Package: RITCH (via r-universe)

August 25, 2024

Type Package

Title R Parser for the ITCH-Protocol

Version 0.1.27

Description Allows to efficiently parse, filter, and write binary ITCH
Files (Version 5.0) containing detailed financial transactions
as distributed by NASDAQ to an R data.table.

License MIT + file LICENSE

URL <https://davzim.github.io/RITCH/>, <https://github.com/DavZim/RITCH>

BugReports <https://github.com/DavZim/RITCH/issues>

Depends R (>= 3.5.0)

Imports data.table, Rcpp (>= 0.12.12), nanotime (>= 0.3.2), bit64 (>=
4.0.5)

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 7.2.3

Suggests tinytest

Roxygen list(markdown = TRUE)

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

RemoteUrl https://github.com/davzim/ritch

RemoteRef HEAD

RemoteSha 9bd51af48d26703bd95ab4f0db6532a497c104c1

Contents

---

add_meta_to_filename   *Adds meta information (date and exchange) to an itch filename*

Description

Note that if date and exchange information are already present, they are overwritten

Usage

```
add_meta_to_filename(file, date, exchange)
```

Arguments

file	the filename
date	the date as a date-class or as a string that is understood by base::as.Date().
exchange	the name of the exchange

Value

the filename with exchanged or added date and exchange information

Examples

```
add_meta_to_filename("03302017.NASDAQ_ITCH50", "2010-12-24", "TEST")
add_meta_to_filename("20170130.BX_ITCH_50.gz", "2010-12-24", "TEST")
add_meta_to_filename("S030220-v50-bx.txt.gz", "2010-12-24", "TEST")
add_meta_to_filename("unknown_file.ITCH_50", "2010-12-24", "TEST")
```

---

count_functions	*Counts the messages of an ITCH-file*

Description

Counts the messages of an ITCH-file

Usage

```
count_messages(
  file,
  add_meta_data = FALSE,
  buffer_size = -1,
  quiet = FALSE,
  force_gunzip = FALSE,
  gz_dir = tempdir(),
  force_cleanup = TRUE
)

count_orders(x)

count_trades(x)

count_modifications(x)

count_system_events(x)

count_stock_directory(x)

count_trading_status(x)

count_reg_sho(x)

count_market_participant_states(x)

count_mwcb(x)

count_ipo(x)

count_luld(x)

count_noii(x)

count_rpii(x)
```

Arguments

file	the path to the input file, either a gz-file or a plain-text file

add_meta_data	if the meta-data of the messages should be added, defaults to FALSE
buffer_size	the size of the buffer in bytes, defaults to 1e8 (100 MB), if you have a large amount of RAM, 1e9 (1GB) might be faster
quiet	if TRUE, the status messages are supressed, defaults to FALSE
force_gunzip	only applies if file is a gz-file and a file with the same (gunzipped) name already exists. if set to TRUE, the existing file is overwritten. Default value is FALSE
gz_dir	a directory where the gz archive is extracted to. Only applies if file is a gz archive. Default is [tempdir()]().
force_cleanup	only applies if file is a gz-file. If force_cleanup=TRUE, the gunzipped raw file will be deleted afterwards.
x	a file or a data.table containing the message types and the counts, as outputted by count_messages

Details

- count_orders: Counts order messages. Message type A and F

- count_trades: Counts trade messages. Message type P, Q and B

- count_modifications: Counts order modification messages. Message type E, C, X, D, and U

- count_system_events: Counts system event messages. Message type S

- count_stock_directory: Counts stock trading messages. Message type R

- count_trading_status: Counts trading status messages. Message type H and h

- count_reg_sho: Counts messages regarding reg SHO. Message type Y

- count_market_participant_states: Counts messages regarding the status of market participants. Message type L

- count_mwcb: Counts messages regarding Market-Wide-Circuit-Breakers (MWCB). Message type V and W

- count_ipo: Counts messages regarding IPOs. Message type K

- count_luld: Counts messages regarding LULDs (limit up-limit down) auction collars. Message type J

- count_noii: Counts Net Order Imbalance Indicatio (NOII) messages. Message type I

- count_rpii: Counts Retail Price Improvement Indicator (RPII) messages. Message type N

Value

a data.table containing the message-type and their counts for count_messages or an integer value for the other functions.

