Package 'rjazz'

October 14, 2022

Type Package

Title Official Client for 'Jazz'

Version 0.1.7

Date 2018-03-19

Depends R (>= 3.1.0)

Imports stats, RCurl

Description This is the official 'Jazz' client. 'Jazz' is a lightweight modular data processing framework, including a web server. It provides data persistence and computation capabilities accessible from 'R' and 'Python' and also through a REST API. <https://github.com/bbvadata/Jazz>
See ?rjazz::rjazz to get a 'Jazz' server.

License Apache License (== 2.0)

NeedsCompilation no

Author Santiago Basaldua [aut, cre],
Banco Bilbao Vizcaya Argentaria, S.A. [cph]

Maintainer Santiago Basaldua <kaalam@kaalam.ai>

Repository CRAN

Date/Publication 2018-03-21 11:25:10 UTC

R topics documented:

rjazz-package 2
create_block_rep 3
create_block_seq 4
create_error_page 5
create_source 6
create_web_resource 7
delete_block 8
delete_source 9
delete_web_source 10
get_block_as_string 11
get_block_attributes 13

get_raw_block 15
get_R_block 16
get_server_version 17
license 18
list_sources 18
list_web_sources 19
new_key 20
notice 20
put_block_flags 21
put_raw_block 22
put_R_block 24
put_strings_as_block 25
set_compatible_data_type 27
set_jazz_host 29
type_const 30

Index **31**

`rjazz-package` *Official Client for Jazz*

Description

How to get a Jazz server

Currently, in March 2018, Jazz is in the middle of major refactoring. The client is released mainly for supporting Jazz development, evaluation purposes and keeping a stable package name for the future. Unless you are interested in Jazz core development, we recommend you to wait until version 0.3.1 before giving Jazz a serious evaluation.

Run a docker image from docker hub (hub.docker.com)

This is fastest way for evaluation purposes. Check dockerhub to find the name of the latest image.

docker run -p8888:8888 kaalam/jazz_neat:0.2.1.99

Compile from source

The most controlable way is to compile the server from source.

git clone https://github.com/bbvadata/Jazz
cd Jazz
./config.sh
cd server
make jazz
./jazz start

You can point a browser at `http://localhost:8888///` to check if the server is running.

create_block_rep *Create a data block in the server by repeating a value a number of times*

Description

Creates a (boolan, integer, real or string) data block in the server and stores it in the persistence by repeating a value a number of times. The type of the block can later be changed to a compatible type: BLOCKTYPE_C_INTEGER to BLOCKTYPE_C_FACTOR or BLOCKTYPE_C_GRADE and BLOCKTYPE_C_REAL to BLOCKTYPE_C_TIMESEC by a later call to set_compatible_data_type().

Usage

```
create_block_rep(source, block_key, val, times, host = .host.)
```

Arguments

source	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
block_key	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
val	The (boolan, integer, real of string) value to be repeated. A single element of type 'logical', 'integer', 'numeric' or 'character'.
times	The number of times to be repeated. A number.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

TRUE or raises an error on failure.

Examples

```
## Not run:
create_source('demo_put')

create_block_rep('demo_put', 'bool_1', TRUE, 3)
any(rep(TRUE, 3) != get_R_block('demo_put', 'bool_1'))

create_block_rep('demo_put', 'int_1', 2L, 4)
any(rep(2L, 4) != get_R_block('demo_put', 'int_1'))

create_block_rep('demo_put', 'real_1', 3.14, 5)
any(rep(3.14, 5) != get_R_block('demo_put', 'real_1'))
```

```
create_block_rep('demo_put', 'str_1', 'Hi!', 6)
any(rep('Hi!', 6) != get_R_block('demo_put', 'str_1'))

create_block_seq('demo_put', 'int_2', 456L, 999L, 123L)
any(seq(456L, 999L, 123L) != get_R_block('demo_put', 'int_2'))

create_block_seq('demo_put', 'real_2', 0.123, 4.56, 0.789)
any(seq(0.123, 4.56, 0.789) != get_R_block('demo_put', 'real_2'))

delete_source('demo_put')

## End(Not run)
```

`create_block_seq` *Create a data block in the server with a simple linear sequence*

Description

Creates an (integer or real) data block in the server and stores it in the persistence with a simple linear sequence. The type of the block can later be changed to a compatible type: BLOCKTYPE_C_INTEGER to BLOCKTYPE_C_FACTOR or BLOCKTYPE_C_GRADE and BLOCKTYPE_C_REAL to BLOCKTYPE_C_TIMESEC by a later call to set_compatible_data_type().

Usage

```
create_block_seq(source, block_key, from, to, by = 1, host = .host.)
```

Arguments

`source`	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
`block_key`	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
`from`	The starting value. A real number or an integer.
`to`	The end value, may not be included, it is the supremum or the infimum when 'by' is negative.
`by`	The increment. It may be negative, in that case 'from' must be bigger than 'to'.
`host`	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

TRUE or raises an error on failure.

