Description

This produces a plain-language table that documents each column in your data. It can optionally use a FormIO schema (via `FieldDictionary()`) to add labels, sections, and descriptions. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
MakeCodebook(
  data,
  form = NULL,
  include = c("input", "all"),
  include_summary = TRUE,
  max_levels = 20,
  quiet = FALSE
)
```

Arguments

`data`	A data.frame of responses, or a list containing `FlatResponses` or `submission_data`.
`form`	Optional form schema, JSON string, or metadata object (see `FieldDictionary()`). If provided, labels and sections are included.
`include`	Which components to include when `form` is provided: `"input"` (default) or `"all"`.
`include_summary`	Logical. If `TRUE` (default), include basic numeric summaries (min, max, mean).
`max_levels`	Maximum number of category levels to list for character or factor columns.
`quiet`	Logical. If `FALSE`, prints a short message.

Details

When a schema is provided, the codebook also includes a `schema_key` column showing which form field key was matched (best-effort). If a column could not be matched to the schema, schema-related columns (type/required/section/etc.) will be `NA`.

Value

A data.frame codebook with one row per column.

Examples

```
df <- data.frame(age = c(10, 12, NA), color = c("red", "blue", "red"))
MakeCodebook(df)
```

`match_field_info_rows` *Match data columns to field dictionary rows*

Description

Match data columns to field dictionary rows

Usage

```
match_field_info_rows(col_names, field_info)
```

Arguments

`col_names`	Column names in the data.
`field_info`	Field dictionary data frame.

Value

Integer vector of matched row indices.

`maybe_prompt_audit_log` *Prompt to start audit logging*

Description

Interactively asks the user to start logging once per session.

Usage

```
maybe_prompt_audit_log()
```

Value

`TRUE` if a log was started, otherwise `FALSE` (invisibly).

`maybe_write_audit` *Write an audit entry if logging is active*

Description

Convenience wrapper that checks state before appending to the log.

Usage

```
maybe_write_audit(action, details = NULL, data = NULL)
```

Arguments

`action`	Action name to record.
`details`	Optional details string.
`data`	Optional data frame to summarize.

Value

The result of `WriteAuditLog()` (invisibly) or `NULL`.

`na_value_for` *Get typed NA for a vector*

Description

Get typed NA for a vector

Usage

```
na_value_for(values)
```

Arguments

`values`	Vector used to infer type.

Value

An NA value of the appropriate type.

`normalize_adjust_updates` *Normalize update inputs*

Description

Coerces supported update formats into a standard data frame with columns `id`, `column`, and `value`.

Usage

```
normalize_adjust_updates(updates)
```

Arguments

`updates`	Updates in data.frame or list form.

Value

A data.frame with normalized columns, or `NULL` if no updates.

`normalize_checkbox_values` *Normalize checkbox values for comparison*

Description

Normalize checkbox values for comparison

Usage

```
normalize_checkbox_values(values)
```

Arguments

`values`	Vector of values.

Value

Normalized character vector.

`normalize_details` *Normalize audit log details*

Description

Normalize audit log details

Usage

```
normalize_details(details, n)
```

Arguments

`details`	Details input.
`n`	Expected length.

Value

Character vector of length `n`.

`normalize_keep_map` *Normalize keep map inputs*

Description

Accepts multiple keep_map formats and returns a named list keyed by `group_id` or `group_key`.

Usage

```
normalize_keep_map(keep_map)
```

Arguments

`keep_map`	Keep decisions in list, named vector, or data.frame form.

Value

A named list of keep choices, or `NULL`.

`normalize_list` *Normalize list inputs*

Description

Normalize list inputs

Usage

```
normalize_list(x)
```

Arguments

`x`	List-like input.

Value

A list (possibly empty).

`normalize_names` *Normalize column names*

Description

Normalize column names

Usage

```
normalize_names(names_vec, style, make_unique, transliterate)
```

Arguments

`names_vec`	Character vector of column names.
`style`	Naming style ("snake", "lower", "upper", "title").
`make_unique`	Logical. Ensure unique names.
`transliterate`	Logical. Convert accents to ASCII.

Value

A character vector of normalized names.

`normalize_row` *Normalize a row to a list*

Description

Normalize a row to a list

Usage