Examples

```
file <- system.file("extdata", "ex20101224.TEST_ITCH_50", package = "RITCH")
count_messages(file)
count_messages(file, add_meta_data = TRUE, quiet = TRUE)

# file can also be a .gz file
gz_file <- system.file("extdata", "ex20101224.TEST_ITCH_50.gz", package = "RITCH")
count_messages(gz_file, quiet = TRUE)

# count only a specific class
msg_count <- count_messages(file, quiet = TRUE)

# either count based on a given data.table outputted by count_messages
count_orders(msg_count)

# or count orders from a file and not from a msg_count
count_orders(file)

### Specific class count functions are:
count_orders(msg_count)
count_trades(msg_count)
count_modifications(msg_count)
count_system_events(msg_count)
count_stock_directory(msg_count)
count_trading_status(msg_count)
count_reg_sho(msg_count)
count_market_participant_states(msg_count)
count_mwcb(msg_count)
count_ipo(msg_count)
count_luld(msg_count)
count_noii(msg_count)
count_rpii(msg_count)
```

---

download_sample_file   *Downloads a sample ITCH File from NASDAQs Server*

Description

The Server can be found at https://emi.nasdaq.com/ITCH/Nasdaq%20ITCH/

Usage

```
download_sample_file(
  choice = c("smallest", "largest", "earliest", "latest", "random", "all"),
  file = NA,
  exchanges = NA,
  dir = ".",
  force_download = FALSE,
  check_md5sum = TRUE,
```

```
  quiet = FALSE
)
```

Arguments

`choice`	which file should be chosen? One of: smallest (default), largest, earliest (date-wise), latest, random, or all.
`file`	the name of a specific file, overrules the choice and exchanges arguments
`exchanges`	A vector of exchanges, can be NASDAQ, BX, or PSX. The default value is to consider all exchanges.
`dir`	The directory where the files will be saved to, default is current working directory.
`force_download`	If the file should be downloaded even if it already exists locally. Default value is FALSE.
`check_md5sum`	If the md5-sum (hash-value) of the downloaded file should be checked, default value is TRUE.
`quiet`	if TRUE, the status messages are suppressed, defaults to FALSE

Details

Warning: the smallest file is around 300 MB, with the largest exceeding 5 GB. There are about 17 files in total. Downloading all might take a considerable amount of time.

Value

an invisible vector of the files

Examples

```
## Not run:
download_sample_file()
file <- download_sample_file()
file

# download a specific sample file
file <- download_sample_file(file = "2019130.BX_ITCH_50.gz")
file

## End(Not run)
```

```
download_stock_directory
                        Downloads the stock directory (stock locate codes) for a given date
                        and exchange
```

Description

The data is downloaded from NASDAQs server, which can be found here https://emi.nasdaq.com/ITCH/Stock_Locate_Codes/

Usage

```
download_stock_directory(exchange, date, cache = FALSE, quiet = FALSE)
```

Arguments

exchange	The exchange, either NASDAQ (equivalent to NDQ), BX, or PSX
date	The date, should be of class Date. If not the value is converted using as.Date.
cache	If the stock directory should be cached, can be set to TRUE to save the stock directories in the working directory or a character for a target directory.
quiet	If the download function should be quiet, default is FALSE.

Value

a data.table of the tickers, the respective stock locate codes, and the exchange/date information

Examples

```
## Not run:
  download_stock_directory("BX", "2019-07-02")
  download_stock_directory(c("BX", "NDQ"), c("2019-07-02", "2019-07-03"))
  download_stock_directory("BX", "2019-07-02", cache = TRUE)

  download_stock_directory("BX", "2019-07-02", cache = "stock_directory")
  dir.exists("stock_directory")
  list.files("stock_directory")

## End(Not run)
```

---

ex20101224.TEST_ITCH_50

ITCH 50 Example Testing Dataset

Description

ITCH 50 Example Testing Dataset

ex20101224.TEST_ITCH_50

The test dataset contains artificial trading data for three made up stocks: ALC, BOB, and CHAR.

The dataset is used in the examples and unit tests of the package.

