

## Certara R School

---

### Lesson 4: Modeling with RsNLME

July 27th, 2022



Keith Nieforth,  
Senior Director, Software Product Management



James Craig,  
R Software Engineer and Shiny Developer

# Certara R School

## Lesson 5: Model Refinement


Aug 23, 2022 @10am EST

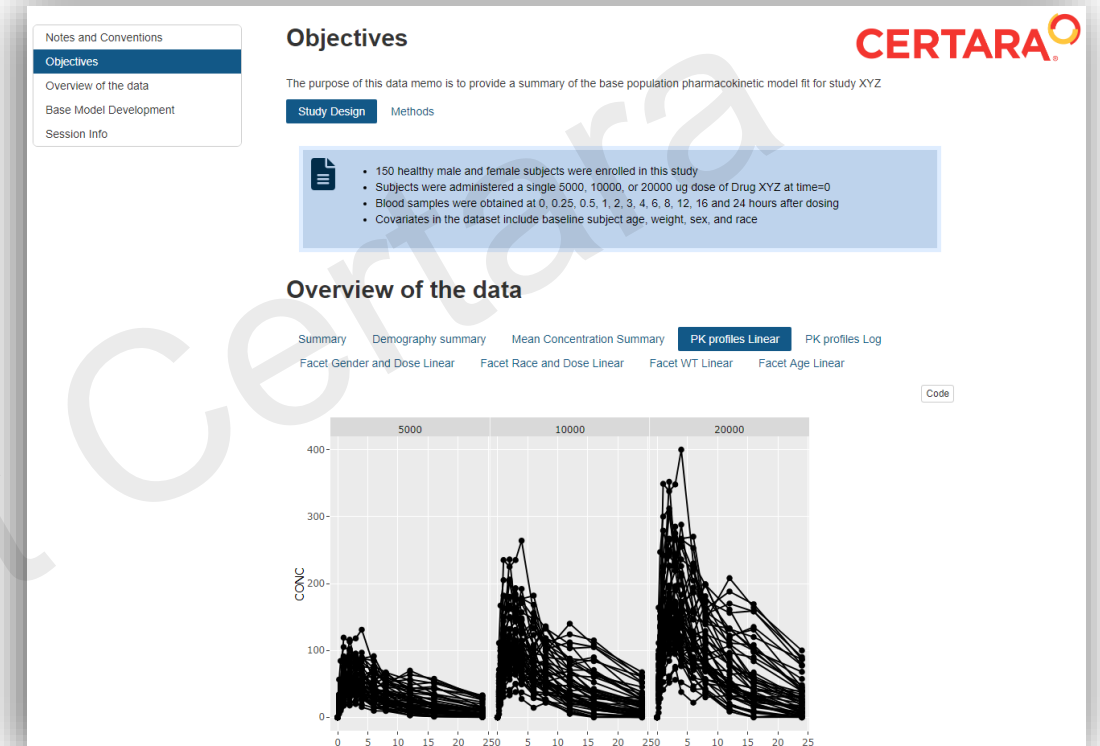
[Register for Lesson 5](#)



# Recap From Last Lecture

## ggquickeda

- Last week we covered exploratory data analysis with ggquickeda
  - GUI for ggplot2
  - Excellent tool that can help you learn ggplot2 code
-  Tip!
  - We are going to start building a report in parallel to our analysis using Rmarkdown
  - Don't worry about the details of this, just follow along
  - We will have a future session on R Markdown



[https://www.youtube.com/watch?v=4Zt29wPi\\_mE](https://www.youtube.com/watch?v=4Zt29wPi_mE)

# What will you learn?

Today we focus on how to set up and execute a model run in RSNLME

## Using Command Line...

```
pkmodel(numCompartments = 1,
        absorption = "FirstOrder",
        data = finaldat,
        columnMap = FALSE,
        modelName = "basemod")

basemod <- basemod %>%
  colMapping(c(id = "ID", time = "TIME", Aa = "AMT", CObs = "CONC"))

basemod <- basemod %>%

  #Set initial parameter estimates
  fixedEffect(effect = c("tvKa", "tvV", "tvCl"), value = c(1.5, 80, 9)) %>%

  #Set initial ETA estimates
  randomEffect(effect = c("nKa", "nv", "nCl"), value = c(0.1, 0.1, 0.1)) %>%

  #Switch to additive error model and set residual error estimate
  residualError(predName = "C", errorType = "Additive", SD = 5)

basemod <- basemod %>%

addCovariate(covariate = c("AGE"),
             type = "Continuous",
             center = "Value",
             centervalue = 45,
             effect = c("v", "Cl")) %>%

addCovariate(covariate = c("WT"),
             type = "Continuous",
             center = "Value",
             centervalue = 70,
             effect=NULL) %>%
```

## Or Using the ModelBuilder User Interface ...

The screenshot displays the CERTARA Model Builder interface. The top navigation bar includes the CERTARA logo and an EXIT button. The main section is titled 'Model Builder' and contains several configuration options:

- Individual/Population:** A toggle switch is set to 'Population'.
- Model Type:** A dropdown menu is set to 'PK'.
- Model Name:** A text field contains 'basemod'.
- PK Structural Model:**
  - Number of Compartments:** A dropdown menu is set to '1'.
  - Elimination:** A dropdown menu is set to 'Linear'.
  - Administration-Absorption:** A dropdown menu is set to 'Intravenous'.
  - Time Lag:** An unchecked checkbox.
  - Options:** A text input field.
- Residual Error Model:**
  - C:** A text field contains 'CObs'.
  - Type:** A dropdown menu is set to 'Multiplicative'.
  - BQL:** An unchecked checkbox.
  - SDDev:** A text field contains '0.1'.
  - Freeze:** An unchecked checkbox.
  - Options:** A text input field.

