Package 'rbedrock'

March 21, 2023

Title Analysis and Manipulation of Data from Minecraft Bedrock Edition

Version 0.3.0

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

RoxygenNote 7.2.0

SystemRequirements cmake, zlib, GNU make, Solaris: g++ is a requirement

NeedsCompilation yes

Depends R (>= 3.6.0)

Imports tibble, R6, stringr, bit64, rappdirs, rlang, dplyr, purrr, magrittr, readr, utils, digest, vctrs, tidyr, fs

Suggests zip, testthat, jsonlite, covr

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

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

Author Reed Cartwright [aut, cre] (<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 2023-03-21 00:20:02 UTC

R topics documented:

`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)

get_acdig_value(x, z, dimension, db)

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

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

read_acdig_value(rawdata)

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.
`rawdata`	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)

get_actors_value(x, z, dimension, db)

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

put_actors_value(value, x, z, dimension, 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(db)
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

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.

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

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

Examples

```
# identify slime chunks
g <- tidyr::expand_grid(x=1:10, z=1:10)
is_slime_chunk <- purrr::pmap_lgl(g, 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 Data3D data (tag 43). Legacy Biomes data is stored as the second map in the Data2D data (tag 45).

`get_biomes_data()` and `get_biomes_value()` load Biomes data from db. `get_biomes_data()` will silently drop keys not representing Data3D data. `get_biomes_value()` supports loading only a single value. `get_biomes_values()` is a synonym for `get_biomes_data()`.

`put_biomes_data()` `put_biomes_values()`, and `put_biomes_value()` update the biome information of chunks. They preserve any existing height data.
`get_legacy_biomes_*()` and `put_legacy_biomes_*()` behave similar to the equivalent non-legacy functions. They get or put 2d biome data.

Usage

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

get_biomes_values(db, x, z, dimension, return_names = TRUE)

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

put_biomes_data(db, data, missing_height = -64L)

put_biomes_values(db, x, z, dimension, values, missing_height = -64L)

put_biomes_value(db, x, z, dimension, value, missing_height = -64L)

get_legacy_biomes_data(db, x, z, dimension, return_names = TRUE)

get_legacy_biomes_values(db, x, z, dimension, return_names = TRUE)

get_legacy_biomes_value(db, x, z, dimension, return_names = TRUE)

put_legacy_biomes_data(db, data, missing_height = 0L)

put_legacy_biomes_values(db, x, z, dimension, values, missing_height = 0L)
```

```
put_legacy_biomes_value(db, x, z, dimension, value, missing_height = 0L)
```

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
return_names	return biome names instead of biome ids.
data	A list of character or integer vectors.
missing_height	if there is no existing height data, use this value for the chunk.
values	a list of arrays containing biome names or ids.
value	an array containing biome names or ids.

Value

get_biomes_value() returns an array with 3 dimensions. get_biomes_data() returns a list of the of the values returned by get_biomes_value().

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.

get_block_entity_data() and get_block_entity_value() load BlockEntity data from db. get_block_entity_data() will silently drop and keys not representing BlockEntity data. get_block_entity_value() supports loading only a single value. get_block_entity_values() is a synonym for get_block_entity_data().

put_block_entity_values(), put_block_entity_value(), and put_block_entity_data() store BlockEntity data into db.

Usage

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

get_block_entity_values(db, x, z, dimension)

get_block_entity_value(db, x, z, dimension)

put_block_entity_values(db, x, z, dimension, values)

put_block_entity_value(db, x, z, dimension, value)

put_block_entity_data(db, data)
```

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
values	A list of nbt objects
value	An nbt object.
data	A named-list specifying key-value pairs.

Value

get_block_entity_data() returns a named-list of nbt data. get_block_entity_values() returns a single nbt value.

Checksums *Load and store Checksums data*

Description

Checksums data (tag 59) holds checksums for several chunk records. These records are 2DMaps (tag 45), SubchunkBlocks (tag 47), BlockEntities (tag 49), and Entities (tag 50).

get_checksums_data() loads Checksums data from a bedrockdb. It will silently drop and keys not representing Checksums data. get_checksums_values() is a synonym for get_checksums_data().

get_checksums_value() loads Checksums data from a bedrockdb. It only supports loading a single value.

update_checksums_data() recalculates Checksums data. It calculates checksums for the specified chunks' SubchunkBlocks, 2DMaps, BlockEntities, and Entities records in db and updates the Checksums record to match.

read_checksums_value() parses a binary Checksums record into a list of checksums.

write_checksums_value() converts Checksums from a named list into binary format.

Usage