The data contains the following count of messages:

- 6 system event (message type S)
- 3 stock directory (message type R)
- 3 trading status (message type H)
- 5000 orders (4997 message type A and 3 F)
- 2000 modifications (198 F, 45 X, 1745 D, and 12 U message types)
- 5000 trades (message type P)

The file is also available as ex20101224.TEST_ITCH_50.gz.

To get real sample ITCH datasets, see the download_sample_file() function.

Examples

```
file <- system.file("extdata", "ex20101224.TEST_ITCH_50", package = "RITCH")

sys <- read_system_events(file)
```

---

filter_itch   *Filters an ITCH file to another ITCH file*

Description

This function allows to perform very fast filter operations on large ITCH files. The messages are written to another ITCH file.

Usage

```
filter_itch(
  infile,
  outfile,
  filter_msg_class = NA_character_,
  filter_msg_type = NA_character_,
  filter_stock_locate = NA_integer_,
  min_timestamp = bit64::as.integer64(NA),
  max_timestamp = bit64::as.integer64(NA),
  filter_stock = NA_character_,
  stock_directory = NA,
  skip = 0,
  n_max = -1,
  append = FALSE,
  overwrite = FALSE,
  gz = FALSE,
  buffer_size = -1,
  quiet = FALSE,
  force_gunzip = FALSE,
  force_cleanup = TRUE
)
```

Arguments

infile	the input file where the messages are taken from, can be a gz-archive or a plain ITCH file.
outfile	the output file where the filtered messages are written to. Note that the date and exchange information from the infile are used, see also [add_meta_to_filename()] for further information.
filter_msg_class	
	a vector of classes to load, can be "orders", "trades", "modifications", ... see also [get_msg_classes()]. Default value is to take all message classes.
filter_msg_type	
	a character vector, specifying a filter for message types. Note that this can be used to only return 'A' orders for instance.
filter_stock_locate	
	an integer vector, specifying a filter for locate codes. The locate codes can be looked up by calling [read_stock_directory()] or by downloading from NAS-DAQ by using [download_stock_directory()]. Note that some message types (e.g., system events, MWCB, and IPO) do not use a locate code.
min_timestamp	an 64 bit integer vector (see also [bit64::as.integer64()]) of minimum timestamp (inclusive). Note: min and max timestamp must be supplied with the same length or left empty.
max_timestamp	an 64 bit integer vector (see also [bit64::as.integer64()]) of maxium timestamp (inclusive). Note: min and max timestamp must be supplied with the same length or left empty.

filter_stock	a character vector, specifying a filter for stocks. Note that this a shorthand for the filter_stock_locate argument, as it tries to find the stock_locate based on the stock_directory argument, if this is not found, it will try to extract the stock directory from the file, else an error is thrown.
stock_directory	
	A data.frame containing the stock-locate code relationship. As outputted by read_stock_directory(). Only used if filter_stock is set. To download the stock directory from NASDAQs server, use download_stock_directory().
skip	Number of messages to skip before starting parsing messages, note the skip parameter applies to the specific message class, i.e., it would skip the messages for each type (e.g., skip the first 10 messages for each class).
n_max	Maximum number of messages to parse, default is to read all values. Can also be a data.frame of msg_types and counts, as returned by count_messages(). Note the n_max parameter applies to the specific message class not the whole file.
append	if the messages should be appended to the outfile, default is false. Note, this is helpful if skip and or n_max are used for batch filtering.
overwrite	if an existing outfile with the same name should be overwritten. Default value is false
gz	if the output file should be gzip-compressed. Note that the name of the output file will be appended with .gz if not already present. The final output name is returned. Default value is false.
buffer_size	the size of the buffer in bytes, defaults to 1e8 (100 MB), if you have a large amount of RAM, 1e9 (1GB) might be faster
quiet	if TRUE, the status messages are suppressed, defaults to FALSE
force_gunzip	only applies if the input file is a gz-archive and a file with the same (gunzipped) name already exists. if set to TRUE, the existing file is overwritten. Default value is FALSE
force_cleanup	only applies if the input file is a gz-archive. If force_cleanup=TRUE, the gunzipped raw file will be deleted afterwards. Only applies when the gunzipped raw file did not exist before.

