

Package ‘gsengine’

November 30, 2025

Type Package

Title Genstat Engine for knitr

Version 2.3-1

Description Provides a custom knitr engine to process Genstat code via socket connection.

Depends R (>= 3.5.0)

Imports base64enc, knitr, jsonlite, rvest, xml2

License GPL-3

Encoding UTF-8

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.3.3

NeedsCompilation no

Author Yuxiao Wang [aut],
Simon Urbanek [aut],
James Curran [aut, cre]

Maintainer James Curran <j.curran@auckland.ac.nz>

Contents

extractTablesToEnv	2
fixTableCols	2
gs.engine	3
gsio	3
processOutput	5
stripGenVerbatim	5
Index	6

`extractTablesToEnv` *Parse HTML tables and assign as data frames/tibbles into the knit env*

Description

Supports three modes: - `mode="auto"`: assign list to a name like `gs_tables_<chunklabel>` - `mode="single_or_list"`: assign a single table (if `n=1`) or a list to a given name - `mode="vector_map"`: map first `k` tables to `k` names

Usage

```
extractTablesToEnv(htmlBlock, saveConfig)
```

Arguments

<code>htmlBlock</code>	Character vector of HTML (from Genstat output)
<code>saveConfig</code>	List produced by <code>normalize_save_tables_option()</code>

Details

Requires: `xml2`, `rvest`

<code>fixTableCols</code>	<i>Inject per-table nth-of-type CSS from <code><col></code> alignment/widths, then remove <code><col>s</code></i>
---------------------------	---

Description

Inject per-table nth-of-type CSS from `<col>` alignment/widths, then remove `<col>s`

Usage

```
fixTableCols(output, io)
```

Arguments

<code>output</code>	Character vector (HTML fragments) to preprocess
<code>io</code>	Environment which contains, among other things, a table counter which is necessary to give the tables unique ids.

Value

A single character string of cleaned HTML

gs.engine

*Genstat Socket Engine for knitr***Description**

This function creates a custom knitr engine for executing Genstat code by connecting to a Genstat server over a socket. The engine sends Genstat code, receives output, and renders it in R Markdown documents with support for formatted tables, warnings, and embedded plots.

Usage

```
gs.engine(
  host = Sys.getenv("GENSTAT_HOST", "localhost"),
  port = as.integer(Sys.getenv("GENSTAT_PORT", "8085")),
  timeout = 1L
)
```

Arguments

host	A character string specifying the host name or IP address of the Genstat server. Defaults to "localhost" or the environment variable 'GENSTAT_HOST'.
port	An integer specifying the port number to connect to on the Genstat server. Defaults to '8085' or the environment variable 'GENSTAT_PORT'.
timeout	numeric, number of seconds to wait if no response is coming.

Value

A function that can be registered as a knitr engine (e.g., via 'knitr::knit_engines\$set(gs = gs.engine())')

Examples

```
## Not run:
knitr::knit_engines$set(gs = gs.engine())

## End(Not run)
```

gsio

*Genstat Messenger TCP/IO Communication Protocol Functions***Description**

`gsio.connect` connects to the Messenger server according to the `host` and `port` arguments. Typically you want to call `gsio.greeting()` on the resulting connection to start the communication.

`gsio.msg` sends a message to the Messenger and waits for a response. Any unmatched, named arguments in `...` will be ignored. As a special case, passing `NULL` will tell `gsio.msg` to not send any message, but still process any pending replies (useful with `wait=FALSE` to check for such condition).

`gsio.greeting` receives a greeting response from the Messenger, i.e., the first response after connecting.

The `close` method closes the connection to the GenStat Messenger.

The following functions are low-level and typically not used directly (use `gsio.msg` instead): `gsio.send` sends content to the Messenger, `gsio.recv` receives a reply from the Messenger.

