

Simulx single page documentation

1. Overview

Version 2024

This documentation is for Simulx.

©Lixoft

Simulx GUI – easy, efficient and flexible application for clinical trial simulations

Now, more than ever, to increase drug success rate and accelerate clinical development it is important to incorporate new technology – like clinical trials simulators. They improve the quality and efficiency of decision making process. Modeling&simulation approach models molecules and mechanisms from the available data and then uses these models to generate new information that can optimize your strategies in terms of time, money and commercial success.



What if it were possible to:

- Use your estimated model and, investigating new dosing regimens and populations, gain unique insights that **improve your chances of success**?
- **Avoid costly surprises** by simulating easily different scenarios with a flexible and user-friendly interface?
- Utilize the outputs from simulations to optimize your clinical trial strategies that **reduces drug development cycle**?

It is possible to do this, and more, with Simulx GUI!

Simulx is an advanced simulation software interconnected with **Monolix** and flexible in building user-designed scenarios. This application combines a user-friendly interface with the highest computational capabilities to help you make faster and more informed decisions.

Simulx has three sections that create an optimal environment to build and analyse simulations:

- **Definition** – create easily new exploration and simulation elements of different types.
- **Exploration** – explore different treatments and effects of model parameters on a typical individual.
- **Simulation** – simulate a clinical trial using a population of individuals in one or several groups with specific treatment or features and use flexible post-processing tools, clear results and interactive plots for analysis.

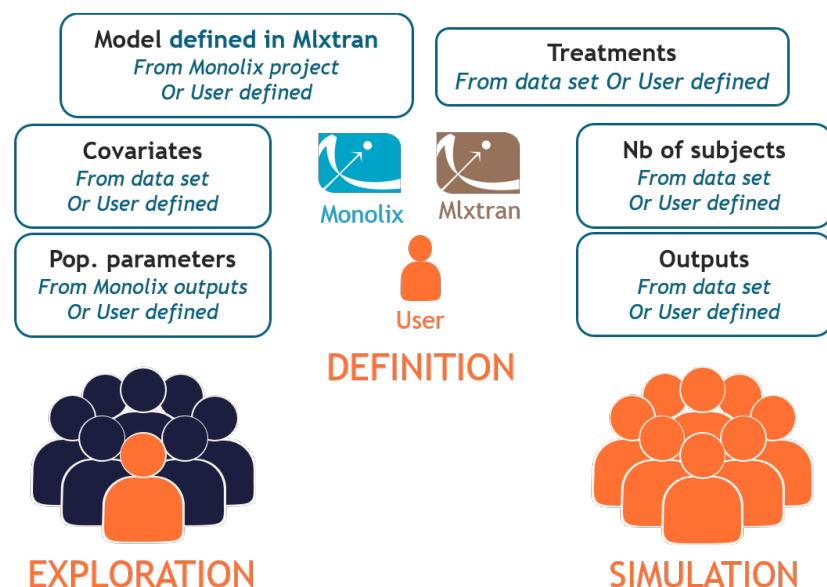
How does it work?

Start with:

- Importing a project from Monolix
- Writing a new project from scratch

play:

- add new dosing regimens, parameters, covariates, outputs, etc.
- explore effects of model parameters and treatments
- combine defined elements into a desired simulation with one or several groups for comparison



- create **outcomes&endpoints** to quantify the simulations outputs
- start immediately the analysis and decision process using automatically generated **results and plots**


and customize:

- plots: add analysis features, stratify data and modify plotting area look
- interface: use dark theme and increase font size and numbers precision
- data: save user files in the result folder
- export: data, plots and settings

Simulx - GUI: a flexible and user-friendly application for simulations

Monika Twarogowska (1), Géraldine Ayrat (1), Pauline Traynard (1), **Jonathan Chauvin (1)**

(1) Lixoft, Antony, France. [Contact: jonathan.chauvin@lixoft.com](mailto:jonathan.chauvin@lixoft.com)



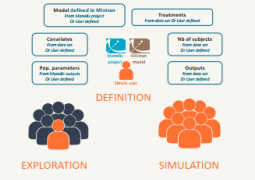
Want to know "What's new" in 2020 release?
[CLICK HERE](#)

INTRODUCTION

Simulx – GUI 2020

- Advanced **simulator** of clinical trials and **decision-making tool**
- **Easy-to-use and interactive interface**
- **Intuitive workflow**
- **Interconnected** with MonolixSuite applications
- **Flexible** in building simulation scenarios
- **Advanced computational capabilities** with C++ engine
- **Immediate visual feedback**
- **Export** of plots and results

How does it work?



DEFINITION **SIMULATION**

EXPLORATION **SIMULATION**

Example

Remifentanyl PKPD dataset - opioid analgesic drug used for sedation tested on 65 healthy adults who received an infusion at different rates, concentration of the drug and EEG measurements

Clinical study questions:

1. What percentage of individuals reach the efficacy target at different dose levels?
2. What is the uncertainty of the result in a small size trial and how does it change if more individuals are recruited?

Methods:


Model: PKPD model from the Monolix library with the population parameters estimated in Monolix

Simulations: qualitative and quantitative comparison of two arms with different treatments and different number of subjects

DEFINITION & EXPLORATION

Import from Monolix

Starting a Simulx project from a Monolix project loads automatically the model developed in Monolix and creates all simulation elements with values estimated by Monolix or read from a dataset.



Definition of simulation elements


Creating new simulation elements is very flexible using single values, regular grids, distributions or external files.

New treatment elements:

- Infusion for 2h with different infusion rates: 25 or 50 µg/min
- Definition of the infusion starting time, the dose amount and the infusion duration
- Other options: cycles, covariate scaling, non-compliance probability

Exploration

Exploration tab allows to simulate a typical individual to investigate different treatments and effects of model parameters.

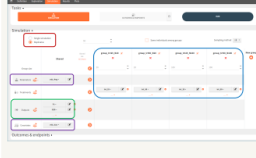


SIMULATION

Simulation of a clinical trial with two treatments

The Simulation tab aim is to simulate populations of individuals in groups, either one or several. It contains a simulation scenario builder which uses simulation elements defined in the "Definition" tab.

- **Four groups** with different treatments (25 or 50 µm/min infusion rate) and different number of individuals (20 or 100)
- **Individual parameters** simulated using population parameters estimated by Monolix and covariates from the original dataset
- **Output:** model predictions for the concentration and EEG
- **Several simulations** of the scenario (**replicates**) for uncertainty assessment




Efficacy target

The Outcomes & Endpoints task is used for post-processing of the simulation outputs. Several outcomes can be combined into an endpoint, and several endpoints can be created. Statistical tests compare endpoints between groups.


- **Outcome:** EEG at time 120 min in the EEG target range [5, 15] Hz
- **Efficacy endpoint:** percentage of individuals in the EEG target
- **Stat test:** Fisher's exact test for the odds ratio between test groups and reference group

PLOTS & RESULTS


Individual output
Concentration vs time with the mean and std.




Output distribution
EEG vs time with the efficacy target limits



Outcome distribution
Number of Individuals in and outside the efficacy target.

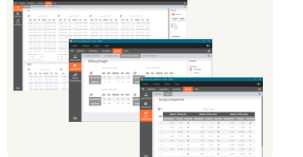


Endpoint distribution
Distribution of the percentage of Individuals in the efficacy target.



Results

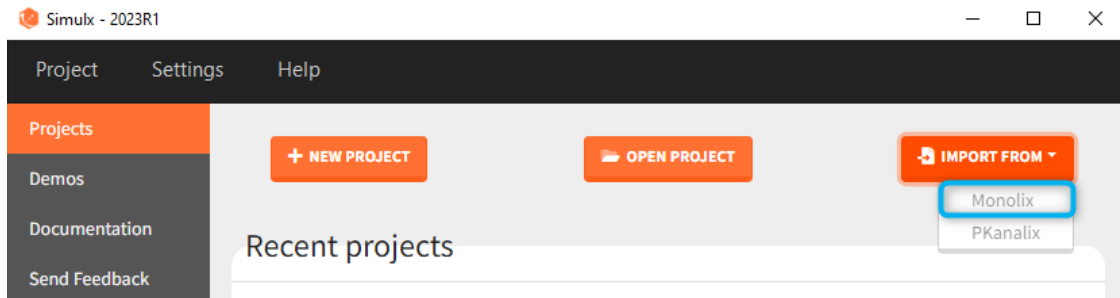
Tables display the percentage of individuals in the EEG target for each replicates. With 20 individuals per arm, this percentage varies from 20 to 65 for the 25µm/min arm and from 50 to 95 for the 50µm/min arm. The difference between arms is significative: 36% over the clinical trial replicates. With 100 individuals per arm, the probability of success is 76% for the 50µm/min arm.



www.lixoft.com www.simulx.lixoft.com

1.1. Import a project from Monolix

There are three methods to start a project in Simulx: **create a new project**, **open an existing project** and **import a project from Monolix or PKanalix**. To import a project from Monolix is as simple as clicking on the button “Import from Monolix” in the Simulx home tab. It is the easiest way to build a simulation scenario, because everything to run a simulation is prepared automatically. As a result, it saves a lot of time. You can always modify current simulation elements, define new ones and change the scenario, so the flexibility of Simulx is not compromised.



1. Simulx project structure with “Import from Monolix”

2. A typical simulation workflow with a project imported from Monolix

Simulx project structure

Importing a project from Monolix creates a Simulx project with pre-defined elements. You can use them to re-simulate the dataset from the Monolix project or as a base for a new simulation scenario. These elements appear in the Definition tab:

- Model, population parameters and individual parameters estimates and output variables are imported from Monolix.
- Occasions, covariates, treatments and regressors are imported from the dataset (used in the imported Monolix project).

Simulx sets default exploration and simulation scenarios – they are ready to run. They contain one exploration group to simulate a typical individual (in the exploration tab) and one simulation group to re-simulate the Monolix project (in the simulation tab).

Default simulation elements

- **model**: mlxtran model with blocks [INDIVIDUAL], [COVARIATE] and [LONGITUDINAL].
- **pop.params.:**
 - **mlx_PopInit** [*no POP.PARAM task results*]: (vector) initial values of the population parameters from Monolix.
 - **mlx_Pop**: (vector) population parameters estimated by Monolix.
 - **mlx_PopUncertainSA** (resp. **mlx_PopUncertainLin**): (matrix) an element which

enables to sample population parameters using the covariance matrix of the estimates computed by Monolix if the Standard Error task (Estimation of the Fisher Information matrix) was performed by stochastic approximation (resp. by linearization). To sample several population parameter sets, this element needs to be used with replicates.

