Package 'surveydown'

June 23, 2026

Title Markdown-Based Programmable Surveys Using 'Quarto' and 'shiny'

Version 1.3.0

Description Generate programmable surveys using markdown and R code chunks. Surveys are composed of two files: a survey.qmd 'Quarto' file defining the survey content (pages, questions, etc), and an app.R file defining a 'shiny' app with global settings (libraries, database configuration, etc.) and server configuration options (e.g., conditional skipping / display, etc.). Survey data collected from respondents is stored in a 'PostgreSQL' database. Features include controls for conditional skip logic (skip to a page based on an answer to a question), conditional display logic (display a question based on an answer to a question), a customizable progress bar, and a wide variety of question types, including multiple choice (single choice and multiple choices), select, text, numeric, multiple choice buttons, text area, and dates. Because the surveys render into a 'shiny' app, designers can also leverage the reactive capabilities of 'shiny' to create dynamic and interactive surveys.

License MIT + file LICENSE

URL https://pkg.surveydown.org

BugReports https://github.com/surveydown-dev/surveydown/issues

Depends R (>= 4.1.0)

Imports cli, DBI, dotenv, fs, htmltools, jsonlite, markdown, miniUI, pool, quarto, rmarkdown, RPostgres, rstudioapi, rvest, shiny, shinyjs, shinyWidgets, utils, xml2, yaml

Suggests glue, knitr, leaflet, testthat

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author John Paul Helveston [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-2657-9191>), Pingfan Hu [aut, cph] (ORCID: <https://orcid.org/0009-0001-4877-4844>), Bogdan Bunea [aut, cph] (ORCID:

<https://orcid.org/0009-0006-2942-0588>),
Stefan Munnes [ctb],
Reed Benkendorf [ctb] (ORCID: <https://orcid.org/0000-0001-9178-3087>),
Umair Durrani [ctb],
Dylan Pieper [ctb]

Maintainer John Paul Helveston <john.helveston@gmail.com>

Repository CRAN

Date/Publication 2026-06-23 05:10:14 UTC

Contents

`sd_add_page` *Add a Page Template to the Current Document*

Description

This function inserts a template for a surveydown page at the current cursor position in the active RStudio document. It provides a basic structure for a new page, including a title, content area, and a next button. If the function call exists in the document, it will be removed before inserting the template.

Usage

```
sd_add_page(page_id = "page_id")
```

Arguments

`page_id`	A character string specifying the ID for the page. Defaults to "page_id".

Details

IMPORTANT: This function should be run outside any division or R code chunk in your 'Quarto' document. Running it inside a division or code chunk may result in an incorrect page structure.

The function performs the following steps:

1. Checks for and removes any existing `sd_add_page()` function call in the document.
2. Inserts a template at the current cursor position.

The template includes:

- A div with class `'sd-page'` and the specified page ID
- A placeholder for the page title
- A placeholder for page contents
- An R code chunk with a placeholder for questions and a next button

Special page_id values:

- When page_id is "end", a thank-you page template with `sd_close()` is inserted

Value

This function does not return a value. It modifies the active document as a side effect by inserting text and potentially removing a function call.

Examples

```
if (interactive()) {
  library(surveydown)

  # Insert a new page template with default ID
  sd_add_page()

  # Insert a new page template with custom ID
  sd_add_page(page_id = "welcome")

  # Insert an end/thank you page
  sd_add_page(page_id = "end")
}
```

---

sd_add_question *Add a Question Template to the Current Document*

Description

This function inserts a template for a surveydown question at the current cursor position in the active RStudio document. It supports various question types and automatically removes the function call before inserting the template if it exists in the document.

Usage

```
sd_add_question(type = "mc", id = NULL, label = NULL, chunk = FALSE)
```

Arguments

type	A character string specifying the type of question template to insert. Default is "mc" (multiple choice). Available options are:

- "mc": Multiple choice (single selection)
- "select": Dropdown selection
- "mc_multiple": Multiple choice (multiple selections)
- "mc_buttons": Multiple choice with button layout (single selection)
- "mc_multiple_buttons": Multiple choice with button layout (multiple selections)
- "text": Short text input
- "textarea": Long text input
- "numeric": Numeric input
- "slider": Slider input
- "date": Date input
- "daterange": Date range input

id	A character string specifying the ID for the question. If not provided, a default ID based on the question type will be used. This ID should be unique within your survey.
label	A character string specifying the label (question text) to display to respondents. If not provided, a default label placeholder will be used.
chunk	Logical. If TRUE, the code will be generated with the R code chunk wrapper. Defaults to FALSE.

Details

The function performs the following steps:

1. Checks for and removes any existing sd_add_question() function call in the document.
2. Inserts the appropriate question template at the current cursor position.
3. If an ID is provided, replaces the default ID in the template with the provided ID.
4. If a label is provided, replaces the default label in the template with the provided label.

Value

This function does not return a value. It modifies the active document as a side effect by inserting text and potentially removing a function call.

Examples

```
if (interactive()) {
  library(surveydown)

  # Insert a default multiple choice question template
  sd_add_question()

  # Insert a text input question with custom ID and label
  sd_add_question("text", id = "user_email", label = "What is your email address?")

  # Insert a slider question template
 sd_add_question("slider", id = "satisfaction", label = "How satisfied were you with our service?")
}
```

---

sd_close *Create a 'Close' Button to Exit the Survey*

Description

This function creates a 'Close' button that, when clicked, will trigger the exit process for the survey. Depending on the server-side configuration, this may show a rating question or a simple confirmation dialog before attempting to close the current browser tab or window.

Usage

```
sd_close(
  label_close = NULL,
  label_previous = NULL,
  show_previous = NULL,
  show_restart = FALSE,
  label_restart = NULL,
  clear_cookies = FALSE
)
```

Arguments

label_close	Character string. The label of the 'Close' button. Defaults to NULL, in which case the word "Exit Survey" will be used.
label_previous	Character string. The label for the 'Previous' button. Defaults to NULL, which uses "← Previous" (or the translated equivalent).
show_previous	Logical. Whether to show the Previous button alongside the Close button. Set to TRUE to allow users to go back before closing. Defaults to FALSE. Note: Unlike sd_nav(), this parameter does NOT read from the show-previous YAML setting.
show_restart	Logical. Whether to show a 'Restart Survey' button alongside the Close button. When TRUE, displays two side-by-side buttons. Defaults to FALSE.
label_restart	Character string. The label for the 'Restart Survey' button. Defaults to NULL, which uses "Restart Survey" (or the translated equivalent).
clear_cookies	Logical. Whether to clear cookies on exit without showing a restart button. Use for use cases where the restart button isn't needed but cookies should be cleared for later resubmission. Defaults to FALSE.

Details

The function generates a 'shiny' action button that, when clicked, triggers the 'show_exit_modal' event. The server-side logic (controlled by the rate_survey parameter in sd_server()) determines whether to show a rating question or a simple confirmation dialog.

The function also includes a custom message handler for closing the window. This is necessary because some browsers may not allow JavaScript to close windows that were not opened by JavaScript. In such cases, the user will be prompted to close the tab manually.

Value

A 'shiny' tagList containing the 'Close' button UI element and associated JavaScript for the exit process.

Note

The actual behavior of the exit process (whether to show a rating question or not) is controlled by the rate_survey parameter in the sd_server() function, not in this UI function.

See Also

sd_server

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_close() in survey.qmd on the last page:
  # --- end
  #
  # Thank you for completing the survey!
  #
  # `r sd_close(label_close = "Exit Survey", show_previous = TRUE)`

  # Find a working directory and start from a template:
  sd_create_survey(template = "default")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

---

sd_completion_code *Generate a Random Completion Code*

Description

This function generates a random completion code with a specified number of digits. The code is returned as a character string.

