Package: dettl (via r-universe)

October 18, 2024

Title Data Extract, Transform, Test and Load

Version 0.1.0

Description Data extract, transform, test and load tool for sanitising your workflow.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

Depends R (>= 3.5.0)

Imports bit64, DBI, gert, R6, testthat, yaml, withr, writexl

Suggests docopt, knitr, mockery, readxl, rmarkdown, RPostgres, RSQLite, vaultr (>= 0.2.2)

VignetteBuilder knitr

Language en-GB

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

RemoteUrl https://github.com/vimc/dettl

RemoteRef master

RemoteSha 79918d4273ac0bb71aa6bf37b564332757e5603a

Contents

`dettl` *Create an import object using functions defined at specified path*

Description

Create an import object using functions defined at specified path

Usage

```
dettl(path, db_name = NULL)
```

Arguments

`path`	Path to directory containing functions for import.
`db_name`	The name of the db to connect to. Connection info must be configured via the `dettl_config.yml`. If name is left blank this will default to using the first db configured.

Value

An Import object.

`dettl_auto_load` *Automatic load function for append mode imports.*

Description

The automatic load function loops over the transformed data and appends each data frame to the matching table in the database. If the appended table contains a key referenced by one of the foreign key constraints then when the data is inserted into the database this returns the value of the key for the new rows. Then loop over all tables in which this is used as a foreign key and update the previous values to use the returned actual values for the referenced key.

Usage

```
dettl_auto_load(transformed_data, con)
```

Arguments

`transformed_data`	The list of transformed data frames to append to tables in the database.
`con`	Connection to the database to add data to.

Details

Expect that this should only be called from within a custom load function if we want to load data to the database in the automatic way but have some special edge cases which we need to add some custom handling for before or after running the automatic load.

Examples

```
path <- dettl:::prepare_test_import(
  system.file("examples", "person_information", package = "dettl"),
  system.file("examples", "dettl_config.yml", package = "dettl")
)
import <- dettl::dettl(file.path(path, "person_information"), "test")
con <- import$get_connection()
data <- list("people" = data.frame(
  name = c("Alice", "Bob"),
  age = c(25, 43),
  height = c(175, 187),
  stringsAsFactors = FALSE))
dettl_auto_load(data, con)
```

`dettl_create_log_table`
Initialise the database by creating log table if it doesn't already exist

---

Description

Initialise the database by creating log table if it doesn't already exist

Usage

```
dettl_create_log_table(path, db_name)
```

Arguments

`path`	Path to import directory containing db connection configuration.
`db_name`	The name of the db to connect to. Connection info must be configured via the `dettl_config.yml`.

Examples

```
path <- dettl:::prepare_test_import(
  system.file("examples", "person_information", package = "dettl"),
  system.file("examples", "dettl_config.yml", package = "dettl"),
  add_log_table = FALSE
)
dettl::dettl_create_log_table(file.path(path, "person_information"), "test")
```

dettl_new *Create new directory and templated code for new dettl process.*

Description

Create new directory and templated code for new dettl process.

Usage

```
dettl_new(name)
```

Arguments

name	The name of the project directory to be created. Should be human readable and meaningful. Any non a-z,0-9,_ characters will be stripped and replaced with _s. Directory name will be prepended with created date.

Examples

```
t <- tempfile()
dir.create(t)
withr::with_dir(t, {
  dettl::dettl_new("test import")
})
```

dettl_run *Run specified stages of an import*

Description

Run specified stages of an import

Usage

```
dettl_run(
  import,
  db_name = NULL,
  comment = NULL,
  dry_run = FALSE,
  allow_dirty_git = FALSE,
  stage = c("extract", "transform"),
  ...
)
```

Arguments

`import`	Path to import directory.
`db_name`	The name of the db to connect to. Connection info must be configured via the ‘dettl_config.yml‘. If name is left blank this will default to using the first db configured.
`comment`	Optional comment to be written to db log table when import is run.
`dry_run`	If TRUE then any changes to the database will be rolled back.
`allow_dirty_git`	If TRUE then skips check that the import is up to date with remote git repo.
`stage`	The stage or stages of the import to be run.
`...`	Additional args passed to run_import for a specific import type see `RImport$run_import()`

Value

The import object

Examples

```
path <- dettl:::prepare_test_import(
  system.file("examples", "person_information", package = "dettl"),
  system.file("examples", "dettl_config.yml", package = "dettl")
)
dettl::dettl_run(file.path(path, "person_information/"), "test",
  comment = "Example import")
dettl::dettl_run(file.path(path, "person_information/"), "test",
  comment = "Example import",
  save = tempfile())
import <- dettl::dettl_run(file.path(path, "person_information/"),
  "test", stage = "extract")
dettl::dettl_run(file.path(path, "person_information/"), "test",
  stage = c("extract", "transform", "load"),
  comment = "Example import")
```

`dettl_save` *Save data*

Description

Saves any extracted and/or transformed data as separate sheet of an xlsx file.