Examples

```
## Not run:
create_source('demo_put')

create_block_rep('demo_put', 'bool_1', TRUE, 3)
any(rep(TRUE, 3) != get_R_block('demo_put', 'bool_1'))

create_block_rep('demo_put', 'int_1', 2L, 4)
any(rep(2L, 4) != get_R_block('demo_put', 'int_1'))

create_block_rep('demo_put', 'real_1', 3.14, 5)
any(rep(3.14, 5) != get_R_block('demo_put', 'real_1'))

create_block_rep('demo_put', 'str_1', 'Hi!', 6)
any(rep('Hi!', 6) != get_R_block('demo_put', 'str_1'))

create_block_seq('demo_put', 'int_2', 456L, 999L, 123L)
any(seq(456L, 999L, 123L) != get_R_block('demo_put', 'int_2'))

create_block_seq('demo_put', 'real_2', 0.123, 4.56, 0.789)
any(seq(0.123, 4.56, 0.789) != get_R_block('demo_put', 'real_2'))

delete_source('demo_put')

## End(Not run)
```

---

create_error_page *Customize the error pages on the server*

Description

Defines a new page for any http error status. These pages are always html (mime = text/html) but can use any number of resources (css, png, js, ...) which must be uploaded appropriately using link_web_resource().

Usage

```
create_error_page(http_status, html, host = .host.)
```

Arguments

http_status	The http status code returned as an error. Only those specified in Jazz API. E.g., 400 (Bad Request) - Syntactical error at top level. (Malformed URI)
html	The html page to be served for that error.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

Returns TRUE on success or throws an error on failure.

Examples

```
## Not run:
page <- '<html>\n<body>\n<br/>Resource was not found on this node.\n</body>\n</html>'
create_error_page(404, page)

## End(Not run)
```

`create_source` *Create a new source on the server*

Description

Creates a new source on the server. You must be authorized to create sources. The server has a maximum number of sources that is configured through the C++ MAX_POSSIBLE_SOURCES and the configuration variable MDB_ENV_SET_MAXDBS including 'sys' and 'www'.

Usage

```
create_source(source, host = .host.)
```

Arguments

`source`	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
`host`	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

TRUE or raises an error on failure.

Examples

```
## Not run:
create_source('demo_put')

create_block_rep('demo_put', 'bool_1', TRUE, 3)
any(rep(TRUE, 3) != get_R_block('demo_put', 'bool_1'))

create_block_rep('demo_put', 'int_1', 2L, 4)
any(rep(2L, 4) != get_R_block('demo_put', 'int_1'))

create_block_rep('demo_put', 'real_1', 3.14, 5)
```

```
any(rep(3.14, 5) != get_R_block('demo_put', 'real_1'))

create_block_rep('demo_put', 'str_1', 'Hi!', 6)
any(rep('Hi!', 6) != get_R_block('demo_put', 'str_1'))

create_block_seq('demo_put', 'int_2', 456L, 999L, 123L)
any(seq(456L, 999L, 123L) != get_R_block('demo_put', 'int_2'))

create_block_seq('demo_put', 'real_2', 0.123, 4.56, 0.789)
any(seq(0.123, 4.56, 0.789) != get_R_block('demo_put', 'real_2'))

delete_source('demo_put')

## End(Not run)
```

`create_web_resource` *High level create a new association of an url from an object*

Description

This completely adds a resource to the web interface. The resource has to be created inside a web source. A block key is created, the resource is uploaded and the link with its url is created.

Usage

```
create_web_resource(web_source, url, type, raw_object, lang = NULL, host = .host.)
```

Arguments

`web_source`	The name of the web source where the url and object will be included. All www resource links and urls are grouped under a "web source" which is just a name to allow removing them with a single call.
`url`	The url that will be used by the server to return the resource with a GET call. These urls cannot start with the name of an existing source or the API will try to execute them as API calls.
`type`	The mime type of the resource. A constant in type_const should be used rather than the corresponding integer value.
`raw_object`	The object to be uploaded via an http PUT call. Interpret "raw" as in "as is". Most cases will be strings, some R raw objects for binary PUT are also possible.
`lang`	The http language definition for the resource in case it has to be defined. Default value is not defined. Valid strings are: "en-US", "es-ES", .. or just two letters as in "jp".
`host`	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

This function returns the newly created key value for further use or throws an error on failure.

Examples

```
## Not run:
set_jazz_host('127.0.0.1:8888')
page <- '<html>\n<body>\n<br/>Hello world!\n</body>\n</html>'
create_web_resource('my_test',
                    '/my_test/hello.html',
                    type_const[['BLOCKTYPE_RAW_MIME_HTML']],
                    page)
# See http://127.0.0.1:8888/my_test/hello.html with a browser.

list_web_sources()
delete_web_source('my_test')

## End(Not run)
```

---

delete_block *Delete a block on the server*

Description

Deletes a block on the server. You must own the block to be able to delete it. It is the lowest level function and it can create harm on the server to delete blocks belonging to meshes or API functions.