Details

Note that this can be especially useful on larger files or where memory is not large enough to filter the datalimits the analysis.

As with the read_itch() functions, it allows to filter for msg_class, msg_type, stock_locate/stock, and timestamp.

Value

the name of the output file (maybe different from the inputted outfile due to adding the date and exchange), silently

Examples

```
infile <- system.file("extdata", "ex20101224.TEST_ITCH_50", package = "RITCH")
outfile <- tempfile(fileext = "_20101224.TEST_ITCH_50")
filter_itch(
  infile, outfile,
  filter_msg_class = c("orders", "trades"),
  filter_msg_type = "R", # stock_directory
  skip = 0, n_max = 100
)

# expecting 100 orders, 100 trades, and 3 stock_directory entries
count_messages(outfile)

# check that the output file contains the same
res  <- read_itch(outfile, c("orders", "trades", "stock_directory"))
sapply(res, nrow)

res2 <- read_itch(infile,  c("orders", "trades", "stock_directory"),
                  n_max = 100)

all.equal(res, res2)
```

---

format_bytes	*Formats a number of bytes*

Description

Formats a number of bytes

Usage

```
format_bytes(x, digits = 2, unit_suffix = "B", base = 1000)
```

Arguments

`x`	the values
`digits`	the number of digits to display, default value is 2
`unit_suffix`	the unit suffix, default value is 'B' (for bytes), useful is also 'B/s' if you have read/write speeds
`base`	the base for kilo, mega, ... definition, default is 1000

Value

the values as a character

Examples

```
format_bytes(1234)
format_bytes(1234567890)
format_bytes(123456789012, unit_suffix = "iB", base = 1024)
```

---

get_exchange_from_filename

Returns the exchange from an ITCH-filename

Description

Returns the exchange from an ITCH-filename

Usage

```
get_exchange_from_filename(file)
```

Arguments

file   a filename

Value

The exchange

Examples

```
get_exchange_from_filename("03302017.NASDAQ_ITCH50")
get_exchange_from_filename("20170130.BX_ITCH_50.gz")
get_exchange_from_filename("S030220-v50-bx.txt.gz")
get_exchange_from_filename("Unknown_file_format")
```

---

get_msg_classes   *Returns the message class data for the message types*

Description

All information is handled according to the official ITCH 5.0 documentation as found here: <http:
//www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/NQTVITCHSpecification.
pdf>

Usage

```
get_msg_classes()
```

Details

- `msg_type` the type of the message
- `msg_class` the group the message belongs to
- `msg_name` the official name of the message
- `doc_nr` the number of the message in the documentation

Value

a data.table with the information of the message-types

See Also

`open_itch_specification()`

Examples

```
get_msg_classes()
```

---

gz_functions	*Compresses and uncompresses files to and from gz-archives*

Description

Allows the compression and uncompression of files

Usage

```
gunzip_file(
  infile,
  outfile = gsub("\\.gz$", "", infile),
  buffer_size = min(4 * file.size(infile), 2e+09)
)

gzip_file(
  infile,
  outfile = NA,
  buffer_size = min(4 * file.size(infile), 2e+09)
)
```

Arguments

infile	the file to be zipped or unzipped
outfile	the resulting zipped or unzipped file
buffer_size	the size of the buffer to read in at once, default is 4 times the file.size (max 2Gb).

Details

Functions are

- `gunzip_file`: uncompresses a gz-archive to raw binary data

-`gzip_file`: compresses a raw binary data file to a gz-archive

Value

The filename of the unzipped file, invisibly

Examples

```
gzfile <- system.file("extdata", "ex20101224.TEST_ITCH_50.gz", package = "RITCH")
file   <- system.file("extdata", "ex20101224.TEST_ITCH_50", package = "RITCH")

# uncompress file
(outfile <- gunzip_file(gzfile, "tmp"))
file.info(outfile)
unlink(outfile)

# compress file
(outfile <- gzip_file(file))
file.info(outfile)
unlink(outfile)
```

---

list_sample_files *Returns a data.table of the sample files on the server*

Description

The Server can be found at <https://emi.nasdaq.com/ITCH/Nasdaq%20ITCH/>

Usage

```
list_sample_files()
```

Value

a data.table of the files

Examples

```
## Not run:
  list_sample_files()

## End(Not run)
```

---

open_itch_sample_server

Opens the ITCH sample page

Description

The server can be found at <https://emi.nasdaq.com/ITCH/Nasdaq%20ITCH/>.

