NONMEM Modeling with with bbr : : **CHEAT SHEET**

Setup

bbr is an R interface for managing NONMEM and other modeling software via bbi. First you need to set up bbr, described briefly below or see the "Getting Started" vignette for more details.

Make sure this is in your .Rprofile:

```
options("bbr.bbi_exe_path" = file.path(getwd(), "bin", "bbi"))
```

Run this once to install bbi to the path set with bbr.bbi_exe_path:

```
bbr::use_bbi()
```

Run this once per modeling directory to create a bbi.yaml (this is necessary for submitting models):

```
bbr::bbi_init(
  .dir = "path/to/model/dir",
  .nonmem_dir = "opt/NONMEM", # location of nonmem
  .nonmem_version = "nm74gf")   # nonmem version
```

Check the version and location of bbi:

```
bbr::bbi_version()
getOption("bbr.bbi_exe_path")
```

Creating a Model

To submit or interact with models in bbr you need a model object. If you have a .ctl file, you can create the model object (and associated .yaml file) using new_model(). The example below expects to find a control stream file in *model/nonmem/1.ctl*

```
MODEL_DIR <- here::here("model/nonmem")

mod1 <- new_model(file.path(MODEL_DIR, 1))
```

If you previously created a model, read it into a model object using read_model().

```
mod2 <- read_model(file.path(MODEL_DIR, 2))
```

Create a new model by modifying a previous model, using copy_model_from(). See the **Example Workflow** section (next page) for more details.

```
mod3 <- copy_model_from(
   .parent_mod = mod2,  .new_model = 3)
```

Submitting a Model

Submit a single model to be run

```
mod1 %>% submit_model()
```

Additional basic commands:

Submit multiple models to run in batch

```
submit_models(list(mod1, mod2, mod3))

model_paths <- file.path(MODEL_DIR, 1:3)

submit_models(purrr::map(model_paths, read_model))
```

Overwrite output from a previous run

```
submit_model(mod1, .overwrite = TRUE)
```

Parallelization (see bbi.yaml sidebar, next page)

```
submit_model(
  mod1, .bbi_args = list(parallel = TRUE, threads = 8))
```

print_bbi_args() gives a complete list of .bbi_args

Model Outputs

The model_summary() function returns model output as a list of lists.

```
sum1 <- mod1 %>% model_summary()
names(sum1)
[1] "run_details" "run_heuristics" "parameters_data"
[4] "parameter_names"  "ofv"  "condition_number"
[7] "shrinkage_details"
```

For a high-level view of model performance, including any heuristics of concern and a basic parameter table, print the sum1 object to the console. This renders nicely in an R markdown with the chunk option *results = "asis"*.

```
```{r results = "asis"}
print(sum1)
```
```

The param_estimates() helper returns a tibble.

```
param_df1 <- sum1 %>% param_estimates()
param_df1
#> # A tibble: 9 x 8
#>   parameter_names estimate stderr    random_effect_sd random_effect_s...
#> 1 THETA1            2.31    8.61e- 2  NA                NA
#> 5 THETA5            4.13    1.36e+ 0  NA                NA
#> 6 OMEGA(1,1)        0.0964  2.00e- 2  0.311             0.0322
#> 7 OMEGA(2,1)        0       1.00e+10  0                 10000000000
#> 9 SIGMA(1,1)        1       1.00e+10  1                 10000000000
#> # ... with 3 more variables: fixed <chr> diag <lgl>, shrinkage <dbl>
```

You can access the model output list directly to quickly check the run

```
h1 <-
sum1$run_details$run_heuristics
h1$has_final_zero_gradient
[1] TRUE
h1$large_condition_number
[1] FALSE
```

METRUM RESEARCH GROUP

Example Workflow

Create initial model

```
MODEL_DIR <- here::here("model/nonmem")
mod1 <- new_model(
  file.path(MODEL_DIR, 1001),
  .description = "base model") %>%
  add_tags(c("ETA-CL", "ETA-KA", "ETA-V",
             "Prop RUV"))
```