```
normalize_row(row_df)
```

Arguments

`row_df`	Data frame row or list.

Value

A list with scalar elements.

`NormalizeColumnNames` *Normalize column names into a clean, readable format*

Description

Designed for non-technical users who want consistent column names after downloading FormIO submissions. Works with either a raw data frame or the list returned by `FlattenSubmissions()`. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
NormalizeColumnNames(
  x,
  style = c("snake", "lower", "upper", "title"),
  make_unique = TRUE,
  transliterate = TRUE,
  return_flat = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`style`	One of `"snake"` (default), `"lower"`, `"upper"`, or `"title"`. Controls casing and separator behavior.
`make_unique`	Logical. If `TRUE`, make duplicated names unique.
`transliterate`	Logical. If `TRUE`, convert accents/special characters to ASCII when possible.
`return_flat`	Logical. If `TRUE` and `x` came from `FlattenSubmissions()`, include the updated list as `flat` in the output.
`quiet`	Logical. If `FALSE`, prints a short summary.

Value

A list with:

data Data frame with normalized column names (the cleaned data)

name_map Data frame with old and new column names

flat If `return_flat` = `TRUE`, the updated FlattenSubmissions-style list (for workflows that expect `FlatResponses`)

Examples

```
## Not run:
flat <- FlattenSubmissions(GetResponses(form_id = "123", api_key = "abc"))
norm <- NormalizeColumnNames(flat)
names(norm$data)
norm$name_map

## End(Not run)
```

`option_suffix` *Extract option suffix from a column name*

Description

Extract option suffix from a column name

Usage

```
option_suffix(name, sep)
```

Arguments

`name`	Column name.
`sep`	Separator between prefix and suffix.

Value

Suffix portion of the name.

pad_right *Pad a string to the right*

Description

Pad a string to the right

Usage

```
pad_right(x, width)
```

Arguments

`x`	Character input.
`width`	Target display width.

Value

Right-padded string.

parse_index_input *Parse index selections*

Description

Parse index selections

Usage

```
parse_index_input(input, max_val)
```

Arguments

`input`	Character string of indices/ranges.
`max_val`	Maximum allowed index.

Value

Integer vector of selected indices.

`parse_keep_choice` *Parse keep selection choices*

Description

Parse keep selection choices

Usage

```
parse_keep_choice(choice, n_rows)
```

Arguments

`choice`	Selection input (numeric, logical, or character).
`n_rows`	Number of available rows.

Value

Integer indices to keep, or `NULL`.

`parse_logical_value` *Parse logical inputs*

Description

Converts common string/number inputs into logical values.

Usage

```
parse_logical_value(x)
```

Arguments

`x`	Input to parse.

Value

`TRUE`, `FALSE`, or `NA`.

`pick_compare_columns` *Pick columns to compare duplicates*

Description

Pick columns to compare duplicates

Usage

```
pick_compare_columns(
  df,
  id_col_name,
  compare_cols,
  quiet = FALSE,
  prompt = TRUE,
  default_cols = NULL
)
```

Arguments

`df`	Data frame of responses.
`id_col_name`	Submission ID column.
`compare_cols`	Optional pre-selected columns.
`quiet`	Logical. Suppress messages.
`prompt`	Logical. Allow interactive prompting.
`default_cols`	Default columns to use when prompting.

Value

Character vector of comparison column names.

`pick_dedup_rows` *Pick rows to keep for deduplication*

Description

Pick rows to keep for deduplication

Usage

```
pick_dedup_rows(df, id_col, time_col, keep)
```

Arguments

`df`	Data frame of responses.
`id_col`	Submission ID column name.
`time_col`	Optional time column name.
`keep`	One of "last" or "first".

Value

Integer indices of rows to keep.

`pick_duplicate_key_columns` *Pick duplicate key columns*

Description

Pick duplicate key columns

Usage

```
pick_duplicate_key_columns(
  df,
  id_col_name,
  key_cols,
  quiet = FALSE,
  prompt = TRUE
)
```

Arguments

`df`	Data frame of responses.
`id_col_name`	Submission ID column.
`key_cols`	Optional pre-selected key columns.
`quiet`	Logical. Suppress messages.
`prompt`	Logical. Allow interactive prompting.

Value

Character vector of key column names.

`pick_first_non_na` *Pick the first non-missing value*

Description

Pick the first non-missing value

Usage

```
pick_first_non_na(x)
```

Arguments

`x`	Vector of values.

Value

The first non-`NA` value (typed), or `NA` if none.

`PlotBarSummary` *Plot a bar chart for a categorical field*

Description

If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
PlotBarSummary(
  x,
  field,
  top_n = 10,
  include_na = FALSE,
  horiz = FALSE,
  main = NULL,
  xlab = NULL,
  col = "steelblue",
  plot = TRUE,
  ...
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`field`	Column name or number to plot.
`top_n`	Number of categories to show (default 10). Use `NULL` for all.
`include_na`	Logical. If `TRUE`, include missing values as "(Missing)".
`horiz`	Logical. If `TRUE`, draw horizontal bars.
`main`	Optional plot title.
`xlab`	Optional x-axis label.
`col`	Bar fill color.
`plot`	Logical. If `FALSE`, return bar data without plotting.
`...`	Additional arguments passed to `barplot()`.

Value

Invisibly returns a list with the field name and bar data.

Examples