Usage

```
open_itch_sample_server()
```

Value

the URL (invisible)

Examples

```
## Not run:
open_itch_sample_server()

## End(Not run)
```

---

open_itch_specification

Opens the ITCH Specification PDF

Description

The specifications can be found as a PDF https://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/NQTVITCHspecification.pdf.

Usage

```
open_itch_specification()
```

Value

the URL (invisible)

Examples

```
## Not run:
open_itch_specification()

## End(Not run)
```

***

read_functions   *Reads certain messages of an ITCH-file into a data.table*

***

Description

For faster file-reads (at the tradeoff of increased memory usages), you can increase the `buffer_size` to 1GB (1e9) or more.

If you access the same file multiple times, you can provide the message counts as outputted from count_messages() to the n_max argument, this allows skipping one pass over the file per read instruction.

If you need to read in multiple message classes, you can specify multiple message classes to `read_itch`, which results in only a single file pass.

If the file is too large to be loaded into the workspace at once, you can specify different skip and n_max to load only a specific range of messages. Alternatively, you can filter certain messages to another file using filter_itch(), which is substantially faster than parsing a file and filtering it.

Note that all read functions allow both plain ITCH files as well as gzipped files. If a gzipped file is found, it will look for a plain ITCH file with the same name and use that instead. If this file is not found, it will be created by unzipping the archive. Note that the unzipped file is NOT deleted by default (the file will be created in the current working directory). It might result in increased disk usage but reduces future read times for that specific file. To force RITCH to delete "temporary" files after uncompressing, use force_cleanup = TRUE (only deletes the files if they were extracted before, does not remove the archive itself).

Usage

```
read_itch(
  file,
  filter_msg_class = NA,
  skip = 0,
  n_max = -1,
  filter_msg_type = NA_character_,
  filter_stock_locate = NA_integer_,
  min_timestamp = bit64::as.integer64(NA),
  max_timestamp = bit64::as.integer64(NA),
  filter_stock = NA_character_,
  stock_directory = NA,
  buffer_size = -1,
  quiet = FALSE,
  add_meta = TRUE,
  force_gunzip = FALSE,
  gz_dir = tempdir(),
  force_cleanup = TRUE
)

read_system_events(file, ..., add_descriptions = FALSE)
```

```
read_stock_directory(file, ..., add_descriptions = FALSE)

read_trading_status(file, ..., add_descriptions = FALSE)

read_reg_sho(file, ..., add_descriptions = FALSE)

read_market_participant_states(file, ..., add_descriptions = FALSE)

read_mwcb(file, ...)

read_ipo(file, ..., add_descriptions = FALSE)

read_luld(file, ...)

read_orders(file, ...)

read_modifications(file, ...)

read_trades(file, ...)

read_noii(file, ..., add_descriptions = FALSE)

read_rpii(file, ..., add_descriptions = FALSE)

get_orders(file, ...)

get_trades(file, ...)

get_modifications(file, ...)
```

Arguments

`file`	the path to the input file, either a gz-archive or a plain ITCH file
`filter_msg_class`	a vector of classes to load, can be "orders", "trades", "modifications", ... see also [get_msg_classes()](). Default value is to take all message classes.
`skip`	Number of messages to skip before starting parsing messages, note the skip parameter applies to the specific message class, i.e., it would skip the messages for each type (e.g., skip the first 10 messages for each class).
`n_max`	Maximum number of messages to parse, default is to read all values. Can also be a data.frame of msg_types and counts, as returned by [count_messages()](). Note the n_max parameter applies to the specific message class not the whole file.
`filter_msg_type`	a character vector, specifying a filter for message types. Note that this can be used to only return 'A' orders for instance.

filter_stock_locate

> an integer vector, specifying a filter for locate codes. The locate codes can be looked up by calling read_stock_directory() or by downloading from NAS-DAQ by using download_stock_directory(). Note that some message types (e.g., system events, MWCB, and IPO) do not use a locate code.

