Package 'rbedrock'

August 25, 2025

Title Analysis and Manipulation of Data from Minecraft Bedrock Edition

Version 0.4.1

Description Implements an interface to Minecraft (Bedrock Edition) worlds. Supports the analysis and management of these worlds and game saves.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.2

SystemRequirements cmake, zlib, GNU make

NeedsCompilation yes

Depends R (>= 3.6.0)

Imports R6, tibble, bit64, rappdirs, rlang, utils, digest

Suggests spelling, zip, testthat, jsonlite, covr

URL https://reedacartwright.github.io/rbedrock/,
https://github.com/reedacartwright/rbedrock

BugReports https://github.com/reedacartwright/rbedrock/issues

Author Reed Cartwright [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0837-9380>),
Rich FitzJohn [ctb],
Christian Stigen Larsen [ctb],
The LevelDB Authors [cph]

Maintainer Reed Cartwright <racartwright@gmail.com>

Repository CRAN

Date/Publication 2025-08-24 23:10:07 UTC

Contents

`ActorDigest` *Read and write Actor Digest Data*

Description

Actor digests store a list of all entities in a chunk; however they are not chunk data and use their own prefix. The key format for actor digest data is acdig:x:z:dimension.

`get_acdig_data()` and `get_acdig_value()` load ActorDigest data from `db`. `get_acdig_value()` supports loading only a single value.

`put_acdig_data()` and `put_acdig_value()` store ActorDigest data into `db`.

`read_acdig_value()` and `write_acdig_value()` decode and encode ActorDigest data respectively.

`create_acdig_keys()` creates keys for ActorDigest data.

Usage

```
get_acdig_data(x, z, dimension, db = default_db())

get_acdig_value(x, z, dimension, db = default_db())

put_acdig_data(values, x, z, dimension, db = default_db())

put_acdig_value(value, x, z, dimension, db = default_db())

read_acdig_value(rawvalue)

write_acdig_value(value)

create_acdig_keys(x, z, dimension)
```

Arguments

`x`, `z`, `dimension`	Chunk coordinates to extract data from. `x` can also be a character vector of db keys.
`db`	A bedrockdb object.
`values`	A list of character vectors. If `x` is missing, the names of `values` will be taken as the keys.
`value`	A character vector.
`rawvalue`	A raw vector.

Value

`get_acdig_values()` returns a vector of actor keys. `get_acdig_data()` returns a named list of the of the values returned by `get_acdig_value()`.

See Also

Actors, Entity

`Actors` *Read and write Actor data*

Description

After 1.18.30, the nbt data of each actor is saved independently in the database, using a key with a prefix and a 16-character storage key: 'actor:0123456789abcdef'. The keys of all actors in a chunk are saved in an ActorDigest record, with format acdig:x:z:dimension'.

Usage

```
get_actors_data(x, z, dimension, db = default_db())

get_actors_value(x, z, dimension, db = default_db())

put_actors_data(values, x, z, dimension, db = default_db())

put_actors_value(value, x, z, dimension, db = default_db())
```

Arguments

`x, z, dimension`	Chunk coordinates to extract data from. `x` can also be a character vector of db keys.
`db`	A bedrockdb object.
`values`	A list of character vectors. If `x` is missing, the names of `values` will be taken as the keys.
`value`	A list of nbt actors data

Details

`get_actors_value()` loads Actors data for a single chunk in `db`. `get_actors_data()` loads Actors data from multiple chunks in `db`.

`put_actors_value()` and `put_actors_data()` store one/multiple chunks Actors data into `db` and update the chunks' ActorDigests. When storing Actors data, an actor's storage key will be recalculated from the actor's `UniqueID`. The actor's position and dimension are not verified to be in the chunk it is assigned to.

See Also

ActorDigest, Entity

bedrockdb *Open a Bedrock Edition world for reading and writing.*

Description

bedrockdb opens a handle to a leveldb database that contains save-game data for a Bedrock Edition world. On success, it returns an R6 class of type 'bedrockdb' that can be used directly for low-level reading and writing access to the db or can be passed to higher-level functions. The handle to the database can be closed by passing it to close.