Usage

```
delete_block(source, block_key, host = .host., silent = FALSE)
```

Arguments

source	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
block_key	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().
silent	(Optional) If this is TRUE, the function returns FALSE instead of throwing an error in case the corresponding PUT or DELETE function returns false.

Value

Returns TRUE if successful. When silent == FALSE (default) throws an error on any failure. Otherwise, it returns FALSE when the corresponding PUT or DELETE function returns false.

Examples

```
## Not run:
create_source('demo_bin')

# When a string is written into a raw block, charToRaw() is applied automatically.
put_raw_block('demo_bin', 'blk_1', 'Hello world!')

a <- get_raw_block('demo_bin', 'blk_1')
# a is raw
rawToChar(a)

# This is the same.
put_raw_block('demo_bin', 'blk_2', charToRaw('Hello again!'))
rawToChar(get_raw_block('demo_bin', 'blk_2'))

# Anything else can be written by serializing as raw.
put_raw_block('demo_bin', 'blk_3', serialize(iris, NULL))

head(unserialize(get_raw_block('demo_bin', 'blk_3')))

# Delete the block or fail
delete_block('demo_bin', 'blk_1')

# Delete will fail, but make it silent
delete_block('demo_bin', 'blk_1', silent = TRUE)

# No need to delete all blocks, they will be deleted by deleting the source.
delete_source('demo_bin')

## End(Not run)
```

---

delete_source *Delete a source on the server*

Description

Deletes a source on the server even if it is not empty. The sources 'sys' and 'www' cannot be deleted.

Usage

```
delete_source(source, host = .host., silent = FALSE)
```

Arguments

source	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.

host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().
silent	(Optional) If this is TRUE, the function returns FALSE instead of throwing an error in case the corresponding PUT or DELETE function returns false.

Value

Returns TRUE if successful. When silent == FALSE (default) throws an error on any failure. Otherwise, it returns FALSE when the corresponding PUT or DELETE function returns false.

Examples

```
## Not run:
create_source('demo_put')

create_block_rep('demo_put', 'bool_1', TRUE, 3)
any(rep(TRUE, 3) != get_R_block('demo_put', 'bool_1'))

create_block_rep('demo_put', 'int_1', 2L, 4)
any(rep(2L, 4) != get_R_block('demo_put', 'int_1'))

create_block_rep('demo_put', 'real_1', 3.14, 5)
any(rep(3.14, 5) != get_R_block('demo_put', 'real_1'))

create_block_rep('demo_put', 'str_1', 'Hi!', 6)
any(rep('Hi!', 6) != get_R_block('demo_put', 'str_1'))

create_block_seq('demo_put', 'int_2', 456L, 999L, 123L)
any(seq(456L, 999L, 123L) != get_R_block('demo_put', 'int_2'))

create_block_seq('demo_put', 'real_2', 0.123, 4.56, 0.789)
any(seq(0.123, 4.56, 0.789) != get_R_block('demo_put', 'real_2'))

delete_source('demo_put')

## End(Not run)
```

---

delete_web_source *Remove a complete web source for the server's "www" source*

Description

Removes a complete web source for the server's "www" source. This also deletes all the resources allocated with the web source.

Usage

```
delete_web_source(web_source, host = .host., silent = FALSE)
```

Arguments

web_source	The name of the web source to be deleted. All www resource links and urls are grouped under a "web source" which is just a name to allow removing them with a single call.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().
silent	(Optional) If this is TRUE, the function returns FALSE instead of throwing an error in case the corresponding PUT or DELETE function returns false.

Details

This is a macro function wrapping calls around put_function().

Value

Returns TRUE if successful. When silent == FALSE (default) throws an error on any failure. Otherwise, it returns FALSE when the corresponding PUT or DELETE function returns false.

Examples

```
## Not run:
set_jazz_host('127.0.0.1:8888')
page <- '<html>\n<body>\n<br/>Hello world!\n</body>\n</html>'
create_web_resource('my_test',
                    '/my_test/hello.html',
                    type_const[['BLOCKTYPE_RAW_MIME_HTML']],
                    page)
# See http://127.0.0.1:8888/my_test/hello.html with a browser.

list_web_sources()
delete_web_source('my_test')

## End(Not run)
```

---

get_block_as_string *Get a data block as an R string*

Description

Converts a data block into an R string with an sprintf compatible format string controlling the precise output format.

Usage

```
get_block_as_string(source, block_key, fmt, host = .host.)
```

Arguments

source	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
block_key	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
fmt	The (sprintf() compatible) format to convert the data as strings. Note: Newlines are NOT added automatically, use \n to add a newline where necessary.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

The output string or raises an error on failure.