On the right side, there is a tabbed interface with 'PML', 'RSNLME', and 'DATA' tabs. The 'PML' tab is active, showing a list of model parameters and their values:

```
1 test()
2 cfmicro(A1,Cl,V)
3 dosepoint(A1)
4 C = A1 / V
5 error(Ceps=0.1)
6 observe(CObs=C * (1 + CEps))
7 stparm(V = tvV * exp(nV))
8 stparm(Cl = tvCl * exp(nCl))
9 fixef( tvV = c(1,))
10 fixef( tvCl = c(1,))
11 ranef(diag(nV,nCl) = c(1,1))
12 }
```

# Creating a Model in RsNLME

## General Steps

- Create a new model object
  - Required inputs are arguments to specify basic structural format, a dataset, and a model name
- Check data mappings and map additional variables as necessary
- Adjust the structural terms of the model as necessary
  - Between and residual error structure, covariate model form, etc.
- Provide initial estimates for parameters

The model is then ready for an estimation run

- All of above can be done at the command line in R or using RsNLME shiny apps which will help you learn the code!

# Built-In Models in RsNLME

## Built-in versus “custom” models

- Today’s lecture will focus on working with “built-in” models in RsNLME
- RsNLME has an extensive library of built-in models that can be specified with the functions `pkmodel`, `pkemaxmodel`, `pkindirectmodel`, `pklinearmodel`, and `linearmodel`, or using the `modelBuilder` user interface
  - Built in models have many optional arguments that can produce thousands of unique PMX models
    - Number of compartments, saturable elimination, elimination compartments, absorption models, lag times, PKPD, indirect response, etc.
  - Helper functions are available to modify elements of built-in models
    - Adding covariates, managing error models, setting initial estimates, etc.
- Models that are not captured by built-in functions can be edited directly
  - We call these “custom” models
    - Multiple metabolite models, models with >3 compartment, etc.
    - See PML library for examples: [https://certara.github.io/R-Certara/articles/pml\\_library.html](https://certara.github.io/R-Certara/articles/pml_library.html)
  - Helper functions do not work with custom models
- In many cases, a built-in model can provide exactly what is needed for a given problem
  - If not, a built-in model can provide something close to what is needed and then be further edited

# Define Base PK Model

## Command Line Approach

- We will start by fitting a 1-compartment model with first order absorption
- To create this model we use the pkmodel function
  - E.g., 

```
basemod <- pkmodel(numCompartments = 1,  
                    absorption = "FirstOrder",  
                    columnMap = FALSE,  
                    data = finaldat)
```
- Let's take a deeper dive into the options and use of these functions

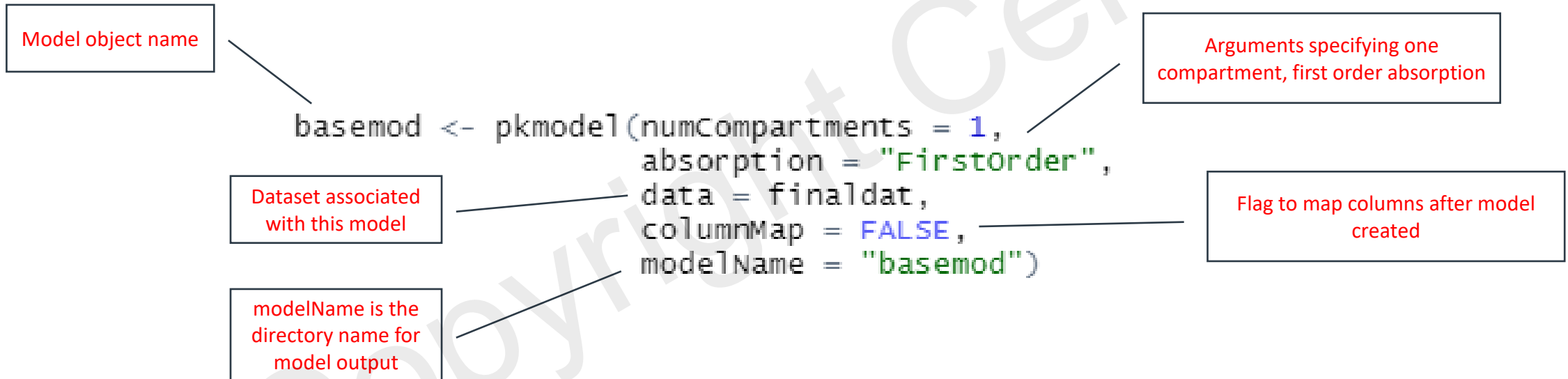
R: Creates a PK model

```
pkmodel(  
  isPopulation = TRUE,  
  parameterization = "Clearance",  
  absorption = "Intravenous",  
  numCompartments = 1,  
  isClosedForm = TRUE,  
  isTlag = FALSE,  
  hasEliminationComp = FALSE,  
  isFractionExcreted = FALSE,  
  isSaturating = FALSE,  
  infusionAllowed = FALSE,  
  isDuration = FALSE,  
  isStdevFrozen = FALSE,  
  data = NULL,  
  ID = NULL,  
  Time = NULL,  
  A1 = NULL,  
  Aa = NULL,  
  A = NULL,  
  A1_Rate = NULL,  
  A1_Duration = NULL,  
  Aa_Rate = NULL,  
  Aa_Duration = NULL,  
  A_Rate = NULL,  
  A_Duration = NULL,  
  A1Strip = NULL,  
  CObs = NULL,  
  C1Obs = NULL,  
  A0Obs = NULL,  
  columnMap = TRUE,  
  modelName = "",  
  workingDir = ""  
)
```