Usage

```
sd_completion_code(digits = 6)
```

Arguments

digits	An integer specifying the number of digits in the completion code. Must be a positive integer. Default is 6.

Value

A character string representing the random completion code.

Examples

```
library(surveydown)

sd_completion_code()  # generates a 6-digit code
sd_completion_code(digits = 8)  # generates an 8-digit code
sd_completion_code(digits = 4)  # generates a 4-digit code
```

```
sd_completion_code(digits = 10)  # generates a 10-digit code
```

`sd_copy_value` *Create a copy of a value*

Description

This function creates a copy of an input value and makes it available as a new output. The new output can then be displayed using `sd_output()`.

Usage

```
sd_copy_value(id, id_copy)
```

Arguments

`id`	Character string. The ID of the input value to copy.
`id_copy`	Character string. The ID for the new copy (must be different from `id`).

Value

`NULL` invisibly. This function is called for its side effects.

See Also

`sd_output()` for displaying the copied value

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_copy_value() to copy an input value to display elsewhere:
  # server <- function(input, output, session) {
  #   sd_copy_value(id = "name", id_copy = "name_copy")
  #   ...
  # }
  # Then in survey.qmd, display it in an R chunk:
  # sd_output("name", type = "value")

  # Find a working directory and start from a template:
  sd_create_survey(template = "default")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

sd_create_messages *Create a messages template file*

Description

This function is depreciated. Creates a template messages.yml file in the project root directory that users can customize to modify system messages.

Usage

```
sd_create_messages(language = "en", path = getwd())
```

Arguments

language	Character string specifying the language to use. See https://shiny.posit.co/r/reference/shiny/1.7.0/dateinpu for supported languages. Also, if "en", "de", "es", "fr", or "it" is chosen, default messages in those langauges will be used, otherwise the default English messages will be used. Defaults to "en".
path	Character string specifying the directory where the messages.yml file should be created. Defaults to the current working directory. The file should be placed in the root project folder of your surveydown survey.

Value

Invisible NULL.

Examples

```
if (interactive()) {
  # Create English template
  sd_create_messages()

  # Create German template
  sd_create_messages(language = "de")

  # Create Japanese template
  # Will use English messages but Japanese date picker - user can modify
  # the messages as desired
  sd_create_messages(language = "ja")
}
```

`sd_create_survey` *Create a new survey template*

Description

This function creates a new survey template by copying template files to the specified directory. You can choose from various predefined templates, including the default built-in template and specialized templates from the surveydown-dev/templates repository.

Usage

```
sd_create_survey(template = "default", path = getwd(), ask = TRUE)
```

Arguments

`template`	A character string specifying the template to use. Default is "default" which uses the built-in package template. Other options include: **default** The default built-in template **conditional_showing** Conditionally showing questions and pages **conditional_skipping** Conditionally skipping to forward pages **conditional_stopping** Conditionally stopping the navigation **conjoint_buttons** Conjoint analysis with button interface **conjoint_tables** Conjoint analysis with table interface **custom_leaflet_map** Survey with interactive Leaflet maps **custom_plotly_chart** Survey with Plotly visualizations **external_redirect** Template with external site redirects **live_polling** Live polling template for real-time surveys **option_shuffling** Survey with shuffled question options **question_types** Showcases all available question types **questions_yml** Survey with questions defined in a YAML file **random_options** Survey with randomized question options **random_options_predefined** Randomized options from predefined sets **reactive_drilldown** Dynamic questions with drill-down capability **reactive_questions** Survey with reactive questions
`path`	A character string specifying the directory where the survey template should be created. Defaults to the current working directory.
`ask`	Logical. If `TRUE` (default), prompts for user confirmation when creating the survey in the current directory. If `FALSE`, bypasses the confirmation prompt and proceeds without asking.

Details

When creating a new survey template, this function will:

1. Check if the specified template is valid
2. Confirm the destination path with the user (if it's the current directory)
3. Download template files from GitHub if a non-default template is specified
4. Copy template files to the destination directory
5. Skip .Rproj files if one already exists in the destination
6. Prompt for confirmation before overwriting existing files

External templates are downloaded from the surveydown-dev/templates GitHub repository.

Value

Invisible `NULL`. The function is called for its side effects.

Examples

```
if (interactive()) {
  # Create a survey with the "question_types" template in the "my_survey" directory
  sd_create_survey(template = "question_types", path = "my_survey")

  # Create a survey using the default template in the "my_survey" directory
  sd_create_survey(path = "my_survey")

  # Create a survey with default template in current directory
  sd_create_survey("default")

  # Create a survey without asking for confirmation
  sd_create_survey(template = "default", path = "my_survey", ask = FALSE)
}
```

---

sd_dashboard *Depreciated Survey Dashboard*

Description

This dashboard was depreciated in version v0.13.0. Now the sdstudio package fully includes the functionality that was previously included in this function.

Usage

```
sd_dashboard(gssencmode = "auto")
```

Arguments

gssencmode	Character string. The GSS encryption mode for the database connection. Defaults to "auto". Options are: • "auto": Tries "prefer" first, then falls back to "disable" if GSSAPI negotiation fails • "prefer": Uses GSSAPI encryption if available, plain connection otherwise • "disable": Disables GSSAPI encryption entirely Set to "disable" if you're having connection issues on a secure connection like a VPN.

---

sd_database *Connect to a 'PostgreSQL' Database with Automatic Cleanup*

Description

This function establishes a connection pool to a 'PostgreSQL' database (e.g. Supabase) and sets up automatic cleanup when the 'shiny' session ends.

Usage

```
sd_database(
  host = NULL,
  dbname = NULL,
  port = NULL,
  user = NULL,
  table = NULL,
  password = Sys.getenv("SURVEYDOWN_PASSWORD"),
  gssencmode = "prefer",
  ignore = FALSE,
  min_size = 1,
  max_size = Inf
)
```

Arguments

host	Character string. The host address of the PostgreSQL database.
dbname	Character string. The name of the PostgreSQL database.
port	Integer. The port number for the PostgreSQL database connection.
user	Character string. The username for the PostgreSQL database connection.
table	Character string. The name of the table to interact with in the Supabase database.
password	Character string. The password for the PostgreSQL database connection. NOTE: While you can provide a hard-coded password here, we do NOT recommend doing so for security purposes. Instead, you should establish a password with surveydown::sd_set_password(), which will create a local .Renviron file

that stores your password as a SURVEYDOWN_PASSWORD environment variable. The password argument uses this as the default value, so if you set a password properly with surveydown::sd_set_password(), then you can safely ignore using the password argument here.

gssencmode — Character string. The GSS encryption mode for the database connection. Defaults to "prefer". NOTE: If you have verified all connection details are correct but still cannot access the database, consider setting this to "disable". This can be necessary if you're on a secure connection, such as a VPN.

ignore — Logical. If TRUE, data will be saved to a local CSV file instead of the database. Defaults to FALSE.

min_size — Integer. The minimum number of connections in the pool. Defaults to 1.

max_size — Integer. The maximum number of connections in the pool. Defaults to Inf.

Value

A list containing the database connection pool (db) and the table name (table), or NULL if in ignore mode or if there's an error.

