Package: survey160r (via r-universe)

June 8, 2026

Title R Client for Survey160 Data

Version 0.16.1

Description Access Survey160 campaign data from R. Reads campaign results from Google Cloud Storage, triggers fresh exports via the Survey160 API, and computes per-campaign recipient-latency reports as in-memory R objects. Persistence and fleet orchestration live in consumer projects (see survey160-shiny).

License MIT + file LICENSE

URL https://github.com/survey160/survey160r

BugReports https://github.com/survey160/survey160r/issues

Encoding UTF-8

Depends R (>= 4.1)

Imports googleCloudStorageR (>= 0.7.0), httr, jsonlite, lubridate, digest, dplyr, rlang, data.table

Suggests askpass, testthat (>= 3.0.0), mockery, withr, lintr, covr

Config/testthat/edition 3

RoxygenNote 7.3.3

Config/pak/sysreqs cmake make libuv1-dev libssl-dev

Repository https://survey160.r-universe.dev

Date/Publication 2026-06-08 13:01:47 UTC

RemoteUrl https://github.com/survey160/survey160r

RemoteRef HEAD

RemoteSha 0e29868869789af568427b6d8d9034d0a65236f3

Contents

`campaign_build_config` *Build a latency config from a campaign id and its CSV*

Description

Pure function. Derives `flow.questions` from the CSV column names via `campaign_discover_questions()` and assembles the rest of the config from the named arguments. No I/O, no API call, no auth precondition.

Usage

```
campaign_build_config(
  campaign_id,
  data,
  field_timezone = "UTC",
  project_id = NULL,
  date_filter = NULL,
  respondent_id_column = NULL
)
```

Arguments

`campaign_id`	Campaign id (numeric or character).
`data`	A data frame of CSV results (or a character vector of column names) used to discover the question flow.
`field_timezone`	Tz used to bucket the Parquet `date` and `hour_local` columns. Default `"UTC"`.
`project_id`	Optional Survey160 project id; defaults to the campaign id as a placeholder.
`date_filter`	Optional character/Date vector restricting which survey dates are processed (interpreted in `field_timezone`).

respondent_id_column
: Optional column name used to dedupe rows by respondent. Default NULL (no dedupe).

Value

A config list ready to pass to campaign_report(), which calls campaign_validate_config() before consuming it.

---

campaign_config_hash *Stable hash of a config*

Description

Hashes a canonical form of the config so the same logical config always produces the same hash even across different YAML serializations.

Usage

```
campaign_config_hash(config)
```

Arguments

config	The config list.

Value

A hex sha256 string.

---

campaign_discover_questions *Discover the question flow from CSV column names*

Description

Scans the column names of an in-memory CSV data frame (as returned by read.csv / s160_gcs_pull_csv) for id.<q>.scriptDate columns and returns the question ids in their original column order. Terminal flow states (refusal, ineligible) are dropped so the result is usable directly as config$flow$questions.

Usage

```
campaign_discover_questions(data)
```

Arguments

data	A data frame or character vector of column names.

Details

Survey160 v2 CSV headers are emitted as `id[<q>]scriptDate` on disk; `read.csv` converts the brackets to dots. Both forms are accepted so callers can pass either a data frame or a character vector of raw header tokens.

Value

A character vector of question ids in flow order.

`campaign_report` *Compute a latency report for one campaign*

Description

Pure function: same `(data, config)` always yields identical output. Implements the algorithm in `campaign_scripts.md` §2.

Usage

```
campaign_report(data, config, run_at = NULL)
```

Arguments

`data`	A data frame with one row per respondent and the per-question timestamp columns named `id.<q>.scriptDate` / `id.<q>.batchDate` for each question in `config$flow$questions`, plus the population column `id.intro.finalText` and the campaign id column.
`config`	Config list from `campaign_build_config()` (or a hand-built list with the same shape).
`run_at`	Optional `POSIXct` timestamp to stamp on every row's `run_at_utc` column. `NULL` (default) uses `Sys.time()`. Fleet runners pass a single timestamp here so every campaign in one fleet pass shares the same `run_at_utc`, making "last fleet run" queries trivial.