```
## Not run:
PlotBarSummary(flat, "region")

## End(Not run)
```

PlotHistogram *Plot a histogram for a numeric field*

Description

If audit logging is active (see StartAuditLog()), this action is recorded.

Usage

```
PlotHistogram(
  x,
  field,
  bins = "Sturges",
  include_na = FALSE,
  main = NULL,
  xlab = NULL,
  col = "steelblue",
  border = "white",
  plot = TRUE,
  ...
)
```

Arguments

x	A data frame of responses, or a list from FlattenSubmissions().
field	Column name(s) or number(s) to plot. When multiple columns are provided, their text is combined into one wordcloud.
bins	Histogram breaks passed to hist(). Default "Sturges".
include_na	Logical. If TRUE, keep NA values (ignored by hist).
main	Optional plot title.
xlab	Optional x-axis label.
col	Bar fill color.
border	Bar border color.
plot	Logical. If FALSE, return histogram data without plotting.
...	Additional arguments passed to hist().

Value

Invisibly returns a list with the field name and histogram object.

Examples

```
## Not run:
PlotHistogram(flat, "age")

## End(Not run)
```

`PlotResponseTimeline` *Plot a response timeline*

Description

Convenience wrapper around `ResponseTimeline()` that draws a simple line chart using base graphics. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
PlotResponseTimeline(
  x,
  date_col = NULL,
  interval = c("day", "week", "month", "hour"),
  tz = "UTC",
  start = NULL,
  end = NULL,
  start_date = NULL,
  end_date = NULL,
  include_empty = TRUE,
  main = NULL,
  xlab = NULL,
  ylab = "Responses",
  col = "steelblue",
  lwd = 2,
  type = "l",
  plot = TRUE,
  ...
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`date_col`	Column name or number for the date/time field. If `NULL`, tries to guess a reasonable timestamp column.
`interval`	One of `"day"` (default), `"week"`, `"month"`, or `"hour"`.
`tz`	Time zone to use for parsing dates (default `"UTC"`).
`start`	Optional start date/time to filter the range. Accepts a `Date`, `POSIXct`, or an ISO-8601 string (e.g., `"2024-03-01"` or `"2024-03-01 14:30:00"`).
`end`	Optional end date/time to filter the range. Accepts a `Date`, `POSIXct`, or an ISO-8601 string (e.g., `"2024-03-31"` or `"2024-03-31 23:59:59"`).
`start_date`	Alias for `start` (useful for readability). Same formats as `start`.
`end_date`	Alias for end (useful for readability). Same formats as end.
`include_empty`	Logical. If `TRUE`, fill missing periods with zeroes.
`main`	Optional plot title.
`xlab`	Optional x-axis label.
`ylab`	Optional y-axis label.

`col`	Line color.
`lwd`	Line width.
`type`	Plot type passed to `plot()`. Default "l".
`plot`	Logical. If `FALSE`, return timeline data without plotting.
`...`	Additional arguments passed to `plot()`.

Value

Invisibly returns the output of `ResponseTimeline()`.

Examples

```
## Not run:
PlotResponseTimeline(flat, date_col = "created", interval = "month")

## End(Not run)
```

PlotWordcloud *Plot a wordcloud for a text field*

Description

Requires the optional `wordcloud` package. If it is not installed, the function will stop with a helpful message. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
PlotWordcloud(
  x,
  field,
  max_words = 100,
  min_freq = 1,
  min_chars = 2,
  remove_stopwords = TRUE,
  stopwords = default_stopwords(),
  seed = NULL,
  colors = NULL,
  ...
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`field`	Column name or number to plot.
`max_words`	Maximum number of words to display.
`min_freq`	Minimum frequency to keep a word.
`min_chars`	Minimum character length for a word to be kept.
`remove_stopwords`	Logical. If `TRUE`, remove common stopwords.

`stopwords`	Character vector of stopwords to remove.
`seed`	Optional random seed for reproducible layout.
`colors`	Vector of colors passed to `wordcloud::wordcloud()`.
`...`	Additional arguments passed to `wordcloud::wordcloud()`.

Value

Invisibly returns a list with the field name(s) and word frequencies.

Examples