```
get_checksums_data(db, x, z, dimension)

get_checksums_values(db, x, z, dimension)

get_checksums_value(db, x, z, dimension)

update_checksums_data(db, x, z, dimension)

read_checksums_value(rawdata)

write_checksums_value(object)
```

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
rawdata	a raw vector holding binary Checksums data
object	a named character vector in the same format as returned by read_checksums_value().

Value

get_checksums_data() returns a named-list of the values returned by get_checksums_value().

get_checksums_value() and read_checksums_value() return a character vector. The names of the character vector indicate which chunk record (tag and subtag) the checksum is for.

write_checksums_value() returns a raw vector.

ChunkVersion *Read and write chunk version data*

Description

Version data (tag 44) and LegacyVersion data (tag 118) store the version number of a chunk. In Minecraft version 1.16.100, chunk version data was moved from tag 118 to tag 44.

get_chunk_version_data() and get_chunk_version_value() load Version data from db. get_chunk_version_data() will silently drop and keys not representing Version data. get_chunk_version_value() supports loading only a single value. get_chunk_version_values() is a synonym for get_chunk_version_data().

put_chunk_version_data(), put_chunk_version_values(), and put_chunk_version_value() store Version data into a bedrockdb.

read_chunk_version_value() decodes Version data.

write_chunk_version_value() encodes Version data.

Usage

```
get_chunk_version_data(db, x, z, dimension)

get_chunk_version_values(db, x, z, dimension)

get_chunk_version_value(db, x, z, dimension)

put_chunk_version_data(db, data)

put_chunk_version_values(db, x, z, dimension, values)

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

read_chunk_version_value(rawdata)

write_chunk_version_value(num)
```

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract version data from. x can also be a character vector of db keys.
data	A named-vector of key-value pairs for Version data.
values	An integer vector
value	A scalar integer vector
rawdata	A scalar raw.
num	A scalar integer.

---

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. Keys that do not contain chunk data are silently dropped.

`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

Data2D *Read and write Data2D data*

Description

Data2D data (tag 45) stores information about surface heights and biomes in a chunk. Data2D data is 768 bytes long and consists of a 256 int16s (heights) followed by 256 uint8s (biomes).

`get_data2d_data()` loads Data2D data from a `bedrockdb`. It will silently drop and keys not representing Data2D data.

`get_data2d_value()` loads Data2D data from a `bedrockdb`. It only supports loading a single value.

`read_data2d_value` decodes binary Data2D data.

`put_data2d_data()`, `put_data2d_values()`, and `put_data2d_value()` store Data2D data into a `bedrockdb`.

`write_data2d_value` encodes Data2D data into a raw vector.

Usage

```
get_data2d_data(db, x, z, dimension)

get_data2d_values(db, x, z, dimension)

get_data2d_value(db, x, z, dimension)

read_data2d_value(rawdata)

put_data2d_data(db, data)

put_data2d_values(db, x, z, dimension, height_maps, biome_maps)

put_data2d_value(db, x, z, dimension, height_map, biome_map)

write_data2d_value(height_map, biome_map)
```

Arguments

`db`	A bedrockdb object.
`x, z, dimension`	Chunk coordinates to extract data from. `x` can also be a character vector of db keys.
`rawdata`	A raw vector.
`data`	A named-vector of key-value pairs for Data2D data.
`height_maps, biome_maps`	Lists of height and biome data. Values will be recycled if necessary to match the number of keys to be written to. If `biome_maps` is missing, `height_maps` should be in the same format as returned by `get_data2d_data()`.

height_map, biome_map
: 16x16 arrays containing height and biome data. Values will be recycled if necessary. If biome_map is missing, height-map should be a list a list() with both "height_map" and "biome_map" elements.

Value

get_data2d_data() returns a list of the of the values returned by get_data2d_value().

get_data2d_value() returns a list with components "height_map" and "biome_map".

Examples