min_timestamp   an 64 bit integer vector (see also bit64::as.integer64()) of minimum times-tamp (inclusive). Note: min and max timestamp must be supplied with the same length or left empty.

max_timestamp   an 64 bit integer vector (see also bit64::as.integer64()) of maxium times-tamp (inclusive). Note: min and max timestamp must be supplied with the same length or left empty.

filter_stock   a character vector, specifying a filter for stocks. Note that this a shorthand for the filter_stock_locate argument, as it tries to find the stock_locate based on the stock_directory argument, if this is not found, it will try to extract the stock directory from the file, else an error is thrown.

stock_directory

> A data.frame containing the stock-locate code relationship. As outputted by read_stock_directory(). Only used if filter_stock is set. To download the stock directory from NASDAQs server, use download_stock_directory().

buffer_size   the size of the buffer in bytes, defaults to 1e8 (100 MB), if you have a large amount of RAM, 1e9 (1GB) might be faster

quiet   if TRUE, the status messages are suppressed, defaults to FALSE

add_meta   if TRUE, the date and exchange information of the file are added, defaults to TRUE

force_gunzip   only applies if the input file is a gz-archive and a file with the same (gunzipped) name already exists. if set to TRUE, the existing file is overwritten. Default value is FALSE

gz_dir   a directory where the gz archive is extracted to. Only applies if file is a gz archive. Default is tempdir().

force_cleanup   only applies if the input file is a gz-archive. If force_cleanup=TRUE, the gun-zipped raw file will be deleted afterwards. Only applies when the gunzipped raw file did not exist before.

...   Additional arguments passed to read_itch

add_descriptions

> add longer descriptions to shortened variables. The added information is taken from the official ITCH documentation see also open_itch_specification()

Details

The details of the different messages types can be found in the official ITCH specification (see also open_itch_specification())

- read_itch: Reads a message class message, can also read multiple classes in one file-pass.

- read_system_events: Reads system event messages. Message type S

- `read_stock_directory`: Reads stock trading messages. Message type `R`

- `read_trading_status`: Reads trading status messages. Message type `H` and `h`

- `read_reg_sho`: Reads messages regarding reg SHO. Message type `Y`

- `read_market_participant_states`: Reads messages regarding the status of market participants. Message type `L`

- `read_mwcb`: Reads messages regarding Market-Wide-Circuit-Breakers (MWCB). Message type `V` and `W`

- `read_ipo`: Reads messages regarding IPOs. Message type `K`

- `read_luld`: Reads messages regarding LULDs (limit up-limit down) auction collars. Message type `J`

- `read_orders`: Reads order messages. Message type `A` and `F`

- `read_modifications`: Reads order modification messages. Message type `E`, `C`, `X`, `D`, and `U`

- `read_trades`: Reads trade messages. Message type `P`, `Q` and `B`

- `read_noii`: Reads Net Order Imbalance Indicatio (NOII) messages. Message type `I`

- `read_rpii`: Reads Retail Price Improvement Indicator (RPII) messages. Message type `N`

For backwards compatability reasons, the following functions are provided as well:

- `get_orders`: Redirects to `read_orders`

- `get_trades`: Redirects to `read_trades`

- `get_modifications`: Redirects to `read_modifications`

Value

a data.table containing the messages

References

https://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/NQTVITCHspecification.pdf

