Package 'shiny.telemetry'

July 17, 2024

Type Package

Title 'Shiny' App Usage Telemetry

Version 0.3.0

Description Enables instrumentation of 'Shiny' apps for tracking user session events such as input changes, browser type, and session duration. These events can be sent to any of the available storage backends and analyzed using the included 'Shiny' app to gain insights about app usage and adoption.

License LGPL-3

URL https://appsilon.github.io/shiny.telemetry/, https://github.com/Appsilon/shiny.telemetry

BugReports https://github.com/Appsilon/shiny.telemetry/issues

Imports checkmate, digest, dplyr (>= 1.1.0), glue, htmltools, httr2, jsonlite, lifecycle, logger, lubridate, odbc, purrr, R6, rlang, RSQLite, shiny, stringr, tidyr

Suggests box, config, DT, mongolite, plotly, plumber, RColorBrewer, RMariaDB, RPostgres, RPostgreSQL, scales, semantic.dashboard (>= 0.1.1), shiny.semantic (>= 0.2.0), shinyjs, testthat (>= 3.1.7), timevis, withr, knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.1

VignetteBuilder knitr

NeedsCompilation no

Author André Veríssimo [aut, cre],
Kamil Żyła [aut],
Krystian Igras [aut],
Recle Vibal [aut],
Arun Kodati [aut],
Wahaduzzaman Khan [aut],
Appsilon Sp. z o.o. [cph]

Maintainer André Veríssimo <opensource+andre@appsilon.com>

Repository CRAN

Date/Publication 2024-07-17 13:40:02 UTC

Contents

`analytics_app` *Run example telemetry analytics dashboard*

Description

Run example telemetry analytics dashboard

Usage

```
analytics_app(data_storage)
```

Arguments

`data_storage`	data_storage instance that will handle all backend read and writes.

Value

An object that represents the analytics app. Printing the object or passing it to `shiny::runApp()` will run it.

build_id_from_secret *Builds id from a secret that can be used in open communication*

Description

This is used in shiny.telemetry, but also externally with the Plumber endpoint.

Usage

```
build_id_from_secret(secret)
```

Arguments

secret	string that contains information that should not be publicly available

Value

A string with an hash of the secret.

Examples

```
build_id_from_secret("some_random_secret_generated_with_uuid::UUIDgenerate")
```

build_token *Builds hash for a call*

Description

Function that takes creates a signature for the `values` using a secret.

Usage

```
build_token(values, secret = NULL)
```

Arguments

values	R object that is going to be signed
secret	string that contains the shared secret to sign the communication. It can be NULL on both telemetry and in plumber API to disable this communication feature

Details

This is used in shiny.telemetry, but also externally with the Plumber endpoint.

Value

A string that contains an hash to uniquely identify the parameters.

Examples

```
build_token(values = list(list(1, 2, 3), 2, 2, 3, "bb"))
build_token(values = list(list(1, 2, 3), 1, 2, 3, "bb"))
build_token(values = list(list(1, 2, 3), 1, 2, 3, "bb"), secret = "abc")
build_token(values = list(list(1, 2, 3), 1, 2, 3, "bb"), secret = "abd")
```

`DataStorage` *Data Storage abstract class to handle all the read/write operations*

Description

Abstract R6 Class that encapsulates all the operations needed by Shiny.telemetry to read and write. This removes the complexity from the functions and uses a unified API.

Active bindings

`event_bucket` string that identifies the bucket to store user related and action data

Methods

Public methods:

- `DataStorage$new()`
- `DataStorage$insert()`
- `DataStorage$read_event_data()`
- `DataStorage$close()`
- `DataStorage$clone()`

Method `new()`**:** initialize data storage object common with all providers

Usage:

```
DataStorage$new()
```

Method `insert()`**:** Insert new data

Usage:

```
DataStorage$insert(app_name, type, session = NULL, details = NULL, time = NULL)
```

Arguments:

`app_name` string with name of dashboard (the version can be also included in this string)

`type` string that identifies the event type to store

`session` (optional) string that identifies a session where the event was logged

`details` atomic element of list with data to save in storage

`time` date time value indicates the moment the record was generated in UTC. By default it should be NULL and determined automatically, but in cases where it should be defined, use `Sys.time()` or `lubridate::now(tzone = "UTC")` to generate it.