Usage

```
bedrockdb(
  path,
  create_if_missing = FALSE,
  error_if_exists = NULL,
  paranoid_checks = NULL,
  write_buffer_size = 4194304L,
  max_open_files = NULL,
  block_size = 163840L,
  cache_capacity = 83886080L,
  bloom_filter_bits_per_key = 10L,
  compression_level = -1L
)

## S3 method for class 'bedrockdb'
close(con, compact = FALSE, ...)

is_bedrockdb(x)
```

Arguments

path	The path to a world folder. If the path does not exist, it is assumed to be the base name of a world folder in the local minecraftWorlds directory.
create_if_missing	Create world database if it doesn't exist.
error_if_exists	Raise an error if the world database already exists.
paranoid_checks	Internal leveldb option
write_buffer_size	Internal leveldb option
max_open_files	Internal leveldb option
block_size	Internal leveldb option
cache_capacity	Internal leveldb option

bloom_filter_bits_per_key
: Internal leveldb option

compression_level
: Internal leveldb option

con
: An database object created by bedrockdb.

compact
: Compact database before closing.

...
: arguments passed to or from other methods.

x
: An object.

Value

On success, bedrockdb returns an R6 class of type 'bedrockdb'.

Examples

```
# open an example works and get all keys
dbpath <- rbedrock_example_world("example1.mcworld")
db <- bedrockdb(dbpath)
keys <- get_keys()
close(db)

## Not run:

# open a world in the minecraftWorlds folder using a world id.
db <- bedrockdb("lrkkYFpUABA=")
# do something with db ...
close(db)

# open a world using absolute path
db <- bedrockdb("C:\\\\minecraftWorlds\\\\my_world")
# do something with db ...
close(db)

## End(Not run)
```

---

bedrock_random *Random Number Generation for Minecraft*

Description

Bedrock Edition's central random number algorithm is MT19937. However, R's MT19937 code is not compatible with Bedrock's. These routines provide an API that is compatible with Bedrock's.

`bedrock_random_seed()` seeds the random number generator.

`bedrock_random_state()` returns the current state of the random number generator as a raw vector.

`bedrock_random_get_uint()` returns a 32-bit random integer. Default range is `[0, 2^32-1]`.

`bedrock_random_get_int()` returns a 31-bit random integer. Default range is `[0, 2^31-1]`.

`bedrock_random_get_float()` returns a random real number. Default range is `[0.0, 1.0)`.

`bedrock_random_get_double()` returns a random real number Default range is `[0.0, 1.0)`.

Usage

```
bedrock_random_seed(value)

bedrock_random_state(new_state = NULL)

bedrock_random_get_uint(n, max)

bedrock_random_get_int(n, min, max)

bedrock_random_get_float(n, min, max)

bedrock_random_get_double(n)
```

Arguments

value	a scalar integer
new_state	a raw vector
n	number of observations.
min, max	lower and upper limits of the distribution. Must be finite. If only one is specified, it is taken as max. If neither is specified, the default range is used.

Examples

```
# seed the global random number generator
bedrock_random_seed(5490L)

# save and restore rng state
saved_state <- bedrock_random_state()
bedrock_random_get_uint(10)
bedrock_random_state(saved_state)
bedrock_random_get_uint(10)
```

---

bedrock_random_create_seed

Random Number Seeds for Minecraft

Description

Minecraft uses several different kind of seeds during world generation and gameplay.

Usage

```
bedrock_random_create_seed(x, z, a, b, salt, type)
```

Arguments

x, z	chunk coordinates
a, b	seed parameters
salt	seed parameter
type	which seed type to use

Details

bedrock_random_create_seed() constructs a seed using the formulas type 1: x*a ^ z*b ^ salt, type 2: x*a + z*b + salt, and type 3: x*a + z*b ^ salt.

Examples

```
# identify slime chunks
g <- expand.grid(x=1:10, z=1:10)
is_slime_chunk <- mapply(g$x, g$z, FUN = function(x,z) {
  seed <- bedrock_random_create_seed(x,z,0x1f1f1f1f,1,0,type=1)
  bedrock_random_seed(seed)
  bedrock_random_get_uint(1,10) == 0
})
```

Biomes *Read and write Biomes data*

Description

Biomes data is stored as the second map in the Data3D data (tag 43).

Usage

```
get_biomes_value(x, z, dimension, db = default_db(), return_names = TRUE)

get_biomes_data(x, z, dimension, db = default_db(), return_names = TRUE)

put_biomes_value(
  value,
  x,
  z,
  dimension,
  db = default_db(),
  missing_height = 0L
)

put_biomes_data(
  values,
  x,
```

```
  z,
  dimension,
  db = default_db(),
  missing_height = 0L
)

biome_id(value)

biome_name(value)
```

Arguments

x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
db	A bedrockdb object.
return_names	return biome names instead of biome ids.
value	An array of biome ids.
missing_height	if there is no existing height data, use this value for the chunk.
values	A (named) list of Biomes data values. If x is missing, the names of values will be taken as the keys.