# The pkmodel function

## Basics of a model in RsNLME

- A “model” in RsNLME consists of a dataset, a block of PML code, a set of “mappings” that link the dataset column names to the model variables, and a model name which is used to name the directory for model output



- Data variables can be mapped in the call to `pkmodel` itself, and/or mappings can be specified after the model is created with the `colMapping` function
  - For today's examples we will use the `columnMap = FALSE` flag in the call to `pkmodel` and map variables in a second step



# Printing RsNLME Models

`print(model)` gives a condensed summary of a model object

- Particularly useful for review of PML block, and for checking data mappings
- Here we see that by default, RsNLME
  - Uses a multiplicative residual error model with an initial estimate of 0.1
  - Assigns an ETA term in the form of `*exp(etaP)` to all parameters
  - Sets initial estimate to 1 for all structural parameters and ETAs
- We also see that the model automatically recognized and mapped ID and TIME in the dataset, but does not know which dataset variables should be used for Aa and Cobs
- Helpful Hint:  
use `print(pkindirectmodel(columnMap=FALSE))` for a quick view of default model function settings

General Model Info

PML Code Block

Model Parameters,  
Error Model Info

Mappings

```
> print(basemod)

Model overview
-----
Is population      : TRUE
Model Type        : PK

PK
-----
Parameterization : Clearance
Absorption        : FirstOrder
Num Compartments  : 1
Dose Tlag?        : FALSE
Elimination Comp? : FALSE
Infusion Allowed? : FALSE
Sequential        : FALSE
Freeze PK         : FALSE

PML
-----
test(){
  cfMicro(A1,c1/v, first = (Aa = Ka))
  dosepoint(Aa)
  C = A1 / v
  error(CEps=0.1)
  observe(Cobs=C * ( 1 + CEps))
  stparm(Ka = tvKa * exp(nKa))
  stparm(V = tvV * exp(nV))
  stparm(c1 = tvC1 * exp(nC1))
  fixef( tvKa = c(,1,))
  fixef( tvV = c(,1,))
  fixef( tvC1 = c(,1,))
  ranef(diag(nKa,nV,nC1) = c(1,1,1))
}

Structural Parameters
-----
Ka v c1
-----
Observations:
Observation Name : Cobs
Effect Name      : C
Epsilon Name     : CEps
Epsilon Type     : Multiplicative
Epsilon frozen   : FALSE
is BQL           : FALSE

Column Mappings
-----
Model Variable Name : Data Column name
id                   : ID
time                 : TIME
Aa                   : ?
Cobs                 : ?
```

# The pkmodel function

## Model arguments and mappings

- The arguments in the call to pkmodel determine the structure of the model, and from that, what variables need to be “mapped”

### IV Bolus Model

```
basemod <- pkmodel(numCompartments = 1,  
  absorption = "Intravenous",  
  infusionAllowed = FALSE,  
  data = finaldat,  
  columnMap = FALSE,  
  modelName = "basemod")  
  
print(basemod)
```

#### Column Mappings

Model variable Name	Data Column name
id	: ID
time	: TIME
A1	: ?
CObs	: ?

### IV Infusion Model

```
basemod <- pkmodel(numCompartments = 1,  
  absorption = "Intravenous",  
  infusionAllowed = TRUE,  
  data = finaldat,  
  columnMap = FALSE,  
  modelName = "basemod")  
  
print(basemod)
```

#### Column Mappings

Model variable Name	Data Column name
id	: ID
time	: TIME
A1_Rate	: ?
A1	: ?
CObs	: ?

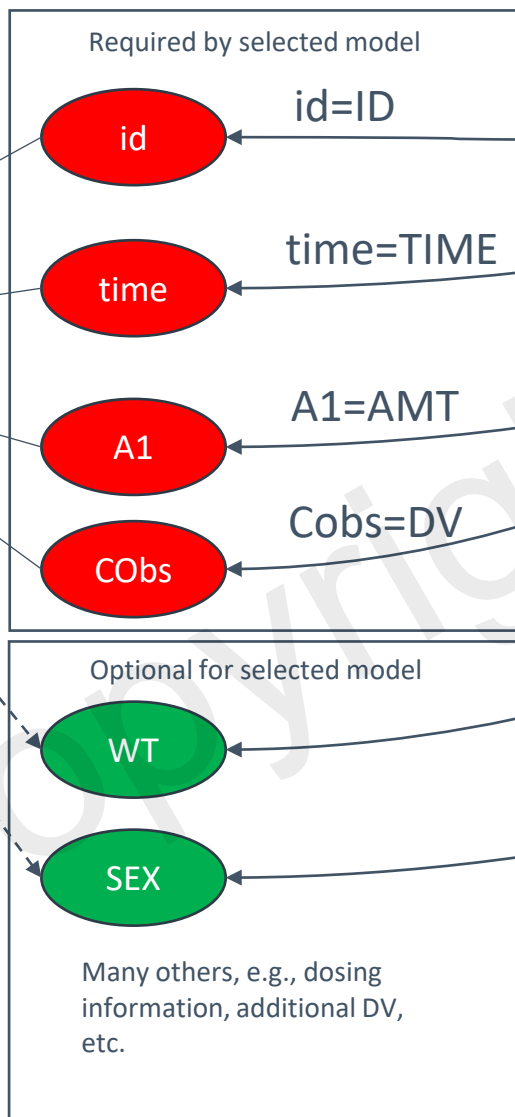
Flag to indicate route of  
input was IV infusion

Here the model is expecting a rate  
of infusion column to be present in  
the dataset, and mapped to the  
model