Submit model

```
mod1 %>% submit_model()
```

View model results

```
sum1 <- mod1 %>% model_summary()
# high-level summary
print(sum1)
# parameter tibble
sum1 %>% param_estimates()
```

Create new model based on initial model

```
mod2 <- copy_model_from(
  mod1, 1002, .inherit_tags = TRUE) %>%
  add_tags(c( "Cov CL-WT", "Cov V-WT" ))
```

Open .ctl to add covariates for WT on CL and V

```
open_model_file(mod2)
```

Compare changes to parent model

```
model_diff(mod2)  # compare control streams
tags_diff(mod2)    # compare tags
```

Submit new model

```
mod2 %>% submit_model()
```

View new model results

```
sum2 <- mod2 %>% model_summary()
mod2 %>% param_estiamtes()
```

Add notes (be sure to reassign to object)

```
mod2 <- mod2 %>%
   add_notes("Parameter estimates did not
change much with wt covariates added")
```

Check model status

```
check_up_to_date(mod2)
```

Tags and Notes

The model object has tags and notes fields, to annotate the model during development. Tags are concise and describe model structure, while notes are free form text to notate decisions and observations. Both are meant to replace descriptions in the $PROB of your control stream.

Note: when adding or modifying these attributes, **you must reassign the modified model object**. You can pipe several modifications together.

```
mod4 <- mod4 %>%
  add_tags("ETA-V2") %>%
  replace_tags("Prop RUV", "Add RUV") %>%
  add_notes("First model to use additive error")
```

Defining a glossary of tags

Tags are most useful when defined in a glossary. See “Details” of ?modify_tags for recommendations.

Modifying the model object

Helper functions exist to add, replace, or remove the tags, notes, description, based_on, and bbi_args fields.

```
mod1 <- mod1 %>% add_description("Base model")
mod1 <- mod1 %>% replace_based_on("1001", "1002")
mod1 <- mod1 %>% add_bbi_args(list(parallel = TRUE))
```

Pull all models in a given directory into a tibble with run_log().

```
log_df <- run_log(MODEL_DIR)
```

Use summary_log() to pull the output of model_summary() into a tibble, or use add_summary() to append this output to a run log tibble.

```
log_df <- run_log(MODEL_DIR) %>%
  add_summary() %>% # joins in summary_log() columns
  collapse_to_string(tags, notes) %>%  # formatting for list columns
  select(run, tags, notes, ofv, param_count, condition_number)
log_df
```

```
# A tibble: 3 x 6
  run   tags                          notes                ofv   param_count condition_number
1 1001  ETA-CL, ETA-KA, ETA-V, Prop RUV NA                   18389.  10          57.5
2 1002  ETA-CL, ETA-KA, ETA-V, Prop RUV, Cov CL… Param estimates did not… 18372.  10          30.3
3 1003  ETA-CL, ETA-KA, ETA-V, Prop RUV, ALAG OFV improved with ALAG 18042.  11          39.3
```

Run Log

Checking model and data are up to date

Pass a model object or run log tibble to check_up_to_date() to verify none of the control streams or data files on disk have changed since the models were run.

Or use config_log() or add_config(), which contain model_has_changed and data_has_changed columns.

Global settings in bbi.yaml

The modeling directory includes a bbi.yaml file. This file contains global defaults for many configuration options. These can be overridden by specifying them in .bbi_args when calling submit_model(). However, if you want the same options for most models, you can change them in this file.

```
// bbi.yaml
...
overwrite: true
parallel: true
threads: 8
```

Helper Functions

Extract paths from model object

```
mod1 %>% get_model_path()
mod1 %>% get_data_path()
mod1 %>%
  build_path_from_model(".ext")
```

Check file contents on disk

```
mod1 %>% tail_lst()
mod1 %>% check_grd()
mod1 %>% check_output_dir()
```