Details

- get_biomes_value() and get_biomes_data() load Biomes data from db. get_biomes_value() loads data for a single chunk, and get_biomes_data() loads data for multiple chunks.
- put_biomes_value() and put_biomes_data() store biomes data into db.

Value

get_biomes_value() returns a Biomes data value. get_biomes_data() returns a named list of Biomes data values. Biomes data values are 16x384x16 arrays containing biome data.

See Also

LegacyBiomes

---

biome_df *Bedrock biome data*

Description

Information about biomes used in Bedrock edition. Generated from the PyMCTranslate project. Colors are generated from the cubiomes project.

Usage

```
biome_df
```

Format

A data.frame with 87 rows and 5 columns.

bedrock_id The numeric id of the biome.

bedrock_name The name of the biome.

java_name The name of the equivalent biome in Java edition.

universal_name The universal name used for the biome in Amulet.

color The color used when mapping biomes.

Source

- https://github.com/gentlegiantJGC/PyMCTranslate/
- https://github.com/Cubitect/cubiomes/

BlockEntity *Load and store BlockEntity NBT data*

Description

BlockEntity data (tag 49) holds a list of NBT values for entity data associated with specific blocks.

Usage

```
get_block_entity_data(x, z, dimension, db = default_db())

get_block_entity_value(x, z, dimension, db = default_db())

put_block_entity_data(values, x, z, dimension, db = default_db())

put_block_entity_value(value, x, z, dimension, db = default_db())
```

Arguments

x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
db	A bedrockdb object.
values	A (named) list of list of NBT objects.
value	A list of NBT objects.

Details

- get_block_entity_value() and get_block_entity_data() load BlockEntity data from db. get_block_entity_value() loads data for a single chunk, and get_block_entity_data() loads data for multiple chunks.
- put_block_entity_value() and put_block_entity_data() store BlockEntity data for one or multiple chunks into db.

Value

`get_block_entity_value()` returns a list of NBT objects. `get_block_entity_data()` returns a named list of lists of NBT objects.

`Blocks` *Read and write Blocks data*

Description

Blocks data stores information about blocks in a world and their properties. Blocks data is stored per-subchunk as SubChunkBlocks data. These functions are wrappers around a SubChunkBlocks functions to make it easy to save and load blocks for an entire chunk.

Usage

```
get_blocks_value(
  x,
  z,
  dimension,
  db = default_db(),
  names_only = FALSE,
  extra_block = !names_only
)

get_blocks_data(
  x,
  z,
  dimension,
  db = default_db(),
  names_only = FALSE,
  extra_block = !names_only
)

put_blocks_value(value, x, z, dimension, db = default_db())

put_blocks_data(values, x, z, dimension, db = default_db())
```

Arguments

`x, z, dimension`	Chunk coordinates to extract data from. `x` can also be a character vector of db keys.
`db`	A bedrockdb object.
`names_only`	A logical scalar. Return only the names of the blocks, ignoring block states.
`extra_block`	A logical scalar. Append the extra block layer to the output (separated by ";"). This is mostly useful if you have waterlogged blocks. If the extra block is air, it will not be appended.

value	A 16x384x16 character array.
values	A (named) list of Blocks values. If x is missing, the names of values will be taken as the keys.

Details

- get_blocks_value() and get_blocks_data() load Blocks data from db. get_blocks_value() loads data for a single chunk, and get_blocks_data() loads data for multiple chunks.
- put_blocks_value() and put_blocks_data() store Blocks data into db.

Value

get_blocks_value() returns a Blocks value. get_blocks_data() returns a named list of Blocks values.

See Also

SubChunkBlocks

---

blocks_str *Convert block data from nbt format to a string*

Description

Block data is stored in nbt format, which includes a block's name and properties. blocks_str() converts the nbt data into strings with the following format: blockname@prop1=val1@prop2=val2. Blocks can have zero or more properties. blocks_nbt() does the inverse operation.

Usage

```
blocks_str(x, names_only = FALSE)

blocks_nbt(x)
```

Arguments

x	block data, either as a list of nbt values or a vector of strings.
names_only	A logical scalar. Return only the names of the blocks, ignoring block properties.

ChunkVersion *Read and write chunk version data*

Description

ChunkVersion data (tag 44) stores the chunk version number for a chunk.

Usage

```
get_chunk_version_value(x, z, dimension, db = default_db())

get_chunk_version_data(x, z, dimension, db = default_db())

put_chunk_version_value(value, x, z, dimension, db = default_db())

put_chunk_version_data(values, x, z, dimension, db = default_db())
```

Arguments

x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
db	A bedrockdb object.
value	An integer
values	A (named) vector of ChunkVersion values. If x is missing, the names of values will be taken as the keys.