Examples

```
if (interactive()) {
  # Assuming SURVEYDOWN_PASSWORD is set in .Renviron
  db <- sd_database(
    host   = "aws-0-us-west-1.pooler.supabase.com",
    dbname = "postgres",
    port   = "6---",
    user   = "postgres.k----------i",
    table  = "your-table-name",
    ignore = FALSE
  )

  # Print the structure of the connection
  str(db)

  # Close the connection pool when done
  if (!is.null(db)) {
    pool::poolClose(db$db)
  }
}
```

---

sd_db_config *Configure database settings*

Description

Set up or modify database configuration settings in a .env file. These settings are used to establish database connections for storing survey responses.

Usage

```
sd_db_config(
  url = NULL,
  host = NULL,
  dbname = NULL,
  port = NULL,
  user = NULL,
  table = NULL,
  password = NULL,
  interactive = NULL
)
```

Arguments

url	Character string. A full PostgreSQL connection URL, e.g. "postgresql://user:password@host:port/ If provided, the URL is parsed to extract host, port, dbname, user, and password. If the password portion is a placeholder (e.g. [YOUR-PASSWORD]), you will be prompted to enter it. Individual parameters override values parsed from the URL.
host	Character string. Database host
dbname	Character string. Database name
port	Character string. Database port
user	Character string. Database user
table	Character string. Table name
password	Character string. Database password
interactive	Logical. Whether to use interactive setup. Defaults to TRUE if no parameters provided

Value

Invisibly returns a list of the current configuration settings

See Also

- sd_db_connect() to connect to the database

Examples

```
if (interactive()) {
  # Interactive setup
  sd_db_config()

  # Quick setup from Supabase connection URL
  sd_db_config(url = "postgresql://postgres.ref:password@host:6543/postgres")

  # Update specific settings
  sd_db_config(table = "new_table")
```

```
  # Update multiple settings
  sd_db_config(
    host = "new_host",
    port = "5433",
    table = "new_table"
  )
}
```

---

sd_db_connect *Connect to a PostgreSQL database for storing survey responses*

Description

Establishes a connection pool to a PostgreSQL database (e.g. Supabase) using credentials from a `.env` file. The survey operating mode (`"database"`, `"preview"`, or `"local"`) is controlled via the `mode` key under `survey-settings` in `survey.qmd`, not by this function.

Usage

```
sd_db_connect(env_file = ".env", ignore = NULL, gssencmode = "auto")
```

Arguments

`env_file`	Character string. Path to the env file. Defaults to `".env"`.
`ignore`	Logical. Deprecated. Use `mode: preview` in `survey.qmd` YAML instead. If `TRUE`, returns `NULL` with a deprecation warning. Defaults to `NULL`.
`gssencmode`	Character string. The GSS encryption mode for the database connection. Defaults to `"auto"`. Options are: • `"auto"`: Tries `"prefer"` first, then falls back to `"disable"` if GSSAPI negotiation fails • `"prefer"`: Uses GSSAPI encryption if available, plain connection otherwise • `"disable"`: Disables GSSAPI encryption entirely NOTE: If you have verified all connection details are correct but still cannot access the database, try setting this to `"disable"`.

Value

A list containing:

- `db`: The database connection pool
- `table`: The database table name

Returns `NULL` if the database connection fails or `ignore = TRUE`.

Examples

```
if (interactive()) {
  db <- sd_db_connect()

  # Close connection when done
  if (!is.null(db$db)) {
    pool::poolClose(db$db)
  }
}
```

`sd_display_question` *Create a placeholder for a reactive survey question*

Description

This function is depreciated - use `sd_output()` instead.

Usage

```
sd_display_question(id)
```

Arguments

`id`	A unique identifier for the question.

Value

A 'shiny' UI element that serves as a placeholder for the reactive question.

`sd_display_value` *Display the value of a survey question*

Description

This function is depreciated - use `sd_output()` instead.

Usage

```
sd_display_value(id, display_type = "inline", wrapper = NULL, ...)
```

Arguments

`id`	The ID of the question to display
`display_type`	The type of display. Can be `"inline"` (default), `"text"`, `"verbatim"`, or `"ui"`.
`wrapper`	A function to wrap the output
`...`	Additional arguments passed to the wrapper function

Value

A 'shiny' UI element displaying the question's value

`sd_get_data` *Fetch data from a database table with automatic reactivity detection*

Description

This function retrieves all data from a specified table in a database. It automatically detects whether it's being used in a reactive context (e.g., within a 'shiny' application) and behaves accordingly. In a reactive context, it returns a reactive expression that automatically refreshes the data at specified intervals.

Usage

```
sd_get_data(db, table = NULL, refresh_interval = NULL)
```

Arguments

`db`	A list containing database connection details created using `sd_db_config()`. Must have elements: • `db`: A `DBI` database connection object • `table`: A string specifying the name of the table to query
`table`	Character string. Database table name to obtain data from, overrides the table provided in the `db` argument. Defaults to `NULL`.
`refresh_interval`	Numeric. The time interval (in seconds) between data refreshes when in a reactive context. Default is `NULL`, meaning the data will not refresh.

Value

In a non-reactive context, returns a data frame containing all rows and columns from the specified table. In a reactive context, returns a reactive expression that, when called, returns the most recent data from the specified database table.

Examples

```
# Non-reactive context example
## Not run:
  library(surveydown)

  # Assuming you have a database connection called db created using
  # sd_database(), you can fetch data with:

  data <- sd_get_data(db)
  head(data)
```

```
  # Reactive context example (inside a surveydown app)

  server <- function(input, output, session) {
    data <- sd_get_data(db, refresh_interval = 10)

    output$data_table <- renderTable({
      data()  # Note the parentheses to retrieve the reactive value
    })
  }

## End(Not run)
```

`sd_get_url_pars` *Get URL Parameters in a 'shiny' Application*

Description

This function retrieves URL parameters from the current 'shiny' session. It must be called from within a 'shiny' reactive context.

Usage

```
sd_get_url_pars(...)
```

Arguments

`...`	Optional. Names of specific URL parameters to retrieve. If none are specified, all URL parameters are returned.

Value

A reactive expression that returns a list of URL parameters.

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_get_url_pars() to parse URL parameters:
  # server <- function(input, output, session) {
  #   # Get all URL parameters
  #   all_params <- sd_get_url_pars()
  #
  #   # Get specific parameters
  #   user_id <- sd_get_url_pars("user_id")
  #
  #   # Use in reactive expressions
  #   url_redirect <- reactive({
  #     params <- sd_get_url_pars()
  #     paste0("https://example.com?id=", params["id"])
```

```
  #   })
  #   sd_server()
  # }

  # Find a working directory and start from a template:
  sd_create_survey(template = "external_redirect")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

`sd_include_folder` *Include a folder to the 'shiny' resource path*

Description

This function includes a specified folder to the 'shiny' resource path, making it accessible for serving static files in a 'shiny' application. It checks for pre-existing resource paths to avoid conflicts with folders already included by the package.

Usage

```
sd_include_folder(folder)
```

Arguments

`folder`	A character string specifying the name of the folder to include. This folder should exist in the root directory of your 'shiny' app.

Value

`NULL` invisibly. The function is called for its side effect of adding a resource path to 'shiny'.

Examples

```
if (interactive()) {
  library(shiny)

  # Create an "images" folder
  dir.create("images")

  # Include the folder in the shiny resource path
  sd_include_folder("images")
}
```

`sd_is_answered` *Check if a question is answered*

Description

This function checks if a given question has been answered by the user. For matrix questions, it checks if all sub-questions (rows) are answered.

Usage

```
sd_is_answered(question_id)
```

Arguments

`question_id`	The ID of the question to check.

Value

A logical value: `TRUE` if the question is answered, `FALSE` otherwise.