Examples

```
## Not run:
create_source('demo_types')

# Write a text file as a block.
txt <- c('Hi all,', '', 'This is a file.', '', 'bye,', 'me')
str <- paste(txt, collapse = '\n')
cat(str)

put_raw_block('demo_types', 'blk_1', str)

# The block is raw (not interpreted as data by the server) and can be converted to any raw type.
set_compatible_data_type('demo_types', 'blk_1', type_const[['BLOCKTYPE_RAW_MIME_TXT']])

# curl 127.0.0.1:8888//demo_types.blk_1 (or open in a in a browser)

get_block_attributes('demo_types', 'blk_1')

# The attribute flags is writable by the user.
put_block_flags('demo_types', 'blk_1', 123000444)

get_block_attributes('demo_types', 'blk_1')

# Unlike the previous block, this block is a data block.
put_R_block('demo_types', 'blk_2', 3:6)

# This trivial block can also be created by the server as..
create_block_seq('demo_types', 'blk_2', 3L, 6)

get_block_attributes('demo_types', 'blk_2')

# The block is interpreted as data by the server, it is an integer and can be converted to
any integer type.
```

```
set_compatible_data_type('demo_types', 'blk_2', type_const[['BLOCKTYPE_C_R_GRADE']])

get_block_attributes('demo_types', 'blk_2')

# This returns all the rows in a single string
get_block_as_string('demo_types', 'blk_2', '

# With some help of R functions, the result of get_block_as_string() can be made integer again.
any(3:6 != as.integer(strsplit(get_block_as_string('demo_types', 'blk_2', '

rs <- c('1', '2.7', '3.14')

# Creating strings into numeric data. (The parse(.., collapse = '\n') is automatic.)
put_strings_as_block('demo_types', 'blk_3', rs, type_const[['BLOCKTYPE_C_R_REAL']])

get_block_attributes('demo_types', 'blk_3')

any(as.numeric(rs) != get_R_block('demo_types', 'blk_3'))

delete_source('demo_types')

## End(Not run)
```

`get_block_attributes` *Get the (header) attributes of a block*

Description

Gets the following attributes of a block: type, length, size, flags and hash64 as an R list.

Usage

```
get_block_attributes(source, block_key, host = .host.)
```

Arguments

`source`	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
`block_key`	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
`host`	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

An R list with the attributes: type, length, size, flags and hash64 or raises an error on failure.

Examples

```
## Not run:
create_source('demo_types')

# Write a text file as a block.
txt <- c('Hi all,', '', 'This is a file.', '', 'bye,', 'me')
str <- paste(txt, collapse = '\n')
cat(str)

put_raw_block('demo_types', 'blk_1', str)

# The block is raw (not interpreted as data by the server) and can be converted to any raw type.
set_compatible_data_type('demo_types', 'blk_1', type_const[['BLOCKTYPE_RAW_MIME_TXT']])

# curl 127.0.0.1:8888//demo_types.blk_1 (or open in a in a browser)

get_block_attributes('demo_types', 'blk_1')

# The attribute flags is writable by the user.
put_block_flags('demo_types', 'blk_1', 123000444)

get_block_attributes('demo_types', 'blk_1')

# Unlike the previous block, this block is a data block.
put_R_block('demo_types', 'blk_2', 3:6)

# This trivial block can also be created by the server as..
create_block_seq('demo_types', 'blk_2', 3L, 6)

get_block_attributes('demo_types', 'blk_2')

# The block is interpreted as data by the server, it is an integer and can be converted to
any integer type.
set_compatible_data_type('demo_types', 'blk_2', type_const[['BLOCKTYPE_C_R_GRADE']])

get_block_attributes('demo_types', 'blk_2')

# This returns all the rows in a single string
get_block_as_string('demo_types', 'blk_2', '

# With some help of R functions, the result of get_block_as_string() can be made integer again.
any(3:6 != as.integer(strsplit(get_block_as_string('demo_types', 'blk_2', '

rs <- c('1', '2.7', '3.14')

# Creating strings into numeric data. (The parse(.., collapse = '\n') is automatic.)
put_strings_as_block('demo_types', 'blk_3', rs, type_const[['BLOCKTYPE_C_R_REAL']])

get_block_attributes('demo_types', 'blk_3')

any(as.numeric(rs) != get_R_block('demo_types', 'blk_3'))
```

```
delete_source('demo_types')

## End(Not run)
```

---

get_raw_block *Generic (low level) GET call to a block in the block API*

Description

Writes data from a block persisted in the server.

Usage

```
get_raw_block(source, block_key, host = .host., buffsize = 1048576)
```

Arguments

source	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
block_key	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().
buffsize	(Default = 1 Mb) the size of the buffer to download binary objects. Must be bigger or equal to the size of the block downloaded.