Value

A list with `consolidated` (one row per (campaign_id, date, hour_local, segment, threshold_min)), `latency_frame` (one row per respondent x segment), `diagnostics` (counts and breakdowns per spec §3.3), and `meta` (algorithm_version, config_hash, run_at_utc).

`campaign_run` *Run the latency report for one campaign*

Description

Analyst-facing one-campaign runner. Given an in-memory campaign CSV (already read by the caller), (optionally) builds the report config from the CSV header and runs the latency algorithm. No I/O; pair with `s160_gcs_pull_csv()` for the GCS source path or `read.csv()` / `readr::read_csv()` / anything else for off-GCS sources.

Usage

```
campaign_run(
  campaign_id,
  data,
  config = NULL,
  run_at = NULL,
  run_by = NULL,
  ...
)
```

Arguments

`campaign_id`	Campaign id (numeric or character).
`data`	In-memory campaign CSV as a data frame (one row per respondent, columns named `id.<q>.scriptDate` / `id.<q>.batchDate` per question plus the population-filter column `id.intro.finalText` and the campaign id column).
`config`	Optional pre-built config. When `NULL` (default), the config is auto-built from `data`'s header; pass any `campaign_build_config()` overrides (`field_timezone`, `project_id`, `date_filter`, `respondent_id_column`) through `....` Mutually exclusive with `....`
`run_at`	Optional `POSIXct` timestamp stamped on every row's `run_at_utc` column. `NULL` (default) uses `Sys.time()`. Fleet runners pass one stamp here so every campaign in a pass shares the same `run_at_utc`.
`run_by`	Optional string stamped on every row's `run_by` provenance column. `NULL` (default) leaves the column as `NA_character_`; callers persisting the result typically fill it at write time.
`...`	Forwarded to `campaign_build_config()` when `config` is `NULL`. Must be empty when `config` is supplied (passing both errors).

Details

Two call shapes:

- Convenience – omit `config` and pass any `campaign_build_config()` overrides through `....` `campaign_run()` derives the config from the CSV header.

- Custom – pre-build the config with `campaign_build_config()` (mutating as needed) and pass it via `config`. `...` must be empty in that case (passing both errors).

Provenance: if `data` carries `source_csv_hash` or `source_csv_path` attributes (set by `s160_gcs_pull_csv` for GCS reads), `campaign_report()` surfaces them on `result$meta`. Analysts pulling CSVs from other sources can attach the attributes themselves before calling, e.g.

```
attr(df, "source_csv_path") <- "dropbox:campaign_1234.csv"
attr(df, "source_csv_hash") <- paste0(
  "sha256:", digest::digest(file = local_path, algo = "sha256"))
```

Value

The list returned by `campaign_report()`: `consolidated`, `latency_frame`, `diagnostics`, `meta` (with `source_csv_hash` and `source_csv_path` from `data`'s attributes, or `NA` when absent).

Examples

```
## Not run:
# GCS source -- pair with s160_gcs_pull_csv().
s160_gcs_init(bucket = "campaign_results")
data   <- s160_gcs_pull_csv(1980)
result <- campaign_run(1980, data, field_timezone = "America/New_York")
result$meta$source_csv_hash

# Off-GCS source -- bring your own CSV.
data   <- read.csv("~/Dropbox/campaign_1980.csv", stringsAsFactors = FALSE)
result <- campaign_run(1980, data)

# Custom config (mutate before running).
config <- campaign_build_config(1980, data, field_timezone = "America/New_York")
config$flow$questions <- c("intro", "q1_custom")
campaign_run(1980, data, config = config)

## End(Not run)
```

---

campaign_validate_config

Validate a latency config against a data frame

Description

Implements the fail-fast checks from spec §2.4. Aborts with a named error on the first failing rule.

Usage

```
campaign_validate_config(config, data)
```

Arguments

config	The config list (typically from campaign_build_config).
data	The data frame the report will run against.

Value

Invisible TRUE on success; otherwise stops with an error.