Details

- get_chunk_version_value() and get_chunk_version_data() load ChunkVersion data from db. get_chunk_version_value() loads data for a single chunk, and get_chunk_version_data() loads data for multiple chunks.
- put_chunk_version_value() and put_chunk_version_data() store ChunkVersion data into db.

Value

get_chunk_version_value() returns a ChunkVersion data value. get_chunk_version_data() returns a named vector of ChunkVersion data values.

See Also

ChunkVersion

chunk_blocks *Extract or replace chunk blocks from an array*

Description

Convenience wrappers around [to extract or replace blocks from an array based on block coordinates.

Usage

```
chunk_blocks(x, ..., drop = TRUE, origin = chunk_origin(x))

chunk_blocks(x, ..., origin = chunk_origin(x)) <- value
```

Arguments

x	Object from which to extract element(s) or in which to replace element(s).
...	block coordinates specifying elements to extract or replace. Can be numeric, logical, or missing. If numeric, the coordinates will be mapped to indices unless there is a single, non-matrix argument.
drop	if TRUE the result is coerced to the lowest possible dimension.
origin	the origin of the chunk array, used for mapping coordinates to indices
value	An array-like R object of similar class as x

chunk_keys *Read and manipulate chunk keys*

Description

Chunk keys are keys to chunk data. A chunk key has a format which indicates the chunk it holds data for and the type of data it holds. This format is either `chunk:x:z:d:t` or `chunk:x:z:d:t:s`, where `x` and `z` indicates the coordinates of the chunk in chunk space, `d` indicates the dimension of the chunk, and `t` and `s` indicate the tag and subtag of the chunk.

`parse_chunk_keys()` splits chunk keys into their individual elements and returns a table with the results.

`create_chunk_keys()` returns a vector of chunk keys formed from its arguments.

`chunk_positions()` returns a matrix containing the chunk coordinates of keys.

`chunk_origins()` returns a matrix containing the block coordinate of the NW corner of keys.

`chunk_tag_str()` and `chunk_tag_int()` convert between integer and character representations of chunk tags.

Usage

```
parse_chunk_keys(keys)

create_chunk_keys(x, z, dimension, tag, subtag)

chunk_positions(keys)

chunk_origins(keys)

chunk_tag_str(tags)

chunk_tag_int(tags)
```

Arguments

keys	A character vector of database keys.
x	Chunk x-coordinate.
z	Chunk z-coordinate.
dimension	Dimension.
tag	The type of chunk data.
subtag	The subchunk the key refers to (Only used for tag 47).
tags	a vector

Examples

```
parse_chunk_keys("chunk:0:0:0:44")
parse_chunk_keys("chunk:0:0:0:47:1")
create_chunk_keys(0, 0, 0, 47, 1)
```

---

chunk_origin *Get or set the coordinates of the origin of a chunk*

Description

Get or set the coordinates of the origin of a chunk

Usage

```
chunk_origin(x)

chunk_origin(x) <- value
```

Arguments

x	an array of block data
value	an integer vector

Data3D *Read and write Data3D data*

Description

Data3D data (tag 43) stores information about surface heights and biomes in a chunk.

Usage

```
get_data3d_value(x, z, dimension, db = default_db())

get_data3d_data(x, z, dimension, db = default_db())

put_data3d_value(value, x, z, dimension, db = default_db())

put_data3d_data(values, x, z, dimension, db = default_db())

read_data3d_value(rawvalue)

write_data3d_value(value)
```

Arguments

x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
db	A bedrockdb object.
value	A Data3D value.
values	A (named) list of Data3D values. If x is missing, the names of values will be taken as the keys.
rawvalue	A raw vector.

Details

- get_data3d_value() and get_data3d_data() load Data3D data from db. get_data3d_value() loads data for a single chunk, and get_data3d_data() loads data for multiple chunks.
- put_data3d_value() and put_data3d_data() store Data3D data into db.
- write_data3d_value() encodes Data3D data into a raw vector. read_data3d_value() decodes binary Data3D data.

Value

get_data3d_value() returns a Data3D value. get_data3d_data() returns a named list of Data3D values. Data3D values are lists containing two elements. The height_map element is a 16x16 matrix containing height data. The biome_map element is a 16x384x16 array containing biome data.

See Also

Data2D

`default_db` *Get/set the default db connection.*

Description

The default db is the db connection that rbedrock uses by default. It defaults to the most recently opened db, but can also be set by the user.