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_is_answered() to conditionally execute code:
  # server <- function(input, output, session) {
  #   observe({
  #     if (sd_is_answered(input$age)) {
  #       message("Age question answered!")
  #     }
  #   })
  #   sd_server()
  # }

  # Find a working directory and start from a template:
  sd_create_survey(template = "default")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

sd_nav *Create Navigation Buttons for Survey Pages*

Description

This function creates both 'Previous' and 'Next' buttons for navigating between pages in a survey-down survey. The buttons are positioned on the left (Previous) and right (Next) of the page. The Previous button allows users to return to previously visited pages, while the Next button maintains the standard forward navigation behavior.

Usage

```
sd_nav(
  page_next = NULL,
  label_previous = NULL,
  label_next = NULL,
  show_previous = NULL,
  show_next = TRUE
)
```

Arguments

page_next	Character string. The ID of the next page to navigate to when the Next button is clicked. If NULL, the survey will navigate to the default next page in sequence.
label_previous	Character string. The label for the 'Previous' button. Defaults to NULL, which uses "← Previous" (or the translated equivalent).
label_next	Character string. The label for the 'Next' button. Defaults to NULL, which uses "Next →" (or the translated equivalent).
show_previous	Logical. Whether to show the Previous button. Set to FALSE for the first page where there is no previous page to navigate to. If NULL (default), uses the show-previous setting from YAML or sd_server().
show_next	Logical. Whether to show the Next button. Set to FALSE to hide the Next button. Defaults to TRUE.

Details

The function generates two 'shiny' action buttons:

- **Previous button:** Positioned on the left, navigates to the last visited page. Uses page history tracking to determine the previous page.
- **Next button:** Positioned on the right, navigates forward. Can be activated by clicking or pressing the Enter key when visible.

The buttons are styled to appear on opposite sides of the page using flexbox layout, and include arrow symbols to indicate direction.

Value

A 'shiny' tagList containing the navigation buttons UI elements.

See Also

sd_next for the legacy single-button navigation (deprecated)

Examples

```
if (interactive()) {
  library(surveydown)

  # Basic usage with both buttons
  sd_nav()

  # First page - hide Previous button
  sd_nav(show_previous = FALSE)

  # Last page - hide Next button
  sd_nav(show_next = FALSE)

  # Hide both navigation buttons
  sd_nav(show_previous = FALSE, show_next = FALSE)

  # Custom labels
  sd_nav(
    label_previous = "Go Back",
    label_next = "Continue"
  )

  # Specify next page explicitly
  sd_nav(page_next = "demographics")
}
```

---

sd_next *Create a 'Next' Button for Page Navigation*

Description

This function creates a 'Next' button for navigating to the specified next page in a Surveydown survey. The button can be activated by clicking or by pressing the Enter key when visible.

Usage

```
sd_next(next_page = NULL, label = NULL)
```

Arguments

next_page	Character string. The ID of the next page to navigate to. This parameter is required.
label	Character string. The label of the 'Next' button. Defaults to NULL, in which case the word "Next" will be used.

Details

The function generates a 'shiny' action button that, when clicked or when the Enter key is pressed, sets the input value to the specified next page ID, facilitating page navigation within the Shiny application. The button is styled to appear centered on the page and includes a class for Enter key functionality.

Value

A 'shiny' tagList containing the 'Next' button UI element.

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_next() in survey.qmd to create navigation:
  # --- welcome
  #
  # Welcome to the survey!
  #
  # `r sd_next(next_page = "page2", label = "Continue")`

  # Find a working directory and start from a template:
  sd_create_survey(template = "default")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

---

sd_output *Output Function for Displaying reactive objects and values*

Description

Output Function for Displaying reactive objects and values

Usage

```
sd_output(
  id,
  type = NULL,
  width = "100%",
```

```
  display = "text",
  inline = TRUE,
  wrapper = NULL,
  ...
)
```

Arguments

id	Character string. A unique identifier for the output element.
type	Character string. Specifies the type of output corresponding with the question id. Can be "question", "value", "label_option", "label_question", or NULL. If "question", it will display a question defined in the server. If "value", it will display the value of question id selected by the respondent. If "label_option", it will display the label of the option for question id selected by the respondent. If "label_question", it will display the label argument value for question id. Finally, if NULL, the function behaves like shiny::uiOutput().
width	Character string. The width of the UI element. Defaults to "100%".
display	Character string. Specifies the display type for "value" outputs. Can be "text", "verbatim", or "ui". Only used when type = "value".
inline	Logical. Whether to render the output inline. Defaults to TRUE.
wrapper	Function. A function to wrap the output. Only used when type = "value".
...	Additional arguments passed to the underlying 'shiny' functions or the wrapper function.

Details

The function behaves differently based on the type parameter:

- If type is NULL, it acts like shiny::uiOutput().
- If type is "question", it creates a placeholder for a reactive survey question.
- If type is "value", it creates an output to display the value of a survey question, with the display style determined by the display parameter.

Value

A 'shiny' UI element, the type of which depends on the input parameters.

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_output() to display reactive questions or values:
  # First, define something in server of app.R:
  # server <- function(input, output, session) {
  #   completion_code <- sd_completion_code(10)
  #   sd_store_value(completion_code)
  #   sd_server()
```

```
  # }

  # Then, display in R chunks of survey.qmd:
  # Your code is: `r sd_output("completion_code", type = 'value')`

  # Find a working directory and start from a template:
  sd_create_survey(template = "reactive_questions")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

`sd_page_gadget` *Show a Shiny gadget for entering a page ID*

Description

This function displays a Shiny gadget that allows the user to input a page ID. Once submitted, it calls sd_add_page() with the specified ID.

Usage

```
sd_page_gadget()
```

Value

The entered page ID (invisibly).

`sd_question` *Create a survey question*

Description

This function creates various types of survey questions for use in a Surveydown survey.

Usage

```
sd_question(
  id,
  type = NULL,
  label = NULL,
  option = NULL,
  options = NULL,
  option_attr = NULL,
  cols = "80",
  direction = "horizontal",
  status = "default",
```

```
  width = "100%",
  height = NULL,
  selected = NULL,
  label_select = "Choose an option...",
  grid = TRUE,
  individual = TRUE,
  justified = FALSE,
  force_edges = TRUE,
  placeholder = NULL,
  resize = NULL,
  row = NULL,
  default = NULL,
  image = NULL,
  yml = "questions.yml",
  matrix_question_width = NULL,
  ...
)
```

Arguments

id	A unique identifier for the question, which will be used as the variable name in the resulting survey data.
type	Specifies the type of question. Possible values are "select", "mc", "mc_multiple", "mc_buttons", "mc_multiple_buttons", "mc_image", "mc_multiple_image", "text", "textarea", "numeric", "slider", "slider_numeric", "date", "daterange", "matrix", and "matrix_multiple". Defaults to NULL.
label	Character string. The label for the UI element, which can be formatted with markdown. Defaults to NULL
option	Named vector for the "select", "radio", "checkbox", and "slider" question types, or numeric vector for "slider_numeric" question type. Can be provided in multiple formats: • Named vector: c("Display A" = "value_a", "Display B" = "value_b") - Names are shown in UI, values are stored in database • Unnamed character vector: c("Option 1", "Option 2") - Values are used as both display labels and stored values (e.g., "Option 1" is shown and stored as "Option 1") • Unnamed numeric vector: c(1, 2, 3) - For non-slider questions, converted to c("1" = "1", "2" = "2", "3" = "3"). For slider_numeric, kept as numeric.
options	Alias for option. Either option or options can be used. If both are provided, option takes precedence. Supports the same formats as option.
option_attr	Character vector. Optional HTML attributes to apply to individual options for "mc" and "mc_multiple" question types. Each element corresponds to an option in the same position. Defaults to NULL.
cols	Integer. Number of columns for the "textarea" question type. Defaults to 80.
direction	Character string. The direction for button groups ("horizontal" or "vertical"). Defaults to "horizontal".