Usage

```
gsio.connect(host = "localhost", port = 8085L, timeout = 1)

gsio.send(io, ...)

gsio.recv(io, wait = TRUE)

gsio.msg(io, ..., all = TRUE, wait = TRUE)

## S3 method for class 'gsio'
close(con, ...)

gsio.greeting(io)
```

Arguments

<code>host</code>	string, name (or IP address) or the host to connect to
<code>port</code>	integer, TCP port to connect to
<code>timeout</code>	real, time (in seconds, can be fractional) for communication attempts.
<code>io</code>	"gsio" object as obtained from <code>gsio.connect()</code>
<code>...</code>	content to send, will be pasted together as string without a separator.
<code>wait</code>	logical scalar, if <code>TRUE</code> then this function does not return until a reply is received. Otherwise will return <code>NULL</code> if no reply was received. Note that if a response header is received, the function will still wait until it can read the entire response in order to prevent inconsistent state of the connection.
<code>all</code>	logical scalar, if <code>TRUE</code> then all replies until "STAT" is received will be collected and returned at the end in a list. Note that if "STAT" is the only reply when the list will be empty. If <code>FALSE</code> then only the first reply will be returned.
<code>con</code>	"gsio" object as obtained from <code>gsio.connect()</code>

Details

A typical communication with the Messenger starts with `gsio.connect()`, followed by `gsio.greeting()`, one or more calls to `gsio.msg()` and final `close()`. In most cases the messages to send to the Messenger are strings representing code to evaluate in GenStat.

Since Messenger 1.1 there are special commands that will be interpreted by the Messenger instead of passing them to GenStat. Those messages start with `"#:"`. Should you need to send a string to GenStat that also starts with `"#:"` (uncommon) then precede the string with `"#:"` to tell the Messenger that it should be passed through, so `"#:#:FOO"` will result in `"#:FOO"` being sent to GenStat.

Which special commands are supported depends on the Messenger version. As of 1.1 `"#:SET_OUTPUT_TYPE:HTML"` will set the output type (valid values are `TEXT`, `RTF` and `HTML`) and restart GenStat with the new setting. `"#:RESTART"` will restart the GenStat server without changing the current output settings.

NOTE: In both cases the GetStat server currently sends *two* STAT responses, so it is prudent to call `gsio.recv(c, wait=FALSE)` immediately after receiving the message response to clear the extra STAT response.

Value

`gsio.connect`: an object of the class "gsio" which can be used for subsequent communication.

`gsio.send`: undefined (currently NULL)

`gsio.recv`: Either NULL (if `wait=FALSE` and no reply is present) or a a reply (see `gsio.msg` return description below for the definition of a reply).

`gsio.msg`: If `all=TRUE` then a list of replies, otherwise a single reply. A reply is a list with the components "cmd" (command), "type" (content type) and "content". Most common commands are "OUT" (text output), "GRAPH" (plot output) and "STAT" (end of evaluation, "status" change). Common types are "TEXT", "RTF" and "HTML" for text output types, "PNG" for graph output and "NULL" if the reply has no contents.

`gsio.greeting`: Greeting reply contents. Messenger verisons 1.1 and higher will reply with a list at least with the components `version` and `outputType`. The latter can have values "TEXT", "HTML" and "RTF". Older versions will only return a string.

See Also

```
link{gsio.msg() }
```

processOutput

implement this function to handle the GS output

Description

implement this function to handle the GS output

Usage

```
processOutput(msg, io, saveConfig = list(enabled = FALSE))
```

stripGenVerbatim	<i>Function to strip stuff that looks like this:<PRE></PRE> from the output.</i>
------------------	---

Description

Function to strip stuff that looks like this:<PRE></PRE> from the output.

Usage

```
stripGenVerbatim(output)
```

Index

`close.gsio(gsio)`, [3](#)

`extractTablesToEnv`, [2](#)

`fixTableCols`, [2](#)

`gs.engine`, [3](#)

`gsio`, [3](#)

`gsio.connect()`, [4](#)

`processOutput`, [5](#)

`stripGenVerbatim`, [5](#)