Package 'knitrdata'

December 2, 2020

Type Package

Title Data Language Engine for 'knitr' / 'rmarkdown'

Version 0.6.0

Description Implements a data language engine for incorporating data directly in 'rmarkdown' documents so that they can be made completely standalone.

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/dmkaplan2000/knitrdata

Depends R (> 3.5.0)

Imports knitr, xfun (>= 0.16), methods

Suggests gpg, rmarkdown, magrittr, shiny, miniUI, rstudioapi (>= 0.11), DT, digest

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author David M. Kaplan [aut, cre, cph] (<https://orcid.org/0000-0001-6087-359X>, dmkaplan2000)

Maintainer David M. Kaplan <dmkaplan2000@gmail.com>

Repository CRAN

Date/Publication 2020-12-02 12:30:02 UTC

R topics documented:

`create_chunk` *Tools for creating (data) chunks and inserting them into Rmarkdown documents*

Description

These helper functions allow one to add the chunk header and tail to text containing chunk contents and then insert that into a Rmarkdown document.

Usage

```
create_chunk(
  text = readLines(file),
  ...,
  chunk_label = NULL,
  chunk_type = "data",
  file = NULL,
  chunk_options_string = NULL,
  split_lines = TRUE,
  newline = platform.newline()
)

insert_chunk(chunk, line, rmd.text = readLines(rmd.file), rmd.file = NULL)
```

Arguments

`text`	Character vector with contents of chunk, one line per element of vector. If the character vector has just a single element, then an attempt will be made to split it into lines using `readLines`.
`...`	Additional chunk options. These are not evaluated, but rather included in the function call as they are entered in the function call.
`chunk_label`	Character string giving the label to be used for the chunk.
`chunk_type`	Character string giving the chunk type. Defaults to `"data"`.
`file`	Path to file containing chunk contents. Ignored if `text` argument supplied. As a consequence, this means that all arguments must be named if the `file` argument is supplied to `create_chunk`.
`chunk_options_string`	Character vector with additional chunk options that will be included in the header after the arguments in `...`.

split_lines	Boolean indicating whether or not the chunk contents should be split along line breaks before returning. Defaults to TRUE.
newline	Character string used to end lines of text. Only relevant if split_lines=FALSE. Defaults to platform.newline().
chunk	Character vector with chunk contents including header and tail. If the character vector has just a single element, then an attempt will be made to split it into lines using readLines.
line	Line number where chunk to be inserted.
rmd.text	Character vector with contents of Rmarkdown document where chunk contents are to be inserted. It should have one element per line of text (as returned by readLines). If the character vector has just a single element, then an attempt will be made to split it into lines using readLines.
rmd.file	Filename of Rmarkdown document where chunk contents are to be inserted. Ignored if rmd.text is supplied.

Details

create_chunk takes in the (possibly encoded by data_encode) contents of a chunk and adds the chunk header and closer, invisibly returning entire chunk contents as a character string.

insert_chunk takes the chunk contents and inserts it at the given line number in the rmd.text or rmd.file.

Note that the additional arguments to create_chunk (...) are not evaluated, but rather they are placed in the chunk header as they appear in the function call as additional chunk options.

Functions

- create_chunk: Silently returns chunk text contents.
- insert_chunk: Invisibly returns the contents of the modified Rmarkdown text as a character vector with each line in an element of the vector including the chunk at the appropriate line number.

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

Other Chunk tools: create_data_chunk_dialog(), insert_data_chunk_dialog(), insert_data_chunk_template(), list_rmd_chunks(), remove_chunks_dialog()