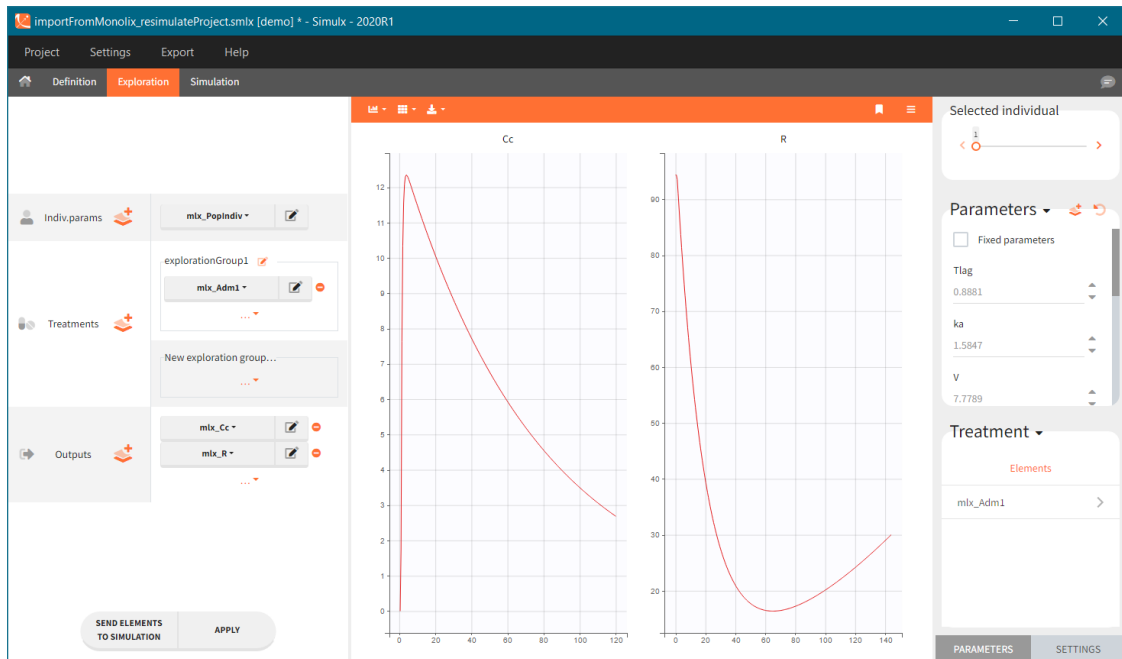
- **indiv.params.:**
 - **mlx_IndivInit** [*no POP.PARAM task results*]: (vector) initial values of the population parameters from Monolix.
 - **mlx_PopIndiv**: (vector) population parameters estimated by Monolix.
 - **mlx_PopIndivCov**: (table) population parameters with the impact of the covariates used in the model (but no random effects).
 - **mlx_EBEs**: (table) EBEs (conditional mode) estimated by Monolix.
 - **mlx_CondMean**: (table) conditional mean estimated by Monolix.
 - **mlx_CondDistSample**: (table) one sample of the conditional distribution (first replicate in Monolix).
- **covariates** [*if used in the model*]:
 - **mlx_Cov**: (table) ids and covariates read from the dataset.
 - **mlx_CovDist**: (distribution) distribution of covariates from the dataset with empirical mean and variance for continuous covariates, set as lognormal for positive continuous covariates, normal for continuous covariate with some negative values, and multinomial low based on frequencies of modalities for categorical covariates.
- **treatment**:
 - **mlx_AdmiD**: (table) ids, amounts and dosing times (+ tinf/rate or washouts) read from the dataset for each administration type.
- **outputs**:
 - **mlx_observationName**: (table) ids and measurement times read from the dataset for each output of the observation model
 - **mlx_predictionName**: (vector) uniform time grid with 250 points on the same time interval as the observations for each continuous output of the structural model.
 - **mlx_TableName**: (vector) uniform time grid with 250 points on the same time interval as the observations for each variable of the structural model defined as table in the OUTPUT block.
- **occasions**:
 - **mlx_Occ** [*if used in the model*]: (table) ids, times and occ(s) read from the dataset.
- **regressors**:
 - **mlx_Reg** [*if used in the model*]: (table) ids, times and regressor values and names

read from the dataset.

Default exploration and simulation scenarios

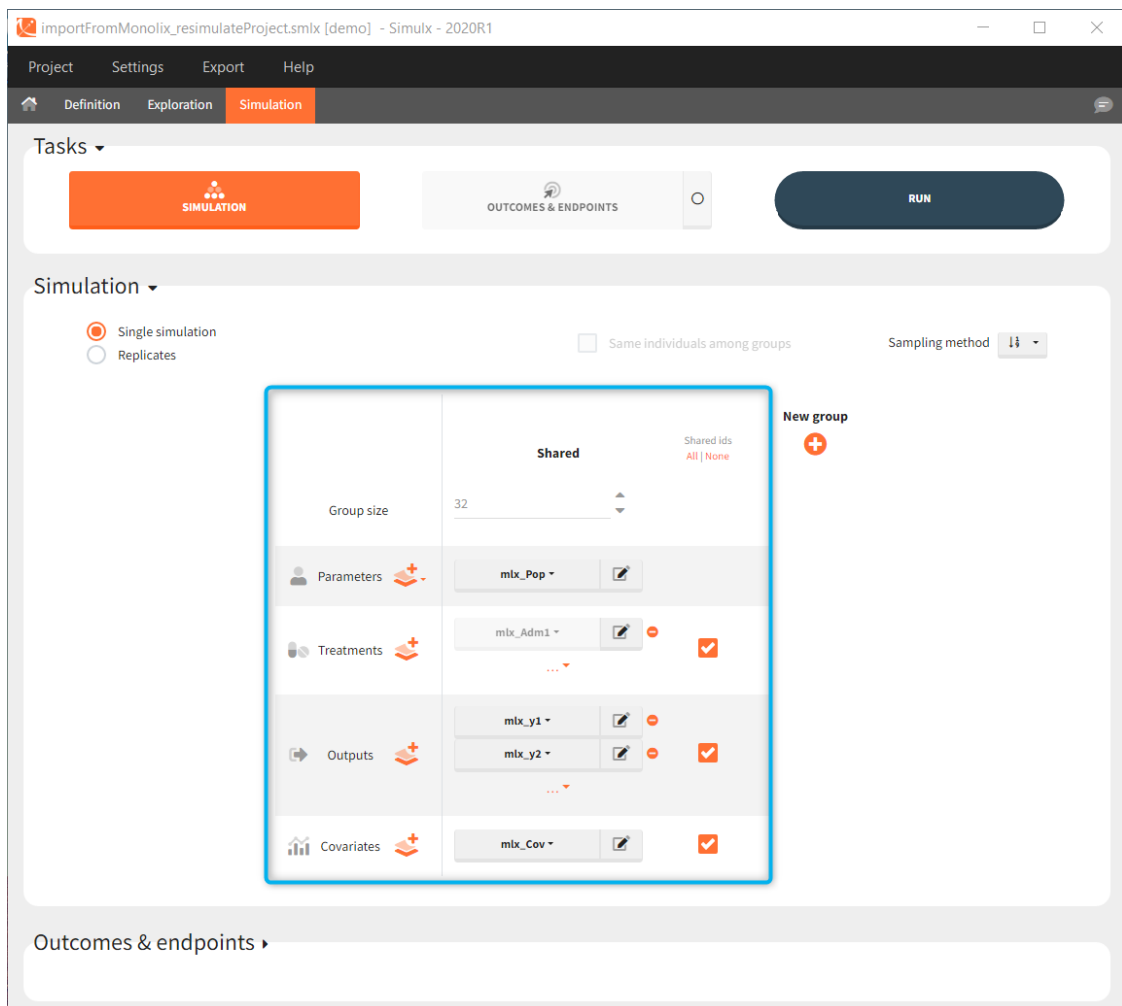
Exploration:

- Individ.params: `mlx_IndivInit` or `mlx_PopIndiv`.
- Treatment: one exploration group with `mlx_AdmlD` for all administration IDs.
- Output: `mlx_predictionName` for all predictions defined in the model.



Simulation:

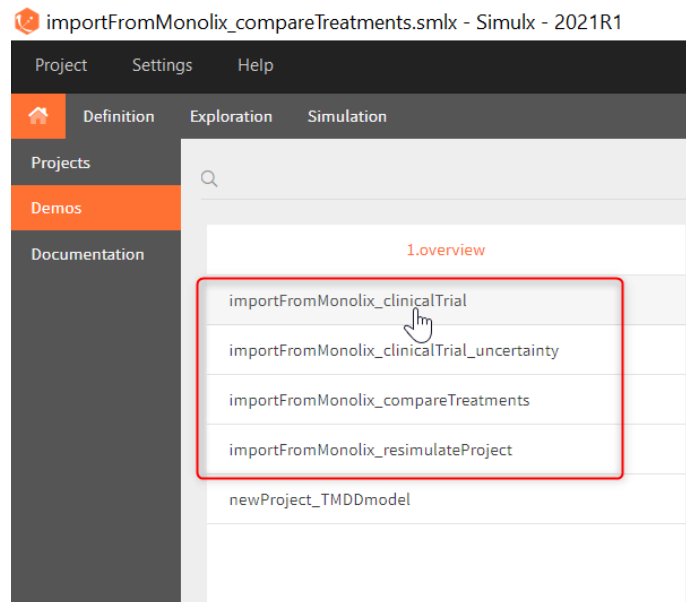
- Size: number of individuals read from the dataset.
- Parameters: `mlx_PopInit` or `mlx_Pop`.
- Treatment: `mlx_AdmlD` for all administration IDs.
- Output: `mlx_observationName` for all observations.
- Covariates: `mlx_cov` if used in the model.
- Regressor: `mlx_reg` if used in the model.



Interface allows to have an overview on all defined elements, modify them and create new ones as well as build simulation scenarios. But, if you modify the imported model, then Simulx will remove all simulation elements. In addition, if you remove occasions, then all occasion-dependent simulation elements will be removed as well.

A typical simulation workflow with a project imported from Monolix

The projects shown here are available as demo projects in the interface "1.overview - importFromMonolix_xxx.smlx":



This example is based on a PK-PD model for Warfarin developed and estimated in Monolix. The Warfarin dataset contains concentration and PCA(%) measurements for 32 individuals, who received different oral doses of the drug. Firstly, the goal of the Simulx project is to use the information from the Monolix project to test the efficacy and safety conditions for different treatments. Secondly, to simulate clinical trials and compare various dosing regimens strategies. Simulations should answer the following questions:

- Which “loading dose” strategy assures a rapid steady state without a concentration peak?
- Do multi-dose treatments meet the efficacy and safety criteria?
- What is the uncertainty of the percentage of individuals in a target due to the variability between individuals and due to the size of a trial group?