Usage

```
default_db(db, check = TRUE)

with_db(db, code, close = is.character(db))

local_db(db, .local_envir = parent.frame(), close = is.character(db))
```

Arguments

db	For `default_db()`, a `bedrockdb` object. For `with_db()` and `local_db()`, a path to the world db to open or an existing `bedrockdb` object.
check	Check the validity of db? Set to `FALSE` to skip the check.
code	Code to execute in the temporary environment.
close	Close db when done? Set to `TRUE` to close db automatically.
.local_envir	The environment to use for scoping.

Details

Invoking `default_db()` returns the current default connection or the most recently opened one. Invoking `default_db(db)` updates the current default and returns the previous set value. `default_db(NULL)` can be used to unset the default db and revert to the last opened one. Closing db will unset it as the default db as well.

`with_db()` and `local_db()` temporarily change the default db.

Value

For `default_db()`, the calculated value of the default db. For `default_db(db)`, the previously manually set value of `default_db()`. For `with_db(db, code)`, the result of evaluating `code` with db as the default db. For `local_db(db)`, the value of db.

See Also

withr::with_connection

Examples

```
dbpath <- rbedrock_example_world("example1.mcworld")
dbz <- bedrockdb(dbpath)
default_db(dbz) # set default
default_db() # returns dbz
default_db(NULL) # unset default
#cleanup
close(dbz)
with_db(dbpath, length(get_keys))
db <- local_db(dbpath)
length(get_keys())
close(db)
unlink(dbpath, recursive = TRUE)
```

`delete_values` *Remove values from a bedrockdb.*

Description

Remove values from a bedrockdb.

Usage

```
delete_values(
  keys,
  db = default_db(),
  report = FALSE,
  readoptions = NULL,
  writeoptions = NULL
)
```

Arguments

`keys`	A character vector of keys.
`db`	A `bedrockdb` object
`report`	A logical indicating whether to generate a report on deleted keys
`readoptions`	A `bedrock_leveldb_readoptions` object
`writeoptions`	A `bedrock_leveldb_writeoptions` object

Value

If `report == TRUE`, a logical vector indicating which keys were deleted.

`FinalizedState` *Load and store FinalizedState data*

Description

FinalizedState data (tag 54) holds a number which indicates a chunk's state of generation.

Usage

```
get_finalized_state_value(x, z, dimension, db = default_db())

get_finalized_state_data(x, z, dimension, db = default_db())

put_finalized_state_value(value, x, z, dimension, db = default_db())

put_finalized_state_data(values, x, z, dimension, db = default_db())
```

Arguments

`x`, `z`, `dimension`	Chunk coordinates to extract data from. `x` can also be a character vector of db keys.
`db`	A bedrockdb object.
`value`	An integer
`values`	A (named) vector of FinalizedState values. If `x` is missing, the names of `values` will be taken as the keys.

Details

FinalizedState data contains the following information.

Value	Name	Description
0	NeedsInstaticking	Chunk needs to be ticked
1	NeedsPopulation	Chunk needs to be populated with mobs
2	Done	Chunk generation is fully complete

- `get_finalized_state_value()` and `get_finalized_state_data()` load FinalizedState data from `db`. `get_finalized_state_value()` loads data for a single chunk, and `get_finalized_state_data()` loads data for multiple chunks.
- `put_finalized_state_value()` and `put_finalized_state_data()` store FinalizedState data into `db`.

Value

`get_finalized_state_value()` returns a ChunkVersion data value. `get_finalized_state_data()` returns a named vector of FinalizedState data values.

get_keys *Get a list of keys stored in a bedrockdb.*

Description

Get a list of keys stored in a bedrockdb.

Usage

```
get_keys(prefix = NULL, db = default_db(), readoptions = NULL)
```

Arguments

prefix	A string specifying chunk prefix or string prefix.
db	A bedrockdb object
readoptions	A bedrock_leveldb_readoptions object

Value

A vector containing all the keys found in the bedrockdb.

If prefix is specified, this vector will be filtered for based on the specified prefix.

get_nbt_data *Read and Write NBT Data*

Description

The Named Binary Tag (NBT) format is used by Minecraft for various data types.