---

required_csv_columns *CSV columns the latency report reads for a given config*

Description

Returns the (dot-form) column names campaign_report() touches for a given config: the per-question scriptDate/batchDate set, the population-filter columns (extracted from config$filters$population), the campaign-id and optional respondent-id columns, plus the fixed non-flow support columns (id.intro.finalText, web_complete, id.ineligible.scriptDate).

Usage

```
required_csv_columns(config, available = NULL)
```

Arguments

config	A config list from campaign_build_config() (or one with the same shape). Build it from a header-only peek (s160_csv_header()) to avoid reading the file twice.
available	Optional character vector of the actual (dot-form) column names present in the file (e.g. from s160_csv_header()). When supplied, columns matching the report's data-dependent patterns (the close-message Text columns) are retained. Strongly recommended.

Details

Some columns the report reads have data-dependent names – the close-message Text columns (id.close*.scriptText/batchText) that detect_survey_mode() greps to tell t2w_external from sms. Pass available (e.g. the result of s160_csv_header()) so these are matched against the real header and retained; omitting it risks projecting them away and misclassifying a t2w_external campaign as sms.

Pass the result as columns = to s160_read_csv() / s160_gcs_pull_csv() to parse only the columns the algorithm needs – the projection yields output identical to a full read.

Value

A character vector of unique dot-form column names.

Examples

```
## Not run:
header <- s160_csv_header(path)
config <- campaign_build_config(1980, header, field_timezone = "America/New_York")
data   <- s160_read_csv(path, columns = required_csv_columns(config, header))

## End(Not run)
```

`s160_api_auth` *Authenticate to the Survey160 API*

Description

Reads service account credentials (`S160_API_USERID` and `S160_API_KEY`) from `~/.Renviron`. On first interactive run, prompts for both values and saves them automatically.

Usage

```
s160_api_auth(base_url = "https://api.survey160.com")
```

Arguments

`base_url`	API base URL. Defaults to `"https://api.survey160.com"`.

Value

Invisible NULL. Stores JWT as side effect.

Examples

```
## Not run:
s160_api_auth()

## End(Not run)
```

`s160_api_campaign_get` *Read a single campaign's attributes*

Description

Wraps the Survey160 API endpoint `GET /campaigns/<campaign_id>`, which returns every column on the `campaigns` table for one campaign. Useful for confirming attributes after a state-changing call (for example, reading `archive_scheduled_date` after scheduling an archive) without dropping to direct database access.

Usage

```
s160_api_campaign_get(campaign_id)
```

Arguments

campaign_id	Campaign ID (numeric or character).

Details

Enriched, API-only fields returned by the endpoint (listlength, list, login, exports, has_texting_started, sandbox_configuration, aggregator, has_assigned_registration) are dropped; the result mirrors the campaigns table only. JSON-valued columns (script, prompt, quotas, ...) come back as length-1 list-columns holding the parsed structure.

The endpoint runs several server-side subqueries on each call; this is a per-campaign read, not appropriate for tight loops over hundreds of IDs. A batch variant would need a backend extension and is out of scope.

Value

A single-row data frame. Scalar columns are scalar; ISO-8601 timestamp columns are coerced to POSIXct in UTC; JSON columns are list-columns of length 1.

Examples

```
## Not run:
s160_api_auth()
info <- s160_api_campaign_get(2107)
info$active
info$script[[1]]  # parsed JSON

## End(Not run)
```

---

s160_api_campaign_results

Download campaign results via API

Description

Triggers a fresh campaign results export, polls GCS until the file is updated, and returns the results as a data frame. Requires both API auth (s160_api_auth) and GCS auth (s160_gcs_init).

Usage

```
s160_api_campaign_results(
  campaign_id,
  filter_open = FALSE,
  timeout = 300,
```

```
  poll_interval = 5,
  destdir = NULL,
  ...
)
```

Arguments

campaign_id	Campaign ID (numeric or character).
filter_open	Logical. Exclude open/uncontacted conversations? Default FALSE.
timeout	Timeout in seconds for export completion. Default 300.
poll_interval	Maximum polling interval in seconds. Default 5. Polling uses exponential back-off starting at the smaller of 2s and this value, capped at this value.
destdir	Directory to save the downloaded CSV. NULL (default) uses a temporary file.
...	Additional arguments passed to read.csv().