Model:

The PK model includes an administration with a first order absorption and a lag time. It has one compartment and a linear elimination. The PD model is an indirect turnover model with inhibition of the production. All individual parameters have log-normal distribution besides the I_{max} parameter, which is logitNormally distributed. In addition, the log-transformed scaled weight covariate explains intra-individual variability of the volume, age covariate has an effect on the clearance and sex covariate on the baseline response. Finally, the combined-1 error model is used in the observation model of the concentration, and the constant model of the response.

0. Re-simulation of the Monolix project

[Demo project “1.overview – importFromMonolix_resimulateProject.smlx”]

You can download the Monolix project for this example, [here](#). To import it in Simulx, first unzip the folder, then start a Simulx session, click on "Import from: Monolix" and browse the .mlxtran file. After importing a project from Monolix, the task buttons "simulation" and "run" in the Simulation tab re-simulate the project. **Plots** and **results** are generated automatically. Results are tables for outputs and individual parameters and plots display model observations as individual outputs and distributions.



1. Exploration of the loading dose strategies

[Demo project "1.overview - importFromMonolix_compareTreatments.smlx"]

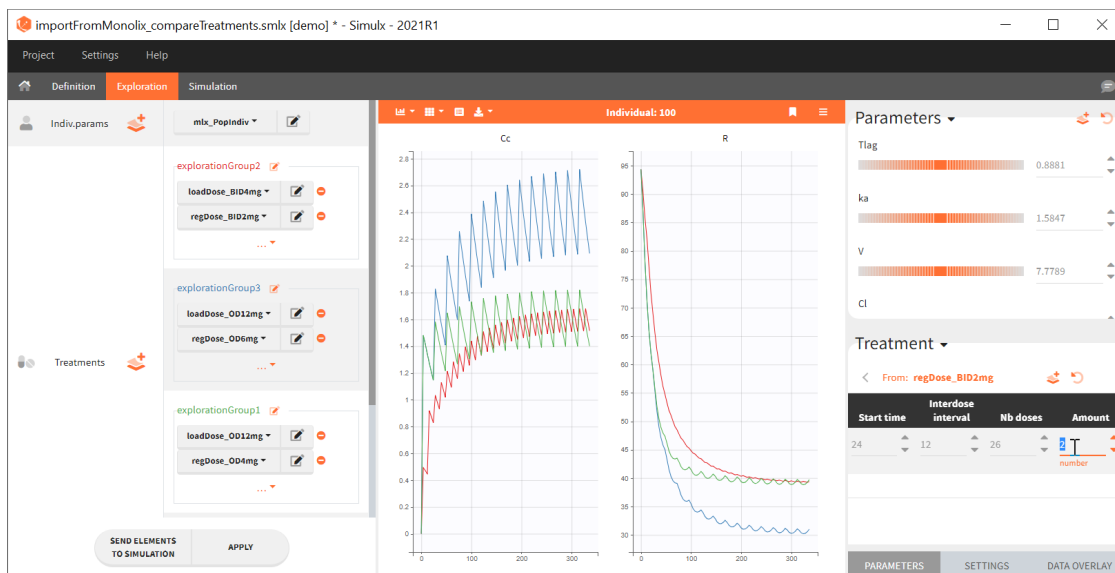
Exploration tab simulates a typical individual. After importing a project from Monolix (the same as in the previous example), you can choose different types of individual parameters elements, eg. equal to population parameters estimated by Monolix or EBEs. Using several exploration groups allows to compare different treatments in one chart. The goal of this example is to test how many days of a “loading dose” are necessary to reach a steady state without a peak of the concentration. Starting dosing regimens are:

- 1 day with a load dose 12mg OD followed by 13 days with a 6mg single dose OD
- 1 day with a load dose 12mg OD followed by 13 days with a 4mg single dose OD
- 1 day with load doses 4mg twice a day (BID) followed by 26 doses of 2mg every 12 hours.

“Loading dose” treatments elements are of manual type (with time of a dose and amount), while multi-dose elements are of regular type. Regular type includes a specification of a treatment period, inter-dose interval and number of doses. You combine treatments elements directly in the exploration tab (in the left panel).

Start time	Interdose interval	Nb doses	Amount
24	12	26	2

Output is the concentration prediction C_c and the response prediction R on a regular time grid over the whole treatment period ($t = 0:1:336$). The plot displays one subplot per output, and all exploration groups (=dosing regimen) together on each subplot. In the right panel, you can edit treatment elements and parameters interactively – predictions are updated on-the-fly for all groups to help you find which exact regimen is the most promising.



2. Treatment comparison: percentage of individuals in the target

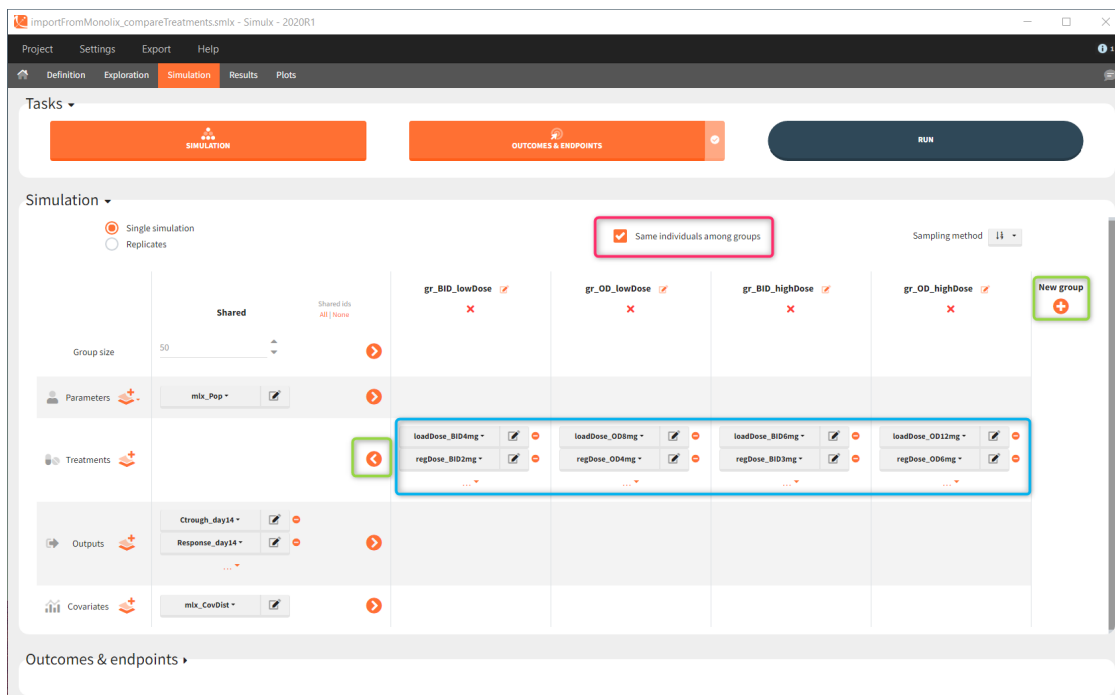
[Demo project "1.overview - importFromMonolix_compareTreatments.smlx"]

Simulation scenario

Exploration tab performed simulation on one individual. In the simulation tab, Simulx simulates a population of individuals. Simulation outputs can be further post-processed to calculate, for example, the percentage of individuals in the target for different treatment arms. This simulation scenario uses the following treatment and output elements.

- BID treatment: One day of a "loading dose" with 4mg or 6mg dose twice a day (BID), followed by 26 doses of 2mg or 3 mg respectively each 12 hours.
- OD treatment: One day of a "loading dose" with 8mg or 12mg dose once a day (OD), followed by 13 doses of 4mg or 6 mg respectively each 24 hours.
- Outputs: manual type using model predictions (Cc and R) at time equal 336h.

In the simulation tab, the button "plus" **adds a new group** and "arrows" (green frames below) move elements from the shared section to the **group specific** section. For treatment, each group has a specific combination of treatments (blue frame). In addition, the option "**same individuals among groups**" removes the effect of intra-individual variability between individuals (red frame). As a consequence, the observed differences between groups are only due to the treatment and simulation can use a smaller number of individuals.



The comparison between groups is in terms of the percentage of individuals in the efficacy and safety target:

- Efficacy: PCA at the end of the treatment should be less than 60%.
- Safety: Ctrough on the last treatment day should be less than 2ug/mL.

You can calculate it in the **outcomes&endpoints** section of the simulation tab. Definition of new outcomes includes selecting an output computed in the simulation scenario and post-processing methods. When you apply threshold condition, then outcome is of a binary (true/false) type.

outcome_efficity

Name

outcome_efficity

Output

Response_day14

relative to baseline (first value of output)

Output processing: average value per id

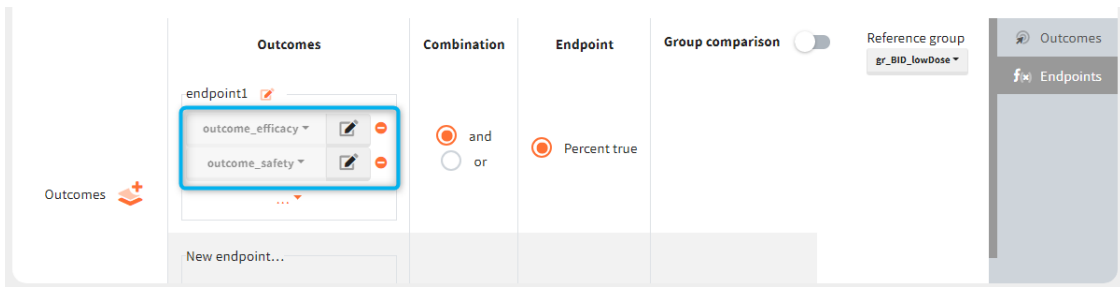
Apply threshold \leq 60

CANCEL

OK

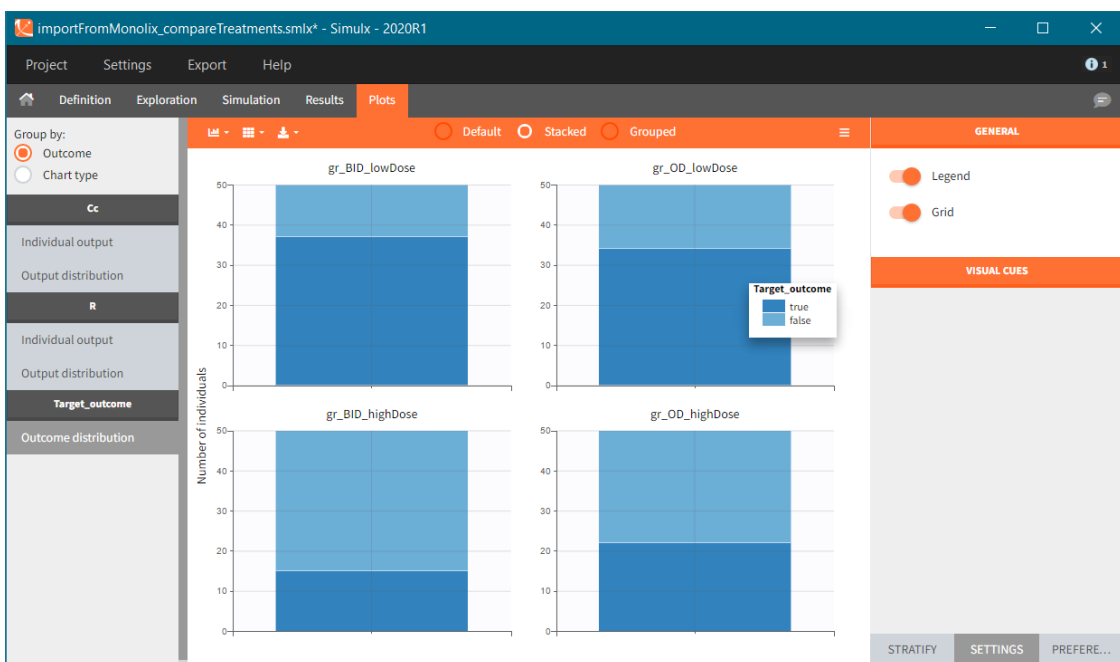
In addition, outcomes can be combined together (blue frame below), and an endpoint summarizes "true" outcomes over all individuals in groups.

