

# Package ‘rbedrock’

May 9, 2026

**Title** Analysis and Manipulation of Data from Minecraft Bedrock Edition

**Version** 0.4.2

**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** 2025-11-09 22:20:08 UTC

## Contents

ActorDigest . . . . .	2
Actors . . . . .	4
bedrockdb . . . . .	5
bedrock_random . . . . .	6
bedrock_random_create_seed . . . . .	7
Biomes . . . . .	8
biome_df . . . . .	9
BlockEntity . . . . .	10
Blocks . . . . .	11
blocks_str . . . . .	12
ChunkVersion . . . . .	13
chunk_blocks . . . . .	14
chunk_keys . . . . .	14
chunk_origin . . . . .	15
Data3D . . . . .	16
default_db . . . . .	17
delete_values . . . . .	18
FinalizedState . . . . .	19
get_keys . . . . .	20
get_nbt_data . . . . .	20
key_prefix . . . . .	21
locate_blocks . . . . .	22
minecraft_worlds . . . . .	23
nbt . . . . .	24
nbt_list_of . . . . .	25
PendingTicks . . . . .	26
put_data . . . . .	27
RandomTicks . . . . .	27
rbedrock_example . . . . .	28
read_level.dat . . . . .	29
simulation_area . . . . .	29
spawning_area . . . . .	30
SubChunkBlocks . . . . .	30
vanilla_block_states_df . . . . .	33

## Index

**34**

## 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

<code>x, z, dimension</code>	Chunk coordinates to extract data from. <code>x</code> can also be a character vector of db keys.
<code>db</code>	A bedrockdb object.
<code>values</code>	A list of character vectors. If <code>x</code> is missing, the names of values will be taken as the keys.
<code>value</code>	A character vector.
<code>rawvalue</code>	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](#)

**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**

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

<code>x, z, dimension</code>	Chunk coordinates to extract data from. <code>x</code> can also be a character vector of db keys.
<code>db</code>	A bedrockdb object.
<code>names_only</code>	A logical scalar. Return only the names of the blocks, ignoring block states.
<code>extra_block</code>	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	<i>Convert block data from nbt format to a string</i>
------------	---

---

### 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	<i>Read and write chunk version data</i>
--------------	--

---

**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

<code>x</code>	Object from which to extract element(s) or in which to replace element(s).
<code>...</code>	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.
<code>drop</code>	if TRUE the result is coerced to the lowest possible dimension.
<code>origin</code>	the origin of the chunk array, used for mapping coordinates to indices
<code>value</code>	An array-like R object of similar class as <code>x</code>

---

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	<i>Get or set the coordinates of the origin of a chunk</i>
--------------	--

---

**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**

<code>x, z, dimension</code>	Chunk coordinates to extract data from. <code>x</code> can also be a character vector of db keys.
<code>db</code>	A bedrockdb object.
<code>value</code>	A Data3D value.
<code>values</code>	A (named) list of Data3D values. If <code>x</code> is missing, the names of values will be taken as the keys.
<code>rawvalue</code>	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	<i>Get/set the default db connection.</i>
------------	---

---

**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	<i>Remove values from a bedrockdb.</i>
---------------	--

---

**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	<i>Load and store FinalizedState data</i>
----------------	---

---

### 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	NeedsInstacking	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 FinalizedState data value. `get_finalized_state_data()` returns a named vector of FinalizedState data values.

---

get_keys	<i>Get a list of keys stored in a bedrockdb.</i>
----------	--

---

**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	<i>Read and Write NBT Data</i>
--------------	--------------------------------

---

**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	<i>Read values stored in a bedrockdb.</i>
------------	---

---

### 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	<i>Locate the coordinates of blocks in a chunk</i>
---------------	--

---

**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 <a href="#">base::grep</a> .
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**

<code>force_default</code>	If TRUE, return most likely world path on the system.
<code>worlds_dir</code>	The path of a <code>minecraftWorlds</code> directory.
<code>id</code>	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 <code>worlds_dir</code> . For <code>import_world()</code> , if <code>id</code> is NULL a unique world id will be generated. How it is generated is controlled by the <code>rbedrock.rand_world_id</code> global options. Possible values are "pretty" and "mcpe".
<code>...</code>	Arguments to customize <code>level.dat</code> settings. Supports dynamic dots via <code>rlang::list2()</code> .
<code>file</code>	The path to an <code>mcworld</code> file. If exporting, it will be created. If importing, it will be extracted.
<code>replace</code>	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