Value

An R raw object or throws an exception on failure.

Examples

```
## Not run:
create_source('demo_bin')

# When a string is written into a raw block, charToRaw() is applied automatically.
put_raw_block('demo_bin', 'blk_1', 'Hello world!')

a <- get_raw_block('demo_bin', 'blk_1')
# a is raw
rawToChar(a)

# This is the same.
put_raw_block('demo_bin', 'blk_2', charToRaw('Hello again!'))
rawToChar(get_raw_block('demo_bin', 'blk_2'))
```

```
# Anything else can be written by serializing as raw.
put_raw_block('demo_bin', 'blk_3', serialize(iris, NULL))

head(unserialize(get_raw_block('demo_bin', 'blk_3')))

# Delete the block or fail
delete_block('demo_bin', 'blk_1')

# Delete will fail, but make it silent
delete_block('demo_bin', 'blk_1', silent = TRUE)

# No need to delete all blocks, they will be deleted by deleting the source.
delete_source('demo_bin')

## End(Not run)
```

`get_R_block` *Read a data block as an R object*

Description

Reads a data block as an R object. The server automatically converts the appropriate data type into logical, integer, numeric or character.

Usage

```
get_R_block(source, block_key, host = .host., buffsize = 1048576)
```

Arguments

`source`	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
`block_key`	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
`host`	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().
`buffsize`	(Default = 1 Mb) the size of the buffer to download binary objects. Must be bigger or equal to the size of the block downloaded.

Value

Returns an R object of type logical, integer, numeric or character or raises an error on failure.

Examples

```
## Not run:
create_source('demo_put')

create_block_rep('demo_put', 'bool_1', TRUE, 3)
any(rep(TRUE, 3) != get_R_block('demo_put', 'bool_1'))

create_block_rep('demo_put', 'int_1', 2L, 4)
any(rep(2L, 4) != get_R_block('demo_put', 'int_1'))

create_block_rep('demo_put', 'real_1', 3.14, 5)
any(rep(3.14, 5) != get_R_block('demo_put', 'real_1'))

create_block_rep('demo_put', 'str_1', 'Hi!', 6)
any(rep('Hi!', 6) != get_R_block('demo_put', 'str_1'))

create_block_seq('demo_put', 'int_2', 456L, 999L, 123L)
any(seq(456L, 999L, 123L) != get_R_block('demo_put', 'int_2'))

create_block_seq('demo_put', 'real_2', 0.123, 4.56, 0.789)
any(seq(0.123, 4.56, 0.789) != get_R_block('demo_put', 'real_2'))

delete_source('demo_put')

## End(Not run)
```

---

get_server_version *Get the version of the Jazz server*

Description

Returns the version of the Jazz server and throws a warning if it does not match the (major version, minor version) of the R client.

Usage

```
get_server_version(host = .host., full = FALSE)
```

Arguments

host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().
full	Returns a list of server properties, including: version, build (DEBUG or RELEASE), artifact (name of the OS where it was built), myname (node name in Jazz), sysname (Linux), hostname (name of the running host), kernel (linux kernel), sysvers (detailed build of the OS), machine (processor type and size of the pointers).

Value

Returns the version of the Jazz server when full = FALSE or a list of server properties if TRUE. Throws an error on failure and a warning if it does not match the (major version, minor version) of the R client.

Examples

```
## Not run:
get_server_version()
get_server_version(full=T)

## End(Not run)
```

---

license *BBVA - Jazz: A lightweight analytical web server for data-driven applications - R client.*

Description

---

list_sources *List all the sources on the Jazz server*

Description

Lists all the Jazz server sources including the 'sys' and 'www' sources. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.

Usage

```
list_sources(host = .host.)
```

Arguments

host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

Returns the server's sources as a vector of string.

Examples

```
## Not run:
list_sources()

## End(Not run)
```

---

list_web_sources *List all the web sources on the Jazz server*

Description

Lists all the web sources on the Jazz server. All www resource links and urls are grouped under a "web source" which is just a name to allow removing them with a single call.

Usage

```
list_web_sources(host = .host.)
```

Arguments

host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

Returns the web sources as a vector of string.

Examples

```
## Not run:
set_jazz_host('127.0.0.1:8888')
page <- '<html>\n<body>\n<br/>Hello world!\n</body>\n</html>'
create_web_resource('my_test',
                    '/my_test/hello.html',
                    type_const[['BLOCKTYPE_RAW_MIME_HTML']],
                    page)
# See http://127.0.0.1:8888/my_test/hello.html with a browser.

list_web_sources()
delete_web_source('my_test')
```

```
## End(Not run)
```

---

new_key *Create a key for a new block using a RNG*

Description

Creates a key for a new block using a RNG. The function uses runif() and does not perform any seed initialization. See .Random.seed for more information on R's random number generation algorithms.