Outcomes & endpoints



Analysis

Similarly to the task button “Simulation”, which runs the simulation, the “Outcomes & endpoint” button performs post-processing, which in this example means calculating the efficacy and safety criteria and the number of individuals in the target. When you run this task, Simulx generates automatically the results in the “endpoint” section and plots for outcomes distributions.



In this example, the outcome distribution compares the number of individuals in the target between groups. The highest number of “true” outcomes is in the group with the low level dose twice a day (gr_BID_lowDose). Failure to satisfy the safety criteria by the two groups with the high dose level causes large numbers of “false” outcomes. In fact, the endpoint results show that for these two groups less than half of the individuals have Ctrough below the safety threshold (blue frame below).

outcome_efficacy		outcome_safety			
	TRUE	FALSE		TRUE	FALSE
gr_BID_lowDose	41	9	gr_BID_lowDose	42	8
gr_OD_lowDose	40	10	gr_OD_lowDose	46	4
gr_BID_highDose	50	0	gr_BID_highDose	16	34
gr_OD_highDose	50	0	gr_OD_highDose	22	28

3. Clinical trial simulation and uncertainty of the results

[Demo project "1.overview - importFromMonolix_clinicalTrial.smlx"]

The previous step shows that the high dose level violates the safety criteria in more than 50% of individuals. Therefore, the simulation of a clinical trial uses only one dose level with the BID and OD administration.

The goal is to analyse the effect of the uncertainty due to the variability between individuals, measurement errors and number of individuals in a trial.

New simulation scenario has four groups which combine different dosing regimens (BID or OD) with different group sizes (30 or 100). Outputs are model observations at the end of the treatment. To simulate the scenario several times (green frames), use the option "replicates". As a result, the endpoints summarize the outcomes not only over the groups but also over the replicates.

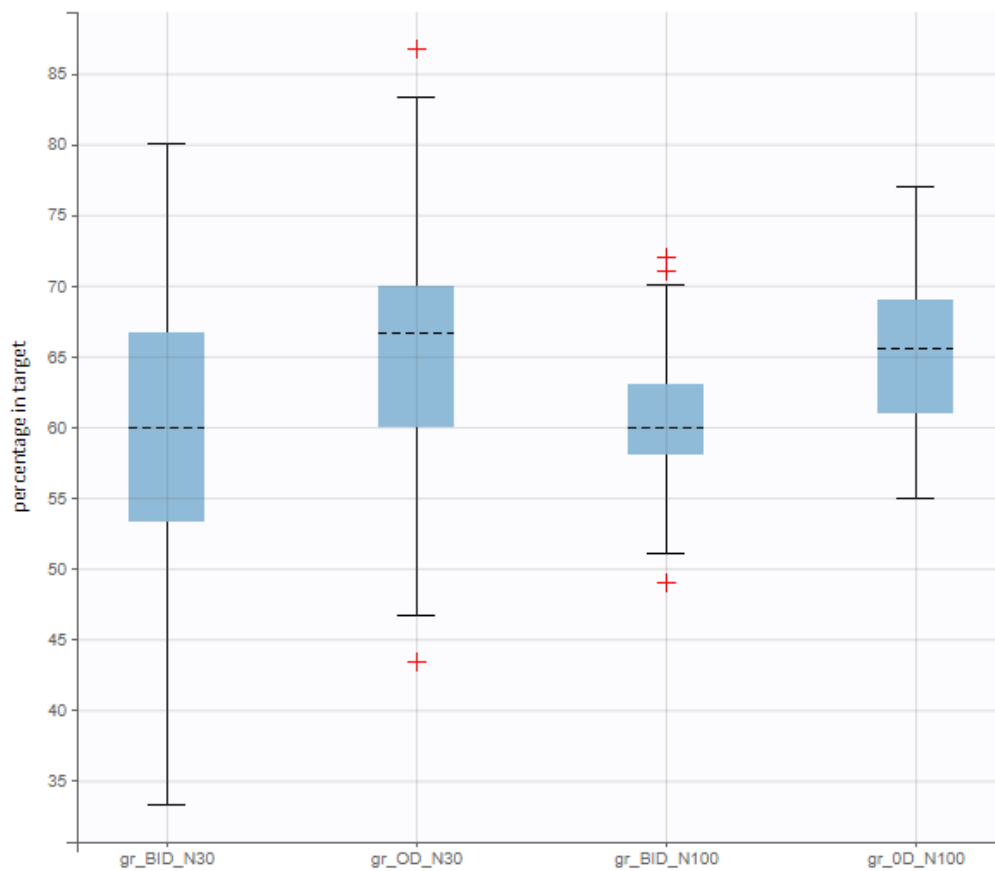
[Starting from the demo project "1.overview - compareTreatments.smlx", change number of replicates to 100, and add two groups: one with treatment as for GR_BID, other as GR_OD. Move "number of ids" as group specific, and set N=30 for one BID-OD pair and N=100 for the other. Change names to indicate group sizes.]

The screenshot displays the Simulx software interface for a clinical trial simulation. The window title is "importFromMonolix_clinicalTrial.smlx* - Simulx - 2020R1". The main menu includes "Project", "Settings", "Export", and "Help". The "Simulation" tab is active, showing a "Tasks" bar with "SIMULATION", "OUTCOMES & ENDPOINTS", and "RUN" buttons. The "Simulation" section is divided into "Simulation" and "Outcomes & endpoints".

In the "Simulation" section, the "Replicates" option is selected and set to 100. The "Same individuals among groups" checkbox is unchecked. The "Sampling method" is set to "iid". A table below shows four groups: "gr_BID_N30", "gr_OD_N30", "gr_BID_N100", and "gr_OD_N100". The "Group size" for "gr_BID_N30" and "gr_OD_N30" is 30, while for "gr_BID_N100" and "gr_OD_N100" it is 100. The "Parameters" are set to "mix_Pop". The "Treatments" are "loadDose_BID4mg", "regDose_BID2mg" for the BID groups, and "loadDose_OD8mg", "regDose_OD4mg" for the OD groups.

In the "Outcomes & endpoints" section, the "Outcomes" are "outcome_efficacy" and "outcome_safety". The "Combination" is set to "and". The "Endpoint" is "Group comparison" with "Statistical test" selected. The "Reference group" is "gr_BID_N30". The "oddsRatio" is set to 1, and the "p-value" is 0.05.

After running both tasks, plots display the endpoint distribution as a box plot with the mean value (dashed line) and standard deviations for each group. As expected, the uncertainty is lower for trial with more individuals. However, the mean values remain at similar levels.



“Group comparison” option in the Outcomes&endpoints section compares the endpoints values across groups (blue frame). Selecting different reference groups and hypothesis (through the odds ratio) allows to change the objectives. Moreover, calculation of new outcomes or endpoints do not require re-running a simulation because the post-processing is a separate task.

The statistical test checks if any of the dosing regimens, BID or OD, is better than the other. Firstly, in the smaller trial so the gr_BID_N30 group is the reference and the hypothesis states that the odds ratio is different from one. For each replicate, if the p-value is lower than selected 0.05, then a test group (GR_OD_N30) is a “success”. Results summary in the “group comparison” section shows that only 5% of the test group replicates are successful (upper image below), which suggests that OD dosing regimen is not significantly better than BID. Larger trial size might give different results. Change reference group to gr_BID_N100 and re-run the outcomes & endpoints task. Percentage of successful replicates for gr_OD_N100 is higher, 8%, (lower image below), but it is comparable to the previous result.

	GR_OD_N30	GR_BID_N100	GR_OD_N100
Target	5	2	7

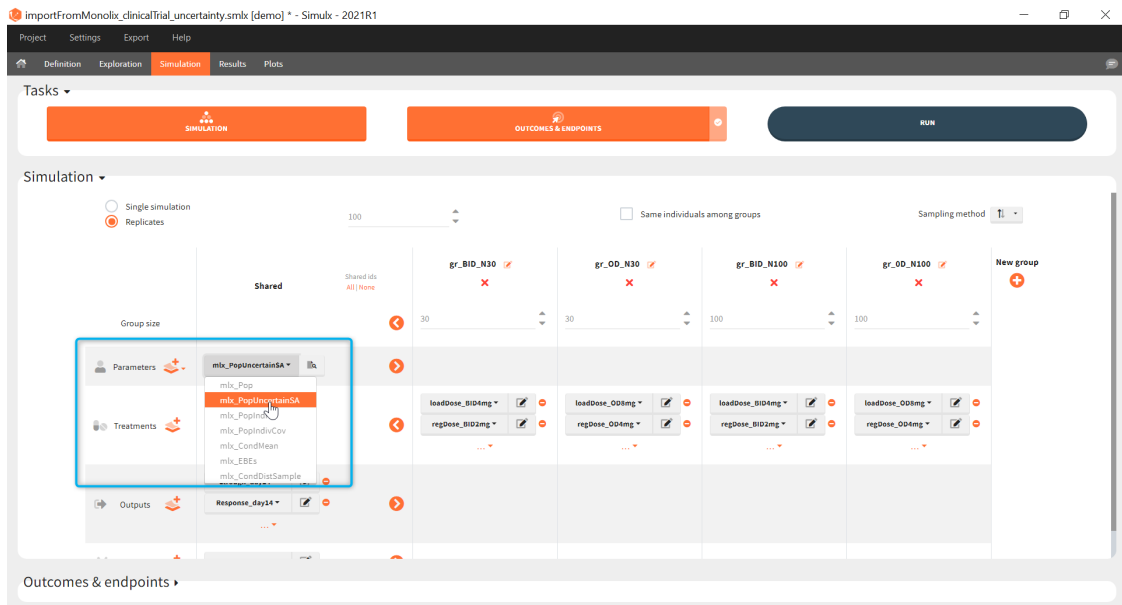
	GR_BID_N30	GR_OD_N30	GR_OD_N100
Target	2	3	8