Usage

```
get_nbt_data(keys, db = default_db(), readoptions = NULL, simplify = TRUE)

get_nbt_value(key, db = default_db(), readoptions = NULL, simplify = TRUE)

put_nbt_data(values, keys, db = default_db(), writeoptions = NULL)

put_nbt_value(value, key, db = default_db(), writeoptions = NULL)

read_nbt(
  rawvalue,
  format = c("little", "big", "network", "network_big"),
  simplify = TRUE
)
```

```
read_nbt_data(
  rawdata,
  format = c("little", "big", "network", "network_big"),
  simplify = TRUE
)

write_nbt(value, format = c("little", "big", "network", "network_big"))

write_nbt_data(values, format = c("little", "big", "network", "network_big"))
```

Arguments

keys	A character vector of keys
db	A bedrockdb object
readoptions	A bedrock_leveldb_readoptions object
simplify	If TRUE, simplifies a list containing a single unnamed nbt value.
key	A single key
values	A list of values. Optionally named.
writeoptions	A bedrock_leveldb_writeoptions object
value	An nbt object or a list of nbt objects
rawvalue	A raw vector
format	A character string specifying which binary NBT format to use.
rawdata	A list of raw vectors

Details

get_nbt_data() and get_nbt_value() load nbt-formatted data from db and parses it.

put_nbt_data() and put_nbt_value() store nbt data into db in binary form.

read_nbt() reads NBT data from a raw vector.

read_nbt_data() calls read_nbt() on each element of a list.

write_nbt() encodes NBT data into a raw vector.

write_nbt_data() calls write_nbt() on each element of a list.

---

key_prefix *Read values stored in a bedrockdb.*

Description

Read values stored in a bedrockdb.

Usage

```
key_prefix(prefix)

starts_with(prefix)

get_data(keys, db = default_db(), readoptions = NULL)

get_value(key, db = default_db(), readoptions = NULL)

has_values(keys, db = default_db(), readoptions = NULL)
```

Arguments

prefix	A string specifying key prefix
keys	A character vector of keys
db	A bedrockdb object
readoptions	A bedrock_leveldb_readoptions object
key	A single key

Value

get_data()returns a named-list of raw vectors.get_value()‘ returns a raw vector.

has_values() returns a logical vector.

---

locate_blocks *Locate the coordinates of blocks in a chunk*

Description

Locate the coordinates of blocks in a chunk

Usage

```
locate_blocks(blocks, pattern, negate = FALSE)
```

Arguments

blocks	A character array containing block data.
pattern	The pattern to look for. Passed to base::grep.
negate	If TRUE, return non-matching elements.

Examples

```
dbpath <- rbedrock_example_world("example1.mcworld")
db <- bedrockdb(dbpath)
blocks <- get_blocks_value(db, x=37, z=10, dimension=0)
locate_blocks(blocks, "ore")
close(db)
```

`minecraft_worlds` *Utilities for working with Minecraft world folders.*

Description

`world_dir_path()` returns the path to the `minecraftWorlds` directory. Use `options(rbedrock.worlds_dir_path = "custom/path")` to customize the path as needed.

`list_worlds()` returns a `data.frame()` containing information about Minecraft saved games.

`create_world()` creates a new Minecraft world.

`export_world()` exports a world to an archive file.

Usage

```
worlds_dir_path(force_default = FALSE)

list_worlds(worlds_dir = worlds_dir_path())

create_world(id = NULL, ..., worlds_dir = worlds_dir_path())

export_world(id, file, worlds_dir = worlds_dir_path(), replace = FALSE)

import_world(file, id = NULL, ..., worlds_dir = worlds_dir_path())

get_world_path(id, worlds_dir = worlds_dir_path())
```

Arguments

`force_default`	If `TRUE`, return most likely world path on the system.
`worlds_dir`	The path of a `minecraftWorlds` directory.
`id`	The path to a world folder. If the path is not absolute or does not exist, it is assumed to be the base name of a world folder in `worlds_dir`. For `import_world()`, if `id` is `NULL` a unique world id will be generated. How it is generated is controlled by the `rbedrock.rand_world_id` global options. Possible values are "pretty" and "mcpe".
`...`	Arguments to customize `level.dat` settings. Supports dynamic dots via `rlang::list2()`.
`file`	The path to an mcworld file. If exporting, it will be created. If importing, it will be extracted.
`replace`	If `TRUE`, overwrite an existing file if necessary.

Examples

```
## Not run:
create_world(LevelName = "My World", RandomSeed = 10)

## End(Not run)
```

---

nbt *Create an NBT value*

Description

The Named Binary Tag (NBT) format is used by Minecraft for various data types. An NBT value holds a 'payload' of data and a 'tag' indicating the type of data held.