Returns: Nothing. This method is called for side effects.

Method `read_event_data()`**:** read all user data from SQLite.

Usage:

```
DataStorage$read_event_data(date_from = NULL, date_to = NULL, app_name = NULL)
```

Arguments:

`date_from` (optional) date representing the starting day of results.

`date_to` (optional) date representing the last day of results.

`app_name` (optional) string identifying the Dashboard-specific event data

Method `close()`**:** Close the connection if necessary

Usage:

```
DataStorage$close()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataStorage$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`DataStorageLogFile` *Data storage class for* `JSON` *Log File*

Description

Implementation of the `DataStorage` R6 class to a `JSON` log file backend using a unified API for read/write operations

Super class

`shiny.telemetry::DataStorage` -> `DataStorageLogFile`

Active bindings

`event_bucket` string that identifies the file path to store user related and action data

Methods

Public methods:

- `DataStorageLogFile$new()`
- `DataStorageLogFile$clone()`

Method `new()`**:** Initialize the data storage class

Usage:

```
DataStorageLogFile$new(log_file_path)
```

Arguments:

`log_file_path` string with path to `JSON` log file user actions

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataStorageLogFile$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
log_file_path <- tempfile(fileext = ".txt")
data_storage <- DataStorageLogFile$new(log_file_path = log_file_path)

data_storage$insert("example", "test_event", "session1")
data_storage$insert("example", "input", "s1", list(id = "id"))
data_storage$insert("example", "input", "s1", list(id = "id2", value = 32))

data_storage$insert(
  "example", "test_event_3_days_ago", "session1",
  time = lubridate::as_datetime(lubridate::today() - 3)
)

data_storage$read_event_data()
data_storage$read_event_data(Sys.Date() - 1, Sys.Date() + 1)

file.remove(log_file_path)
```

DataStorageMariaDB *Data storage class with MariaDB / MySQL provider*

Description

Implementation of the `DataStorage` R6 class to MariaDB backend using a unified API for read/write operations

Super classes

`shiny.telemetry::DataStorage` -> `shiny.telemetry::DataStorageSQLFamily` -> `DataStorageMariaDB`

Methods

Public methods:

- `DataStorageMariaDB$new()`
- `DataStorageMariaDB$clone()`

Method `new()`**:** Initialize the data storage class

Usage:

```
DataStorageMariaDB$new(
  username = NULL,
  password = NULL,
  hostname = "127.0.0.1",
  port = 3306,
  dbname = "shiny_telemetry"
)
```

Arguments:

`username` string with a MariaDB username.

`password` string with the password for the username.

`hostname` string with hostname of MariaDB instance.

`port` numeric value with the port number of MariaDB instance.

`dbname` string with the name of the database in the MariaDB instance.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataStorageMariaDB$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
data_storage <- DataStorageMariaDB$new(user = "mariadb", password = "mysecretpassword")

data_storage$insert("example", "test_event", "session1")
data_storage$insert("example", "input", "s1", list(id = "id1"))
data_storage$insert("example", "input", "s1", list(id = "id2", value = 32))

data_storage$insert(
  "example", "test_event_3_days_ago", "session1",
  time = lubridate::as_datetime(lubridate::today() - 3)
)

data_storage$read_event_data()
data_storage$read_event_data(Sys.Date() - 1, Sys.Date() + 1)
data_storage$close()

## End(Not run)
```

`DataStorageMongoDB` *Data storage class with MongoDB provider*

Description

Implementation of the `DataStorage` R6 class to MongoDB backend using a unified API for read/write operations

Super class

shiny.telemetry::DataStorage -> DataStorageMongoDB

Methods

Public methods:

- DataStorageMongoDB$new()
- DataStorageMongoDB$clone()

Method `new()`**:** Initialize the data storage class

Usage:

```
DataStorageMongoDB$new(
  hostname = "localhost",
  port = 27017,
  username = NULL,
  password = NULL,
  authdb = NULL,
  dbname = "shiny_telemetry",
  options = NULL,
  ssl_options = mongolite::ssl_options()
)
```

Arguments:

`hostname` the hostname or IP address of the MongoDB server.

`port` the port number of the MongoDB server (default is 27017).

`username` the username for authentication (optional).

`password` the password for authentication (optional).

`authdb` the default authentication database (optional).

`dbname` name of database (default is "shiny_telemetry").

`options` Additional connection options in a named list format (e.g., list(ssl = "true", replicaSet = "myreplicaset")) (optional).

`ssl_options` additional connection options such as SSL keys/certs (default is `mongolite::ssl_options()`).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataStorageMongoDB$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
data_storage <- DataStorageMongoDB$new(
  host = "localhost",
  db = "test",
  ssl_options = mongolite::ssl_options()
)
```

```
data_storage$insert("example", "test_event", "session1")
data_storage$insert("example", "input", "s1", list(id = "id1"))
data_storage$insert("example", "input", "s1", list(id = "id2", value = 32))

data_storage$insert(
  "example", "test_event_3_days_ago", "session1",
  time = lubridate::as_datetime(lubridate::today() - 3)
)

data_storage$read_event_data()
data_storage$read_event_data(Sys.Date() - 1, Sys.Date() + 1)
data_storage$close()

## End(Not run)
```

DataStorageMSSQLServer

Data storage class with MS SQL Server provider

Description

Implementation of the DataStorage R6 class to MS SQL Server backend using a unified API for read/write operations. This provider requires a configured and named ODBC driver to be set up on your system, for example, "ODBC Driver 17 for SQL Server" or "ODBC Driver 18 for SQL Server".

Note that MS SQL Server support requires a subtly different database schema: the time field is stored as a DATETIME rather than a TIMESTAMP.

Super classes

shiny.telemetry::DataStorage -> shiny.telemetry::DataStorageSQLFamily -> DataStorageMSSQLServer

Methods

Public methods:

- DataStorageMSSQLServer$new()
- DataStorageMSSQLServer$clone()

Method new(): Initialize the data storage class

Usage:

```
DataStorageMSSQLServer$new(
  username = NULL,
  password = NULL,
  hostname = "127.0.0.1",
  port = 1433,
  dbname = "shiny_telemetry",
  driver = "ODBC Driver 17 for SQL Server",
  trust_server_certificate = "NO"
)
```

Arguments:

`username` string with a MS SQL Server username.

`password` string with the password for the username.

`hostname` string with hostname of the MS SQL Server instance.

`port` numeric value with the port number of MS SQL Server instance.

`dbname` string with the name of the database in the MS SQL Server instance.

`driver` string with the name of the `ODBC` driver class for MS SQL, for example "`ODBC` Driver 17 for SQL Server".

`trust_server_certificate` string with "NO" or "YES", setting whether or not to trust the server's certificate implicitly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataStorageMSSQLServer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
data_storage <- DataStorageMSSQLServer$new(
  user = "sa",
  password = "my-Secr3t_Password",
  hostname = "localhost",
  port = 1433,
  dbname = "shiny_telemetry",
  driver = "ODBC Driver 18 for SQL Server",
  trust_server_certificate = "YES"
)

data_storage$insert("example", "test_event", "session1")
data_storage$insert("example", "input", "s1", list(id = "id1"))
data_storage$insert("example", "input", "s1", list(id = "id2", value = 32))

data_storage$insert(
  "example", "test_event_3_days_ago", "session1",
  time = lubridate::as_datetime(lubridate::today() - 3)
)

data_storage$read_event_data()
data_storage$read_event_data(Sys.Date() - 1, Sys.Date() + 1)
data_storage$close()