# Mapping Data Columns

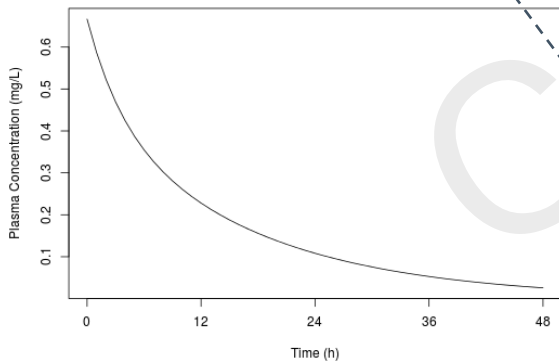
The mapping process connects your dataset to your model – Recall that RsNLME is very flexible with dataset variable names

The Model Expects:



The Dataset Has:

	A	B	C	D	E	F	G	H	I
1	ID	TIME	DV	AMT	WT	CRCL	AGE	SEX	CNTR
2	1	0	0	100	82	90.3	82	0	0
3	1	0.25	1.1191	0	82	90.3	82	0	0
4	1	0.5	1.2574	0	82	90.3	82	0	0
5	1	1	0.91638	0	82	90.3	82	0	0
6	1	2	0.80113	0	82	90.3	82	0	0
7	1	4	0.66857	0	82	90.3	82	0	0
8	1	6	0.59021	0	82	90.3	82	0	0
9	1	8	0.58108	0	82	90.3	82	0	0
10	1	12	0.41931	0	82	90.3	82	0	0
11	1	23.9	0.19041	0	82	90.3	82	0	0
12	1	24	0	100	82	90.3	82	0	0
13	1	36	0.60738	0	82	90.3	82	0	0
14	1	47.9	0.43271	0	82	90.3	82	0	0
15	2	0	0	100	120	111.9	36	0	0
16	2	0.25	0.56653	0	120	111.9	36	0	0
17	2	0.5	0.52905	0	120	111.9	36	0	0
18	2	1	0.5168	0	120	111.9	36	0	0
19	2	2	0.44628	0	120	111.9	36	0	0
20	2	4	0.36115	0	120	111.9	36	0	0



# Mapping Data Columns

## The colMapping function

- Links column names in the dataset to model variable names
- Base model:


```
basemod <- pkmodel(numCompartments = 1,  
  absorption = "FirstOrder",  
  data = finaldat,  
  columnMap = FALSE,      #map columns after model creation  
  modelName = "basemod")  
  
print(basemod)
```

- Map missing columns:

```
basemod <- basemod %>%  
  colMapping(c(id = "ID", time = "TIME", Aa = "AMT", Cobs = "CONC"))
```


- As you build and modify models in the normal course of analysis, always print and check mappings, and update as needed
  - E.g., adding covariate effects, using multiple dosing dataset variables (ADDL, II), using MDV, switching datasets for a simulation run, and more

## Review print(model) output to check mappings



```
-----  
Column Mappings  
-----  
Model Variable Name : Data Column name  
id                  : ID  
time                : TIME  
Aa                  : ?  
Cobs                : ?
```

Missing!



```
-----  
Column Mappings  
-----  
Model Variable Name : Data Column name  
id                  : ID  
time                : TIME  
Aa                  : AMT  
Cobs                : CONC
```

All expected variables mapped

# Refining Initial Model and Setting Initial Parameter Estimates

Commonly used functions to edit built in models:

- [fixedEffect](#) - Specifies the initial values, lower bounds, upper bounds, and units for fixed effects in a model
- [randomEffect](#) - Use to set or update the covariance matrix of random effects in a model object.
- [residualError](#) - Use to change or update residual error model for model object
- [structuralParameter](#) - Use to specify the relationship of the structural parameter with corresponding fixed effect, random effect, and covariate.
- [addCovariate](#) - Add a continuous, categorical, or occasion covariate to model object and set covariate effect on structural parameters.
- [removeCovariate](#) - Remove a covariate from a model

# Refining Initial Model and Setting Initial Parameter Estimates

Commonly used functions to edit built in models:

```
basemod <- basemod %>%  
  
#Set initial parameter estimates  
fixedEffect(effect = c("tvKa", "tvV", "tvCl"), value = c(1.5, 80, 9)) %>%  
  
#Set initial ETA estimates  
randomEffect(effect = c("nKa", "nV", "nCl"), value = c(0.1, 0.1, 0.1)) %>%  
  
#Switch to additive error model and set residual error estimate  
residualError(predName = "C", errorType = "Additive", SD = 5)  
  
print(basemod)
```

## Base Model

```
PML  
-----  
test(){  
  cfMicro(A1,C1/V, first = (Aa = Ka))  
  dosepoint(Aa)  
  C = A1 / V  
  error(CEps=0.1)  
  observe(CObs=C * ( 1 + CEps))  
  stparm(Ka = tvKa * exp(nKa))  
  stparm(V = tvV * exp(nV))  
  stparm(Cl = tvCl * exp(nCl))  
  fixef( tvKa = c(,1,))  
  fixef( tvV = c(,1,))  
  fixef( tvCl = c(,1,))  
  ranef(diag(nKa,nV,nCl) = c(1,1,1))  
}
```



## Refined Base Model

```
PML  
-----  
test(){  
  cfMicro(A1,C1/V, first = (Aa = Ka))  
  dosepoint(Aa)  
  C = A1 / V  
  error(CEps=5)  
  observe(CObs=C + CEps)  
  stparm(Ka = tvKa * exp(nKa))  
  stparm(V = tvV * exp(nV))  
  stparm(Cl = tvCl * exp(nCl))  
  fixef( tvKa = c(,1.5,))  
  fixef( tvV = c(,80,))  
  fixef( tvCl = c(,9,))  
  ranef(diag(nKa,nV,nCl) = c(0.1,0.1,0.1))  
}
```