Usage

```
unnbt(x)

nbt_compound(...)

nbt_byte(x)

nbt_byte_array(x)

nbt_byte_list(x)

nbt_byte_array_list(x)

nbt_short(x)

nbt_short_list(x)

nbt_int(x)

nbt_int_array(x)

nbt_int_list(x)

nbt_int_array_list(x)

nbt_float(x)

nbt_float_list(x)

nbt_double(x)

nbt_double_list(x)
```

```
nbt_long(x)

nbt_long_array(x)

nbt_long_list(x)

nbt_long_array_list(x)

nbt_string(x)

nbt_raw_string(x)

nbt_string_list(x)

nbt_raw_string_list(x)

nbt_empty_list(x = list())

nbt_compound_list(x)

nbt_nested_list(x)
```

Arguments

x	An nbt payload.
...	NBT objects, possibly named.

Details

- `nbt_*()` family of functions create nbt data types.
- `unnbt()` recursively strips NBT metadata from an NBT value.

`nbt_list_of` *Create a list of nbt objects.*

Description

Create a list of nbt objects.

Usage

```
nbt_list_of(...)
```

Arguments

...	NBT objects, possibly named.

PendingTicks *Load and store PendingTicks NBT data*

Description

PendingTicks data (tag 51) holds a list of NBT values for pending ticks.

Usage

```
get_pending_ticks_data(x, z, dimension, db = default_db())

get_pending_ticks_value(x, z, dimension, db = default_db())

put_pending_ticks_data(values, x, z, dimension, db = default_db())

put_pending_ticks_value(value, x, z, dimension, db = default_db())
```

Arguments

x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
db	A bedrockdb object.
values	A (named) list of list of NBT objects.
value	A list of NBT objects.

Details

- get_pending_ticks_value() and get_pending_ticks_data() load PendingTicks data from db. get_pending_ticks_value() loads data for a single chunk, and get_pending_ticks_data() loads data for multiple chunks.
- put_pending_ticks_value() and put_pending_ticks_data() store PendingTicks data for one or multiple chunks into db.

Value

get_pending_ticks_value() returns a list of NBT objects. get_pending_ticks_data() returns a named list of lists of NBT objects.

put_data *Write values to a bedrockdb.*

Description

Write values to a bedrockdb.

Usage

```
put_data(values, keys, db = default_db(), writeoptions = NULL)

put_value(value, key, db = default_db(), writeoptions = NULL)
```

Arguments

values	A list of raw values.
keys	A character vector of keys.
db	A bedrockdb object
writeoptions	A bedrock_leveldb_writeoptions object
value	A raw vector that contains the information to be written.
key	A key that will be used to store data.

Value

An invisible copy of db.

RandomTicks *Load and store RandomTicks NBT data*

Description

RandomTicks data (tag 59) holds a list of NBT values for random ticks.

Usage

```
get_random_ticks_data(x, z, dimension, db = default_db())

get_random_ticks_value(x, z, dimension, db = default_db())

put_random_ticks_data(values, x, z, dimension, db = default_db())

put_random_ticks_value(value, x, z, dimension, db = default_db())
```

Arguments

x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
db	A bedrockdb object.
values	A (named) list of list of NBT objects.
value	A list of NBT objects.

Details

- get_random_ticks_value() and get_random_ticks_data() load RandomTicks data from db. get_random_ticks_value() loads data for a single chunk, and get_random_ticks_data() loads data for multiple chunks.
- put_random_ticks_value() and put_random_ticks_data() store RandomTicks data for one or multiple chunks into db.

Value

get_random_ticks_value() returns a list of NBT objects. get_random_ticks_data() returns a named list of lists of NBT objects.

---

rbedrock_example *Get path to rbedrock example*

Description

rbedrock comes bundled with a number of sample files in its inst/extdata directory. This function make them easy to access.

Usage

```
rbedrock_example(path = NULL)

rbedrock_example_world(path)
```

Arguments

path	Name of file or directory. If NULL, the examples will be listed.

Examples

```
rbedrock_example()
rbedrock_example("example1.mcworld")
rbedrock_example_world("example1.mcworld")
```

read_leveldat *Read and write data from a world's level.dat file.*

Description

Read and write data from a world's level.dat file.

Usage

```
read_leveldat(path)

write_leveldat(object, path, version = 8L)
```

Arguments

path	The path to a world folder. If the path does not exist, it is assumed to be the base name of a world folder in the local minecraftWorlds directory.
object	NBT data to be written to level.dat.
version	The level.dat format version for the file header.

Value

`read_leveldat` returns nbt data.

`write_leveldat` returns a copy of the data written.

Examples

```
# Fix level.dat after opening a world in creative.
dbpath <- rbedrock_example_world("example1.mcworld")
dat <- read_leveldat(dbpath)
dat$hasBeenLoadedInCreative <- FALSE
write_leveldat(dat, dbpath)
```

simulation_area *Calculate a player-based simulation area*

Description

Calculate a player-based simulation area

Usage

```
simulation_area(sim_distance, x = 0, z = 0)
```

Arguments

sim_distance	A sim distance setting
x, z	Chunk coordinates where a player is standing

Value

A data.frame containing the chunk coordinates in the simulation area.