Usage

```
new_key(host = .host.)
```

Arguments

host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

This function returns the newly created key value for further use.

Examples

```
## Not run:
new_key()
new_key()
new_key()

## End(Not run)
```

---

notice *BBVA - Jazz: A lightweight analytical web server for data-driven applications - R client.*

Description

put_block_flags *Write flags into a block's header*

Description

Writes a 32 bit integer named flags into a block's header. The server does not use that value in the block API.

Usage

```
put_block_flags(source, block_key, flags, host = .host.)
```

Arguments

source	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
block_key	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
flags	The (integer) value written into the block's header as the flags attribute.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

TRUE or raises an error on failure.

Examples

```
## Not run:
create_source('demo_types')

# Write a text file as a block.
txt <- c('Hi all,', '', 'This is a file.', '', 'bye,', 'me')
str <- paste(txt, collapse = '\n')
cat(str)

put_raw_block('demo_types', 'blk_1', str)

# The block is raw (not interpreted as data by the server) and can be converted to any raw type.
set_compatible_data_type('demo_types', 'blk_1', type_const[['BLOCKTYPE_RAW_MIME_TXT']])

# curl 127.0.0.1:8888//demo_types.blk_1 (or open in a in a browser)

get_block_attributes('demo_types', 'blk_1')

# The attribute flags is writable by the user.
```

```
put_block_flags('demo_types', 'blk_1', 123000444)

get_block_attributes('demo_types', 'blk_1')

# Unlike the previous block, this block is a data block.
put_R_block('demo_types', 'blk_2', 3:6)

# This trivial block can also be created by the server as..
create_block_seq('demo_types', 'blk_2', 3L, 6)

get_block_attributes('demo_types', 'blk_2')

# The block is interpreted as data by the server, it is an integer and can be converted to
any integer type.
set_compatible_data_type('demo_types', 'blk_2', type_const[['BLOCKTYPE_C_R_GRADE']])

get_block_attributes('demo_types', 'blk_2')

# This returns all the rows in a single string
get_block_as_string('demo_types', 'blk_2', '

# With some help of R functions, the result of get_block_as_string() can be made integer again.
any(3:6 != as.integer(strsplit(get_block_as_string('demo_types', 'blk_2', '

rs <- c('1', '2.7', '3.14')

# Creating strings into numeric data. (The parse(.., collapse = '\n') is automatic.)
put_strings_as_block('demo_types', 'blk_3', rs, type_const[['BLOCKTYPE_C_R_REAL']])

get_block_attributes('demo_types', 'blk_3')

any(as.numeric(rs) != get_R_block('demo_types', 'blk_3'))

delete_source('demo_types')

## End(Not run)
```

`put_raw_block` *Write a raw object or a string as a block.*

Description

Writes a raw object or a string as a block in persistence. That block can be stored as raw, converted to a compatible raw type or converted into data by the server.

Usage

```
put_raw_block(source, block_key, block_val, host = .host.)
```

Arguments

source	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
block_key	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
block_val	The content of the block. This function is for data blocks and the type is automatic. The block must be an array of: boolean, integer, double or string. For blocks other than data blocks, such as web resources, use the appropriate function.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

TRUE or raises an error on failure.

Examples

```
## Not run:
create_source('demo_bin')

# When a string is written into a raw block, charToRaw() is applied automatically.
put_raw_block('demo_bin', 'blk_1', 'Hello world!')

a <- get_raw_block('demo_bin', 'blk_1')
# a is raw
rawToChar(a)

# This is the same.
put_raw_block('demo_bin', 'blk_2', charToRaw('Hello again!'))
rawToChar(get_raw_block('demo_bin', 'blk_2'))

# Anything else can be written by serializing as raw.
put_raw_block('demo_bin', 'blk_3', serialize(iris, NULL))

head(unserialize(get_raw_block('demo_bin', 'blk_3')))

# Delete the block or fail
delete_block('demo_bin', 'blk_1')

# Delete will fail, but make it silent
delete_block('demo_bin', 'blk_1', silent = TRUE)

# No need to delete all blocks, they will be deleted by deleting the source.
delete_source('demo_bin')

## End(Not run)
```

put_R_block *Write an R object as a data block*

Description

Writes an R object as a data block of type BLOCKTYPE_C_BOOL, BLOCKTYPE_C_OFFS_CHARS, BLOCKTYPE_C_R_INTEGER or BLOCKTYPE_C_R_REAL for R objects of type logical, character, integer or numeric.

Usage

```
put_R_block(source, block_key, sexp, host = .host.)
```

Arguments

source	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
block_key	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
sexp	The R object written into the block. Must be logical, character, integer or numeric.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

TRUE or raises an error on failure.