```
## Not run:
PlotWordcloud(flat, "feedback")

## End(Not run)
```

`print.FlagSuspiciousSubmissionsResult`

Print method for FlagSuspiciousSubmissions results

Description

Prints the full result list and then prints `decision_sentence` as the final line so users can quickly see the overall survey-level decision.

Usage

```
## S3 method for class 'FlagSuspiciousSubmissionsResult'
print(x, ...)
```

Arguments

`x`	A `FlagSuspiciousSubmissionsResult` object.
`...`	Passed to `base::print()`.

Value

Invisibly returns `x`.

`print_column_selection_table` *Print a column selection table*

Description

Print a column selection table

Usage

```
print_column_selection_table(
  df_names,
  right = FALSE,
  max_rows = 25,
  color_numbers = TRUE,
  quiet = FALSE
)
```

Arguments

`df_names`	Column names to display.
`right`	Logical. Reserved for layout (unused).
`max_rows`	Maximum rows before splitting into two columns.
`color_numbers`	Logical. Colorize the index numbers when possible.
`quiet`	Logical. Suppress printed output when `TRUE`.

Value

Invisibly returns `NULL`.

`print_table_left` *Print a data frame for review*

Description

Print a data frame for review

Usage

```
print_table_left(df, quiet = FALSE)
```

Arguments

`df`	Data frame to print.
`quiet`	Logical. Suppress printed output when `TRUE`.

Value

Invisibly returns `NULL`.

RenameCols *Rename columns from a flattened FormIO dataset*

Description

Renames columns in a list returned by `FlattenSubmissions()`. You can run it interactively (prompt for each column) or non-interactively with `rename_map`.

Usage

```
RenameCols(
  x,
  NamesDF = TRUE,
  renameDF = TRUE,
  rename_map = NULL,
  quiet = FALSE
)
```

Arguments

`x`	List from `FlattenSubmissions()` containing `FlatResponses` and `ColumnNames`.
`NamesDF`	Logical. Include the rename table in the output. Default `TRUE`.
`renameDF`	Logical. Include the renamed flat object in the output. Default `TRUE`.
`rename_map`	Optional data frame with `OldNames` and `NewNames` columns for non-interactive renaming.
`quiet`	Logical. If `TRUE`, suppresses preview output.

Value

A named list with:

- `renamedDF`: rename mapping table (`Number`, `OldNames`, `NewNames`)
- `flat`: updated flattened object with renamed `FlatResponses`

Examples

```
## Not run:
# Interactive
RenameCols(flat)

# Non-interactive map
map_df <- data.frame(
  OldNames = c("submissionId", "age"),
  NewNames = c("submission_id", "age_years"),
  stringsAsFactors = FALSE
)
RenameCols(flat, rename_map = map_df, quiet = TRUE)

## End(Not run)
```

`resolve_credentials` *Resolve FormIO credentials*

Description

Resolve FormIO credentials

Usage

```
resolve_credentials(
  form_id = NULL,
  api_key = NULL,
  reenter.credentials = FALSE
)
```

Arguments

`form_id`	Optional form ID.
`api_key`	Optional API key.
`reenter.credentials`	Logical. Force re-entry of credentials.

Value

A list with `form_id` and `api_key`.

`resolve_field_name` *Resolve a single field name*

Description

Converts a numeric index or character label into a validated column name.

Usage

```
resolve_field_name(df, field)
```

Arguments

`df`	Data frame of responses.
`field`	Column name or index.

Value

A valid column name from `df`.

`resolve_field_names` *Resolve one or more field names*

Description

Converts numeric indices or character labels into validated column names.

Usage

```
resolve_field_names(df, field)
```

Arguments

`df`	Data frame of responses.
`field`	Column name(s) or index/indices.

Value

A character vector of valid column names.

`resolve_form_id` *Resolve a form ID from a metadata object*

Description

Resolve a form ID from a metadata object

Usage

```
resolve_form_id(form)
```

Arguments

`form`	Metadata list.

Value

Form ID string or `NULL`.

`resolve_id_col` *Resolve submission ID column name*

Description

Resolve submission ID column name

Usage

```
resolve_id_col(df, id_col)
```

Arguments

`df`	Data frame of responses.
`id_col`	Column name or index.

Value

Valid column name for submission IDs.

`resolve_keep_for_group` *Resolve keep choices for a duplicate group*

Description

Resolve keep choices for a duplicate group

Usage

