Databind

Release 0.1.0

Adam Thompson-Sharpe

Jun 23, 2021

CONTENTS

Contents:

CHAPTER

ONE

GETTING STARTED

Get started with Databind.

1.1 Installation

Databind is installed using cargo. With cargo installed, run `cargo install databind` to get the latest version. If Rust is in your path, then you should be able to access the CLI by running `databind` in any command line.

1.2 Creating a Project

To create a new project, use the `databind create` command.

```
USAGE:
databind create [OPTIONS] <NAME>

FLAGS:
-h, --help       Prints help information
-V, --version    Prints version information

OPTIONS:
--description <DESCRIPTION>    The pack description [default: A databind pack]
--path <PATH>                  The path to create the pack in

ARGS:
    <NAME>    The name of the project
```

Example use:

`databind create my_project` to create a new project in a folder called `my_project`.

`databind create --description "My first project" my_project` to create a new project with the description `My first project`.

`databind create --path . my_project` to create a new project in the current directory. Only works if empty.

1.3 Writing Code

Below is the default `main.databind` file. `.databind` files **can only be used** to contain function definitions.

```
:func main
:tag load
tellraw @a "Hello, World!"
:endfunc
```

First, a function named main is defined. The name can be changed, it doesn't have to be main. Then, it is tagged with `load`. This tag is normal to datapacks and means that a function will run when the datapack is initially loaded. After that, an ordinary `tellraw`, and then `:endfunc` to close the function definition.

When compiled, this will create a file called `main.mcfunction` that contains the following:

```
tellraw @a "Hello, World!"
```

A `load.json` file will also be generated in `minecraft/tags/functions` to give the function a load tag.

1.4 Building

To build your project, run `databind` in the root directory of your project. Alternatively, you can run `databind <PATH>` where `<PATH>` is the path to your project.

1.5 Additional Files

You are able to create as many `.databind` files and as many namespaces as you'd like. You are also able to mix normal `.mcfunction` files with `.databind` files, meaning you don't have to have a project that only uses Databind. This is helpful if you want to convert a normal datapack to a Databind project. Databind files cannot contain anything other than function definitions, so something such as this alone in a `.databind` file:

```
say Hello, World!
```

Would not generate any output.

1.6 See Examples

If you want to see some examples of language features, go to the *Examples*. Otherwise, you may continue to the next page.

CHAPTER

TWO

SYNTAX

A table of the syntax for different operations.

Syntax	Operation
`:var varName .= <int>`	Define a new variable
`:obj objectiveName <objective>`	Define a new scoreboard objective
`:sobj objectiveName <objective> <target> <assignment operator> <int>`	Set the value of an objective for a given target (eg. `@a` or `PlayerName`)
`:def defName <text replacement>`	Define a text replacement for the preprocessor. See examples for more information
`:var varName <assignment operator> <int>`	Update the value of an existing variable
`:tvar varName`	Used to test variables in `if` commands (eg. `execute if :tvar varName matches 1`)
`:func name`	Define a function. Generates a new mcfunction file
`:endfunc`	Close a function definition.
`:call <function>`	Call a function. Can infer namespace based on directory (see function calling example)
`:while <condition>`	Create a while loop. Condition should be something passable to an `if` command.
`:endwhile`	Close a while loop.
Assignment Operators	
`+=`	Add to a variable.
`-=`	Subtract from a variable.
`=`	Set the value of a variable.

CHAPTER

THREE

DATABIND CLI

3.1 What Can Be Transpiled

Databind transpiles Databind projects (see *Creating a Project*). Databind will look for included files (`**/*.databind` by default) and leave other files alone.

Note that the namespace inference used for `:func` assumes a proper file structure (`<datapack>/data/<namespace>/functions` for functions), but it **does not check if this is the case.** A `minecraft/tags/functions/` folder may be generated in an unexpected place if an invalid folder is passed.