status — Character string. The status for button groups. Defaults to "default".

width — Character string. The width of the UI element. Defaults to "100%".

height — Character string. The height of the input for the "textarea" question type. Defaults to "100px".

selected — Value. The selected value(s) for certain input elements.

label_select — Character string. The label for the select input. Defaults to "Choose an option...".

grid — Logical. Whether to show tick marks and labels under each position of a "slider" question. Defaults to TRUE. Only the major (labeled) tick marks are shown; the small minor tick marks between positions are hidden by surveydown's styling.

individual — Logical. Whether buttons in a group should be individually styled. Defaults to TRUE.

justified — Logical. Whether buttons in a group should fill the width of the parent div. Defaults to FALSE.

force_edges — Logical. Whether to force edges for slider input. Defaults to TRUE.

placeholder — Character string. Placeholder text for "text" and "textarea" question types.

resize — Character string. Resize option for textarea input. Defaults to NULL.

row — List. Used for "matrix" and "matrix_multiple" type questions. Contains the row labels and their corresponding IDs.

default — Numeric, length 1 (for a single sided slider), or 2 for a two sided (range based) slider. Values to be used as the starting default for the slider. Defaults to the median of values.

image — Character vector. Required for "mc_image" and "mc_multiple_image" question types: image paths or URLs, one per option, in the same order as option. Paths are used as-is in the image src, so they should resolve against the survey's images or www folder (e.g. "images/cat.png") or be full URLs.

For image-choice questions, the option names control captions:

- Named options (e.g. c("Cat" = "cat", "Dog" = "dog")) show the names as text captions beneath each image.
- An unnamed option vector (e.g. c("cat", "dog")) shows the images only, with no captions. The values are still stored in the data as usual.

This differs from the other multiple-choice types, where an unnamed option vector uses the values as both the displayed labels and the stored values.

yml — Character string. The name of the YAML file to load question configurations from. Defaults to "questions.yml". Custom YAML files can be specified, either in the root directory or subdirectories (e.g., "folder/custom.yml").

matrix_question_width — The width of the matrix question column. Accepts numeric (e.g., 40), character without percent (e.g., "40"), or character with percent (e.g., "40%") - all are treated equivalently as percentages. Defaults to NULL, which auto-calculates the width based on the longest row label (using a heuristic of 20% base + 0.5% per character, bounded between 30% and 80%). The remaining width is automatically distributed equally among the option columns.

... — Additional arguments, often specific to different input types. Examples include pre, sep, step, and animate for "slider" and "slider_numeric" question types, etc.

Details

The function supports various question types:

- "select": A dropdown selection
- "mc": Multiple choice (single selection)
- "mc_multiple": Multiple choice (multiple selections allowed)
- "mc_buttons": Multiple choice with button-style options (single selection)
- "mc_multiple_buttons": Multiple choice with button-style options (multiple selections allowed)
- "mc_image": Multiple choice where each option is an image card (single selection)
- "mc_multiple_image": Multiple choice where each option is an image card (multiple selections allowed)
- "text": Single-line text question
- "textarea": Multi-line text question
- "numeric": Numeric question
- "slider": Slider question
- "slider_numeric": Extended numeric slider question
- "date": Date question
- "daterange": Date range question
- "matrix": Matrix-style question with rows and columns (single selection per row, radio buttons)
- "matrix_multiple": Matrix-style question where each row allows multiple selections (checkboxes)

For "matrix" and "matrix_multiple" type questions, use the row parameter to define the rows of the matrix. Each element in the row list should have a name (used as the row ID) and a value (used as the row label). Each row becomes its own sub-question with the ID <question_id>_<row_id>, stored as a separate column in the data.

Value

A 'shiny' UI element wrapped in a div with a data attribute for question ID.

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_question() to create questions in R chunks of survey.qmd:
  # sd_question(
  #   id    = "favorite_penguin_static",
  #   type  = "mc",
  #   label = "Which type of penguin do you like the best?",
  #   option = c(
  #     "Adélie"    = "adelie",
```

```
#     "Chinstrap" = "chinstrap",
#     "Gentoo"    = "gentoo"
#   )
#  )

# Use sd_question() to create reactive questions in app.R under server:
# server <- function(input, output, session) {
#   sd_question(
#     id    = "favorite_penguin_reactive",
#     type  = "mc",
#     label = "Which type of penguin do you like the best?",
#     option = c(
#       "Adélie"    = "adelie",
#       "Chinstrap" = "chinstrap",
#       "Gentoo"    = "gentoo"
#     )
#    )
#   sd_server()
# }

# Find a working directory and start from a template:
sd_create_survey(template = "default")
# This creates survey.qmd and app.R - launch the survey using app.R
}
```

`sd_question_custom` *Create a Custom Question with a Shiny Widget*

Description

This function creates a custom survey question that incorporates any Shiny widget and captures its interaction value. It allows for the integration of interactive visualizations (e.g., maps, plots) or other custom Shiny outputs into a survey, storing the result of user interaction as survey data.

Usage

```
sd_question_custom(id, label, output, value, height = "400px")
```

Arguments

`id`	Character string. A unique identifier for the question.
`label`	Character string. The label text for the question, which can include HTML formatting.
`output`	Shiny UI element. The output of a Shiny widget (e.g., `leafletOutput()`, `plotlyOutput()`).
`value`	Reactive expression that returns the value to be stored in the survey data when the user interacts with the widget.
`height`	Character string. The height of the widget output. Defaults to "400px".

Details

The function creates a custom question container that includes:

- A visible widget output that users can interact with
- A hidden text input that stores the value from the interaction
- Automatic tracking of user interaction for progress monitoring

The value to be stored is controlled by the reactive expression provided to the `value` parameter, which should update whenever the user interacts with the widget in the desired way.

Value

None (called for side effects)

See Also

`sd_question()` for standard question types

Examples

```
if (interactive()) {
  library(surveydown)
  library(leaflet)

  server <- function(input, output, session) {
    # Create map output
    output$usa_map <- renderLeaflet({
      leaflet() |>
        addTiles() |>
        setView(lng = -98.5795, lat = 39.8283, zoom = 4)
    })

    # Reactive value for selected location
    selected_location <- reactiveVal(NULL)

    # Click observer
    observeEvent(input$usa_map_click, {
      click <- input$usa_map_click
      if (!is.null(click)) {
        selected_location(
          sprintf("Lat: %0.2f, Lng: %0.2f", click$lat, click$lng)
        )
      }
    })

    # Create the custom question
    sd_question_custom(
      id = "location",
      label = "Click on your location:",
      output = leafletOutput("usa_map", height = "400px"),
      value = selected_location
```

```
    )

    sd_server()
  }

  shinyApp(ui = sd_ui(), server = server)
}
```

`sd_question_gadget` *Show a Shiny gadget for selecting a question type*

Description

This function displays a Shiny gadget that allows the user to select a question type from a dropdown menu. Once submitted, it calls sd_add_question() with the specified type.

Usage

```
sd_question_gadget(chunk = FALSE)
```

Arguments

`chunk`	Logical. If `TRUE`, the code will be generated with the R code chunk wrapper. Defaults to `FALSE`.

Value

The selected question type (invisibly).

`sd_reactive` *Create a reactive value that is also stored in survey data*

Description

This function creates a reactive value similar to Shiny's reactive() function, but also automatically stores the calculated value in the survey data.

Usage