```
resolve_keep_for_group(keep_map, group_id, group_key, n_rows)
```

Arguments

`keep_map`	Normalized keep map.
`group_id`	Numeric group ID.
`group_key`	Character group key.
`n_rows`	Number of rows in the group.

Value

Integer indices to keep, or `NULL`.

`resolve_version_id` *Resolve a version ID from metadata*

Description

Resolve a version ID from metadata

Usage

```
resolve_version_id(form, version = "latest")
```

Arguments

`form`	Metadata list with versions.
`version`	Version identifier or "latest".

Value

Version ID string or `NULL`.

`ResolveRepeats` *Resolve repeated answers into one row per submission*

Description

Automatically collapses repeated values within each submission ID using a consistent strategy (or simple heuristics when `strategy` = `"auto"`). This is a non-interactive alternative to `FixDups()`.

Usage

```
ResolveRepeats(
  x,
  id_col = 1,
 strategy = c("auto", "concat", "first", "last", "sum", "mean", "count", "count_yes"),
  sep = ", ",
  unique = TRUE,
  return_flat = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`id_col`	Integer or character. Submission ID column (default 1).
`strategy`	One of `"auto"`, `"concat"`, `"first"`, `"last"`, `"sum"`, `"mean"`, `"count"`, or `"count_yes"`.
`sep`	Separator used for concatenation (default `", "`).
`unique`	Logical. If `TRUE`, remove duplicate values before concatenating.
`return_flat`	Logical. If `TRUE` and `x` came from `FlattenSubmissions()`, include the updated list as `flat` in the output.
`quiet`	Logical. If `FALSE`, prints a short summary.

Details

Note: some FormIO components (for example uploads or address blocks) can produce list-columns or nested data-frame columns even after flattening. These values are converted to readable JSON/text before collapsing so the output remains stable and export-friendly. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Value

A list with:

data Collapsed data frame with one row per submission (the cleaned data)

summary Data frame describing how each column was handled

flat If `return_flat` = `TRUE`, the updated FlattenSubmissions-style list (for workflows that expect `FlatResponses`)

Examples

```
## Not run:
flat <- FlattenSubmissions(GetResponses(form_id = "123", api_key = "abc"))
resolved <- ResolveRepeats(flat, id_col = "submissionId")
head(resolved$data)

## End(Not run)
```

`ResponseTimeline` *Summarize responses over time*

Description

Counts responses per day/week/month (or hour), using a timestamp column. If `date_col` is `NULL`, the function tries common timestamp names automatically (e.g., `created`, `modified`, `_createdAt`). If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
ResponseTimeline(
  x,
  date_col = NULL,
  interval = c("day", "week", "month", "hour"),
  tz = "UTC",
  start = NULL,
  end = NULL,
  start_date = NULL,
  end_date = NULL,
  include_empty = TRUE,
  quiet = FALSE
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`date_col`	Column name or number for the date/time field. If `NULL`, tries to guess a reasonable timestamp column.
`interval`	One of `"day"` (default), `"week"`, `"month"`, or `"hour"`.
`tz`	Time zone to use for parsing dates (default `"UTC"`).
`start`	Optional start date/time to filter the range. Accepts a `Date`, `POSIXct`, or an ISO-8601 string (e.g., `"2024-03-01"` or `"2024-03-01 14:30:00"`).
`end`	Optional end date/time to filter the range. Accepts a `Date`, `POSIXct`, or an ISO-8601 string (e.g., `"2024-03-31"` or `"2024-03-31 23:59:59"`).
`start_date`	Alias for `start` (useful for readability). Same formats as `start`.
`end_date`	Alias for `end` (useful for readability). Same formats as `end`.
`include_empty`	Logical. If `TRUE`, fill missing periods with zeroes.
`quiet`	Logical. If `FALSE`, prints a short summary.

Value

A list with:

data Data frame with `period` and `count` columns

summary List with metadata about the calculation

Examples

```
## Not run:
ResponseTimeline(flat, date_col = "created", interval = "week")

## End(Not run)
```

`ReviewDuplicateSubmissions`
Interactively review and resolve duplicate submissions

Description

This function walks you through duplicate groups defined by key columns (for example `email`, `username`, `project_id`). For each duplicate group, it shows selected columns so you can compare submissions and choose which ones to keep. You can keep multiple submissions or drop all submissions for a given group if needed.