# Refining Initial Model and Setting Initial Parameter Estimates

## Adding and Removing Covariate Effects

- The addCovariate function is used to add covariates to a model, and to assign covariate effects to parameters

### Base Model

```
basemod <- basemod %>%  
addCovariate(covariate = c("AGE"),  
             type = "Continuous",  
             center = "value",  
             centervalue = 45,  
             effect=NULL)
```

```
basemod <- basemod %>%  
addCovariate(covariate = c("AGE"),  
             type = "Continuous",  
             center = "value",  
             centervalue = 45,  
             effect = c("V", "Cl"))
```

```
PML  
-----  
test(){  
  cfMicro(A1,Cl/V, first = (Aa = Ka))  
  dosepoint(Aa)  
  C = A1 / V  
  error(CEps=5)  
  observe(CObs=C + CEps)  
  stparm(Ka = tvKa * exp(nKa))  
  stparm(V = tvV * exp(nV))  
  stparm(Cl = tvCl * exp(nCl))  
  fixef( tvKa = c(,1.5,))  
  fixef( tvV = c(,80,))  
  fixef( tvCl = c(,9,))  
  ranef(diag(nKa,nV,nCl) = c(0.1,0.1,0.1))  
}
```

```
PML  
-----  
test(){  
  cfMicro(A1,Cl/V, first = (Aa = Ka))  
  dosepoint(Aa)  
  C = A1 / V  
  error(CEps=5)  
  observe(CObs=C + CEps)  
  stparm(Ka = tvKa * exp(nKa))  
  stparm(V = tvV * exp(nV))  
  stparm(Cl = tvCl * exp(nCl))  
  fixef( tvKa = c(,1.5,))  
  fixef( tvV = c(,80,))  
  fixef( tvCl = c(,9,))  
  ranef(diag(nKa,nV,nCl) = c(0.1,0.1,0.1))  
}
```

### Refined Base Model

```
PML  
-----  
test(){  
  cfMicro(A1,Cl/V, first = (Aa = Ka))  
  dosepoint(Aa)  
  C = A1 / V  
  error(CEps=5)  
  observe(CObs=C + CEps)  
  stparm(Ka = tvKa * exp(nKa))  
  stparm(V = tvV * exp(nV))  
  stparm(Cl = tvCl * exp(nCl))  
  fcovariate(AGE)  
  fixef( tvKa = c(,1.5,))  
  fixef( tvV = c(,80,))  
  fixef( tvCl = c(,9,))  
  ranef(diag(nKa,nV,nCl) = c(0.1,0.1,0.1))  
}
```

```
PML  
-----  
test(){  
  cfMicro(A1,Cl/V, first = (Aa = Ka))  
  dosepoint(Aa)  
  C = A1 / V  
  error(CEps=5)  
  observe(CObs=C + CEps)  
  stparm(Ka = tvKa * exp(nKa))  
  stparm(V = tvV * ((AGE/45)^dvdAGE) * exp(nV))  
  stparm(Cl = tvCl * ((AGE/45)^dclAGE) * exp(nCl))  
  fcovariate(AGE)  
  fixef( tvKa = c(,1.5,))  
  fixef( tvV = c(,80,))  
  fixef( tvCl = c(,9,))  
  fixef( dvdAGE(enable=c(0)) = c(,0,))  
  fixef( dclAGE(enable=c(1)) = c(,0,))  
  ranef(diag(nKa,nV,nCl) = c(0.1,0.1,0.1))  
}
```

# Refining Initial Model and Setting Initial Parameter Estimates

## Adding and Removing Covariate Effects

- The `removeCovariate` function is used to remove covariates from a model

### Base Model

```
basemod <- basemod %>%  
  removeCovariate(covariate = c("AGE"),  
    paramName = c("c1")) %>%  
  removeCovariate(covariate = c("AGE"),  
    paramName = c("v"))
```

```
PML  
-----  
test(){  
  cfMicro(A1,c1/v, first = (Aa = Ka))  
  dosepoint(Aa)  
  C = A1 / V  
  error(CEps=5)  
  observe(CObs=C + CEps)  
  stparm(Ka = tvKa * exp(nKa))  
  stparm(V = tvV * ((AGE/45)^dvdAGE) * exp(nv))  
  stparm(c1 = tvC1 * ((AGE/45)^dc1dAGE) * exp(nc1))  
  fcovariate(AGE)  
  fixef( tvKa = c(,1.5,))  
  fixef( tvV = c(,80,))  
  fixef( tvC1 = c(,9,))  
  fixef( dvdAGE(enable=c(0)) = c(,0,))  
  fixef( dc1dAGE(enable=c(1)) = c(,0,))  
  ranef(diag(nKa,nv,nc1) = c(0.1,0.1,0.1))  
}
```



### Refined Base Model

```
PML  
-----  
test(){  
  cfMicro(A1,c1/v, first = (Aa = Ka))  
  dosepoint(Aa)  
  C = A1 / V  
  error(CEps=5)  
  observe(CObs=C + CEps)  
  stparm(Ka = tvKa * exp(nKa))  
  stparm(V = tvV * exp(nv))  
  stparm(c1 = tvC1 * exp(nc1))  
  fcovariate(AGE)  
  fcovariate(WT)  
  fcovariate(SEX())  
  fcovariate(RACE())  
  fcovariate(DOSEGRP)  
  fixef( tvKa = c(,1.5,))  
  fixef( tvV = c(,80,))  
  fixef( tvC1 = c(,9,))  
  ranef(diag(nKa,nv,nc1) = c(0.1,0.1,0.1))  
}
```

- When setting up a base model it is useful to add all covariates to the model as in the refined base model above, but not assign any covariate effects to parameters – This allows covariate variables in the dataset to be recognized as covariates and available for diagnostic plots.