Examples

```
# Use a temporary directory ----------------------------
owd = getwd()
td = tempdir()
setwd(td)

# Test --------------
```

```
library(knitrdata)
library(magrittr) # For pipe operator

# Create new Rmarkdown document
if (file.exists("test.create_chunks.Rmd"))
  file.remove("test.create_chunks.Rmd")
rmarkdown::draft("test.create_chunks.Rmd","github_document","rmarkdown",
                 edit=FALSE)

# List all chunks in document
chunklst = list_rmd_chunks(file="test.create_chunks.Rmd")
chunklst

# Remove the pressure chunk
xx = split_rmd_by_chunk(file="test.create_chunks.Rmd",chunk_label="pressure")
txt = c(xx$pre_chunk,xx$post_chunk)
writeLines(txt,"test.create_chunks.Rmd")

# List chunks again
chunklst = list_rmd_chunks(file="test.create_chunks.Rmd")
chunklst

# Remove all but setup chunk
remove_chunks(file="test.create_chunks.Rmd",
              chunk_labels = 2:nrow(chunklst),
              output.file="test.create_chunks.Rmd")

# List all chunks again
chunklst = list_rmd_chunks(file="test.create_chunks.Rmd")
chunklst

# Create some binary data
x = data.frame(a=1:10,b=(1:10)^2)
saveRDS(x,"test.create_chunks.RDS")

# Push chunks into Rmarkdown document
# Insert in reverse order to not have to figure out line number
txt = create_chunk(chunk_label="plot",c("x","plot(b~a,data=x)"),chunk_type="r") %>%
  insert_chunk(11,rmd.file="test.create_chunks.Rmd")
txt = data_encode("test.create_chunks.RDS","base64") %>%
 create_chunk(chunk_label="thedata",output.var="x",format="binary",loader.function=readRDS) %>%
  insert_chunk(11,txt)
txt = create_chunk(chunk_label="loadknitrdata","library(knitrdata)",chunk_type="r") %>%
  insert_chunk(11,txt)

writeLines(txt,"test.create_chunks.Rmd")

# List all chunks again
chunklst = list_rmd_chunks(file="test.create_chunks.Rmd")
chunklst

# Render document to test
if (rmarkdown::pandoc_available(version="1.12.3"))
```

```
  rmarkdown::render("test.create_chunks.Rmd")

# Clean up --------------
file.remove("test.create_chunks.Rmd","test.create_chunks.RDS",
            "test.create_chunks.md","test.create_chunks.html")
unlink("test.create_chunks_files",recursive=TRUE)

setwd(owd)
```

`create_data_chunk_dialog` *Invoke Shiny gadget to create a data chunk*

Description

As different elements of the data chunk are specified, other options will be modified as is likely to be useful. For example, if a binary file is uploaded, then the `format` will be set to `binary`, the `encoding` will be set to `base64` and the `Encode data?` option will be checked. If these options are not appropriate, then they can be altered afterwards.

Usage

```
create_data_chunk_dialog(
  title = "Data chunk creator",
  infobar = "<big><b>Fill out, then click above to create chunk</b></big>"
)
```

Arguments

`title`	Text to place in title bar of gadget
`infobar`	HTML content to place in information bar at top of gadget

Details

When the `Create chunk` button is clicked, the function will return the chunk contents including header and tail.

Value

Invisibly returns the text of the data chunk as a character vector, one line per element.

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

Other Chunk tools: `create_chunk()`, `insert_data_chunk_dialog()`, `insert_data_chunk_template()`, `list_rmd_chunks()`, `remove_chunks_dialog()`

Examples

```
## Not run:
create_data_chunk_dialog()

## End(Not run)
```

`data_decode` *Decode and encode text and binary data files*

Description

These helper functions allow one to encode as text a binary or text data file using either `base64` or `gpg` encoding (`data_encode`) and decode text-encoded data back into its original binary or text format (`data_decode`).

Usage

```
data_decode(data, encoding, as_text = FALSE, options = list())

data_encode(file, encoding, options = list(), output = NULL)
```

Arguments

`data`	Encoded data as a character string
`encoding`	Either `'base64'` or `'gpg'`
`as_text`	A boolean indicating if decoded data should be treated as text (`TRUE`) or binary (`FALSE`). Defaults to `FALSE`, meaning binary.
`options`	A list containing extra arguments for the encoding/decoding functions. For `base64` encoding, `linewidth` (defaults to 64) and and `newline` (defaults to `platform.newline()`) optional arguments are possible. For gpg encoding, see the description below for details regarding the required `receiver` option to define the key to use for encryption. For further details and potentially other additional arguments, see the help of the corresponding underlying encoding functions: `base64_encode` and `gpg_encrypt`.
`file`	Path to file containing data to be encoded
`output`	Path where encoded data is to be stored. Optional; if `NULL` then encoded data will not be written to a file.

Details

Encoding and decoding in `base64` format uses functionality from the `xfun` package, whereas encoding and decoding using gpg uses functionality from the `gpg` package. See those packages for more details on the encoding and decoding process and setting up a gpg keyring.

`data_encode` takes the name of a file containing the binary or text data to be encoded and returns the encoded data as a character string. The encoded data is returned silently to avoid outputing to

the screen large amounts of encoded data. If you want to visualize the encoded data, use the cat function. For larger data files, set the output argument to a path where the encoded data will be stored.

data_encode takes a character string of encoded data and returns either a character string of decoded data (if as_text=TRUE) or a raw vector of decoded binary data (if as_text=FALSE).

For both functions, the options input argument takes a list of additional input arguments that are passed directly to the encoding or decoding functions in the respective packages that handle the actual data translation. See base64_encode and gpg_encrypt for details.