Usage

```
ReviewDuplicateSubmissions(
  x,
  id_col = 1,
  key_cols = NULL,
  compare_cols = NULL,
  return_flat = FALSE,
  quiet = FALSE,
  keep_map = NULL,
  prompt = TRUE,
  default_keep = "all"
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`id_col`	Integer or character. Submission ID column (default 1).
`key_cols`	Optional. Columns used to define "duplicate submissions" (for example `username`, `email`, `project_id`). If `NULL`, you will be prompted to choose columns interactively (unless `prompt = FALSE`). The default suggestions are the first three columns that do not start with `"form-"` or `"form_"`. Tip: choose columns that are stable within a submission (not multi-row fields).
`compare_cols`	Optional. Character or integer vector of columns to show when comparing duplicates. If `NULL` (default), you will be prompted to choose columns interactively (unless `prompt = FALSE`).
`return_flat`	Logical. If `TRUE` and `x` came from `FlattenSubmissions()`, include the updated list as `flat` in the output.
`quiet`	Logical. If `FALSE`, prints guidance and summaries.
`keep_map`	Optional. Non-interactive decisions for which rows to keep, keyed by submission ID. Supported formats: • Named list: `list(id1 = c(1,3), id2 = "all")` • Named character vector: `c(id1 = "1,3", id2 = "all")` • Data frame with columns `id` and `keep` Values can be row numbers (within each duplicate group), ranges like `"1:3"`, or keywords `"all"` / `"none"`.
`prompt`	Logical. If `TRUE` (default), ask interactively. If `FALSE`, uses `keep_map` and `default_keep` without prompting.
`default_keep`	Default action when `prompt = FALSE` and no `keep_map` entry exists for a given ID. Accepts the same formats as `keep_map` values. Default `"all"`.

Details

This is designed for non-technical users who want full control over which duplicates are kept. If you want an automatic (non-interactive) approach, use `DeduplicateSubmissions()`.

If audit logging is active (see `StartAuditLog()`), each decision is logged.

Value

A list with:

data Data frame after duplicate review

summary List with counts and whether the review stopped early

decisions Data frame describing what was kept/dropped for each ID

flat If `return_flat = TRUE` and `x` was a FlattenSubmissions list, the updated list

Examples

```
## Not run:
flat <- FlattenSubmissions(GetResponses(form_id = "123", api_key = "abc"))
out <- ReviewDuplicateSubmissions(flat, id_col = "form_submissionid")
View(out$decisions)

# Non-interactive example (scriptable)
df <- data.frame(
  submissionId = c("a", "a", "b", "b", "b"),
  email = c("x@example.com", "x@example.com", "x@example.com", "x@example.com", "x@example.com"),
  status = c("draft", "final", "test", "final", "final"),
  stringsAsFactors = FALSE
)
# Group 1 will be the email=x@example.com group (the only group in this example)
keep_map <- list(`1` = 2)
out2 <- ReviewDuplicateSubmissions(
  df,
  id_col = "submissionId",
  key_cols = "email",
  compare_cols = c("submissionId", "email", "status"),
  keep_map = keep_map,
  prompt = FALSE,
  quiet = TRUE
)

## End(Not run)
```

`sanitize_sheet_name` *Sanitize Excel sheet names*

Description

Sanitize Excel sheet names

Usage

```
sanitize_sheet_name(name)
```

Arguments

`name`	Sheet name candidate.

Value

Sanitized sheet name.

`select_version_row` *Select the best version row*

Description

Select the best version row

Usage

```
select_version_row(versions)
```

Arguments

`versions`	Versions data.frame.

Value

A single-row data.frame (or list) for the best version.

`set_audit_state` *Persist audit log state*

Description

Stores the audit state in R options.

Usage

```
set_audit_state(state)
```

Arguments

`state`	Audit state list.

Value

The stored state (invisibly).

StartAuditLog *Start an audit log for FormIOr actions*

Description

Creates a new audit log file and turns on automatic logging for FormIOr functions. Each subsequent action (downloads, cleaning, exports, etc.) will append a new row to the log.

Usage

```
StartAuditLog(file = NULL, overwrite = FALSE, append = FALSE, quiet = FALSE)
```

Arguments

`file`	Path to the audit log file. Required.
`overwrite`	Logical. If `FALSE` (default), stop if the file exists.
`append`	Logical. If `TRUE`, append to an existing log instead of erroring. Useful when resuming a previous session.
`quiet`	Logical. If `FALSE`, prints a short message.

Details

The audit log is designed to be "Excel friendly": it is a simple CSV/TSV with one row per action, including a timestamp, an action name, optional details, and (when available) row/column counts for the dataset being processed.

If you do *not* start a log, many FormIOr functions will (once per R session) ask whether you want to begin logging. Starting a log here avoids that prompt and gives you control over the file location.

Value

Invisibly returns the first log entry (a data.frame row).

Examples

```
## Not run:
log_file <- tempfile(fileext = ".csv")
StartAuditLog(log_file, overwrite = TRUE)