4. Propagation of parameter uncertainty to the predictions

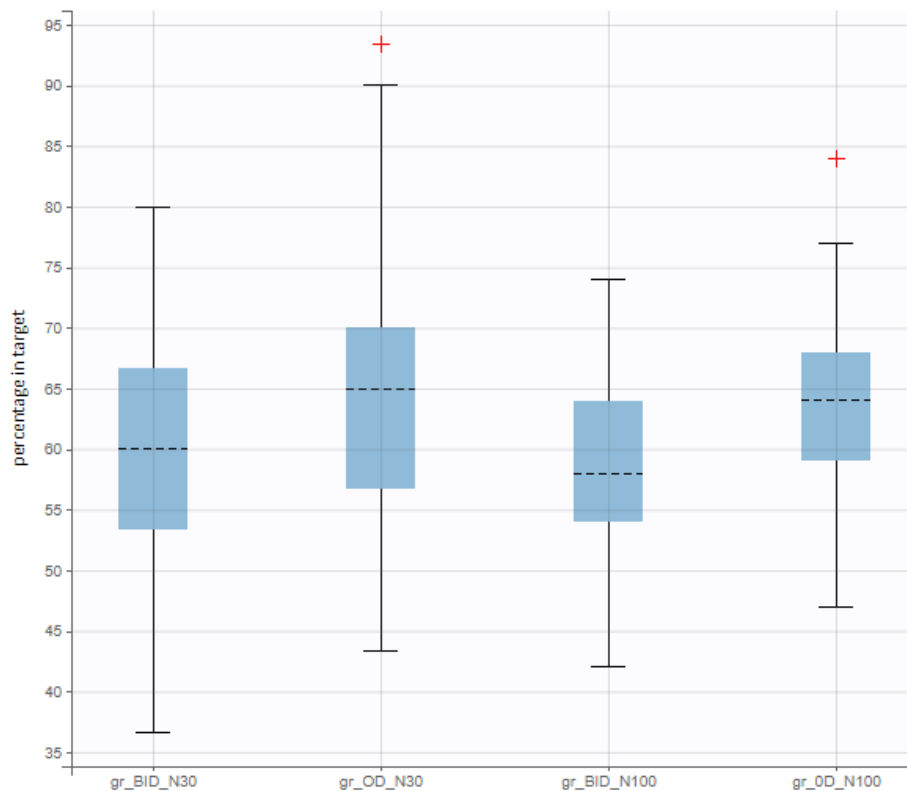
[Demo project in 2021 version only "1.overview - importFromMonolix_clinicalTrial_uncertainty.smlx"]

In the previous step, we replicated the study 100 times to check the impact of sampling different individuals on the endpoint, and on the power of the study. For all these replicates, we sampled individuals using the same population parameters (so same typical values and deviations of random effects).

The goal of this final step is to check the impact of the uncertainty of population parameter estimates on the final endpoint and study power. For this, starting from the 2021 version, it is possible to use the population parameter element `mlx_PopUncertainSA` (resp. `mlx_PopUncertainLin`) if standard errors have been computed by stochastic approximation in the original Monolix project (resp. by linearization). Now the 100 replicates will be samples using each time different population parameter values, which are sampled from the variance-covariance matrix of the estimates imported from Monolix.

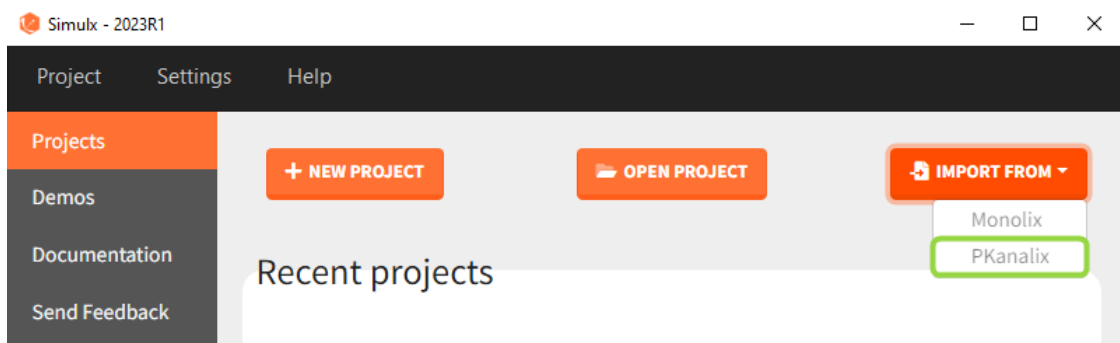


When running again simulation and outcomes with this population parameter element, we obtain a higher uncertainty on the endpoint than in step 3, since now we observe in addition the uncertainty related to the estimation of parameters in Monolix.



1.2. Import a project from PKanalix

There are three methods to start a project in Simulx: **create a new project**, **open an existing project** and **from Monolix to Simulx or import project from PKanalix**. To import a project from PKanalix, simply click the "Import from PKanalix" option of the "Import from" in the Simulx Home tab. It is the most convenient way to create a simulation scenario, as everything is automatically prepared for running a simulation. You can modify current simulation elements, define new ones and change the scenario at any time, such that the flexibility of Simulx is not compromised.



1. Simulx project structure with "Import from PKanalix"

2. A typical simulation workflow with a project imported from PKanalix

Simulx project structure

Importing a project from PKanalix creates a Simulx project with pre-defined elements. You can use them to re-simulate the dataset from the PKanalix project or as a base for a new simulation scenario. These elements appear in the Definition tab:

- Model, individual parameters estimates and output variables are imported from PKanalix.
- Occasions, treatments and regressors are imported from the dataset.

Moreover, exploration and simulation scenarios are set and ready to run. They contain one exploration group to simulate a typical individual (in the exploration tab) and one simulation group to re-simulate the PKanalix project (in the simulation tab).

Default simulation elements

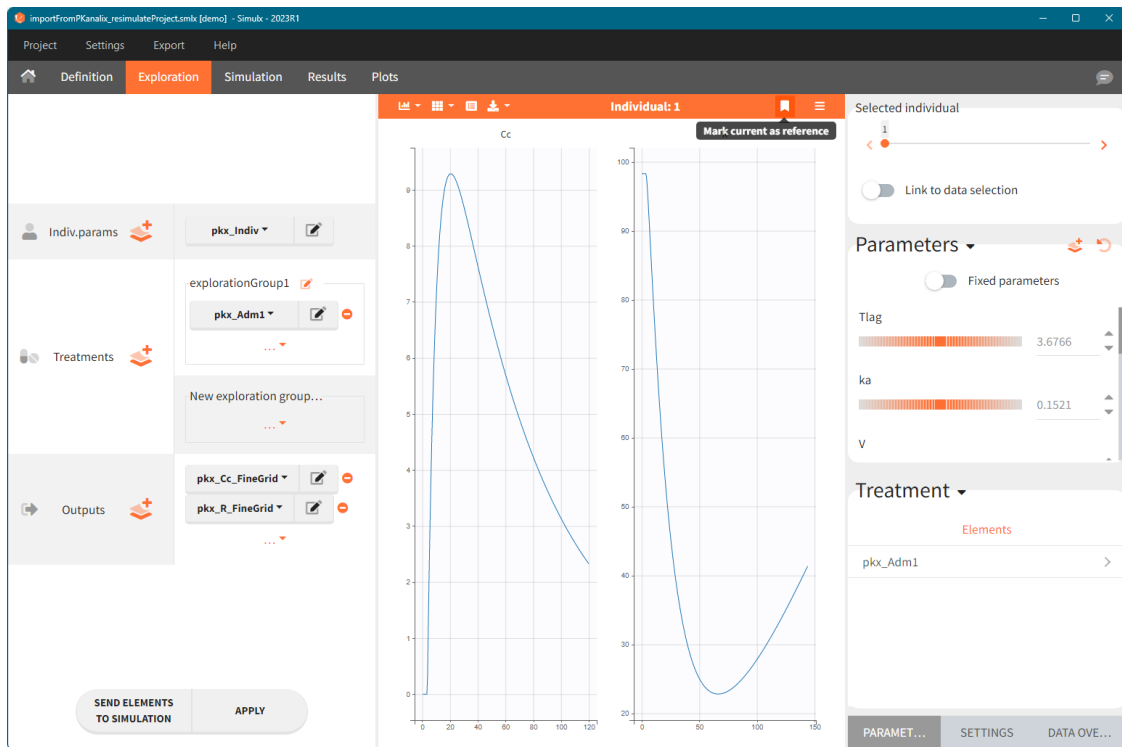
- **model**: same structural model as in the CA Model tab of PKanalix. Model is written in mlxtran language
- **Indiv.params.:**

- **pkx_IndivInit** [*no POP.PARAM task results*]: (vector) individual parameters corresponding to the initial values of the parameters in PKanalix
 - **pkx_Indiv**: (table) individual parameters estimated by CA task
 - **pkx_IndivGeoMean**: (vector) geometric mean of individual parameters over all subjects
- **treatment**:
 - **pkx_AdmiD**: (table) ids, amounts and dosing times (+ tinf/rate or washouts) read from the dataset for each administration type.
- **outputs**:
 - **pkx_outputName_FineGrid** (vector): for each continuous output of the structural model, a vector with a uniform time grid with 250 points on the same time interval as the observations.
 - **pkx_outputName_OriginalTimes** (table): for each output of the model, a table with values for ids and observation times corresponding to the measurements read from the dataset
 - **pkx_TableName**: (vector) for each variable of the structural model defined as table in the OUTPUT block, a vector with a uniform time grid with 250 points on the same time interval as the observations.
- **occasions**:
 - **pkx_Occ** [*if used in the model*]: (table) ids, times and occ(s) read from the dataset.
 - **regressors**:
 - **pkx_Reg** [*if used in the model*]: (table) ids, times and regressor values and names read from the dataset.