Value

A data frame with one row per survey response.

Examples

```
## Not run:
s160_gcs_init(bucket = "campaign_results")
s160_api_auth()
df <- s160_api_campaign_results(1980)
df <- s160_api_campaign_results(1980, filter_open = TRUE, timeout = 600)

## End(Not run)
```

---

s160_csv_header *Read just the header (column names) of a CSV*

Description

Peeks the first line of a CSV and returns its column names in the same make.names()-munged (dot-form) form the readers produce, without parsing the body. Pair with campaign_build_config() + required_csv_columns() to derive a column-projection set for a large file before reading it:

Usage

```
s160_csv_header(path, encoding = "UTF-8")
```

Arguments

path	Path to the CSV.
encoding	File encoding for the header peek ("UTF-8" default), kept consistent with the body read so a UTF-8/BOM file munges to the same names regardless of reader.

Details

```
header <- s160_csv_header(path)
config <- campaign_build_config(id, header, field_timezone = tz)
data   <- s160_read_csv(path, columns = required_csv_columns(config))
```

Value

Character vector of dot-form column names.

`s160_gcs_campaign_results_files`
List files in a campaign's GCS folder

Description

Returns the file names inside a campaign's folder in the results bucket. Returns `character(0)` with a message if the campaign has no files.

Usage

```
s160_gcs_campaign_results_files(campaign_id, bucket = NULL)
```

Arguments

`campaign_id`	Campaign ID (numeric or character). Must be a single value.
`bucket`	Source GCS bucket. `NULL` (default) falls back to the global bucket set by `s160_gcs_init()`.

Value

Character vector of file names (without the campaign_id prefix).

Examples

```
## Not run:
s160_gcs_init(bucket = "campaign_results")
s160_gcs_campaign_results_files(1980)

## End(Not run)
```

s160_gcs_campaign_results_list

List all campaign IDs in the current bucket

Description

Returns a sorted character vector of campaign IDs (top-level folder names) in the results bucket. Objects at the bucket root (not inside a folder) are excluded.

Usage

```
s160_gcs_campaign_results_list(bucket = NULL)
```

Arguments

bucket	Source GCS bucket. NULL (default) falls back to the global bucket set by s160_gcs_init().

Value

Character vector of campaign IDs, sorted.

Examples

```
## Not run:
s160_gcs_init(bucket = "campaign_results")
s160_gcs_campaign_results_list()

## End(Not run)
```

s160_gcs_campaign_results_read

Read campaign results CSV from GCS into a data frame

Description

Downloads the CSV from GCS and reads it into R. By default, the file is downloaded to a temporary location and cleaned up automatically. Set destdir to keep a local copy.

Usage

```
s160_gcs_campaign_results_read(
  campaign_id,
  filename = NULL,
  destdir = NULL,
  bucket = NULL,
  columns = NULL,
  ...
)
```

Arguments

campaign_id	Campaign ID (numeric or character). Must be a single value.
filename	File name in the campaign folder. Defaults to <campaign_id>_raw_data_download.csv (the standard export filename). Must not contain path separators.
destdir	Directory to save the downloaded file. When NULL (default), a temporary file is used and cleaned up automatically. Use "." for the current directory.
bucket	Source GCS bucket. NULL (default) falls back to the global bucket set by s160_gcs_init().
columns	Optional character vector of (dot-form) column names to keep, e.g. from required_csv_columns(). When set, only those columns are parsed (via data.table::fread's column projection), cutting read time and memory on wide exports. NULL (default) reads every column.
...	Additional arguments forwarded to the CSV reader (data.table::fread, or utils::read.csv when data.table is unavailable), e.g. na.strings, nrows, sep.

Details

GCS path: gs://<bucket>/<campaign_id>/<filename>

Value

A data frame with one row per survey response.

Examples

```
## Not run:
s160_gcs_init(bucket = "campaign_results")
df <- s160_gcs_campaign_results_read(1980)
df <- s160_gcs_campaign_results_read(1980, destdir = ".")
df <- s160_gcs_campaign_results_read(1980, destdir = "~/data")

## End(Not run)
```

---

s160_gcs_campaign_results_status

Check campaign results export status

Description

Returns GCS file metadata for the campaign's export file without triggering a new export. Requires GCS auth (s160_gcs_init).