3.2 Using the CLI

```
USAGE:
    databind [FLAGS] [OPTIONS] <DATAPACK>
    databind [FLAGS] [OPTIONS] <SUBCOMMAND>

FLAGS:
    -h, --help                 Prints help information
        --ignore-config        Ignore the config file. Used for testing
        --random-var-names     Add characters to the end of variable names. Does not␣
↪work when using variables across
                               multiple files
        --var-display-names    Change the display name of variables in-game to hide␣
↪extra characters. Only relevant with
                               --random-var-names
    -V, --version              Prints version information

OPTIONS:
    -c, --config <config>    Configuration for the transpiler
    -o, --out <output>       The output file or directory [default: out]

ARGS:
    <DATAPACK>    The Databind project to transpile

SUBCOMMANDS:
    create    Create a new project
    help      Prints this message or the help of the given subcommand(s)
```

3.2.1 From an Installation

When installed, you can access the CLI by running `databind` in any command line. Running `databind --help` will output the text above.

3.2.2 With `cargo run`

After building Databind yourself, you can use `cargo run` to run it. Everything works almost the exact same. You just need to add two dashes (`--`) after `run` (eg. `cargo run -- --help`).

CHAPTER

FOUR

DATABIND CONFIGURATION

4.1 Configuration File

Databind can be configured via the `databind.toml` generated in the project's root. A config file can also be passed with the `-c` or `--config` option.

This table represents the default values of the options if no config changes are made.

Option	Notes
`random_var_names = false`	(Not well-supported) Whether to randomly add characters to the end of variable names
`var_display_names = false`	Whether to update scoreboard display names for randomized variables
`inclusions = ["**/*.databind"]`	Specify what files to transpile using globs
`exclusions = []`	Specify what files not to copy over/transpile using globs
`output = "out"`	The output file or folder

4.2 Example Config

Below is a configuration file with all of the above settings.

```
random_var_names = false
var_display_names = false
inclusions = ["**/*.databind"]
exclusions = []
output = "out"
```

4.3 CLI Arguments

Most options that can be set in the `databind.toml` file can also be set using CLI arguments. The CLI arguments use dashes instead of underscores (eg. `--random-var-names` instead of `random_var_names`) and may have different names or shorthand.

Example use:

```
databind -c config.toml -o ./target ./datapack
```

CHAPTER

FIVE

EXAMPLES

Various examples on how to use Databind and its features.

Contents:

5.1 Text Replacement Examples

Examples using text replacement definitions.

Contents:

5.1.1 Long Execute Commands

Using definitions for a long execute command.

Example

`example/src/data/example/functions/main.databind`

```
:def LONG_EXECUTE execute as @a[scores={custom_item_obj=1..},nbt={SelectedItem:{id:
↪"minecraft:carrot_on_a_stick",tag:{custom_item:1b}}}] at @s

:func tick
:tag tick
LONG_EXECUTE run summon lightning_bolt ^ ^ ^5
LONG_EXECUTE run summon lightning_bolt ^ ^ ^-5
LONG_EXECUTE run summon lightning_bolt ^5 ^ ^
LONG_EXECUTE run summon lightning_bolt ^-5 ^ ^
:endfunc
```

Transpiled

`example/out/data/example/functions/tick.mcfunction`

```
execute as @a[scores={custom_item_obj=1..},nbt={SelectedItem:{id:"minecraft:carrot_on_
↪a_stick",tag:{custom_item:1b}}}] at @s run summon lightning_bolt ^ ^ ^5
execute as @a[scores={custom_item_obj=1..},nbt={SelectedItem:{id:"minecraft:carrot_on_
↪a_stick",tag:{custom_item:1b}}}] at @s run summon lightning_bolt ^ ^ ^-5
```

(continues on next page)

(continued from previous page)

```
execute as @a[scores={custom_item_obj=1..},nbt={SelectedItem:{id:"minecraft:carrot_on_
↪a_stick",tag:{custom_item:1b}}}] at @s run summon lightning_bolt ^5 ^ ^
execute as @a[scores={custom_item_obj=1..},nbt={SelectedItem:{id:"minecraft:carrot_on_
↪a_stick",tag:{custom_item:1b}}}] at @s run summon lightning_bolt ^-5 ^ ^
```

5.2 Function Examples

Examples using functions.

Contents:

5.2.1 Calling

Different ways to call a function.