# Run some FormIOr steps (each will append a row when possible)
flat <- FlattenSubmissions(FoodTypes)
norm <- NormalizeColumnNames(flat, quiet = TRUE)

StopAuditLog()
read.csv(log_file, stringsAsFactors = FALSE)

## End(Not run)
```

StopAuditLog *Stop audit logging for FormIOr*

Description

Turns off automatic logging. The log file is not deleted. You can start a new log later using `StartAuditLog()`.

Usage

```
StopAuditLog(quiet = FALSE)
```

Arguments

`quiet`	Logical. If `FALSE`, prints a short message.

Value

Invisibly returns `TRUE` when logging was active, `FALSE` otherwise.

Examples

```
## Not run:
StopAuditLog()

## End(Not run)
```

strategy_fun_value *Provide a typed default for vapply*

Description

Provide a typed default for vapply

Usage

```
strategy_fun_value(strategy, values)
```

Arguments

`strategy`	Strategy name.
`values`	Vector used to infer type.

Value

A typed NA value matching the strategy output.

`strip_style_safe` *Strip ANSI styling safely*

Description

Strip ANSI styling safely

Usage

```
strip_style_safe(x)
```

Arguments

`x`	Character input with optional styling.

Value

Plain text without ANSI styling.

`summarize_column` *Summarize a single column*

Description

Summarize a single column

Usage

```
summarize_column(x, max_levels = 20, include_summary = TRUE)
```

Arguments

`x`	Column vector.
`max_levels`	Maximum category levels to list.
`include_summary`	Logical. Include numeric summaries.

Value

A list of summary metrics.

`SummaryByField` *Summarize a single field in a FormIO response dataset*

Description

Designed for non-technical users: it accepts either a data frame or the list produced by `FlattenSubmissions()` and returns a simple summary. Numeric fields get descriptive statistics; categorical fields get counts and percents. If audit logging is active (see `StartAuditLog()`), this action is recorded.

Usage

```
SummaryByField(
  x,
  field,
  top_n = 10,
  include_na = FALSE,
  digits = 2,
  quiet = FALSE
)
```

Arguments

`x`	A data frame of responses, or a list from `FlattenSubmissions()`.
`field`	Column name or number to summarize.
`top_n`	For categorical fields, the number of top values to return (default 10). Use `NULL` to return all values.
`include_na`	Logical. If `TRUE`, include missing values as "(Missing)" in categorical summaries.
`digits`	Number of decimal places for percentages/statistics.
`quiet`	Logical. If `FALSE`, prints a short summary.

Value

A list with:

field Resolved column name

type Detected type: "numeric", "categorical", or "date"

total Total rows

missing Missing value count

distinct Distinct non-missing values

summary Data frame of summary stats or counts

Examples

```
## Not run:
flat <- FlattenSubmissions(GetResponses(form_id = "123", api_key = "abc"))
SummaryByField(flat, "age")
SummaryByField(flat, "favorite_food", top_n = 5)

## End(Not run)
```

`SuspiciousSurveyDemo` *Suspicious Survey Demo Dataset*

Description

A synthetic survey-response dataset designed to test `FlagSuspiciousSubmissions()`.

Usage

```
data("SuspiciousSurveyDemo")
```

Format

A data frame with 262 rows and 10 variables:

submissionId Character submission identifier. Some IDs are repeated intentionally.

created POSIXct timestamp in UTC.

age Numeric respondent age.

satisfaction_score Numeric score (0-100).

region Categorical region (`North`, `South`, `East`, `West`).

program Categorical program (`A`, `B`, `C`).

consent Categorical consent indicator (`Yes`, `No`).

comment Short free-text category.

phase_expected Expected phase label (`pre`, `post`).

mostly_missing Mostly missing text field for completeness-threshold tests.

Details

It includes:

- A pre-change (anonymous) phase and post-change (identity-verified) phase.
- Intentional distribution shifts between phases (numeric and categorical).
- Repeated submission IDs with close timestamps.
- Repeated answer fingerprints with different submission IDs.
- A low-completeness field (`mostly_missing`) for completeness-filter testing.

Source

Synthetic data generated in `data-raw/make_suspicious_survey_demo.R`.

Examples

```
data("SuspiciousSurveyDemo")
str(SuspiciousSurveyDemo)

out <- FlagSuspiciousSubmissions(
  SuspiciousSurveyDemo,
  id_col = "submissionId",
  time_col = "created",
  cutoff_time = "2026-01-06 00:00:00",
  group_action = "split_only",
  quiet = TRUE
)
out$comparability$non_comparable
```

to_character *Coerce to a single character string*

Description

Coerce to a single character string

Usage