# Refining Initial Model and Setting Initial Parameter Estimates

structuralParameter function is used to specify the relationship of a parameter with it's corresponding fixed effect, random effect and covariate

```
#Change style of structural parameter
basemod <- basemod %>%
  structuralParameter(paramName = 'c1',
                     style = "Normal")

print(basemod)
```

- "LogNormal" (Default): The structural parameter is defined as  $\text{Product} * \exp(\text{Eta})$ .
- "LogNormal1": The structural parameter is defined as  $\text{Sum} * \exp(\text{Eta})$ .
- "LogNormal2": The structural parameter is defined as  $\exp(\text{Sum} + \text{Eta})$ .
- "LogitNormal": The structural parameter is defined as  $\text{ilogit}(\text{Sum} + \text{Eta})$ .
- "Normal": The structural parameter is defined as  $\text{Sum} + \text{Eta}$ .

```
#Remove eta term from Ka
basemod <- basemod %>%
  structuralParameter(paramName = "Ka",
                     hasRandomEffect = FALSE)

print(basemod)
```

```
PML
-----
test(){
  cfMicro(A1,c1/V, first = (Aa = Ka))
  dosepoint(Aa)
  C = A1 / V
  error(CEps=0.2)
  observe(CObs=C * exp( CEps))
  stparm(Ka = tvKa * exp(nKa))
  stparm(V = tvV * exp(nv))
  stparm(c1 = tvC1 + nC1)
  fixef( tvKa = c(,1.5,))
  fixef( tvV = c(,80,))
  fixef( tvC1 = c(,9,))
  ranef(diag(nKa,nV,nC1) = c(0.1,0.1,0.1))
}
```

```
PML
-----
test(){
  cfMicro(A1,c1/V, first = (Aa = Ka))
  dosepoint(Aa)
  C = A1 / V
  error(CEps=0.2)
  observe(CObs=C * exp( CEps))
  stparm(Ka = tvKa)
  stparm(V = tvV * exp(nv))
  stparm(c1 = tvC1 + nC1)
  fixef( tvKa = c(,1.5,))
  fixef( tvV = c(,80,))
  fixef( tvC1 = c(,9,))
  ranef(diag(nV,nC1) = c(0.1,0.1))
}
```

- The structuralParameter style argument also controls how covariate effects are implemented in the model
  - Combine as products, or sums, etc.

# Recap

## Create a new RsNLME Model

- Use the `pkmodel` function to create a new model
- Print the model and check variable mappings, map missing variables if needed
- Set initial parameter estimates, and fine-tune the model structure if necessary
- The created model is ready to be fit
- The concepts covered also apply to the other model creation functions in RsNLME
  - `pkemaxmodel`, `pkindirectmodel`, `pklinearmodel`, `linearmodel`

# Fitting Models

## The fitmodel function

- The fitmodel function is used to execute an NLME estimation run
- The common arguments supplied to fitmodel are a model name, and optionally a “host” and engine parameter settings for the run
- RsNLME Hosts
  - A host is essentially a set of instructions that define a NLME engine configuration, with the main configuration difference being the parallelMethod used by the host
    - The NlmeParallelMethod “none” will not parallelize runs and will use a single computing core
    - The NlmeParallelMethod “LOCAL\_MPI”, also referred to as “parallelizing within model” will run a single model across several computer cores and is useful for speeding up the fitting of complex models
    - The NlmeParallelMethod “multicore”, also referred to as “parallelizing by model” will send individual models to different cores, and is therefore useful for speeding up jobs like bootstrapping and simulation that employ many model replicates
- Host Setup is optional
  - If no host is defined, RsNLME will select a host configuration based on the RsNLME installation and job type

# Fitting Models

## Some Example Host Configurations

```
## Set up commonly used hosts
# host setup: run locally without MPI
localHost <- NlmeParallelHost(sharedDirectory = Sys.getenv("NLME_ROOT_DIRECTORY"),
                             installationDirectory = Sys.getenv("INSTALLDIR"),
                             parallelMethod = NlmeParallelMethod("None"),
                             hostName = "Local",
                             numCores = 1)

# host setup: run locally with MPI enabled
localMPIHost <- NlmeParallelHost(sharedDirectory = Sys.getenv("NLME_ROOT_DIRECTORY"),
                                installationDirectory = Sys.getenv("INSTALLDIR"),
                                parallelMethod = NlmeParallelMethod("LOCAL_MPI"),
                                hostName = "Local_MPI",
                                numCores = 4)

# host setup: run locally with multicore enabled
localMultiCoreHost <- NlmeParallelHost(sharedDirectory = Sys.getenv("NLME_ROOT_DIRECTORY"),
                                       installationDirectory = Sys.getenv("INSTALLDIR"),
                                       parallelMethod = NlmeParallelMethod("multicore"),
                                       hostName = "Local_MultiCore",
                                       numCores = 4)
```

- These are local host configurations, it's also possible to set up remote hosts for e.g., grid runs. See: [https://certara.github.io/R-RsNLME/articles/remote\\_execution.html](https://certara.github.io/R-RsNLME/articles/remote_execution.html)