For gpg encoding and decoding, in addition to installing the gpg package, a gpg keyring must be installed and properly configured. For encoding, the receiver and potentially the signer arguments must be supplied as elements of the options input argument.

Value

Returns either the decoded data (data_decode) or the encoded data (data_encode).

Functions

- data_decode: Returns decoded data as either a character string (as_text=TRUE) or raw vector (as_text=FALSE).
- data_encode: Returns data encoded as a character string using base64 or gpg encoding.

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

See also base64_encode and gpg_encrypt, platform.newline.

Examples

```
# Use a temporary directory -----------------------------
owd = getwd()
td = tempdir()
setwd(td)

# Do some data encoding and decoding ------------------
library(knitrdata)

x = data.frame(a=1:5,b=letters[1:5])
write.csv(x,"test.csv")
saveRDS(x,"test.RDS")

enccsv = data_encode("test.csv","base64")
encrds = data_encode("test.RDS","base64")

csv = data_decode(enccsv,"base64",as_text=TRUE)
cat(csv)
```

```
rds = data_decode(encrds,"base64",as_text=FALSE)
writeBin(rds,"test_output.RDS")
y = readRDS("test_output.RDS")
y

params = list(run_gpg=FALSE)
if (requireNamespace("gpg") && params$run_gpg) {
  k = gpg::gpg_keygen("test","test@test.org")
  encgpg = data_encode("test.csv","gpg",options = list(receiver=k))

  cat(data_decode(encgpg,"gpg",as_text=TRUE))

  gpg::gpg_delete(k,secret=TRUE)
}

# Cleanup -----------------------------------
file.remove("test.csv","test.RDS","test_output.RDS")

setwd(owd)
```

`insert_data_chunk_dialog` *Invoke Rstudio addin to insert a data chunk in active source document*

Description

As different elements of the data chunk are specified, other options will be modified as is likely to be useful. For example, if a binary file is uploaded, then the `format` will be set to `binary`, the `encoding` will be set to `base64` and the `Encode data?` option will be checked. If these options are not appropriate, then they can be altered afterwards.

Usage

```
insert_data_chunk_dialog(title = "Data chunk inserter", chunk = NULL)
```

Arguments

`title`	Text to place in title bar of gadget.
`chunk`	Text content of the data chunk. If not given (as is typically the case), the `create_data_chunk_dialog` will be used to generate chunk contents.

Details

When the `Create chunk` button is clicked, the contents of the data chunk will be inserted at the current cursor location of the active source document in the Rstudio editor.

Value

Returns `TRUE` if a chunk was inserted, `FALSE` otherwise.

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

Other Chunk tools: create_chunk(), create_data_chunk_dialog(), insert_data_chunk_template(), list_rmd_chunks(), remove_chunks_dialog()

Examples

```
## Not run:
insert_data_chunk_dialog()

## End(Not run)
```

---

insert_data_chunk_template

Insert an empty data chunk template in active source document

---

Description

This function is essentially the equivalent for data chunks of the "Insert a new code chunk" menu item available in Rstudio when a Rmarkdown document is open. It places at the current cursor location an empty data chunk that can then be modified and filled in by hand.

Usage

```
insert_data_chunk_template()
```

Value

Returns TRUE if a chunk was inserted, FALSE otherwise.

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

Other Chunk tools: create_chunk(), create_data_chunk_dialog(), insert_data_chunk_dialog(), list_rmd_chunks(), remove_chunks_dialog()

Examples

```
## Not run:
insert_data_chunk_template()

## End(Not run)
```

`is.file.binary` *Functions to assess if files are binary, DOS text or UNIX text format*

Description

These functions attempt to determine if a file is binary or text. In addition, `file.type` attempts to determine the newline character(s) used in the file.

Usage

```
is.file.binary(file, bin.ints = c(1:8, 14:25), nbytes = 1000, nbin = 2)

file.type(file, bin.ints = c(1:8, 14:25), nbytes = 1000, nbin = 2)
```

Arguments

`file`	The path to the file to be examined
`bin.ints`	List of integers with the ASCII values of control characters that are to be considered when when looking for signs a file is binary. Default includes most ASCII control characters except things like NULL, LF, CR and HT that might actually appear in an ASCII file.
`nbytes`	Number of bytes to read in from the beginning of the file.
`nbin`	An integer indicating the threshold on the number of control characters above which a file is considered binary. Defaults to 2.

Details

A file is assessed to be binary using a heuristic based on finding more than `nbin` ASCII control (i.e., non-printing) characters in the first `nbytes` of the file. This works well for standard ASCII text, but it may be less effective for complex UTF8 text (e.g., Chinese).