---

spawning_area *Calculate a player-based spawning area*

Description

Calculate a player-based spawning area

Usage

```
spawning_area(sim_distance, x = 0, z = 0)
```

Arguments

sim_distance	A sim distance setting
x, z	Chunk coordinates where a player is standing (can be fractional)

Value

A data.frame containing the chunk coordinates in the spawning area.

SubChunkBlocks *Load and store SubChunkBlocks data*

Description

SubChunkBlocks data (tag 47) stores information about the blocks in a world. Each chunk is divided into multiple 16x16x16 subchunks, and the blocks for each subchunk are stored separately. Blocks are stored per subchunk in a palette-based format. Block storage is separated into multiple layers where each layer has its own block palette and block ids. In practices subchunks have either one or two layers, and the extra layer is most-often used to store water for water-logged blocks.

Usage

```
get_subchunk_blocks_value(x, z, dimension, subchunk, db = default_db())

get_subchunk_blocks_data(x, z, dimension, subchunk, db = default_db())

put_subchunk_blocks_value(
  value,
  x,
  z,
  dimension,
  subchunk,
  db = default_db(),
  version = 9L
)

put_subchunk_blocks_data(
  values,
  x,
  z,
  dimension,
  subchunk,
  db = default_db(),
  version = 9L
)

read_subchunk_blocks_value(rawvalue, subchunk_position = NA_integer_)

write_subchunk_blocks_value(value, subchunk_position, version = 9L)

subchunk_blocks_value_as_array(
  value,
  names_only = FALSE,
  extra_block = !names_only
)

subchunk_blocks_array_as_value(r)

subchunk_origins(keys)

subchunk_coords(ind, origins = subchunk_origins(names(ind)))
```

Arguments

x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
subchunk	Subchunk indexes
db	A bedrockdb object.

value	A SubChunkBlocks data value
version	Which format of subchunk data to use
values	A (named) list of SubChunkBlocks data values. If x is missing, the names of values will be taken as the keys.
rawvalue	A raw vector
subchunk_position	Optional, an integer. When reading a value, it will be used if the value's position attribute is missing. When writing a value, it will be used in place of the value's position attribute.
names_only	A logical scalar. Return only the names of the blocks, ignoring block states.
extra_block	A logical scalar. Append the extra block layer to the output (separated by ";"). This is mostly useful if you have waterlogged blocks. If the extra block is air, it will not be appended.
r	A character array
keys	A character vector of keys.
ind	Numeric vector or a named list of numeric vectors containing indexes for blocks in a subchunk.
origins	A matrix of subchunk origins.

Details

The format description can be found at https://gist.github.com/Tomcc/a96af509e275b1af483b25c543cfbf37.

- get_subchunk_blocks_value() and get_subchunk_blocks_data() load SubChunkBlocks data from db. get_subchunk_blocks_value() loads data for a single subchunk, and get_subchunk_blocks_data() loads data for multiple subchunks.
- put_subchunk_blocks_value() and put_subchunk_blocks_data() store SubChunkBlocks data into db.
- write_subchunk_blocks_value() encodes SubChunkBlocks data into a raw vector. read_subchunk_blocks_value() decodes binary SubChunkBlocks data.
- subchunk_blocks_value_as_array() converts SubChunkBlocks data into a character array.
- subchunk_origins() returns a matrix containing the block coordinate of the lower NW corner of subchunk keys.
- subchunk_coords() determines the block coordinates of blocks based on their array indexes and their subchunk origins.

Value

get_subchunk_blocks_value() returns a SubChunkBlocks data value. get_biomes_data() returns a named list of SubChunkBlocks data values.

vanilla_block_states_df *Bedrock block data*

Description

Information about blocks used in Bedrock edition. Generated from the PyMCTranslate project.

Usage

```
vanilla_block_states_df

vanilla_block_list

vanilla_block_property_type_list
```

Format

vanilla_block_states_df:
A data.frame in long-format with 1131 rows and 5 columns. Block data version is 18168865.

name Block name.
property Property name.
type Property type.
default Default value.
allowed Allowed values.

vanilla_block_list:
List of blocks names. Includes blocks without properties, which don't show up in vanilla_block_states_df.

vanilla_block_property_type_list:
List of properties (names) and their types (values).

Source

- https://github.com/gentlegiantJGC/PyMCTranslate/

Index