## End(Not run)
```

DataStoragePlumber *Data storage class with SQLite provider*

Description

Implementation of the DataStorage R6 class to SQLite backend using a unified API for read/write operations

Super class

shiny.telemetry::DataStorage -> DataStoragePlumber

Active bindings

event_read_endpoint string field that returns read action endpoint

event_insert_endpoint string field that returns insert action endpoint

Methods

Public methods:

- DataStoragePlumber$new()
- DataStoragePlumber$clone()

Method new(): Initialize the data storage class

Usage:

```
DataStoragePlumber$new(
  hostname = "127.0.0.1",
  port = 80,
  protocol = "http",
  path = NULL,
  secret = NULL,
  authorization = NULL
)
```

Arguments:

hostname string with hostname of plumber instance,

port numeric value with port number of plumber instance.

protocol string with protocol of the connection of the plumber instance.

path string with sub-path of plumber deployment.

secret string with secret to sign communication with plumber (can be NULL for disabling communication signing).

authorization string to use in HTTP headers for authorization (for example: to authenticate to a plumber deployment behind a connect server).

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
DataStoragePlumber$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## Not run:
# Make sure the PLUMBER_SECRET environment variable is valid before
# running these examples (NULL or a valid secret)

data_storage <- DataStoragePlumber$new(
  hostname = "connect.appsilon.com",
  path = "shiny_telemetry_plumber",
  port = 443,
  protocol = "https",
  authorization = Sys.getenv("CONNECT_AUTHORIZATION_KEY"),
  secret = Sys.getenv("PLUMBER_SECRET")
)

data_storage <- DataStoragePlumber$new(
  hostname = "127.0.0.1",
  path = NULL,
  port = 8087,
  protocol = "http",
  secret = Sys.getenv("PLUMBER_SECRET")
)

data_storage$insert("example", "test_event", "session1")
data_storage$insert("example", "input", "s1", list(id = "id"))
data_storage$insert("example", "input", "s1", list(id = "id2", value = 32))

data_storage$insert(
  "example", "test_event_3_days_ago", "session1",
  time = lubridate::as_datetime(lubridate::today() - 3)
)

data_storage$read_event_data()
data_storage$read_event_data(Sys.Date() - 1, Sys.Date() + 1)

## End(Not run)
```

DataStoragePostgreSQL *Data storage class with PostgreSQL provider*

Description

Implementation of the DataStorage R6 class to PostgreSQL backend using a unified API for read/write operations

Super classes

shiny.telemetry::DataStorage -> shiny.telemetry::DataStorageSQLFamily -> DataStoragePostgreSQL

Methods

Public methods:

- DataStoragePostgreSQL$new()
- DataStoragePostgreSQL$clone()

Method `new()`**:** Initialize the data storage class

Usage:

```
DataStoragePostgreSQL$new(
  username = NULL,
  password = NULL,
  hostname = "127.0.0.1",
  port = 5432,
  dbname = "shiny_telemetry",
  driver = "RPostgreSQL"
)
```

Arguments:

`username` string with a PostgreSQL username.

`password` string with the password for the username.

`hostname` string with hostname of PostgreSQL instance.

`port` numeric value with the port number of PostgreSQL instance.

`dbname` string with the name of the database in the PostgreSQL instance.

`driver` string, to select PostgreSQL driver among `c("RPostgreSQL", "RPostgres")`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataStoragePostgreSQL$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
data_storage <- DataStoragePostgreSQL$new(user = "postgres", password = "mysecretpassword")

data_storage$insert("example", "test_event", "session1")
data_storage$insert("example", "input", "s1", list(id = "id1"))
data_storage$insert("example", "input", "s1", list(id = "id2", value = 32))

data_storage$insert(
  "example", "test_event_3_days_ago", "session1",
  time = lubridate::as_datetime(lubridate::today() - 3)
)
```

```
data_storage$read_event_data()
data_storage$read_event_data(Sys.Date() - 1, Sys.Date() + 1)
data_storage$close()