# Fitting Models

Printing a model fit object provides a basic summary

```
basemodfit <- fitmodel(basemod)  
print(basemodfit)
```



```
> print(basemodfit)  
$overall  
  Scenario RetCode   LogLik   -2LL    AIC    BIC nParm nObs nSub EpsShrinkage Condition  
1: WorkFlow      1 -7187.912 14375.82 14389.82 14427.68    7 1650  150    0.10699  73.06982  
  
$theta  
#Scenario Parameter Estimate Units   Stderr   CV%   2.5%CI   97.5%CI Var.Inf.factor  
1: WorkFlow   tvKa  1.195375    NA 0.03937092 3.293605  1.118152  1.272597      NA  
2: WorkFlow   tvV  80.425584    NA 2.83247767 3.521861 74.869937 85.981231      NA  
3: WorkFlow   tvCl  7.731636    NA 0.42486324 5.495127  6.898305  8.564967      NA  
4: WorkFlow   CEps 13.789523    NA 0.77473096 5.618258 12.269959 15.309088      NA  
  
$omega  
  Label    nKa    nV    nCl  
1:  nKa 0.07185426 0.00000000 0.00000000  
2:   nV 0.00000000 0.1687598 0.00000000  
3:  nCl 0.00000000 0.00000000 0.3953833  
  
$omega_correlation  
  Label nKa nV nCl  
1:  nKa  1  0  0  
2:   nV  0  1  0  
3:  nCl  0  0  1  
  
$eta_shrinkage  
  Label    nKa    nV    nCl  
1: Shrinkage 0.3216746 0.05676357 0.06095907  
  
$omega_stderr  
  Label    nKa    nV    nCl  
1:  nKa 0.01546955 0.00000000 0.00000000  
2:   nV 0.00000000 0.02327878 0.00000000  
3:  nCl 0.00000000 0.00000000 0.04707696
```

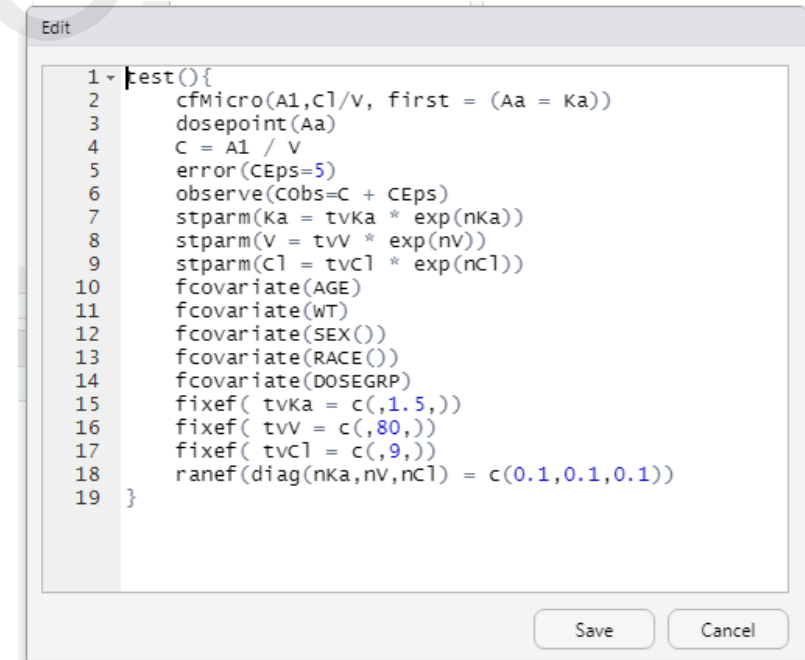
- After a model is fit, it can be passed to other functions that assist with diagnostic review and post-processing
  - Create an xpose database object from the model and use xpose functions
  - Pass directly to modelResultsUI

# Editing Models Directly with the editModel Function

Use to directly edit PML text

- The editModel command is very useful for working with customized models or to quickly edit PML directly
- Recall that all PML syntax is supported in RsNLME – The “built-in” models cover an extensive range of model types, but it is often necessary to customize models to suit unique problems
- The editModel command allows free-text editing of PML code
- Note, once you use editModel, you can no longer use the built-in model functions
- We use copyModel to make a copy of our base model, and then call editModel to edit

```
basemodNew <- copyModel(basemod, modelName = "basemodNew")  
basemodNew <- editModel(basemodNew)
```



```
1 test(){  
2   cfMicro(A1,C1/V, first = (Aa = Ka))  
3   dosepoint(Aa)  
4   C = A1 / V  
5   error(CEps=5)  
6   observe(CObs=C + CEps)  
7   stparm(Ka = tvKa * exp(nKa))  
8   stparm(V = tvV * exp(nV))  
9   stparm(C1 = tvC1 * exp(nC1))  
10  fcovariate(AGE)  
11  fcovariate(WT)  
12  fcovariate(SEX())  
13  fcovariate(RACE())  
14  fcovariate(DOSEGRP)  
15  fixef( tvKa = c(,1.5,))  
16  fixef( tvV = c(,80,))  
17  fixef( tvC1 = c(,9,))  
18  ranef(diag(nKa,nV,nC1) = c(0.1,0.1,0.1))  
19 }
```

# Create a Model with Certara.ModelBuilder

The modelBuilderUI is a R shiny app that provides a graphical user interface for model creation

- The modelbuilderUI allows you to build a model interactively with real-time code update
  - No knowledge of pkmodel function, etc., is required (but it helps!)
  - modelbuilderUI returns code to you based on your selections to help you learn RsNLME command line
- Load the Certara.RsNLME.ModelBuilder package
  - `>library(Certara.RsNLME.ModelBuilder)`
- Supply a model name and dataset in a call to modelBuilderUI

```
basemodui <- modelBuilderUI(finaldat, modelName = "basemodui")
```
- Copy the generated model code into your script

The screenshot displays the Certara Model Builder interface. At the top, the Certara logo is visible. The main section is titled 'Model Builder' and includes a 'Model Type' dropdown set to 'PK' and a 'Model Name' field containing 'basemodel'. Below this, the 'PK Structural Model' section shows 'Number of Compartments' set to 1, 'Elimination' set to Linear, and 'Administration-Absorption' set to Intravenous. There is a checkbox for 'Time Lag' which is currently unchecked. An 'Options' button is located below these settings. The 'Residual Error Model' section shows 'C' set to CObs, 'Type' set to Multiplicative, and 'StDev' set to 0.1. There are checkboxes for 'BQL' and 'Freeze', both of which are unchecked. A 'NEXT' button is at the bottom left. On the right side, there are tabs for 'PML', 'RSNLME', and 'DATA'. The 'RSNLME' tab is active, showing a preview of the generated R code for the model.