```
sd_reactive(id, expr, blank_na = TRUE)
```

Arguments

`id`	Character string. The id (name) of the value to be stored in the data.
`expr`	An expression that calculates a value based on inputs
`blank_na`	Logical. If TRUE, NA values are converted to empty strings. Default is TRUE.

Value

A reactive expression that can be called like a function

Examples

```
# This example shows how sd_reactive would be used in the app.R file
if (interactive()) {
  library(surveydown)
  library(shiny)

  # Demo app setup
  server <- function(input, output, session) {
    # Create a reactive value that is stored in survey data
    product <- sd_reactive("product", {
      as.numeric(input$first_number) * as.numeric(input$second_number)
    })

    # Display the result
    output$result <- renderText({
      paste("The product is:", product())
    })

    # The rest of your survey setup...
    sd_server()
  }

  # In your survey.qmd file, you would use:
  # The product is: `r sd_output("product", type = "value")`
}
```

---

sd_redirect *Create a Redirect Element for 'shiny' Applications*

Description

This function creates a UI element that redirects the user to a specified URL. It can be used in both reactive and non-reactive contexts within 'shiny' applications.

Usage

```
sd_redirect(
  id,
  url,
  button = TRUE,
  label = "Click here",
  delay = NULL,
  newtab = FALSE
)
```

Arguments

id	A character string of a unique id to be used to identify the redirect button in the survey body. In reactive contexts, this becomes the output ID, while the actual button gets the ID id + "_btn" to avoid input/output conflicts.
url	A character string specifying the URL to redirect to.
button	A logical value indicating whether to create a button (TRUE) or a text element (FALSE) for the redirect. Default is TRUE.
label	A character string for the button or text label. Defaults to NULL, in which case the words "Click here" will be used.
delay	An optional numeric value specifying the delay in seconds before automatic redirection. If NULL (default), no automatic redirection occurs.
newtab	A logical value indicating whether to open the URL in a new tab (TRUE) or in the current tab (FALSE). Default is FALSE.

Value

In a reactive context, creates an output with the specified ID that can be displayed using sd_output(). The actual button element gets the ID id + "_btn" to prevent input/output conflicts. In a non-reactive context, returns the redirect element directly.

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_redirect() to create redirect in R chunks of survey.qmd:
  # sd_redirect(
  #   id     = "redirect",
  #   url    = "https://www.google.com",
  #   label  = "Redirect to Google",
  #   button = TRUE,
  #   newtab = TRUE
  # )

  # sd_redirect() also supports reactive redirections.
  # Find a working directory and start from a template:
  sd_create_survey(template = "external_redirect")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

---

sd_server *Server logic for a surveydown survey*

Description

This function defines the server-side logic for a 'shiny' application used in surveydown. It handles various operations such as conditional display, progress tracking, page navigation, database updates for survey responses, and exit survey functionality.

All survey settings are configured in the `survey.qmd` YAML header under `survey-settings`. See the surveydown documentation for available options.

Usage

```
sd_server(db = NULL)
```

Arguments

db	A list containing database connection information created using `sd_database()` function. Defaults to `NULL`. If `NULL`, will be auto-detected from the calling environment or remain `NULL` (ignore mode).

Details

The function performs the following tasks:

- Initializes variables and reactive values, including the `all_data` reactive list for accessing question responses.
- Implements conditional display logic for questions.
- Tracks answered questions and updates the progress bar.
- Handles page navigation and skip logic.
- Manages required questions.
- Performs database operation.
- Controls auto-scrolling behavior based on survey settings.
- Uses sweetalert for warning messages when required questions are not answered.
- Handles the exit survey process based on survey settings.

All survey behavior is configured through the `survey-settings` section in the `survey.qmd` YAML header. Available settings include:

- `show-previous`: Show Previous button (default: FALSE)
- `use-cookies`: Enable cookie-based session management (default: TRUE)
- `auto-scroll`: Auto-scroll to next question (default: FALSE)
- `rate-survey`: Show rating question on exit (default: FALSE)
- `all-required`: Make all questions required (default: FALSE)
- `start-page`: ID of the page to start on (default: first page)
- `system-language`: Language for system messages (default: "en")
- `highlight-unanswered`: Highlight unanswered questions (default: TRUE)
- `highlight-color`: Color for highlighting (default: "gray")

- capture-metadata: Capture browser/IP info (default: TRUE)
- required: Vector of required question IDs
- shuffled: Vector of question IDs to shuffle (MC options or matrix rows)
- all-shuffled: Shuffle all MC options and matrix rows (default: FALSE)

Value

This function does not return a value; it sets up the server-side logic for the 'shiny' application and exposes the all_data reactive list in the parent environment.

Progress Bar

The progress bar is updated based on the last answered question. It will jump to the percentage corresponding to the last answered question and will never decrease, even if earlier questions are answered later. The progress is calculated as the ratio of the last answered question's index to the total number of questions.

Database Operations

If db is provided, the function will update the database with survey responses. If db is NULL (ignore mode), responses will be saved to a local CSV file.

Accessing Question Values

The sd_server() function exposes the all_data reactive values list that provides a reliable way to access question responses in your server logic. Unlike directly accessing input$question_id, which only works for questions on the current page, all_data$question_id works for all questions by automatically restoring values from the database when needed.

Use all_data in conditional logic:

```
sd_skip_if(
  all_data$age < 18 ~ "parental_consent",
  all_data$employed == "yes" ~ "employment_questions"
)
```

Or in custom reactive expressions:

```
output$custom_text <- renderText({
  paste("You selected:", all_data$favorite_color)
})
```

The all_data list automatically stays synchronized with question responses and includes restored values from previous sessions (via cookies or page refreshes), making it safe to use even when navigating backward or refreshing the page.

See Also

sd_database(), sd_ui()

Examples

```
if (interactive()) {
  library(surveydown)

  # Basic sd_server() usage in app.R:
  # library(surveydown)
  #
  # db <- sd_database(
  #   host   = "db.xxxx.supabase.co",
  #   dbname = "postgres",
  #   port   = "5432",
  #   user   = "postgres",
  #   table  = "my_survey"
  # )
  #
  # server <- function(input, output, session) {
  #   sd_server(db = db)
  # }
  #
  # shiny::shinyApp(ui = sd_ui(), server = server)

  # All settings are configured in survey.qmd YAML header:
  # ---
  # survey-settings:
  #   show-previous: true
  #   auto-scroll: true
  #   required:
  #     - age
  #     - name
  # ---

  # Using all_data for conditional logic:
  # server <- function(input, output, session) {
  #   sd_skip_if(
  #     all_data$age < 18 ~ "parental_consent",
  #     all_data$country == "USA" ~ "usa_specific"
  #   )
  #
  #   sd_server(db = db)
  # }

  # Find a working directory and start from a template:
  sd_create_survey(template = "default")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

---

sd_setup *Required Set Up Function*

Description

This function is depreciated and no longer needed.

Usage

```
sd_setup()
```

Details

The function configures the 'shiny' application to use Bootstrap 5 for styling and enables 'shinyjs' for JavaScript functionalities within the application.

Value

This function does not return a value. It is called for its side effects of setting up the 'shiny' application.

---

sd_set_password *Set password for surveydown survey*

Description

This function sets your surveydown password, which is used to access the 'PostgreSQL' data (e.g. Supabase). The password is saved in a `.Renviron` file and adds `.Renviron` to `.gitignore`.