## End(Not run)
```

`DataStorageSQLFamily` *Data storage abstract class for SQL providers*

Description

Abstract subclass of the `DataStorage` R6 class for the SQL family of providers

Super class

`shiny.telemetry::DataStorage` -> `DataStorageSQLFamily`

Methods

Public methods:

- `DataStorageSQLFamily$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataStorageSQLFamily$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`DataStorageSQLite` *Data storage class with SQLite provider*

Description

Implementation of the `DataStorage` R6 class to SQLite backend using a unified API for read/write operations

Super classes

`shiny.telemetry::DataStorage` -> `shiny.telemetry::DataStorageSQLFamily` -> `DataStorageSQLite`

Methods

Public methods:

- DataStorageSQLite$new()
- DataStorageSQLite$clone()

Method `new()`**:** Initialize the data storage class

Usage:

```
DataStorageSQLite$new(db_path = "user_stats.sqlite")
```

Arguments:

`db_path` string with path to `SQLite` file.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataStorageSQLite$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
db_path <- tempfile(fileext = ".sqlite")
data_storage <- DataStorageSQLite$new(db_path = db_path)

data_storage$insert("example", "test_event", "session1")
data_storage$insert("example", "input", "s1", list(id = "id1"))
data_storage$insert("example", "input", "s1", list(id = "id2", value = 32))

data_storage$insert(
  "example", "test_event_3_days_ago", "session1",
  time = lubridate::as_datetime(lubridate::today() - 3)
)

data_storage$read_event_data()
data_storage$read_event_data(Sys.Date() - 1, Sys.Date() + 1)

file.remove(db_path)
```

---

date_from_null *Common date_from to recognize as NULL*

Description

Common date_from to recognize as NULL

Usage

```
date_from_null
```

Format

An object of class `character` of length 1.

`date_to_null` *Common date_to to recognize as NULL*

Description

Common date_to to recognize as NULL

Usage

```
date_to_null
```

Format

An object of class `character` of length 1.

`Telemetry` *Telemetry class to manage analytics gathering at a global level*

Description

An instance of this class will define metadata and data storage provider for gathering telemetry analytics of a Shiny dashboard.

The `name` and `version` parameters will describe the dashboard name and version to track using analytics, allowing to store the analytics data from multiple dashboards in the same data storage provider. As well as discriminate different versions of the dashboard.

The default data storage provider uses a local SQLite database, but this can be customizable when instantiating the class, by using another one of the supported providers (see `DataStorage`).

Debugging

Events are logged at the `DEBUG` level using the `logger` package. To see the logs, you can set:

```
logger::log_threshold("DEBUG", namespace = "shiny.telemetry")
```

Active bindings

`data_storage` instance of a class that inherits from `DataStorage`. See the documentation on that class for more information.

`app_name` string with name of dashboard

Methods

Public methods:

- Telemetry$new()
- Telemetry$start_session()
- Telemetry$log_navigation()
- Telemetry$log_navigation_manual()
- Telemetry$log_login()
- Telemetry$log_logout()
- Telemetry$log_click()
- Telemetry$log_browser_version()
- Telemetry$log_button()
- Telemetry$log_all_inputs()
- Telemetry$log_input()
- Telemetry$log_input_manual()
- Telemetry$log_custom_event()
- Telemetry$log_error()
- Telemetry$clone()

Method `new()`**:** Constructor that initializes Telemetry instance with parameters.

Usage:

```
Telemetry$new(
  app_name = "(dashboard)",
 data_storage = DataStorageSQLite$new(db_path = file.path("telemetry.sqlite"))
)
```

Arguments:

`app_name` (optional) string that identifies the name of the dashboard. By default it will store data with `(dashboard)`.

`data_storage` (optional) `DataStorage` instance where telemetry data is being stored. It can take any of data storage providers by this package, By default it will store in a SQLite local database in the current working directory with filename `telemetry.sqlite`

`version` (optional) string that identifies the version of the dashboard. By default it will use `v0.0.0`.