Examples

```
file <- system.file("extdata", "ex20101224.TEST_ITCH_50", package = "RITCH")
od <- read_orders(file, quiet = FALSE) # note quiet = FALSE is the default
tr <- read_trades(file, quiet = TRUE)

## Alternatively
od <- read_itch(file, "orders", quiet = TRUE)

ll <- read_itch(file, c("orders", "trades"), quiet = TRUE)

od
tr
str(ll, max.level = 1)

## additional options:

# take only subset of messages
od <- read_orders(file, skip = 3, n_max = 10)

# a message count can be provided for slightly faster reads
msg_count <- count_messages(file, quiet = TRUE)
od <- read_orders(file, n_max = msg_count)

## .gz archive functionality
# .gz archives will be automatically unzipped
gz_file <- system.file("extdata", "ex20101224.TEST_ITCH_50.gz", package = "RITCH")
od <- read_orders(gz_file)
# force a decompress and delete the decompressed file afterwards
od <- read_orders(gz_file, force_gunzip = TRUE, force_cleanup = TRUE)

## read_itch()
otm <- read_itch(file, c("orders", "trades"), quiet = TRUE)
str(otm, max.level = 1)

## read_system_events()
se <- read_system_events(file, add_descriptions = TRUE, quiet = TRUE)
se

## read_stock_directory()
sd <- read_stock_directory(file, add_descriptions = TRUE, quiet = TRUE)
sd

## read_trading_status()
ts <- read_trading_status(file, add_descriptions = TRUE, quiet = TRUE)
ts

## read_reg_sho()
## Not run:
# note the example file has no reg SHO messages
rs <- read_reg_sho(file, add_descriptions = TRUE, quiet = TRUE)
rs
```

```
## End(Not run)

## read_market_participant_states()
## Not run:
# note the example file has no market participant states
mps <- read_market_participant_states(file, add_descriptions = TRUE,
                                      quiet = TRUE)
mps

## End(Not run)

## read_mwcb()
## Not run:
# note the example file has no circuit breakers messages
mwcb <- read_mwcb(file, quiet = TRUE)
mwcb

## End(Not run)

## read_ipo()
## Not run:
# note the example file has no IPOs
ipo <- read_ipo(file, add_descriptions = TRUE, quiet = TRUE)
ipo

## End(Not run)

## read_luld()
## Not run:
# note the example file has no LULD messages
luld <- read_luld(file, quiet = TRUE)
luld

## End(Not run)

## read_orders()
od <- read_orders(file, quiet = TRUE)
od

## read_modifications()
mod <- read_modifications(file, quiet = TRUE)
mod

## read_trades()
tr <- read_trades(file, quiet = TRUE)
tr

## read_noii()
## Not run:
# note the example file has no NOII messages
noii <- read_noii(file, add_descriptions = TRUE, quiet = TRUE)
noii
```

```
## End(Not run)

## read_rpii()
## Not run:
# note the example file has no RPII messages
rpii <- read_rpii(file, add_descriptions = TRUE, quiet = TRUE)
rpii

## End(Not run)
```

---

write_itch   *Writes a data.frame or a list of data.frames of ITCH messages to file*

Description

Note that additional information, e.g., columns that were added, will be dropped in the process and only ITCH-compliant information is saved.

Usage

```
write_itch(
  ll,
  file,
  add_meta = TRUE,
  append = FALSE,
  compress = FALSE,
  buffer_size = 1e+08,
  quiet = FALSE,
  append_warning = TRUE
)
```

Arguments

ll	a data.frame or a list of data.frames of ITCH messages, in the format that the [read_functions()](#) return
file	the filename of the target file. If the folder to the file does not exist, it will be created recursively
add_meta	if date and file information should be added to the filename. Default value is TRUE. Note that adding meta information changes the filename.
append	if the information should be appended to the file. Default value is FALSE
compress	if the file should be gzipped. Default value is FALSE. Note that if you compress a file, buffer_size matters a lot, with larger buffers you are more likely to get smaller filesizes in the end. Alternatively, but slower, is to write the file without compression fully and then gzip the file using another program.
buffer_size	the maximum buffer size. Default value is 1e8 (100MB). Accepted values are > 52 and < 5e9

quiet	if TRUE, the status messages are suppressed, defaults to FALSE
append_warning	if append is set, a warning about timestamp ordering is given. Set append_warning = FALSE to silence the warning. Default value is TRUE

Details

Note that the ITCH filename contains the information for the date and exchange. This can be specified explicitly in the file argument or it is added if not turned off add_meta = FALSE.

Value

the filename (invisibly)

Examples

```
infile <- system.file("extdata", "ex20101224.TEST_ITCH_50", package = "RITCH")
sys <- read_system_events(infile, quiet = TRUE)
outfile <- tempfile()
write_itch(sys, outfile)

# create a list of events, stock directory, and orders and write to a file
sdir <- read_stock_directory(infile, quiet = TRUE)
od   <- read_orders(infile, quiet = TRUE)

ll <- list(sys, sdir, od)
write_itch(ll, outfile)
```

Index