Default exploration and simulation scenarios

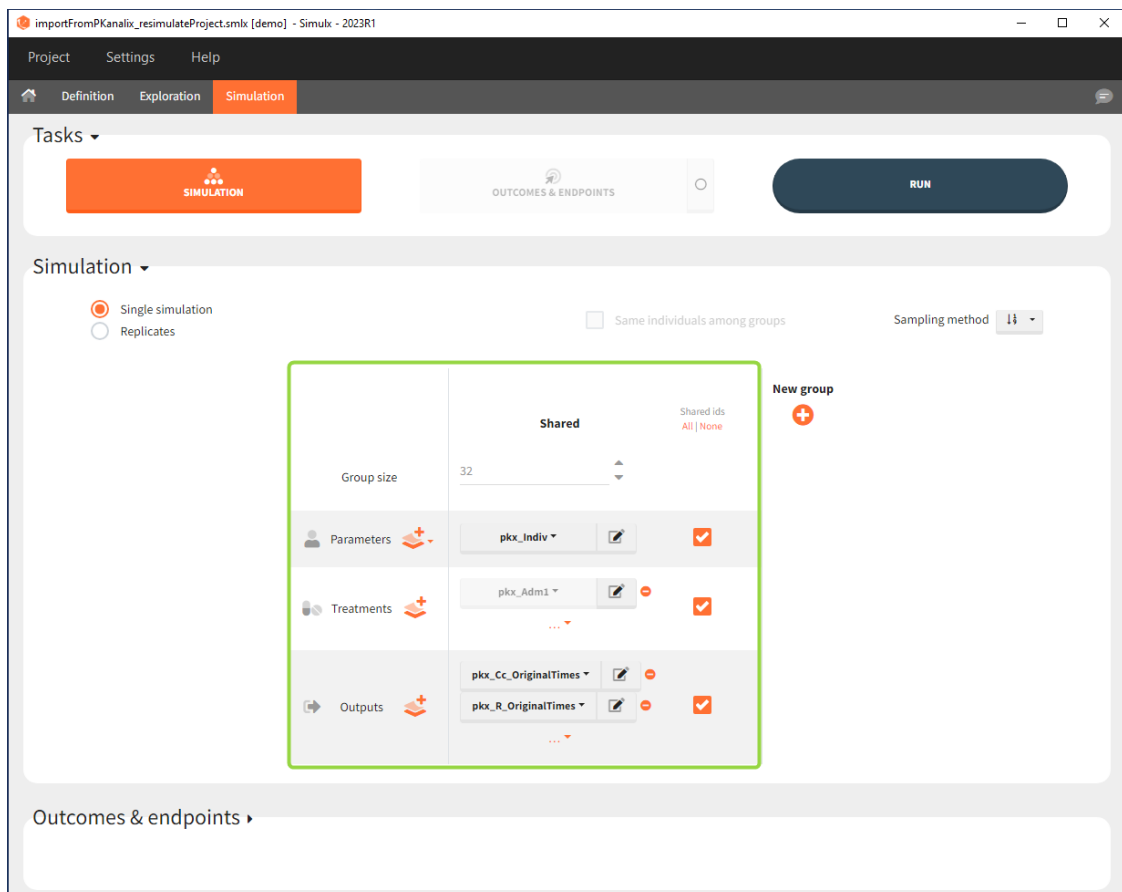
Exploration:

- Individ.params: pkx_Indiv or pkx_IndivInit.
- Treatment: one exploration group with pkx_AdmiD for all administration IDs.
- Output: pkx_outputName_FineGrid for all predictions defined in the model on a fine time grid.



Simulation:

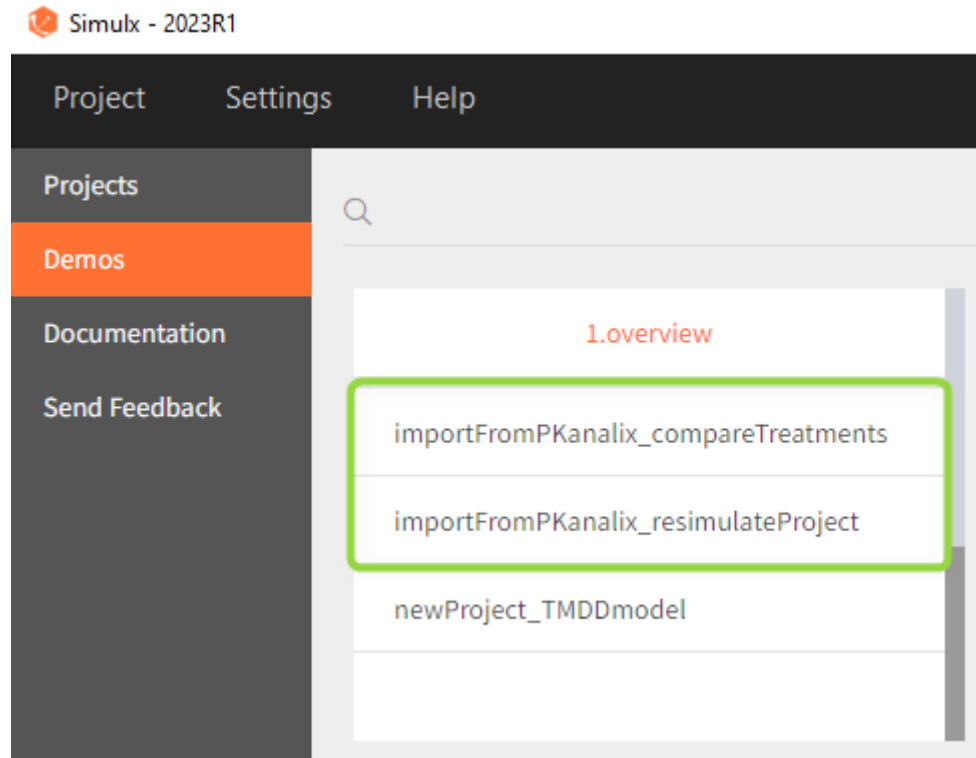
- Size: number of individuals read from the dataset.
- Parameters: pkx_Indiv or pkx_IndivInit.
- Treatment: mx_AdmId for all administration IDs.
- Output: pkx_outputName_OriginalTimes for all model outputs.
- Regressor: pkx_reg if used in the model.



Interface allows to have an overview on all defined elements, modify them and create new ones as well as build simulation scenarios. But, if you modify the imported model, then Simulx will remove all simulation elements. In addition, if you remove occasions, then all occasion-dependent simulation elements will be removed as well.

A typical simulation workflow with a project imported from PKanalix

The projects shown here are available as demo projects in the interface "1.overview – importFromPKanalix_xxx.smlx":



This example is based on a PK-PD model for Warfarin developed and estimated in Monolix. The Warfarin dataset contains concentration and PCA(%) measurements for 32 individuals, who received different oral doses of the drug with the amount 1.5mg/kg. Both the PKPD model and the PKPD parameter estimates come from the compartment analysis (CA) performed in PKanalix. The aim of the Simulx project is to use the information from the PKanalix project to test the efficacy and safety conditions for different treatments. Possible questions to answer with simulations:

- Which "loading dose" strategy assures a rapid steady state without a concentration peak?
- Do multi-dose treatments meet the efficacy and safety criteria?

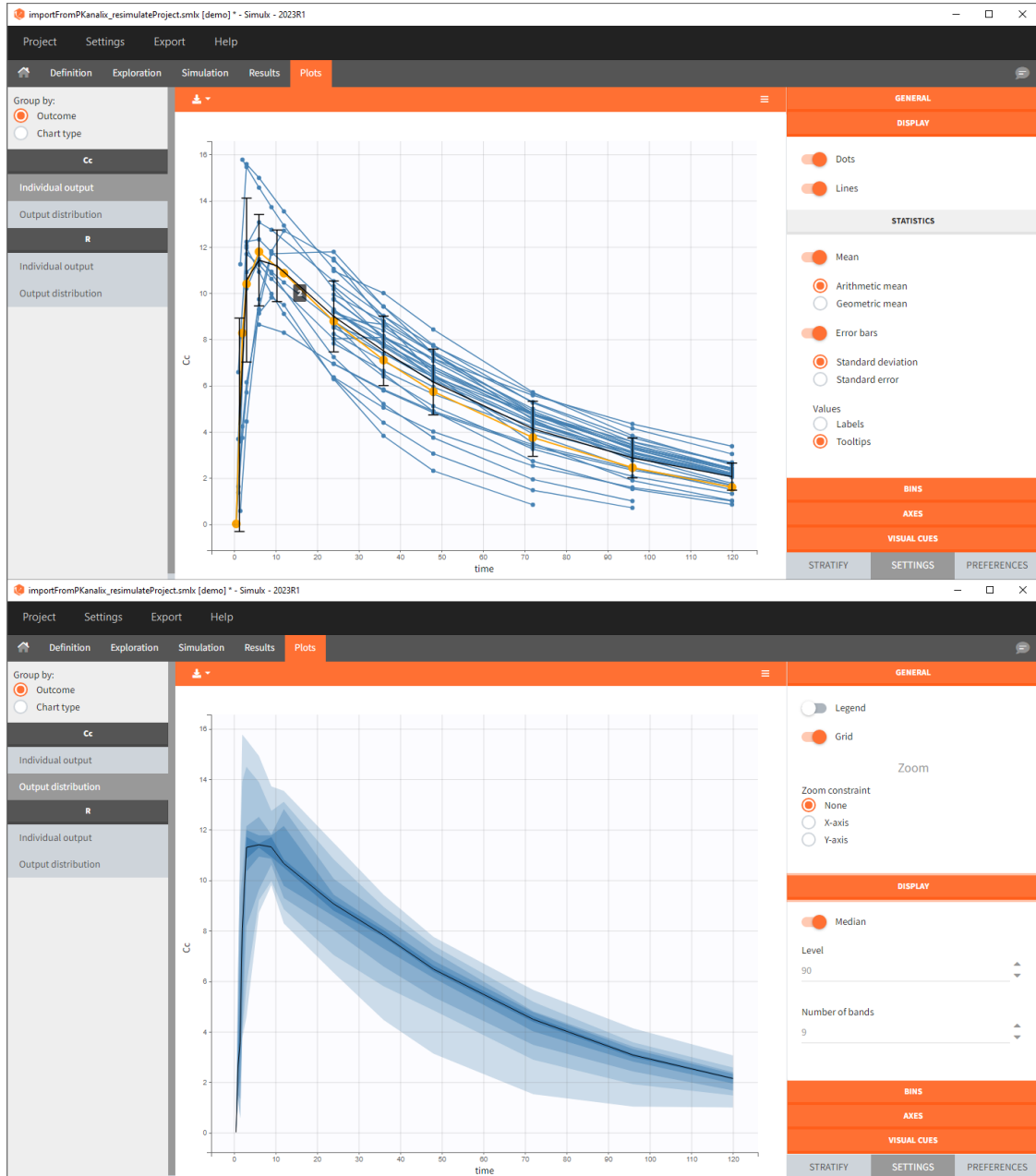
Model:

The PK model includes an administration with a first order absorption and a lag time. It has one compartment and a linear elimination. The PD model is an indirect turnover model with inhibition of the production.