Examples

```
## Not run:
create_source('demo_types')

# Write a text file as a block.
txt <- c('Hi all,', '', 'This is a file.', '', 'bye,', 'me')
str <- paste(txt, collapse = '\n')
cat(str)

put_raw_block('demo_types', 'blk_1', str)

# The block is raw (not interpreted as data by the server) and can be converted to any raw type.
set_compatible_data_type('demo_types', 'blk_1', type_const[['BLOCKTYPE_RAW_MIME_TXT']])

# curl 127.0.0.1:8888//demo_types.blk_1 (or open in a in a browser)

get_block_attributes('demo_types', 'blk_1')
```

```
# The attribute flags is writable by the user.
put_block_flags('demo_types', 'blk_1', 123000444)

get_block_attributes('demo_types', 'blk_1')

# Unlike the previous block, this block is a data block.
put_R_block('demo_types', 'blk_2', 3:6)

# This trivial block can also be created by the server as..
create_block_seq('demo_types', 'blk_2', 3L, 6)

get_block_attributes('demo_types', 'blk_2')

# The block is interpreted as data by the server, it is an integer and can be converted to
any integer type.
set_compatible_data_type('demo_types', 'blk_2', type_const[['BLOCKTYPE_C_R_GRADE']])

get_block_attributes('demo_types', 'blk_2')

# This returns all the rows in a single string
get_block_as_string('demo_types', 'blk_2', '

# With some help of R functions, the result of get_block_as_string() can be made integer again.
any(3:6 != as.integer(strsplit(get_block_as_string('demo_types', 'blk_2', '

rs <- c('1', '2.7', '3.14')

# Creating strings into numeric data. (The parse(.., collapse = '\n') is automatic.)
put_strings_as_block('demo_types', 'blk_3', rs, type_const[['BLOCKTYPE_C_R_REAL']])

get_block_attributes('demo_types', 'blk_3')

any(as.numeric(rs) != get_R_block('demo_types', 'blk_3'))

delete_source('demo_types')

## End(Not run)
```

`put_strings_as_block` *Write a vector of strings as a data block*

Description

Writes a data block by converting a vector of strings into a vector of the appropriate type.

Usage

```
put_strings_as_block(source, block_key, txt, type, fmt = NA, host = .host.)
```

Arguments

source	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.
block_key	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
txt	A single string with the rows separated by \n or a vector of strings. The latter will be converted into the former automatically. The block will initially be a block of strings and then be converted into a binary block.
type	The destination type. Possible formats are: BLOCKTYPE_C_BOOL..BLOCKTYPE_C_R_REAL.
fmt	(Optional) The (sscanf() compatible) format to convert the strings to binary data. Note: Each input line produces one data element row by row. Newline is not part of fmt. By default, this is set automatically to the simplest format for the type assuming strings do not contain any other characters than the decimal representation of the numbers.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

TRUE or raises and error on failure.

Examples

```
## Not run:
create_source('demo_types')

# Write a text file as a block.
txt <- c('Hi all,', '', 'This is a file.', '', 'bye,', 'me')
str <- paste(txt, collapse = '\n')
cat(str)

put_raw_block('demo_types', 'blk_1', str)

# The block is raw (not interpreted as data by the server) and can be converted to any raw type.
set_compatible_data_type('demo_types', 'blk_1', type_const[['BLOCKTYPE_RAW_MIME_TXT']])

# curl 127.0.0.1:8888//demo_types.blk_1 (or open in a in a browser)

get_block_attributes('demo_types', 'blk_1')

# The attribute flags is writable by the user.
put_block_flags('demo_types', 'blk_1', 123000444)

get_block_attributes('demo_types', 'blk_1')

# Unlike the previous block, this block is a data block.
put_R_block('demo_types', 'blk_2', 3:6)
```

```
# This trivial block can also be created by the server as..
create_block_seq('demo_types', 'blk_2', 3L, 6)

get_block_attributes('demo_types', 'blk_2')

# The block is interpreted as data by the server, it is an integer and can be converted to
any integer type.
set_compatible_data_type('demo_types', 'blk_2', type_const[['BLOCKTYPE_C_R_GRADE']])

get_block_attributes('demo_types', 'blk_2')

# This returns all the rows in a single string
get_block_as_string('demo_types', 'blk_2', '

# With some help of R functions, the result of get_block_as_string() can be made integer again.
any(3:6 != as.integer(strsplit(get_block_as_string('demo_types', 'blk_2', '

rs <- c('1', '2.7', '3.14')

# Creating strings into numeric data. (The parse(.., collapse = '\n') is automatic.)
put_strings_as_block('demo_types', 'blk_3', rs, type_const[['BLOCKTYPE_C_R_REAL']])

get_block_attributes('demo_types', 'blk_3')

any(as.numeric(rs) != get_R_block('demo_types', 'blk_3'))

delete_source('demo_types')

## End(Not run)
```

`set_compatible_data_type` *Change the type of a data block to a binary compatible type*

Description

Changes the type of a data block to a binary compatible type without changing its contents. This is only valid between integer types (integer, sorted and categorical) and real types (double and time). This can also be used to change within raw types.