Usage

```
sd_set_password(password)
```

Arguments

`password`	Character string. The password to be set for the database connection.

Details

The function performs the following actions:

1. Creates a `.Renviron` file in the root directory if it doesn't exist.
2. Adds or updates the `SURVEYDOWN_PASSWORD` entry in the `.Renviron` file.
3. Adds `.Renviron` to `.gitignore` if it's not already there.

Value

None. The function is called for its side effects.

Examples

```
## Not run:
  # Set a temporary password for demonstration
  temp_password <- paste0(sample(letters, 10, replace = TRUE), collapse = "")

  # Set the password
  sd_set_password(temp_password)

  # After restarting R, verify the password was set
  cat("Password is :", Sys.getenv('SURVEYDOWN_PASSWORD'))

## End(Not run)
```

---

sd_show_if *Define show conditions for survey questions and pages*

Description

This function is used to define conditions under which certain questions or pages in the survey should be shown. It takes one or more formulas where the left-hand side is the condition and the right-hand side is the target question ID or page ID. If called with no arguments, it will return NULL and set no conditions.

Usage

```
sd_show_if(...)
```

Arguments

...	One or more formulas defining show conditions. The left-hand side of each formula should be a condition based on input values, and the right-hand side should be the ID of the question or page to show if the condition is met.

Value

A list of parsed conditions, where each element contains the condition and the target question or page ID. Returns NULL if no conditions are provided.

See Also

sd_skip_if()

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_show_if() to conditionally show/hide questions:
  # server <- function(input, output, session) {
  #   sd_show_if(
  #     input$has_car == "yes" ~ "car_make",
  #     input$employed == "yes" ~ "job_title",
  #     input$age >= 18 ~ "adult_questions_page"
  #   )
  #   sd_server()
  # }

  # Find a working directory and start from a template:
  sd_create_survey(template = "conditional_showing")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

`sd_skip_forward` *Define forward skip conditions for survey pages (Deprecated)*

Description

This function is deprecated. Please use `sd_skip_if()` instead.

This function is used to define conditions under which certain pages in the survey should be skipped ahead to (forward only). It takes one or more formulas where the left-hand side is the condition and the right-hand side is the target page ID.

Usage

```
sd_skip_forward(...)
```

Arguments

`...`	One or more formulas defining skip conditions. The left-hand side of each formula should be a condition based on input values, and the right-hand side should be the ID of the page to skip to if the condition is met. Only forward skipping (to pages later in the sequence) is allowed.

Value

A list of parsed conditions, where each element contains the condition and the target page ID.

`sd_skip_if` *Define skip conditions for survey pages*

Description

This function is used to define conditions under which certain pages in the survey should be skipped. It takes one or more formulas where the left-hand side is the condition and the right-hand side is the target page ID. Only forward skipping is allowed to prevent navigation loops.

Usage

```
sd_skip_if(...)
```

Arguments

`...`	One or more formulas defining skip conditions. The left-hand side of each formula should be a condition based on input values, and the right-hand side should be the ID of the page to skip to if the condition is met. Only forward skipping (to pages later in the sequence) is allowed.

Value

A list of parsed conditions, where each element contains the condition and the target page ID.

See Also

`sd_show_if()`

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_skip_if() to skip pages based on answers:
  # server <- function(input, output, session) {
  #   sd_skip_if(
  #     input$age < 18 ~ "underage_page",
  #     input$country == "USA" ~ "usa_page",
  #     input$consent == "no" ~ "exit_page"
  #   )
  #   sd_server()
  # }

  # Find a working directory and start from a template:
  sd_create_survey(template = "conditional_skipping")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

`sd_stop_if` *Define stop conditions for survey questions*

Description

This function is used to define custom stop conditions for survey questions. When a user attempts to navigate to the next page, these conditions are checked first (before required question checks). If any condition is TRUE (i.e., the stop condition is met), a modal dialog displays the corresponding error messages and highlights the invalid questions in red.

Usage

```
sd_stop_if(...)
```

Arguments

`...`	One or more formulas defining stop conditions. The left-hand side of each formula should be a condition that, when TRUE, will stop navigation and show an error (e.g., `nchar(input$zip) != 5`). The right-hand side should be the error message to display when the condition is TRUE.

Details

The function performs the following:

- Automatically determines which page each stop condition belongs to based on the input variables referenced in the conditions
- Groups stop conditions by page and shows them together when multiple conditions are triggered on the same page
- Displays error messages as a numbered list when multiple stop conditions are triggered on the same page, or as a single message for individual triggers
- Highlights invalid question containers in red
- Takes higher priority than required question checks - stop conditions are checked first before required question checks

Value

This function does not return a value; it sets up stop logic for the survey.

See Also

`sd_show_if`, `sd_skip_if`

Examples

```
if (interactive()) {
  library(surveydown)

  # Define a server with custom stop conditions
  server <- function(input, output, session) {

    sd_stop_if(
      # Stop if zip code is not exactly 5 digits
      nchar(input$zip) != 5 ~ "Zip code must be 5 digits",
      # Stop if year of birth is before 1900
      as.numeric(input$yob) < 1900 ~ "Year of birth must be after 1900",
      # Stop if phone number is not exactly 10 digits
      nchar(input$phone) != 10 ~ "Phone number must be 10 digits"
    )

    sd_server()
  }
}
```

---

sd_store_value *Store a value in the survey data*

Description

This function allows storing additional values to be included in the survey data, such as respondent IDs or other metadata. When a database connection is provided, it implements session persistence - if a value already exists for the current session, storage is skipped to maintain consistency across page refreshes.

Usage

```
sd_store_value(value, id = NULL, db = NULL, auto_assign = TRUE)
```

Arguments

value	The value to be stored. This can be any R object that can be coerced to a character string.
id	(Optional) Character string. The id (name) of the value in the data. If not provided, the name of the value variable will be used. Cannot be one of the reserved IDs: "session_id", "time_start", "time_end", "exit_survey_rating", "current_page", "browser", or "ip_address".
db	(Optional) Database connection object created with sd_db_connect(). If provided, enables session persistence. If not provided, will automatically look for a variable named 'db' in the calling environment, or fall back to the database connection from the session.

auto_assign Logical. If TRUE (default), automatically assigns the stored value back to the original variable in the calling environment. This eliminates the need for explicit assignment when session persistence is desired. If FALSE, the function only returns the value without modifying the original variable.

Value

The value that was stored (either the new value or existing value from database if session persistence applies). This allows the function to be used in variable assignments.