0. Re-simulation of the PKanalix project

[Demo project "1.overview – importFromPKanalix_resimulateProject.smlx"]

You can download the PKanalix project for this example, [here](#). To import it in Simulx, first unzip the folder, then start a Simulx session, click on "Import from: PKanalix" and browse the .pkx file. After importing a project from PKanalix, the task buttons "simulation" and "run" in the Simulation tab re-simulate the project. **Plots** and **results** are generated automatically. Results are tables for outputs and individual parameters and plots display model observations as individual outputs and distributions.



1. Exploration of the loading dose strategies

[Demo project "1.overview - importFromPKanalix_compareTreatments.smlx"]

Definition

The **Exploration tab** simulates a typical individual. After importing a project (the same as in the previous example), you can choose different types of individual parameters elements, eg. individual parameters estimated in the CA task or geometric mean estimated over all individual parameters. Using several exploration groups allows to compare different treatments in one chart. The goal of this example is to test how many days of a “loading dose” are necessary to reach a steady state without a peak of the concentration. Starting dosing regimens are:

- 1 day with a load dose 12mg OD followed by 13 days with a 6mg single dose OD
- 1 day with a load dose 12mg OD followed by 13 days with a 4mg single dose OD
- 1 day with load doses 4mg twice a day (BID) followed by 26 doses of 2mg every 12 hours.

“Loading dose” treatments elements are of manual type (with time of a dose and amount), while multi-dose elements are of regular type. Regular type includes a specification of a treatment period, inter-dose interval and number of doses. You combine treatments elements directly in the exploration tab (in the left panel).

New treatment with regular schedule

REGULAR MANUAL EXTERNAL

Name: regDose_BID4mg Adm: 1

Values

Start time	Interdose interval	Nb doses	Amount
24	12	26	2

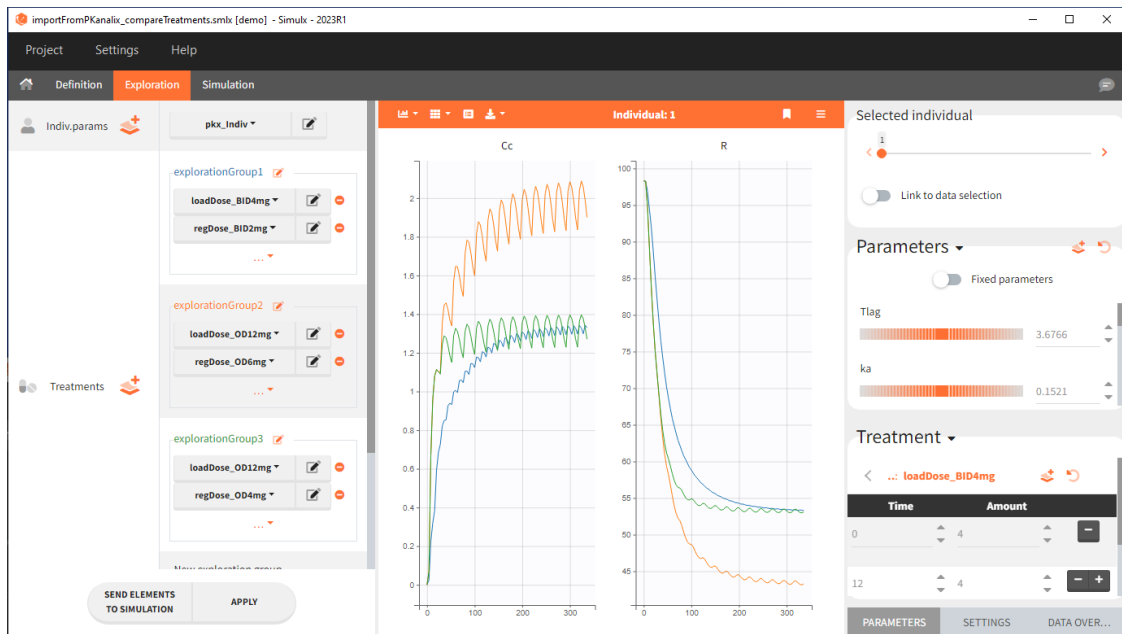
Repeat cycle

Non-compliance probability

CANCEL OK

Exploration

Output is the concentration prediction C_c and the response prediction R on a regular time grid over the whole treatment period ($t = 0:1:336$). The plot displays one subplot per output, and all exploration groups (=dosing regimen) together on each subplot. In the right panel, you can edit treatment elements and parameters interactively – predictions are updated on-the-fly for all groups to help you find which exact regimen is the most promising.



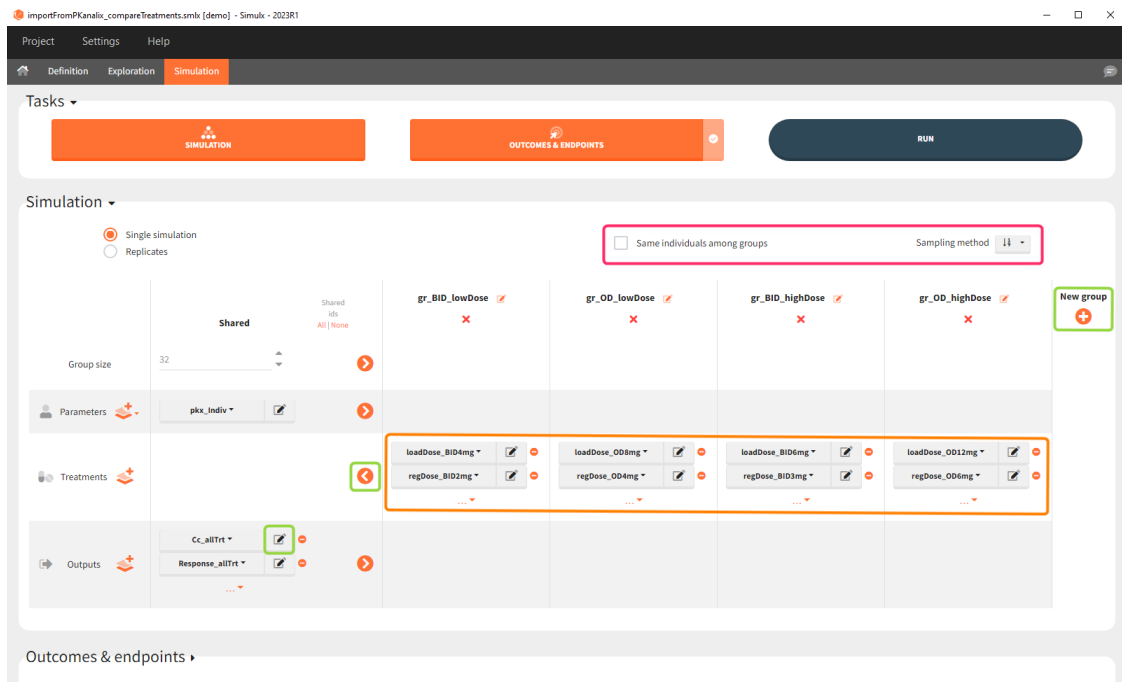
2. Treatment comparison: percentage of individuals in the target

[Demo project "1.overview – importFromPKanalix_compareTreatments.smlx"]

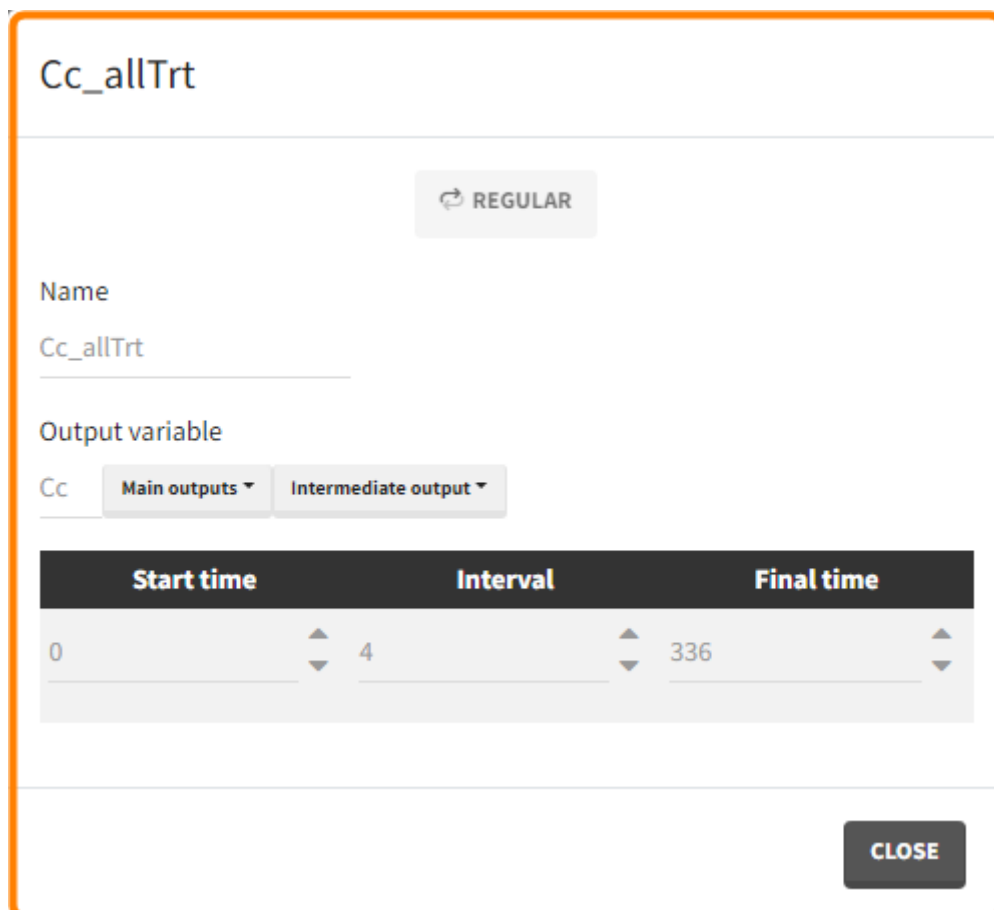
Simulation scenario

Exploration tab performed simulation on one individual. In the simulation tab, Simulx simulates a population of individuals – in this case all individuals from the original dataset used in the imported PKanalix project. Simulation outputs can be further post-processed to calculate, for example, the percentage of individuals in the target for different treatment arms. This simulation scenario uses the following treatment and output elements.

- BID treatment: One day of a “loading dose” with 4mg or 6mg dose twice a day (BID), followed by 26 doses of 2mg or 3 mg respectively each 12 hours.
- OD treatment: One day of a “loading dose” with 8mg or 12mg dose once a day (OD), followed by 13 doses of 4mg or 6 mg respectively each 24 hours.
- Outputs: regular type using model predictions (Cc and R) up to time 336h.