For text files, line endings are assessed by `file.type` by searching first for DOS line endings (`\r\n`) in the first `nbytes` of the input file, and then by searching for UNIX line endings (`\n`). If neither is found, then `NA_character_` is returned for the line ending.

Value

For `is.file.binary`, a boolean value is returned, whereas a list is returned for `file.type`.

Functions

- `is.file.binary`: A boolean that will be `TRUE` if a file is considered to be binary.
- `file.type`: Returns a list with up to two elements: `type` & `newline`. `type` can either by `"binary"` or `"text"`. `newline` will be `NULL` for binary files, `"\r\n"` for DOS formatted text files, `"\n"` for UNIX formatted text files and `NA_character_` for text files without any newline characters in the first `nbytes` of the file.

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

See also platform.newline.

---

list_rmd_chunks *Tools for working with existing chunks in Rmarkdown documents*

Description

These helper functions allow one to identify all the chunks in a Rmarkdown document, split the document into pieces by a specific chunk so that one can either work with the chunk contents or remove the chunk, and remove several chunks at once.

Usage

```
list_rmd_chunks(
  text = readLines(file),
  file = NULL,
  chunk.start.pattern = "^```[{](.+)[}] *$",
  chunk.end.pattern = "^``` *$"
)

split_rmd_by_chunk(text = readLines(file), chunk_label, file = NULL, ...)

remove_chunks(
  text = readLines(file),
  chunk_labels,
  file = NULL,
  output.file = NULL,
  ...
)
```

Arguments

text	Character vector with contents of chunk, one element per line of text. If the character vector has just a single element, then an attempt will be made to split it into lines using readLines.
file	Path to file containing chunk contents. Ignored if text argument supplied. As a consequence, this means that all arguments must be named if the file argument is supplied.
chunk.start.pattern	Regular expression used to identify chunk starts. The default looks for lines beginning with three back quotes, followed by curly braces with some sort of

text between them and then only spaces till the end of the line. This should generally work, but if the Rmarkdown document has chunks that have unusual headers, then this argument can be useful. In particular, if the document has chunks that begin without curly braces, these will not be recognized.

`chunk.end.pattern`
: Regular expression used to identify the chunk end. Default should generally work.

`chunk_label`
: Character string giving the chunk label or the chunk number (as returned by `list_rmd_chunks`.

`...`
: Additional arguments to be passed to `list_rmd_chunks` (e.g., `chunk.start.pattern`).

`chunk_labels`
: A vector of numeric or character chunk labels (as returned by `list_rmd_chunks`.

`output.file`
: Name of a file where Rmd document with desired chunks removed is to be saved.

Details

`list_rmd_chunks` takes a Rmarkdown document and returns a `data.frame` listing the essential information of every chunk, including chunk type (language engine), label and start and end line numbers.

`split_rmd_by_chunk` takes a Rmarkdown document and a chunk label or number and returns the Rmarkdown document split into 4 pieces: the part before the chunk, the chunk header, the chunk contents, the chunk tail and the part after the chunk. These can then be used to either work with the chunk contents or remove the chunk from the Rmarkdown document.

`remove_chunks` removes several chunks, designated by their text or numeric labels, all at once from a Rmarkdown document.

Note that the regular expression used by default to identify chunk starts is not guaranteed to be exactly the same as that used by `knitr` and may not work if the Rmarkdown document has unusual chunks. In particular, each chunk must have the chunk type and chunk options enclosed in curly braces. If code chunks exist without curly braces, then these will generally be ignored, but they could potentially cause problems in unusual cases.

Functions

- `list_rmd_chunks`: Returns a data frame with 4 columns: the chunk type, the chunk label, the line number of the beginning of the chunk and the line number of the end of the chunk.
- `split_rmd_by_chunk`: Returns a list with the contents of the Rmarkdown document broken into 4 pieces: pre-chunk, chunk header, chunk contents, chunk tail, and post-chunk.
- `remove_chunks`: Silently returns a character vector with the contents of the Rmd file after having removed the desired chunks

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

David M. Kaplan <dmkaplan2000@gmail.com>

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

Other Chunk tools: create_chunk(), create_data_chunk_dialog(), insert_data_chunk_dialog(), insert_data_chunk_template(), remove_chunks_dialog()