```
heights <- matrix(63,16,16)
biomes <- matrix(1,16,16)
# Pass heights and biomes as separate parameters
dat <- write_data2d_value(heights, biomes)
# Pass them as a list.
obj <- list(height_map = heights, biome_map = biomes)
dat <- write_data2d_value(obj)
# Pass them as scalars
dat <- write_data2d_value(63, 1)
```

Data3D *Read and write Data3D data*

Description

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

get_data3d_data() loads Data3D data from db. It will silently drop keys not representing Data3D data.

get_data3d_value() loads Data3D data from db. It only supports loading a single value.

put_data3d_data(), put_data3d_values(), and put_data3d_value() store Data3D data into db.

read_data3d_value() decodes binary Data3D data.

write_data3d_value encodes Data3D data into a raw vector.

Usage

```
get_data3d_data(db, x, z, dimension)

get_data3d_values(db, x, z, dimension)

get_data3d_value(db, x, z, dimension)

put_data3d_data(db, data)

put_data3d_values(db, x, z, dimension, height_maps, biome_maps)
```

```
put_data3d_value(db, x, z, dimension, height_map, biome_map)

read_data3d_value(rawdata)

write_data3d_value(height_map, biome_map)
```

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
data	A named-vector of key-value pairs for Data3D data.
height_maps, biome_maps	Lists of height and biome data. Values will be recycled if necessary to match the number of keys to be written to. If biome_maps is missing, height_maps should be in the same format as returned by get_data3d_data().
height_map	16x16 array containing height data. Values will be recycled if necessary. If biome_map is missing, height-map should be a list a list() with both "height_map" and "biome_map" elements.
biome_map	16xNx16 array containing biome data.
rawdata	A raw vector.

Value

get_data3d_data() returns a list of the of the values returned by get_data3d_value().

get_data3d_value() returns a list with components "height_map" and "biome_map".

---

delete_values *Remove values from a bedrockdb.*

Description

Remove values from a bedrockdb.

Usage

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

Arguments

db	A bedrockdb object
keys	A character vector of keys.
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.

Entity *Load and store Entity NBT data*

Description

Entity data (tag 50) holds a list of NBT values for mobs and other entities in the game. After 1.18.30, entity data was migrated to a new actor digest format and no longer saved with chunk data.

get_entity_data() and get_entity_value() load Entity data from db. get_entity_data() will silently drop and keys not representing Entity data. get_entity_value() supports loading only a single value. get_entity_values() is a synonym for get_entity_data().

put_entity_values(), put_entity_value(), and put_entity_data() store BlockEntity data into db.

Usage

```
get_entity_data(db, x, z, dimension)

get_entity_values(db, x, z, dimension)

get_entity_value(db, x, z, dimension)

put_entity_values(db, x, z, dimension, values)

put_entity_value(db, x, z, dimension, value)

put_entity_data(db, data)
```

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
values	A list of nbt objects
value	An nbt object.
data	A named-list specifying key-value pairs.

Value

`get_entity_data()` returns a named-list of nbt data. `get_entity_values()` returns a single nbt value.

`FinalizedState` *Load and store FinalizedState data*

Description

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

`get_finalized_state_data()` and `get_finalized_state_value()` load FinalizedState data from `db`. `get_finalized_state_data()` will silently drop and keys not representing FinalizedState data. `get_finalized_state_value()` supports loading only a single value. `get_finalized_state_values()` is a synonym for `get_finalized_state_data()`.

`put_finalized_state_data()`, `put_finalized_state_values()`, and `put_finalized_state_value()` store FinalizedState data into a `bedrockdb`.

`read_finalized_state_value()` parses a binary FinalizedState record.

`write_finalized_state_value()` converts a FinalizedState value to a raw vector.

Usage

```
get_finalized_state_data(db, x, z, dimension)

get_finalized_state_values(db, x, z, dimension)

get_finalized_state_value(db, x, z, dimension)

put_finalized_state_data(db, data)

put_finalized_state_values(db, x, z, dimension, values)

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

read_finalized_state_value(rawdata)

write_finalized_state_value(value)
```

Arguments

`db`	A bedrockdb object.
`x, z, dimension`	Chunk coordinates to extract data from. `x` can also be a character vector of db keys.
`data`	A named-vector of key-value pairs for FinalizedState data.
`values`	An integer vector
`value`	a scalar integer
`rawdata`	a raw vector

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

Value

`get_finalized_state_data()` returns a named integer vector of the values returned by `get_finalized_state_value()`.

`get_chunk_blocks_data` *Load block data from one or more chunks*

Description

These functions return block data as strings containing the block name and block states. The strings' format is `blockname@state1=value1@state2=value2` etc. Blocks may have 0 or more states.

`get_chunk_blocks_value()` is an alias for `get_chunk_blocks_data()`

`get_chunk_blocks_value()` loads block data from a `bedrockdb`. It only supports loading a single value.

`put_chunk_blocks_data()`, `put_chunk_blocks_values()`, and `put_chunk_blocks_value()` stores block data into a `bedrockdb`.

Usage