Usage

```
s160_gcs_campaign_results_status(campaign_id, bucket = NULL)
```

Arguments

campaign_id	Campaign ID (numeric or character).
bucket	Source GCS bucket. NULL (default) falls back to the global bucket set by s160_gcs_init().

Value

Named list with name, updated, and size, or NULL if no export file exists.

Examples

```
## Not run:
s160_gcs_init(bucket = "campaign_results")
s160_gcs_campaign_results_status(1980)

## End(Not run)
```

---

s160_gcs_init *Initialize GCS connection*

Description

Authenticates to GCS using the Survey160 Desktop OAuth client and sets the global bucket.

Usage

```
s160_gcs_init(bucket)
```

Arguments

bucket	GCS bucket name (e.g. "campaign_results").

Details

On first run, prompts for the client secret (get it from your team lead) and saves it to ~/.Renviron. Subsequent runs read it automatically. Also opens a browser for Google sign-in on first use; the OAuth token is cached in a platform-dependent directory (run gargle::gargle_oauth_sitrep() to locate it).

The authenticated Google account needs Storage Object Viewer permission on the target bucket.

Value

Invisible NULL. Sets global bucket as side effect.

Examples

```
## Not run:
s160_gcs_init(bucket = "campaign_results")

## End(Not run)
```

s160_gcs_pull_csv *Read a campaign CSV from GCS, hashing it for provenance*

Description

Thin wrapper over s160_gcs_campaign_results_read that also computes a sha256 of the downloaded CSV bytes. The hash and the canonical gs:// path travel back on the returned data frame as the source_csv_hash and source_csv_path attributes; campaign_report() reads them and copies them onto result$meta so downstream consumers (e.g. persistence layers) don't have to fish them off attributes.

Usage

```
s160_gcs_pull_csv(campaign_id, filename = NULL, bucket = NULL, columns = NULL)
```

Arguments

campaign_id	Campaign id (numeric or character).
filename	Optional override for the CSV filename.
bucket	Source GCS bucket. NULL (default) falls back to the global bucket set by s160_gcs_init(); pass an explicit value to skip the global entirely.
columns	Optional character vector of (dot-form) column names to keep (e.g. from required_csv_columns()). Forwarded to s160_gcs_campaign_results_read() to parse only those columns.

Value

A data frame with attributes source_csv_hash and source_csv_path set.

s160_read_csv *Read a campaign CSV from a local path, hashing it for provenance*

Description

Local-source sibling of s160_gcs_pull_csv(). Reads the CSV via data.table::fread (falling back to utils::read.csv) and stamps source_csv_hash and source_csv_path attributes on the returned data frame so downstream campaign_report() / campaign_run() surface them on result$meta. Use for backfills (archived campaign CSVs stored on disk, Dropbox, S3 mounts, etc.).

Usage

```
s160_read_csv(path, columns = NULL, hash = TRUE, ...)
```

Arguments

path	Path to the CSV. Recorded verbatim on attr(., "source_csv_path").
columns	Optional character vector of (dot-form) column names to keep (e.g. from required_csv_columns()). When set, only those columns are parsed, cutting read time and memory on wide exports. NULL (default) reads every column.
hash	When TRUE (default), compute the sha256 of the file for source_csv_hash. Set FALSE to skip the hashing pass (a full second read of the file) on large backfills where provenance hashing is not needed; source_csv_hash is then NA.
...	Forwarded to the CSV reader (data.table::fread, or utils::read.csv when data.table is unavailable), e.g. na.strings, sep. stringsAsFactors defaults to FALSE.

Value

A data frame with source_csv_hash and source_csv_path attributes set.

Examples

```
## Not run:
data <- s160_read_csv("~/Dropbox/archive/campaign_500.csv")
attr(data, "source_csv_hash")
campaign_run(500, data, field_timezone = "America/New_York")

## End(Not run)
```

Index