Usage

```
set_compatible_data_type(source, block_key, type, host = .host.)
```

Arguments

`source`	The Jazz source. Jazz persistence is organized in sources. All sources except 'sys' and 'www' are user defined. Sources are 15 char long alphanumeric or underscore.

block_key	The key identifying the block. Keys are 15 alphanumeric or underscore characters. They can be user defined or created by new_key(). Also, meshes use block keys internally.
type	The new type of the block. Must be binary compatible with the exiting type. BLOCKTYPE_C_INTEGER to BLOCKTYPE_C_FACTOR or BLOCKTYPE_C_GRADE and BLOCKTYPE_C_REAL to BLOCKTYPE_C_TIMESEC.
host	(Optional) the name of the jazz server host (including the port). Usually set just once via set_jazz_host().

Value

TRUE or raises an error on failure.

Examples

```
## Not run:
create_source('demo_types')

# Write a text file as a block.
txt <- c('Hi all,', '', 'This is a file.', '', 'bye,', 'me')
str <- paste(txt, collapse = '\n')
cat(str)

put_raw_block('demo_types', 'blk_1', str)

# The block is raw (not interpreted as data by the server) and can be converted to any raw type.
set_compatible_data_type('demo_types', 'blk_1', type_const[['BLOCKTYPE_RAW_MIME_TXT']])

# curl 127.0.0.1:8888//demo_types.blk_1 (or open in a in a browser)

get_block_attributes('demo_types', 'blk_1')

# The attribute flags is writable by the user.
put_block_flags('demo_types', 'blk_1', 123000444)

get_block_attributes('demo_types', 'blk_1')

# Unlike the previous block, this block is a data block.
put_R_block('demo_types', 'blk_2', 3:6)

# This trivial block can also be created by the server as..
create_block_seq('demo_types', 'blk_2', 3L, 6)

get_block_attributes('demo_types', 'blk_2')

# The block is interpreted as data by the server, it is an integer and can be converted to
any integer type.
set_compatible_data_type('demo_types', 'blk_2', type_const[['BLOCKTYPE_C_R_GRADE']])

get_block_attributes('demo_types', 'blk_2')
```

```
# This returns all the rows in a single string
get_block_as_string('demo_types', 'blk_2', '

# With some help of R functions, the result of get_block_as_string() can be made integer again.
any(3:6 != as.integer(strsplit(get_block_as_string('demo_types', 'blk_2', '

rs <- c('1', '2.7', '3.14')

# Creating strings into numeric data. (The parse(.., collapse = '\n') is automatic.)
put_strings_as_block('demo_types', 'blk_3', rs, type_const[['BLOCKTYPE_C_R_REAL']])

get_block_attributes('demo_types', 'blk_3')

any(as.numeric(rs) != get_R_block('demo_types', 'blk_3'))

delete_source('demo_types')

## End(Not run)
```

`set_jazz_host` *Set the name of the Jazz server to avoid passing it in all function calls*

Description

Sets the name of the Jazz server to avoid passing it in all function calls. It simply assigns it to the global variable .host.

Usage

```
set_jazz_host(host)
```

Arguments

`host`	The name of the jazz server host (including the port).

Value

Returns TRUE if the argument is a single string. No other checks done.

Examples

```
## Not run:
set_jazz_host('127.0.0.1:8888')
page <- '<html>\n<body>\n<br/>Hello world!\n</body>\n</html>'
create_web_resource('my_test',
                    '/my_test/hello.html',
                    type_const[['BLOCKTYPE_RAW_MIME_HTML']],
                    page)
# See http://127.0.0.1:8888/my_test/hello.html with a browser.
```

```
list_web_sources()
delete_web_source('my_test')

## End(Not run)
```

---

type_const *A set of server constants stored in a list*

Description

This is a global variable of type list that contains a set of constants used in the Jazz API. All constants are integers. Direct usage of the integer values is possible but not recommended.

Examples

```
## Not run:
type_const[['BLOCKTYPE_RAW_MIME_HTML']]

## End(Not run)
```

Index

create_block_rep, 3
create_block_seq, 4
create_error_page, 5
create_source, 6
create_web_resource, 7

delete_block, 8
delete_source, 9
delete_web_source, 10

get_block_as_string, 11
get_block_attributes, 13
get_R_block, 16
get_raw_block, 15
get_server_version, 17

license, 18
list_sources, 18
list_web_sources, 19

new_key, 20
notice, 20

put_block_flags, 21
put_R_block, 24
put_raw_block, 22
put_strings_as_block, 25

rjazz *(*rjazz-package*)*, 2
rjazz-package, 2

set_compatible_data_type, 27
set_jazz_host, 29

type_const, 30