```
get_chunk_blocks_data(
  db,
  x,
  z,
  dimension,
  names_only = FALSE,
  extra_block = FALSE
)

get_chunk_blocks_values(
  db,
  x,
  z,
  dimension,
  names_only = FALSE,
  extra_block = FALSE
)
```

```
get_chunk_blocks_value(
  db,
  x,
  z,
  dimension,
  names_only = FALSE,
  extra_block = FALSE
)

put_chunk_blocks_data(db, data, version = 9L)

put_chunk_blocks_values(db, x, z, dimension, values, version = 9L)

put_chunk_blocks_value(db, x, z, dimension, value, version = 9L)
```

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
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.
data	A named list of 16xNx16 character() arrays
version	Which format of subchunk data to use
values	A list of 16xNx16 character() arrays
value	A 16xNx16 character array

Value

`get_chunk_blocks_data()` returns a list of the of the values returned by `read_chunk_blocks_value()`.

`get_chunk_blocks_value()` return a 16xNx16 character array. The axes represent the x, y, and z dimensions in that order. The size of the y-axis is based on the highest subchunk in the coordinate. Missing subchunks are considered air.

---

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

Description

Get a list of keys stored in a bedrockdb.

Usage

```
get_keys(db, starts_with = NULL, readoptions = NULL)
```

Arguments

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

Value

A vector containing all the keys found in the bedrockdb.

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

---

get_nbt_data *Read and Write NBT Data*

Description

get_nbt_data() and get_nbt_value() load nbt-formatted data from db and parses it. get_nbt_values() is a synonym for get_nbt_data().

put_nbt_values(), put_nbt_value(), and put_nbt_data() 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.

Usage

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

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

get_nbt_values(db, keys, readoptions = NULL, simplify = TRUE)

put_nbt_values(db, keys, values, writeoptions = NULL)

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

put_nbt_data(db, data, writeoptions = NULL)

read_nbt(rawdata, simplify = TRUE)

read_nbt_data(data, simplify = TRUE)

write_nbt(object)

write_nbt_data(data)
```

Arguments

db	A bedrockdb object
keys	A character vector of keys.
readoptions	A bedrock_leveldb_readoptions object
simplify	If TRUE, simplifies a list containing a single unnamed nbtnode.
key	A single key.
values	A list of nbt objects
writeoptions	A bedrock_leveldb_writeoptions object
value	An nbt object.
data	A named-list specifying key-value pairs.
rawdata	A raw vector
object	An nbt object or a list of nbt objects

Details

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

---

get_values *Read values stored in a bedrockdb.*

Description

get_values() and get_data() are synonyms.

Usage

```
get_values(db, keys, starts_with, readoptions = NULL)

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

get_value(db, key, readoptions = NULL)

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

Arguments

db	A bedrockdb object
keys	A character vector of keys.
starts_with	A string specifying chunk prefix or string prefix.
readoptions	A bedrock_leveldb_readoptions object
key	A single key.

Value

`get_values()` returns a named-list of raw vectors.

`get_value()` returns a raw vector.

`has_values()` returns a logical vector.

HSA *Read and write HardcodedSpawnArea (HSA) data*

Description

HardcodedSpawnArea (HSA) data (tag 57) stores information about any structure spawning locations in a chunk. An HSA is defined by a bounding box that specifies the location of an HSA in a chunk and a tag that specifies the type: 1 = NetherFortress, 2 = SwampHut, 3 = OceanMonument, and 5 = PillagerOutpost.

`get_hsa_data()` loads HardcodedSpawnArea data from a `bedrockdb`. It will silently drop and keys not representing HSA data. `get_hsa_values()` is a synonym for `get_hsa_data()`.

`get_hsa_value()` loads HSA data from a `bedrockdb`. It only supports loading a single value.

`read_hsa_value()` decodes HSA data.

`put_hsa_data()` puts HSA data into a `bedrockdb`. HSA bounding boxes will be split across chunks and

`put_hsa_values()` and `put_hsa_value()` store HSA data into a `bedrockdb`.

`write_hsa_value()` encodes HSA data.

Usage