Examples

```
if (interactive()) {
  library(surveydown)

  # Use sd_store_value() to store custom values in the database:
  # server <- function(input, output, session) {
  #   # Store a generated value
  #   respondentID <- sample(1:1000, 1)
  #   sd_store_value(respondentID, "respID", db)
  #
  #   # Store with automatic ID detection
  #   completion_code <- sample(0:9, 6, replace = TRUE)
  #   sd_store_value(completion_code, db = db)
  #
  #   # Store without auto-assignment
  #   stored_val <- sd_store_value(42, "my_value", db, auto_assign = FALSE)
  #
  #   sd_server()
  # }

  # Find a working directory and start from a template:
  sd_create_survey(template = "reactive_questions")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

---

sd_ui *Create the UI for a surveydown survey*

Description

This function creates the user interface for a surveydown survey, including necessary CSS and JavaScript files, and applies custom styling. It retrieves theme and progress bar settings from the survey.qmd file.

Usage

```
sd_ui()
```

Details

The function reads the following settings from the survey.qmd YAML header:

- `theme`: The theme to be applied to the survey.
- `barcolor`: The color of the progress bar (accepts hex colors like #FF0000 or #F00, or CSS color names like red, blue, green).
- `barposition`: The position of the progress bar (`'top'`, `'bottom'`, or `'none'`).

If `barcolor` is not specified or is `NULL`, the default theme color will be used. If `barposition` is not specified, it defaults to 'top'.

Value

A 'shiny' UI object

See Also

`sd_server()` for creating the server-side logic of the survey

Examples

```
if (interactive()) {
  library(surveydown)

  # Basic sd_ui() usage in app.R:
  # library(surveydown)
  #
  # ui <- sd_ui()
  #
  # server <- function(input, output, session) {
  #   sd_server()
  # }
  #
  # shiny::shinyApp(ui = ui, server = server)

  # Find a working directory and start from a template:
  sd_create_survey(template = "default")
  # This creates survey.qmd and app.R - launch the survey using app.R
}
```

---

sd_value *Access question values from survey responses (alias)*

Description

This is an alias for `sd_values()`.

Usage

```
sd_value(..., as_numeric = NULL, as_vector = NULL)
```

Arguments

`...`	One or more question IDs to retrieve. Each can be provided as an unquoted name (e.g., `age`) or a quoted string (e.g., `"age"`).
`as_numeric`	Logical or NULL. Controls numeric type conversion:

Value	Behavior
`NULL` (default)	Auto-detect: checks if value looks numeric, converts if so. Non-numeric values stay as character.
`TRUE`	Force convert: always calls `as.numeric()`. Non-convertible values become `NA`.
`FALSE`	Never convert: returns value as-is, no detection.

`as_vector`	Logical or NULL. Controls splitting of pipe-separated values:

Value	Behavior
`NULL` (default)	Auto-detect: checks if pipe symbol exists, splits if found.
`TRUE`	Force split: always splits on pipe.
`FALSE`	Never split: returns value as-is, no detection.

Details

This function provides a functional interface to access question values from the `all_data` reactive values list. It is equivalent to using `all_data$question_id` but allows programmatic access using a string variable or unquoted name. Multiple question IDs can be provided to retrieve multiple values at once.

Value

If a single question ID is provided, returns the value of that question. If multiple question IDs are provided, returns an unnamed vector of values. Returns `NULL` for any question ID that doesn't exist.

See Also

`sd_values()`

Examples

```
## Not run:
  library(surveydown)

  server <- function(input, output, session) {
    # Single value access - all equivalent:
    age1 <- all_data$age
    age2 <- sd_value(age)        # Unquoted (recommended)
    age3 <- sd_value("age")      # Quoted (also works)
```

```
  # Multiple values at once:
  values <- sd_value(age, name, country)
  # Returns values with auto-detection (numeric converted, pipes split)

  # Default behavior (NULL) auto-detects and converts numeric values:
  age <- sd_value(age)
  # If age is "25", returns 25 (numeric)
  # If age is "hello", returns "hello" (character, not convertible)

  # Force numeric conversion (may produce NA):
  val <- sd_value(some_field, as_numeric = TRUE)
  # "123" becomes 123, "hello" becomes NA

  # Never convert to numeric (e.g., ZIP codes with leading zeros):
  zip <- sd_value(zip_code, as_numeric = FALSE)
  # "01234" stays as "01234", not converted to 1234

  # Default behavior (NULL) auto-detects pipe and splits:
  fruits <- sd_value(fav_fruits)
  # If value is "apple|banana|orange", returns c("apple", "banana", "orange")
  # If value is "apple", returns "apple" (no pipe, no split)

  # Force split (even if no pipe present):
  vals <- sd_value(some_field, as_vector = TRUE)

  # Never split (keep as single string):
  raw <- sd_value(fav_fruits, as_vector = FALSE)
  # "apple|banana|orange" stays as "apple|banana|orange"

  # Check number of selections:
  if (length(sd_value(fav_fruits)) > 3) {
    # User selected more than 3 fruits
  }

  # Check if specific option was selected:
  if ("apple" %in% sd_value(fav_fruits)) {
    # User selected apple
  }

  sd_server(db = db)
}

## End(Not run)
```

---

sd_values *Access question values from survey responses*

Description

This function provides a functional interface to access question values from the `all_data` reactive values list. It is equivalent to using `all_data$question_id` but allows programmatic access using a string variable or unquoted name. Multiple question IDs can be provided to retrieve multiple values at once.

Usage

```
sd_values(..., as_numeric = NULL, as_vector = NULL)
```

Arguments

`...`	One or more question IDs to retrieve. Each can be provided as an unquoted name (e.g., `age`) or a quoted string (e.g., `"age"`).
`as_numeric`	Logical or NULL. Controls numeric type conversion:

Value	Behavior
`NULL` (default)	Auto-detect: checks if value looks numeric, converts if so. Non-numeric values stay as character.
`TRUE`	Force convert: always calls `as.numeric()`. Non-convertible values become `NA`.
`FALSE`	Never convert: returns value as-is, no detection.

`as_vector`	Logical or NULL. Controls splitting of pipe-separated values:

Value	Behavior
`NULL` (default)	Auto-detect: checks if pipe symbol exists, splits if found.
`TRUE`	Force split: always splits on pipe.
`FALSE`	Never split: returns value as-is, no detection.

Value

If a single question ID is provided, returns the value of that question. If multiple question IDs are provided, returns an unnamed vector of values. Returns `NULL` for any question ID that doesn't exist.

Examples

```
## Not run:
  library(surveydown)

  server <- function(input, output, session) {
    # Single value access - all equivalent:
    age1 <- all_data$age
    age2 <- sd_values(age)        # Unquoted (recommended)
    age3 <- sd_values("age")      # Quoted (also works)

    # Multiple values at once:
    values <- sd_values(age, name, country)
    # Returns values with auto-detection (numeric converted, pipes split)
```

```
  # Default behavior (NULL) auto-detects and converts numeric values:
  age <- sd_values(age)
  # If age is "25", returns 25 (numeric)
  # If age is "hello", returns "hello" (character, not convertible)

  # Force numeric conversion (may produce NA):
  val <- sd_values(some_field, as_numeric = TRUE)
  # "123" becomes 123, "hello" becomes NA

  # Never convert to numeric (e.g., ZIP codes with leading zeros):
  zip <- sd_values(zip_code, as_numeric = FALSE)
  # "01234" stays as "01234", not converted to 1234

  # Default behavior (NULL) auto-detects pipe and splits:
  fruits <- sd_values(fav_fruits)
  # If value is "apple|banana|orange", returns c("apple", "banana", "orange")
  # If value is "apple", returns "apple" (no pipe, no split)

  # Force split (even if no pipe present):
  vals <- sd_values(some_field, as_vector = TRUE)

  # Never split (keep as single string):
  raw <- sd_values(fav_fruits, as_vector = FALSE)
  # "apple|banana|orange" stays as "apple|banana|orange"

  # Check number of selections:
  if (length(sd_values(fav_fruits)) > 3) {
    # User selected more than 3 fruits
  }

  # Check if specific option was selected:
  if ("apple" %in% sd_values(fav_fruits)) {
    # User selected apple
  }

  sd_server(db = db)
}

## End(Not run)
```

`sd_version` *Check Surveydown Version*

Description

This function checks if the local surveydown package is up-to-date with the latest online version. It compares the local version with the latest version available on GitHub and provides information about whether an update is needed.

Usage

```
sd_version()
```

Value

No return value, called for side effects (prints version information and update status to the console).

Examples

```
surveydown::sd_version()
```

Index