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

---

**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	<i>Load and store RandomTicks NBT data</i>
-------------	--

---

**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	<i>Get path to rbedrock example</i>
------------------	-------------------------------------

---

**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_level.dat	<i>Read and write data from a world's level.dat file.</i>
----------------	---

---

**Description**

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

**Usage**

```
read_level.dat(path)
```

```
write_level.dat(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\_level.dat returns nbt data.

write\_level.dat 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_level.dat(dbpath)
dat$hasBeenLoadedInCreative <- FALSE
write_level.dat(dat, dbpath)
```

---

simulation_area	<i>Calculate a player-based simulation area</i>
-----------------	---

---

**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	<i>Calculate a player-based spawning area</i>
---------------	---

---

**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	<i>Load and store SubChunkBlocks data</i>
----------------	---

---

**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

## \* datasets

- biome\_df, 9
- vanilla\_block\_states\_df, 33

ActorDigest, 2, 4

Actors, 3, 4

base::grep, 22

bedrock\_random, 6

bedrock\_random\_create\_seed, 7

bedrock\_random\_get\_double  
(bedrock\_random), 6

bedrock\_random\_get\_float  
(bedrock\_random), 6

bedrock\_random\_get\_int  
(bedrock\_random), 6

bedrock\_random\_get\_uint  
(bedrock\_random), 6

bedrock\_random\_seed (bedrock\_random), 6

bedrock\_random\_state (bedrock\_random), 6

bedrockdb, 5

biome\_df, 9

biome\_id (Biomes), 8

biome\_name (Biomes), 8

Biomes, 8

BlockEntity, 10

Blocks, 11

blocks\_nbt (blocks\_str), 12

blocks\_str, 12

chunk\_blocks, 14

chunk\_blocks<- (chunk\_blocks), 14

chunk\_keys, 14

chunk\_origin, 15

chunk\_origin<- (chunk\_origin), 15

chunk\_origins (chunk\_keys), 14

chunk\_positions (chunk\_keys), 14

chunk\_tag\_int (chunk\_keys), 14

chunk\_tag\_str (chunk\_keys), 14

ChunkVersion, 13

close.bedrockdb (bedrockdb), 5

create\_acdig\_keys (ActorDigest), 2

create\_chunk\_keys (chunk\_keys), 14

create\_world (minecraft\_worlds), 23

Data3D, 8, 16

default\_db, 17

delete\_values, 18

Entity, 3, 4

export\_world (minecraft\_worlds), 23

FinalizedState, 19

get\_acdig\_data (ActorDigest), 2

get\_acdig\_value (ActorDigest), 2

get\_actors\_data (Actors), 4

get\_actors\_value (Actors), 4

get\_biomes\_data (Biomes), 8

get\_biomes\_value (Biomes), 8

get\_block\_entity\_data (BlockEntity), 10

get\_block\_entity\_value (BlockEntity), 10

get\_blocks\_data (Blocks), 11

get\_blocks\_value (Blocks), 11

get\_chunk\_version\_data (ChunkVersion),  
13

get\_chunk\_version\_value (ChunkVersion),  
13

get\_data (key\_prefix), 21

get\_data3d\_data (Data3D), 16

get\_data3d\_value (Data3D), 16

get\_finalized\_state\_data  
(FinalizedState), 19

get\_finalized\_state\_value  
(FinalizedState), 19

get\_keys, 20

get\_nbt\_data, 20

get\_nbt\_value (get\_nbt\_data), 20

get\_pending\_ticks\_data (PendingTicks),  
26

- get\_pending\_ticks\_value (PendingTicks), 26
- get\_random\_ticks\_data (RandomTicks), 27
- get\_random\_ticks\_value (RandomTicks), 27