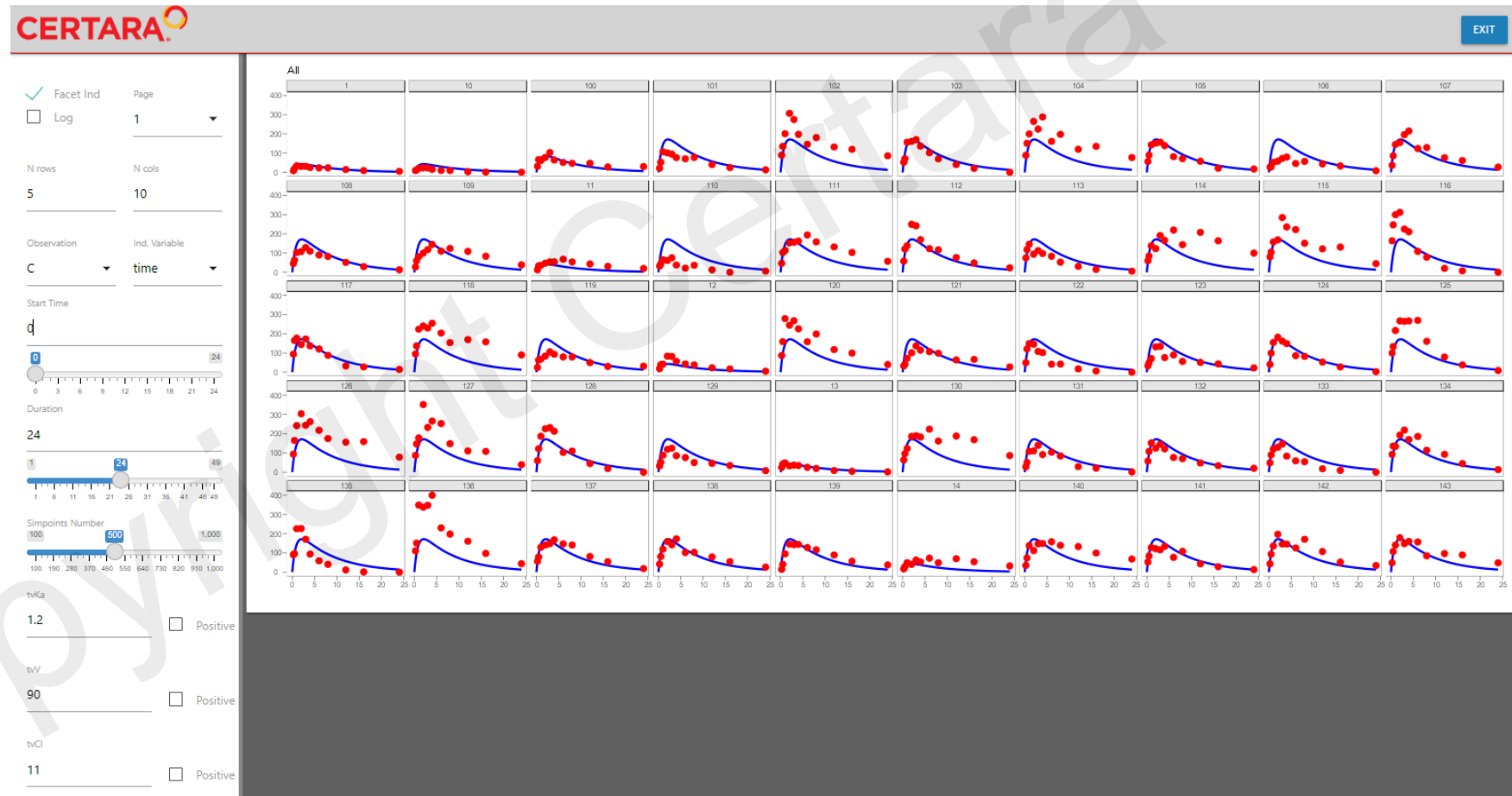
```
1 test(){
2   cfmicro(A1,C1/V)
3   dosepoint(A1)
4   C = A1 / V
5   error(Ceps=0.1)
6   observe(CObs=C * ( 1 + CEps))
7   stparam(V = tvV * exp(nV))
8   stparam(C1 = tvC1 * exp(nC1))
9   fixef( tvV = c(1,))
10  fixef( tvC1 = c(1,))
11  ranef(diag(nV,nC1) = c(1,1))
12 }
```

Below the code preview is an 'Options' button.

# Set Initial Parameter Estimates with estimatesUI

The estimatesUI allows you to interactively set initial parameter estimates for your model

```
basemodui <- estimatesUI(basemodui)
```





# Assignment Lesson 4

We encourage you to try these scripts on your own!

- Run Lesson 4 Script on your own computer
- Try to improve base model (lower -2LL value) by refining model (model structure, covariates, etc.)
- Bring your questions to office hours Aug 3<sup>rd</sup> 10AM EST ([https://certara.github.io/R-Certara/articles/lesson\\_4.html](https://certara.github.io/R-Certara/articles/lesson_4.html))
- We look forward to seeing you there!

# Questions

---



# Certara R School

## Lesson 5: Model Refinement

Aug 23, 2022 @10am EST

[Register for Lesson 5](#)