```
get_hsa_data(db, x, z, dimension)

get_hsa_values(db, x, z, dimension)

get_hsa_value(db, x, z, dimension)

read_hsa_value(rawdata)

put_hsa_data(db, data, merge = TRUE)

put_hsa_values(db, x, z, dimension, values)

put_hsa_value(db, x, z, dimension, value)

write_hsa_value(value)
```

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
rawdata	A scalar raw.
data	A table containing HSA coordinates.
merge	Merge the new HSAs with existing HSAs.
values	A list of tables containing HSA coordinates and tags.
value	A table containing HSA coordinates

Value

get_hsa_data() returns a table in the same format as get_hsa_value().

get_hsa_value() and read_hsa_value() return a table with columns indicating the coordinates of the HSA bounding box and the location of the HSS at the center of the bounding box. get_hsa_value() also records the dimension of the bounding box.

Examples

```
dbpath <- rbedrock_example_world("example1.mcworld")
db <- bedrockdb(dbpath)
# view all HSA in a world
hsa <- get_hsa_data(db, get_keys(db))
hsa
# add an HSA to a world
dat <- data.frame(x1 = 0, x2 = 15, z1 = 0, z2 = 15,
                  y1 = 40, y2 = 60, tag = "SwampHut")
put_hsa_data(db, dat, merge = TRUE)
close(db)
```

---

list_biomes *List Minecraft Bedrock Edition biomes.*

Description

List Minecraft Bedrock Edition biomes.

Usage

```
list_biomes()

biome_id(x)
```

Arguments

x	A character vector containing biome name.

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 stringr::str_detect.
negate	If TRUE, return non-matching elements.

Examples

```
dbpath <- rbedrock_example_world("example1.mcworld")
db <- bedrockdb(dbpath)
blocks <- get_chunk_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_byte *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.
nbt_*() family of functions create nbt data types. unnbt() recursively strips NBT metadata from an NBT value.

payload() reads an nbt value's payload.

get_nbt_tag() returns the NBT tag corresponding to and NBT value.

Usage

```
nbt_byte(x)

nbt_short(x)

nbt_int(x)

nbt_long(x)

nbt_float(x)

nbt_double(x)

nbt_byte_array(x)

nbt_string(x)

nbt_raw_string(x)

nbt_int_array(x)

nbt_long_array(x)

nbt_compound(...)

nbt_list(...)

is_nbt(x)

payload(x)

unnbt(x)

get_nbt_tag(x)
```

Arguments

x	An nbt value
...	Arguments to collect into an NBT compound or NBT list value. Supports dynamic dots via `rlang::list2()`.

`PendingTicks` *Load and store PendingTicks NBT data*

Description

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

`get_pending_ticks_data()` and `get_pending_ticks_value()` load PendingTicks data from db. `get_pending_ticks_data()` will silently drop and keys not representing PendingTicks data. `get_pending_ticks_value()` supports loading only a single value. `get_pending_ticks_values()` is a synonym for `get_pending_ticks_data()`.

`put_pending_ticks_values()`, `put_pending_ticks_value()`, and `put_pending_ticks_data()` store PendingTicks data into db.

Usage

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

get_pending_ticks_values(db, x, z, dimension)

get_pending_ticks_value(db, x, z, dimension)

put_pending_ticks_values(db, x, z, dimension, values)

put_pending_ticks_value(db, x, z, dimension, value)

put_pending_ticks_data(db, data)
```

Arguments

`db`	A bedrockdb object.
`x, z, dimension`	Chunk coordinates to extract data from. `x` can also be a character vector of db keys.
`values`	A list of nbt objects
`value`	An nbt object.
`data`	A named-list specifying key-value pairs.

Value

`get_pending_ticks_data()` returns a named-list of nbt data. `get_pending_ticks_values()` returns a single nbt value.

`put_values` *Write values to a bedrockdb.*

Description

Write values to a bedrockdb.

Usage

```
put_values(db, keys, values, writeoptions = NULL)

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

put_data(db, data, writeoptions = NULL)
```

Arguments

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

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.

get_random_ticks_data() and get_random_ticks_value() load RandomTicks data from db. get_random_ticks_data() will silently drop and keys not representing RandomTicks data. get_random_ticks_value() supports loading only a single value. get_random_ticks_values() is a synonym for get_random_ticks_data().

put_random_ticks_values(), put_random_ticks_value(), and put_random_ticks_data() store RandomTicks data into db.

Usage

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

get_random_ticks_values(db, x, z, dimension)

get_random_ticks_value(db, x, z, dimension)

put_random_ticks_values(db, x, z, dimension, values)

put_random_ticks_value(db, x, z, dimension, value)

put_random_ticks_data(db, data)
```

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
values	A list of nbt objects
value	An nbt object.
data	A named-list specifying key-value pairs.