```
to_character(x)
```

Arguments

x	Value to coerce.

Value

A single character value or `NA_character_`.

to_text *Convert a value to display text*

Description

Convert a value to display text

Usage

```
to_text(x)
```

Arguments

x	Value to convert.

Value

Character representation.

`to_update_text` *Format update values for logging*

Description

Converts values into a short, human-readable string (or list text).

Usage

```
to_update_text(x)
```

Arguments

`x`	Value to format.

Value

A single character representation (or `NA_character_`).

`tokenize_words` *Tokenize text into words*

Description

Splits text into lowercase word tokens and filters by minimum length.

Usage

```
tokenize_words(values, min_chars = 2)
```

Arguments

`values`	Vector of text values.
`min_chars`	Minimum number of characters per token.

Value

A character vector of word tokens.

`try_fetch_schema` *Attempt to fetch the schema from metadata*

Description

Attempt to fetch the schema from metadata

Usage

```
try_fetch_schema(form, version = "latest")
```

Arguments

`form`	Metadata list.
`version`	Version identifier.

Value

A schema list or `NULL`.

`try_fetch_schema_url` *Try fetching a schema URL*

Description

Try fetching a schema URL

Usage

```
try_fetch_schema_url(url, form_id, api_key)
```

Arguments

`url`	Schema endpoint URL.
`form_id`	Form ID.
`api_key`	API key.

Value

A schema list or `NULL`.

`unique_sheet_names` *Make unique sheet names*

Description

Make unique sheet names

Usage

```
unique_sheet_names(names_vec, count)
```

Arguments

`names_vec`	Candidate names.
`count`	Number of sheets.

Value

A character vector of unique, sanitized sheet names.

`update_column_names` *Update stored column name metadata*

Description

Update stored column name metadata

Usage

```
update_column_names(column_names_df, new_names)
```

Arguments

`column_names_df`	Column metadata data.frame (optional).
`new_names`	New column names.

Value

Updated metadata data.frame (or unchanged input).

`values_equal` *Compare two values for equality*

Description

Compare two values for equality

Usage

```
values_equal(x, y)
```

Arguments

`x`	First value.
`y`	Second value.

Value

`TRUE` if the normalized values are equal.

`write_delimited_sheets` *Write CSV/TSV sheets*

Description

Write CSV/TSV sheets

Usage

```
write_delimited_sheets(
  sheets,
  path,
  ext = "csv",
  overwrite = FALSE,
  include_row_names = FALSE
)
```

Arguments

`sheets`	Named list of data frames.
`path`	Output path.
`ext`	File extension ("csv" or "tsv").
`overwrite`	Logical. Overwrite existing files.
`include_row_names`	Logical. Include row names.

Value

A list describing the written files.

`write_excel_sheets` *Write Excel sheets*

Description

Write Excel sheets

Usage

```
write_excel_sheets(
  sheets,
  path,
  overwrite = FALSE,
  include_row_names = FALSE,
  quiet = FALSE
)
```

Arguments

`sheets`	Named list of data frames.
`path`	Output path.
`overwrite`	Logical. Overwrite existing file.
`include_row_names`	Logical. Include row names.
`quiet`	Logical. Suppress messages.

Value

A list describing the written file(s).

`WriteAuditLog` *Write a simple audit log entry*

Description

This writes a small CSV/TSV log of key actions (e.g., downloads, cleaning, exports). It is designed to be easy for non-technical users to open in Excel. If you are using automatic logging, you generally do not need to call this directly; it is used behind the scenes by other FormIOr functions.

Usage

```
WriteAuditLog(
  action,
  details = NULL,
  file = NULL,
  data = NULL,
  user = NULL,
  append = TRUE,
  quiet = FALSE
)
```

Arguments

`action`	Short action name (e.g., `"export"`). Can be a vector.
`details`	Optional details or notes. Can be a character vector or list.
`file`	Path to the audit log file. If `NULL`, uses the active audit log file when one is running.
`data`	Optional data.frame (or list with `FlatResponses` / `submission_data`) used to record row/column counts.
`user`	Optional user name. Defaults to the system user if available.
`append`	Logical. If `TRUE` (default), append to the existing file.
`quiet`	Logical. If `FALSE`, prints a short message.

Value

The data.frame entry that was written.

Examples

```
df <- data.frame(a = 1:3)
log_path <- tempfile(fileext = ".csv")
WriteAuditLog(
  "export",
  details = "Exported survey data",
  data = df,
  file = log_path,
  quiet = TRUE
)
```

Index