Method `start_session()`**:** Setup basic telemetry

Usage:

```
Telemetry$start_session(
  track_inputs = TRUE,
  track_values = FALSE,
  login = TRUE,
  logout = TRUE,
  browser_version = TRUE,
  navigation_input_id = NULL,
  session = shiny::getDefaultReactiveDomain(),
  username = NULL,
```

```
  track_anonymous_user = TRUE,
  track_errors = TRUE
)
```

Arguments:

`track_inputs` flag that indicates if the basic telemetry should track the inputs that change value. `TRUE` by default

`track_values` flag that indicates if the basic telemetry should track the values of the inputs that are changing. `FALSE` by default. This parameter is ignored if `track_inputs` is `FALSE`

`login` flag that indicates if the basic telemetry should track when a session starts. `TRUE` by default.

`logout` flag that indicates if the basic telemetry should track when the session ends. `TRUE` by default.

`browser_version` flag that indicates that the browser version should be tracked.`TRUE` by default.

`navigation_input_id` string or vector of strings that represent input ids and which value should be tracked as navigation events. i.e. a change in the value represent a navigation to a page or tab. By default, no navigation is tracked.

`session` `ShinySession` object or NULL to identify the current Shiny session.

`username` Character with username. If set, it will overwrite username from session object.

`track_anonymous_user` flag that indicates to track anonymous user. A cookie is used to track same user without login over multiple sessions, This is only activated if none of the automatic methods produce a username and when a username is not explicitly defined.`TRUE` by default.

`track_errors` flag that indicates if the basic telemetry should track the errors. `TRUE` by default. if using shiny version < `1.8.1`, it can auto log errors only in UI output functions. By using latest versions of shiny, it can auto log all types of errors.

Returns: Nothing. This method is called for side effects.

Method `log_navigation()`**:** Log an input change as a navigation event

Usage:

```
Telemetry$log_navigation(input_id, session = shiny::getDefaultReactiveDomain())
```

Arguments:

`input_id` string that identifies the generic input in the Shiny application so that the function can track and log changes to it.

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_navigation_manual()`**:** Log a navigation event manually by indicating the id (as input id)

Usage:

```
Telemetry$log_navigation_manual(
  navigation_id,
  value,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`navigation_id` string that identifies navigation event.

`value` string that indicates a value for the navigation

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_login()`**:** Log when session starts

Usage:

```
Telemetry$log_login(
  username = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`username` string with username from current session

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_logout()`**:** Log when session ends

Usage:

```
Telemetry$log_logout(
  username = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`username` string with username from current session

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_click()`**:** Log an action click

Usage:

```
Telemetry$log_click(id, session = shiny::getDefaultReactiveDomain())
```

Arguments:

`id` string that identifies a manual click to the dashboard.

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_browser_version()`**:** Log the browser version

Usage:

```
Telemetry$log_browser_version(session = shiny::getDefaultReactiveDomain())
```

Arguments:

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_button()`**:** Track a button and track changes to this input (without storing the values)

Usage:

```
Telemetry$log_button(
  input_id,
  track_value = FALSE,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`input_id` string that identifies the button in the Shiny application so that the function can track and log changes to it.

`track_value` flag that indicates if the basic telemetry should track the value of the input that are changing. `FALSE` by default.

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_all_inputs()`**:** Automatic tracking of all input changes in the App. Depending on the parameters, it may only track a subset of inputs by excluding patterns or by including specific vector of `input_ids`.