- get\_subchunk\_blocks\_data (SubChunkBlocks), 30
- get\_subchunk\_blocks\_value (SubChunkBlocks), 30
- get\_value (key\_prefix), 21
- get\_world\_path (minecraft\_worlds), 23
- has\_values (key\_prefix), 21
- import\_world (minecraft\_worlds), 23
- is\_bedrockdb (bedrockdb), 5
- key\_prefix, 21
- list\_worlds (minecraft\_worlds), 23
- local\_db (default\_db), 17
- locate\_blocks, 22
- minecraft\_worlds, 23
- nbt, 24
- nbt\_byte (nbt), 24
- nbt\_byte\_array (nbt), 24
- nbt\_byte\_array\_list (nbt), 24
- nbt\_byte\_list (nbt), 24
- nbt\_compound (nbt), 24
- nbt\_compound\_list (nbt), 24
- nbt\_double (nbt), 24
- nbt\_double\_list (nbt), 24
- nbt\_empty\_list (nbt), 24
- nbt\_float (nbt), 24
- nbt\_float\_list (nbt), 24
- nbt\_int (nbt), 24
- nbt\_int\_array (nbt), 24
- nbt\_int\_array\_list (nbt), 24
- nbt\_int\_list (nbt), 24
- nbt\_list\_of, 25
- nbt\_long (nbt), 24
- nbt\_long\_array (nbt), 24
- nbt\_long\_array\_list (nbt), 24
- nbt\_long\_list (nbt), 24
- nbt\_nested\_list (nbt), 24
- nbt\_raw\_string (nbt), 24
- nbt\_raw\_string\_list (nbt), 24
- nbt\_short (nbt), 24
- nbt\_short\_list (nbt), 24
- nbt\_string (nbt), 24
- nbt\_string\_list (nbt), 24
- parse\_chunk\_keys (chunk\_keys), 14
- PendingTicks, 26
- put\_acdig\_data (ActorDigest), 2
- put\_acdig\_value (ActorDigest), 2
- put\_actors\_data (Actors), 4
- put\_actors\_value (Actors), 4
- put\_biomes\_data (Biomes), 8
- put\_biomes\_value (Biomes), 8
- put\_block\_entity\_data (BlockEntity), 10
- put\_block\_entity\_value (BlockEntity), 10
- put\_blocks\_data (Blocks), 11
- put\_blocks\_value (Blocks), 11
- put\_chunk\_version\_data (ChunkVersion), 13
- put\_chunk\_version\_value (ChunkVersion), 13
- put\_data, 27
- put\_data3d\_data (Data3D), 16
- put\_data3d\_value (Data3D), 16
- put\_finalized\_state\_data (FinalizedState), 19
- put\_finalized\_state\_value (FinalizedState), 19
- put\_nbt\_data (get\_nbt\_data), 20
- put\_nbt\_value (get\_nbt\_data), 20
- put\_pending\_ticks\_data (PendingTicks), 26
- put\_pending\_ticks\_value (PendingTicks), 26
- put\_random\_ticks\_data (RandomTicks), 27
- put\_random\_ticks\_value (RandomTicks), 27
- put\_subchunk\_blocks\_data (SubChunkBlocks), 30
- put\_subchunk\_blocks\_value (SubChunkBlocks), 30
- put\_value (put\_data), 27
- RandomTicks, 27
- rbedrock\_example, 28
- rbedrock\_example\_world (rbedrock\_example), 28
- read\_acdig\_value (ActorDigest), 2
- read\_data3d\_value (Data3D), 16
- read\_leveldat, 29
- read\_nbt (get\_nbt\_data), 20
- read\_nbt\_data (get\_nbt\_data), 20

`read_subchunk_blocks_value`  
(SubChunkBlocks), 30

`simulation_area`, 29

`spawning_area`, 30

`starts_with(key_prefix)`, 21

`subchunk_blocks_array_as_value`  
(SubChunkBlocks), 30

`subchunk_blocks_value_as_array`  
(SubChunkBlocks), 30

`subchunk_coords` (SubChunkBlocks), 30

`subchunk_origins` (SubChunkBlocks), 30

SubChunkBlocks, 11, 30

`unnbt(nbt)`, 24

`vanilla_block_list`  
(vanilla\_block\_states\_df), 33

`vanilla_block_property_type_list`  
(vanilla\_block\_states\_df), 33

`vanilla_block_states_df`, 33

`with_db(default_db)`, 17

`withr::with_connection`, 17

`worlds_dir_path` (minecraft\_worlds), 23

`write_acdig_value` (ActorDigest), 2

`write_data3d_value` (Data3D), 16

`write_leveldat` (read\_leveldat), 29

`write_nbt` (get\_nbt\_data), 20

`write_nbt_data` (get\_nbt\_data), 20

`write_subchunk_blocks_value`  
(SubChunkBlocks), 30