`function` command

Built into mcfunctions. Requires a namespace.

`example/src/data/example/functions/main.databind`

```
:func example_func
say Hello, World!
:endfunc

function example:example_func
```

`:call` (infer namespace)

Add namespaces to functions while transpiling. Allows more freedom with directory names.

`example/src/data/example/functions/main.databind`

```
:func example_func
say Hello, World!
:endfunc

:call example_func
```

Transpiled, `:call example_func` becomes `function example:example_func`.

`:call` (explicit namespace)

`example/src/data/example/functions/main.databind`

```
:func example_func
say Hello, World!
:endfunc

:call example:example_func
```

Effectively the same as the `function` command.

5.2.2 Simple Function

Example

A function that increments a counter and logs when it's run.

`example/src/data/example/functions/main.databind`

```
:func load
:tag load
:var counter .= 0
:endfunc

:func example
tellraw @a "Example_function run"
:var counter += 1
:endfunc
```

Transpiled

`example/out/data/example/functions/load.mcfunction`

```
scoreboard objectives add counter dummy
scoreboard players set --databind counter 0
```

`example/out/data/example/functions/example.mcfunction`

```
tellraw @a "Example_function run"
scoreboard players add --databind counter 1
```

5.3 Objective Examples

Examples using objectives.

Contents:

5.3.1 Create Objective

Create a scoreboard objective.

Example

```
# Create an objective points and set everyone's score to 100
:obj points dummy
:sobj points @a = 100
```

Transpiled

```
scoreboard objectives add points dummy
scoreboard players set @a points 100
```

5.4 Variable Examples

Examples using variables.

Contents:

5.4.1 Create, Modify & Test

Example

```
# Create a variable called example and set it to 2
:var example .= 2
# Add 1 to example
:var example += 1
# Subtract 2 from example
:var example -= 2
# Set example to 1
:var example = 1
# Say something if example is 1
execute if :tvar example matches 1 run say Variable example is equal to 1!
```

Transpiled

```
scoreboard objectives add example dummy
scoreboard players set --databind example 2
scoreboard players add --databind example 1
scoreboard players remove --databind example 2
scoreboard players set --databind example 1
execute if score --databind example matches 1 run say Variable example is equal to 1!
```

5.5 While Examples

Examples using while loops.

Contents:

5.5.1 For Loop

A for loop-like while loop.

Example

`example/src/data/example/functions/main.databind`

```
:func load
:tag load
:var i .= 10
:while :tvar i matches 1..
tellraw @a "Variable i is above 0"
:var i -= 1
:endwhile
tellraw @a "Variable i is at 0"
:endfunc
```

Transpiled

When while loops are transpiled, functions with random characters at the end are created. In transpiled examples, these characters will be `abcd`.

`example/out/data/example/functions/load.mcfunction`

```
scoreboard objectives add i dummy
scoreboard players set --databind i 10
function example:while_abcd
tellraw @a "Variable i is at 0"
```

`example/out/data/example/functions/while_abcd.mcfunction`

```
execute if score --databind i matches 1.. run function example:condition_abcd
```

`example/out/data/example/functions/condition_abcd.mcfunction`

```
tellraw @a "Variable i is above 0"
scoreboard objectives remove --databind i 1
function example:loop_abcd
```

5.5.2 Loop Until False

Use an integer as a boolean to loop until false.

Example

`example/src/data/example/functions/main.databind`

```
:func load
:tag load
:var bool .= 1
:while :tvar bool matches 1
tellraw @a "Bool is true"
:endwhile
:endfunc
```

Transpiled

When while loops are transpiled, functions with random characters at the end are created. In transpiled examples, these characters will be `abcd`.

`example/out/data/example/functions/load.mcfunction`

```
scoreboard objectives add bool dummy
scoreboard players set --databind bool 1
function example:while_abcd
```

`example/out/data/example/functions/while_abcd.mcfunction`

```
execute if score --databind bool matches 1 run function example:condition_abcd
```

`example/out/data/example/functions/condition_abcd.mcfunction`

```
tellraw @a "Bool is true"
function example:while_abcd
```