Usage

```
dettl_save(import, file, stage)
```

Arguments

`import`	The import object to save the data for.
`file`	File path at which to save the data.
`stage`	The stage or stages to save. 'extract' and/or 'transform'

Examples

```
path <- dettl:::prepare_test_import(
  system.file("examples", "person_information", package = "dettl"),
  system.file("examples", "dettl_config.yml", package = "dettl")
)
import <- dettl::dettl(file.path(path, "person_information"), "test")
import$extract()
import$transform()
t <- tempfile()
dettl::dettl_save(import, t, "extract")
t2 <- tempfile()
dettl::dettl_save(import, t2, "transform")
t3 <- tempfile()
dettl::dettl_save(import, t3, c("extract", "transform"))
```

`prepare_test_import` *Prepare example import inside a git repo*

Description

Copies an example import to a new temp directory, sets up git for the directory and creates a test SQLite DB in the temp directory as test.sqlite.

Usage

```
prepare_test_import(
  example_dir = "example",
  dettl_config = "dettl_config.yml",
  create_db = TRUE,
  add_data = FALSE,
  add_job_table = FALSE,
  add_log_table = TRUE,
  add_fk_data = FALSE,
  add_cyclic_fks = FALSE
)
```

Arguments

`example_dir`	The example directory to copy to temp.
`dettl_config`	Path to the dettl config file.
`create_db`	If TRUE then test SQLite db will be created

`add_data`	If TRUE data is bootstrapped to people table in test DB.
`add_job_table`	If TRUE also bootstrap job table related to people table.
`add_log_table`	If TRUE then also bootstrap log table.
`add_fk_data`	If TRUE then bootstrap three tables with foreign key
`add_cyclic_fks`	If TRUE then bootstrap two tables with cyclic foreign key constraints. constraints for testing automatic reading of foreign key constraints from db.

Details

This should only be called from a test, vignette or roxygen example.

Examples

```
dettl:::prepare_test_import(
  system.file("examples", "person_information", package = "dettl"),
  system.file("examples", "dettl_config.yml", package = "dettl")
)
```

`RImport` *Manage R based data import.*

Description

Manage R based data import.

Manage R based data import.

Details

This object should not be initialised directly. Use `dettl` to create the object.

Import can be run by working with import object returned by `dettl` or by running top-level functions. Run the import by working with this object if you want to step through the import process stage by stage and inspect the data after each stage.

Super class

`dettl::Import` -> `RImport`

Methods

Public methods:

- `RImport$reload()`
- `RImport$read_config()`
- `RImport$get_extracted_data()`
- `RImport$get_transformed_data()`
- `RImport$extract()`

- `RImport$transform()`
- `RImport$pre_modify_checks()`
- `RImport$run_import()`

Method `reload()`**:** Reload the objects sources to refresh source code or repair a broken Postgres connection.

Usage:

```
RImport$reload()
```

Method `read_config()`**:** Read and parse config from path.

Usage:

```
RImport$read_config()
```

Method `get_extracted_data()`**:** Get the extracted data created by the extract step

Usage:

```
RImport$get_extracted_data()
```

Returns: The extracted data

Method `get_transformed_data()`**:** Get the transformed data created by the transform step

Usage:

```
RImport$get_transformed_data()
```

Returns: The transformed data

Method `extract()`**:** Run the extract stage of the data import

Usage:

```
RImport$extract()
```

Method `transform()`**:** Run the transform stage of the data import

Usage:

```
RImport$transform()
```

Method `pre_modify_checks()`**:** Run suite of checks to verify that db can be modified

Usage:

```
RImport$pre_modify_checks(dry_run, allow_dirty_git)
```

Arguments:

`dry_run` Whether to run in dry run mode. If TRUE then any database changes will be rolled back. Defaults to FALSE.

`allow_dirty_git` If TRUE then skips check that the import is up to date with remote git repo. FALSE by default.

Method `run_import()`**:** Run multiple stages of the data import

Usage:

```
RImport$run_import(
  comment = NULL,
  dry_run = FALSE,
  allow_dirty_git = FALSE,
  stage = c("extract", "transform"),
  save = FALSE
)
```

Arguments:

`comment` Optional comment to be written to db log table when import is run.

`dry_run` If TRUE then any changes to the database will be rolled back.

`allow_dirty_git` If TRUE then skips check that the import is up to date

`stage` The stage or stages of the import to be run.

`save` Path and name to save data from each stage at, if TRUE then will save to a tempfile.

Examples

```
path <- dettl:::prepare_test_import(
  system.file("examples", "person_information", package = "dettl"),
  system.file("examples", "dettl_config.yml", package = "dettl"))
import_path <- file.path(path, "person_information")

import <- dettl::dettl(import_path, db_name = "test")
import$extract()
import$transform()
import$load()
```

`SqlImport` *Manage SQL based data import.*

Description

Manage SQL based data import.

Manage SQL based data import.

Details

This object should not be initialised directly. Use `dettl` to create the object.

Import can be run by working with import object returned by `dettl` or by running top-level functions. Run the import by working with this object if you want to step through the import process stage by stage and inspect the data after each stage.

Super class

`dettl::Import` -> `SqlImport`

Methods

Public methods:

- `SqlImport$reload()`
- `SqlImport$read_config()`

Method `reload()`**:** Reload the objects sources to refresh source code or repair a broken Postgres connection.

Usage:

```
SqlImport$reload()
```

Method `read_config()`**:** Read and parse config from path.

Usage:

```
SqlImport$read_config()
```

Examples

```
path <- dettl:::prepare_test_import(
  system.file("examples", "sql_example", package = "dettl"),
  system.file("examples", "dettl_config.yml", package = "dettl"))
import_path <- file.path(path, "sql_example")

import <- dettl::dettl(import_path, db_name = "test")
import$run_import(stage = c("extract", "transform", "load"))
```

Index