Examples

```
# Use a temporary directory ----------------------------
owd = getwd()
td = tempdir()
setwd(td)

# Test --------------
library(knitrdata)
library(magrittr) # For pipe operator

# Create new Rmarkdown document
if (file.exists("test.create_chunks.Rmd"))
  file.remove("test.create_chunks.Rmd")
rmarkdown::draft("test.create_chunks.Rmd","github_document","rmarkdown",
                 edit=FALSE)

# List all chunks in document
chunklst = list_rmd_chunks(file="test.create_chunks.Rmd")
chunklst

# Remove the pressure chunk
xx = split_rmd_by_chunk(file="test.create_chunks.Rmd",chunk_label="pressure")
txt = c(xx$pre_chunk,xx$post_chunk)
writeLines(txt,"test.create_chunks.Rmd")

# List chunks again
chunklst = list_rmd_chunks(file="test.create_chunks.Rmd")
chunklst

# Remove all but setup chunk
remove_chunks(file="test.create_chunks.Rmd",
              chunk_labels = 2:nrow(chunklst),
              output.file="test.create_chunks.Rmd")

# List all chunks again
chunklst = list_rmd_chunks(file="test.create_chunks.Rmd")
chunklst

# Create some binary data
x = data.frame(a=1:10,b=(1:10)^2)
saveRDS(x,"test.create_chunks.RDS")

# Push chunks into Rmarkdown document
# Insert in reverse order to not have to figure out line number
txt = create_chunk(chunk_label="plot",c("x","plot(b~a,data=x)"),chunk_type="r") %>%
  insert_chunk(11,rmd.file="test.create_chunks.Rmd")
txt = data_encode("test.create_chunks.RDS","base64") %>%
```

```
 create_chunk(chunk_label="thedata",output.var="x",format="binary",loader.function=readRDS) %>%
  insert_chunk(11,txt)
txt = create_chunk(chunk_label="loadknitrdata","library(knitrdata)",chunk_type="r") %>%
  insert_chunk(11,txt)

writeLines(txt,"test.create_chunks.Rmd")

# List all chunks again
chunklst = list_rmd_chunks(file="test.create_chunks.Rmd")
chunklst

# Render document to test
if (rmarkdown::pandoc_available(version="1.12.3"))
  rmarkdown::render("test.create_chunks.Rmd")

# Clean up --------------
file.remove("test.create_chunks.Rmd","test.create_chunks.RDS",
            "test.create_chunks.md","test.create_chunks.html")
unlink("test.create_chunks_files",recursive=TRUE)

setwd(owd)
```

`platform.newline` *Platform independent newline string*

Description

A simple function to determine the appropriate newline string for a given operating system.

Usage

```
platform.newline(os = .Platform$OS.type)
```

Arguments

`os`	Name of the operating system. Defaults to `.Platform$OS.type`

Value

For Windows, this should return `'\r\n'`, whereas for other operating system it will return `'\n'`.

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

See also `file.type`.

remove_chunks_dialog *Invoke Rstudio addin to remove chunks from the active source document*

Description

The dialog will present a data table list of chunks in the source document. Select the rows that correspond to the chunks that you wish to remove and hit the Remove chunks button to remove them.

Usage

```
remove_chunks_dialog(title = "Eliminate (data) chunks")
```

Arguments

title	Text to place in title bar of gadget.

Details

When the dialog is started, if the cursor is positioned inside a chunk in the source document, then the row corresponding to this chunk will be selected by default.

Value

Returns TRUE if one or more chunks were removed, FALSE otherwise.

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

Other Chunk tools: create_chunk(), create_data_chunk_dialog(), insert_data_chunk_dialog(), insert_data_chunk_template(), list_rmd_chunks()

Examples

```
## Not run:
remove_chunks_dialog()

## End(Not run)
```

unlock_gpg_key_passphrase *Functions that attempts to unlock a gpg key for decryption*

Description

This function will attempt to unlock a specific GPG key by first encrypting a small amount of information using the (public) key and then immediately decrypting it using the (private) key, thereby causing the keyring to temporarily store the key passphrase. This can be helpful if one is trying to knit a document with encrypted data chunks, but the key for those data chunks is locked with a passphrase. See the package vignette section `Workarounds for GPG data chunk error: Password callback did not return a string value` for more details.

Usage

```
unlock_gpg_key_passphrase(
  id = ifelse(is.null(name), ifelse(is.null(email), NULL,
    gpg::gpg_list_keys()$id[gpg::gpg_list_keys()$email == email]),
    gpg::gpg_list_keys()$id[gpg::gpg_list_keys()$name == name]),
  name = NULL,
  email = NULL
)
```

Arguments

`id`	Identifier of the GPG key
`name`	Name associated with the desired GPG key
`email`	Email associated with the desired GPG key

Value

Will return the identifier of the GPG key that was unlocked.

Author(s)

David M. Kaplan <dmkaplan2000@gmail.com>

See Also

See also `data_encode`, `gpg_encrypt`.

Index