Usage:

```
Telemetry$log_all_inputs(
  track_values = FALSE,
  excluded_inputs = c("browser_version"),
  excluded_inputs_regex = NULL,
  include_input_ids = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`track_values` flag that indicates if the basic telemetry should track the values of the inputs that are changing. `FALSE` by default. This parameter is ignored if `track_inputs` is `FALSE`.

`excluded_inputs` vector of input_ids that should not be tracked. By default it doesn't track browser version, which is added by this package.

`excluded_inputs_regex` vector of input_ids that should not be tracked. All Special characters will be escaped.

`include_input_ids` vector of input_ids that will be tracked. This input_ids should be an exact match and will be given priority over exclude list.

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_input()`**:** Track changes of a specific input id.

Usage:

```
Telemetry$log_input(
  input_id,
  track_value = FALSE,
  matching_values = NULL,
```

```
  input_type = "text",
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`input_id` string (or vector of strings) that identifies the generic input in the Shiny application so that the function can track and log changes to it.
When the `input_id` is a vector of strings, the function will behave just as calling `log_input` one by one with the same arguments.

`track_value` flag that indicates if the basic telemetry should track the value of the input that are changing. `FALSE` by default.

`matching_values` An object specified possible values to register.

`input_type` `"text"` to registered bare input value, `"json"` to parse value from `JSON` format.

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for its side effects.

Method `log_input_manual()`**:** Log a manual input value.

This can be called in telemetry and is also used as a layer between log_input family of functions and actual log event. It creates the correct payload to log the event internally.

Usage:

```
Telemetry$log_input_manual(
  input_id,
  value = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`input_id` string that identifies the generic input in the Shiny application so that the function can track and log changes to it.

`value` (optional) scalar value or list with the value to register.

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_custom_event()`**:** Log a manual event

Usage:

```
Telemetry$log_custom_event(
  event_type,
  details = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`event_type` string that identifies the event type

`details` (optional) scalar value or list with the value to register.

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `log_error()`**:** Log an error event

Usage:

```
Telemetry$log_error(
  output_id,
  message,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`output_id` string that refers to the output element where the error occurred.

`message` string that describes the error.

`session` `ShinySession` object or NULL to identify the current Shiny session.

Returns: Nothing. This method is called for side effects.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Telemetry$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`DataStorage` which this function wraps.

Examples

```
log_file_path <- tempfile(fileext = ".txt")
telemetry <- Telemetry$new(
  data_storage = DataStorageLogFile$new(log_file_path = log_file_path)
) # 1. Initialize telemetry with default options

#
# Use in a shiny application

if (interactive()) {
  library(shiny)

  shinyApp(
    ui = fluidPage(
      use_telemetry(), # 2. Add necessary javascript to Shiny
      numericInput("n", "n", 1),
      plotOutput('plot')
    ),
    server = function(input, output) {
      telemetry$start_session() # 3. Minimal setup to track events
      output$plot <- renderPlot({ hist(runif(input$n)) })
    }
  )
}
```

```
#
# Manual logging of Telemetry that can be used inside Shiny Application
# to further customize the events to be tracked.

session <- shiny::MockShinySession$new() # Create dummy session (only for example purposes)
class(session) <- c(class(session), "ShinySession")

telemetry$log_click("a_button", session = session)

telemetry$log_error("global", message = "An error has occured")

telemetry$log_custom_event("a_button", list(value = 2023), session = session)
telemetry$log_custom_event("a_button", list(custom_field = 23), session = session)

# Manual call login with custom username
telemetry$log_login("ben", session = session)

# Read all data
telemetry$data_storage$read_event_data()

file.remove(log_file_path)

#
# Using SQLite

db_path <- tempfile(fileext = ".sqlite")
telemetry_sqlite <- Telemetry$new(
  data_storage = DataStorageSQLite$new(db_path = db_path)
)

telemetry_sqlite$log_custom_event("a_button", list(value = 2023), session = session)
telemetry_sqlite$log_custom_event("a_button", list(custom_field = 23), session = session)

# Read all data from time range
telemetry_sqlite$data_storage$read_event_data("2020-01-01", "2055-01-01")

file.remove(db_path)
```

`use_telemetry` *Function that adds telemetry HTML elements to UI*

Description

Function that adds telemetry HTML elements to UI

Usage

```
use_telemetry(id = "")
```

Arguments

`id`	(optional) string with id representing the namespace

Value

A `shiny.tag` object to be included in the UI of a Shiny app.

Index