Value

get_random_ticks_data() returns a named-list of nbt data. get_random_ticks_values() returns a single nbt value.

---

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, old = FALSE)

write_leveldat(object, path, old = FALSE, 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.
old	Read/write to 'level.dat_old' instead.
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) holds information about the blocks in a subchunks. Each chunk is divided into multiple 16x16x16 subchunks, and each subchunk is stored separately and indicated by the use of the subtag. Blocks are stored in a palette-based format. Subchunks can have two layers of blocks, and the extra layer is most-often used to store water for water-logged blocks.

These functions return block data as strings containing the block name and block states. The strings' format is blockname@state1=value1@state2=value2 etc. Blocks may have 0 or more states.

get_subchunk_blocks_data() loads SubchunkBlocks data from a bedrockdb. It will silently drop and keys not representing SubchunkBlocks data. get_subchunk_blocks_values() is a synonym for get_subchunk_blocks_data().

get_subchunk_blocks_value() loads SubchunkBlocks data from a bedrockdb. It only supports loading a single value.

`get_subchunk_blocks_from_chunk()` loads SubchunkBlocks data from a `bedrockdb`. It supports efficiently loading subchunk block data from a single chunk.

`put_subchunk_blocks_data()`, `put_subchunk_blocks_values()`, and `put_subchunk_blocks_value()` store SubchunkBlocks data into a `bedrockdb`.

`read_subchunk_blocks_value()` decodes binary SubchunkBlock data.

`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.

Usage

```
get_subchunk_blocks_data(
  db,
  x,
  z,
  dimension,
  subchunk,
  names_only = FALSE,
  extra_block = FALSE
)

get_subchunk_blocks_values(
  db,
  x,
  z,
  dimension,
  subchunk,
  names_only = FALSE,
  extra_block = FALSE
)

get_subchunk_blocks_value(
  db,
  x,
  z,
  dimension,
  subchunk,
  names_only = FALSE,
  extra_block = FALSE
)

get_subchunk_blocks_from_chunk(
  db,
  x,
  z,
  dimension,
```

```
  names_only = FALSE,
  extra_block = FALSE
)

put_subchunk_blocks_data(db, data, version = 9L)

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

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

read_subchunk_blocks_value(
  rawdata,
  missing_offset = NA,
  names_only = FALSE,
  extra_block = FALSE
)

write_subchunk_blocks_value(object, version = 9L, missing_offset = NA_integer_)

subchunk_origins(keys)

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

Arguments

db	A bedrockdb object.
x, z, dimension	Chunk coordinates to extract data from. x can also be a character vector of db keys.
subchunk	Subchunk indexes to extract data from.
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.
data	A named list of 16x16x16 character() arrays
version	Which format of subchunk data to use
values	A list of 16x16x16 character() arrays
value	A 16x16x16 character array
rawdata	a raw vector holding binary SubchunkBlock data
missing_offset	subchunk offset to use if one is not found in rawdata
object	A 16x16x16 character array.
keys	A character vector of database 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

If a subchunk contains only air it will not be stored in the database, and missing subchunks are considered air.

Value

`get_subchunk_blocks_data()` returns a list of the of the values returned by `read_subchunk_blocks_value()`.

`get_subchunk_blocks_value()` and `read_subchunk_blocks_value()` return a 16x16x16 character array. The axes represent the `x`, `y`, and `z` dimensions in that order.

`get_subchunk_blocks_from_chunk()` returns a list of the of the values returned by `read_subchunk_blocks_value()`.

`read_subchunk_blocks_value()` returns a 16x16x16 character array. The axes represent the `x`, `y`, and `z` dimensions in that order.

`subchunk_coords()` returns a 3-column matrix of block coordinates.

Index