The **Simulation tab** consists of the same element blocks as the Exploration tab. the button “plus” **adds a new group** and “arrows” (green frames below) move elements from the shared section to the **group specific** section. For treatment, each group has a specific combination of treatments (orange frame). As output a new vector was defined. Click on the “edit” icon to see the time discretization. Both output vectors, for the concentration as well as for the response range from time point 0 to time point 336 with step size 4 (unit is in hours h).



The option **“Same individual among groups”** removes the effect of intra-individual variability between individuals (red frame). As a consequence, the observed differences between groups are only due to the treatment itself. This case

Outcomes & endpoints

The efficacy and safety criteria correspond to the following conditions:

- Efficacy: at the end of the treatment PCA should not exceed 60%.
- Safety: on the last treatment day concentration should not exceed 1.5µg/mL

You can calculate it in the **outcomes&endpoints** section of the simulation tab. Definition of new outcomes includes selecting an output computed in the simulation scenario and post-processing methods. When you apply threshold condition, then outcome is of a binary (true/false) type.

outcome_efficiency

Name
outcome_efficiency

Output
Response_allTrt

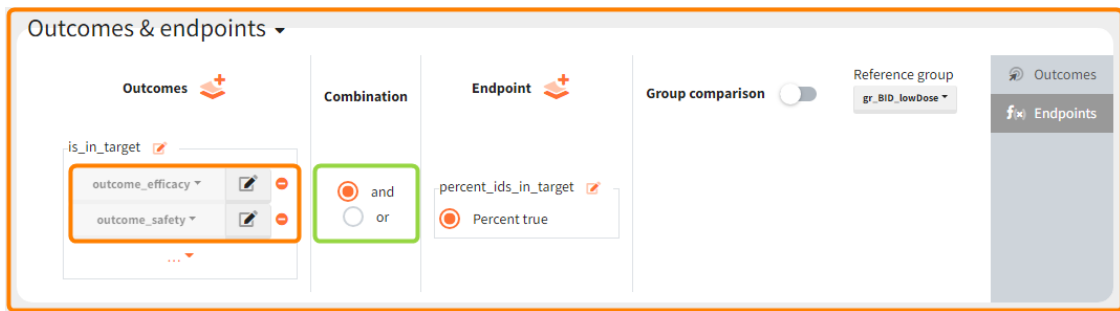
Relative to
none

Output processing:
last value per id

Apply threshold ≤ 60

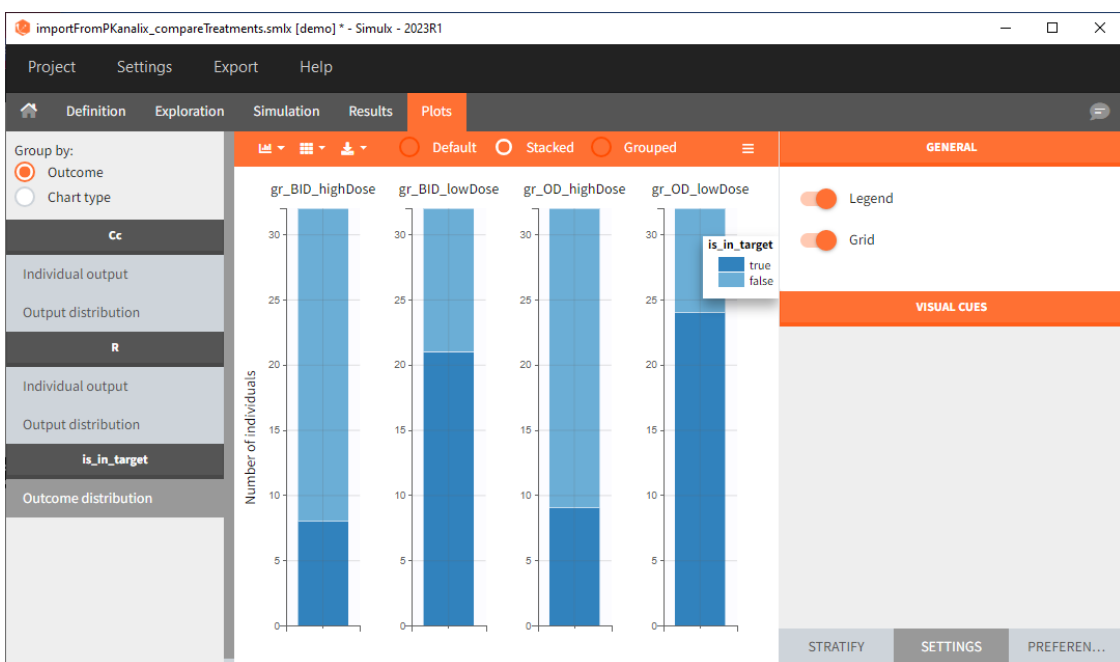
CANCEL OK

In addition, the defined outcomes can be combined together with a logical operator (green frames below), and an endpoint summarizes “true” outcomes over all individuals in groups.



Analysis

Analogous to the "Simulation" button that runs the simulation, the "Outcomes & Endpoint" button performs the post-processing – calculates the efficacy and safety criteria and the number of people in the target. In addition, the execution of this task automatically generates the results in the "Endpoint" section and graphs for the outcomes distributions.



In this example, the outcome distribution compares the number of individuals in the target between treatment groups. Stack charts show the highest number of "true" results ($\hat{=}$ at time $t=336h$ $PCA \leq 60\%$ AND $Cc \leq 1.5\mu g/mL$) in the group with the low dose once daily (gr_OD_lowDose).

Large number of "false" in the two groups with the high doses (gr_OD_highDose, gr_BID_highDose) relates to the violation of the safety criteria. The endpoint results show that for these two groups, less than 1/3 of the subjects have concentration below the safety threshold (green frame below).

importFromPKanalix_compareTreatments.smlx [demo] * - Simulx - 2023R1

Project Settings Export Help

Definition Exploration Simulation Results Plots

OUTCOMES [SUMMARY] OUTCOMES [TABLE] ENDPOINTS [TABLE]

SIMULATION

ENDPOINTS

Summary

DISPLAY

Paginate

COLLAPSE ALL EXPAND ALL

3 Items selected

is_in_target			outcome_efficity			outcome_safety		
	TRUE	FALSE		TRUE	FALSE		TRUE	FALSE
gr_BID_lowDose	21	11	gr_BID_lowDose	25	7	gr_BID_lowDose	27	5
gr_OD_lowDose	24	8	gr_OD_lowDose	25	7	gr_OD_lowDose	31	1
gr_BID_highDose	8	24	gr_BID_highDose	31	1	gr_BID_highDose	9	23
gr_OD_highDose	9	23	gr_OD_highDose	31	1	gr_OD_highDose	10	22

Conclusion

In the higher multi-dose regimens, although 31 of 32 subjects reach the efficacy criterion, only 9 to 10 of these subjects meet the safety criterion.

The group with the lower dose administered once per day, gr_OD_lowDose, have the highest success rate: 24 out of 32 subjects, i.e. 75% of the subjects meet both the safety and the efficacy criteria.

Go to [typical simulation workflow with a project imported from Monolix](#), to see how you can extend this analysis to simulations of clinical trials and assessment of the uncertainty.

1.3. New project from scratch

New possibilities, as well as unexpected and unwanted effects, arise during different phases of drugs development. Simulx allows to create a **new project from scratch**, which **gives freedom** in simulating new scenarios. Moreover, it creates easily a space for testing new hypothesis. This **helps to explore the full potential of drugs**, and better prepare for their possible failures.

To create a new project from scratch:

- Select “New project” after opening the Simulx.
- Browse, or write new, mlxtran model (see [here](#) for details about the model structure).
- Use **Definition tab** to specify simulation elements required by the model. Explore in the **Exploration tab** the effects of treatments and parameters. Build a simulation scenario in the **Simulation tab** and run the simulation.

Demo project: 1.overview/newProject_TMDDmodel

The following example shows how to build “from scratch” a simulation that aims at comparing different treatments among two populations: healthy volunteers and patients with rheumatoid arthritis. It uses a **TMDD model** for a ligand and receptor concentrations. In addition, the synthesis of the receptor depends on the population type – lower for healthy subjects. All simulation elements are user-defined. The basic assumption is that a treatment is effective if the free receptor stays below 10% of the baseline. The goal is to test if the dose levels effective on healthy volunteers allow rheumatoid arthritis patients to reach the efficiency target.

Model

When building a project from scratch, loading a model is mandatory and has to be done in the first place. It sets automatically the structure of a simulation based on the model type and its elements.

- Structural model is a TMDD model with the QE approximation, two compartments and oral administration.
- Statistical model defines an effect of the patient state (healthy or with RA) on the synthesis rate of the receptor. It also includes the power law relationship between the weight covariate and the volume of the central compartment.

Covariates and their transformations are described in the [COVARIATE] block, the statistical model of the individual parameters is in the [INDIVIDUAL] block, while the [LONGITUDINAL] block contains the structural model with output definitions.

Definition of the scenario elements:

To guide building a project from scratch, Simulx defines default simulation elements automatically using information from the loaded model. These elements can be removed or modified, as well as new elements can be created. In this example, the definition includes the following new elements used in the exploration and simulation.

- **Pop.Param.:** Values come from an external table with at least two mandatory rows. The first row contains names of all population parameters as used in the model,

while the second has the parameter values.

- **Indiv.Params.:** Individual parameters appear in the exploration tab. In this example, they are equal to population parameters in order to explore the effect of parameters on the model predictions for a typical individual.
- **Covariates:** The aim of this simulation is to compare two populations (healthy and with rheumatoid arthritis). There are two covariate elements with different values of the “patient” covariate, but the same “weight” covariate.

New covariates from distribution

Name
cov_healthy

Continuous covariates

Covariate	Distribution	Typical	sd
Weight	LOGNORMAL	70	0.1

Categorical covariates

Covariate	Category	Probability	Auto fill
Patient	HV	1	<input type="checkbox"/>
	RA	0	<input checked="" type="checkbox"/>

CANCEL OK

- **Treatment** has two cycles (green frame). In each cycle patients receive an oral dose scaled by their weight (blue frame) once per day for 7 days (red frame) followed by 7 days without a treatment. Three different dose levels: 3mg/kg, 6mg/kg and 12mg/kg require three separate treatment elements. An option “duplicate” allows to clone a treatment, and then change only the name and the amount. Note that the “View” option has the scaling formula.

New treatment with regular schedule

REGULAR MANUAL EXTERNAL

Name
trt_amt3

Adm
1

Values

Start time	Interdose interval	Nb doses	Amount
0	1	7	3

Repeat

Cycle duration: 14 Number of repetitions: 1

Scale amount by a covariate

Covariate: Weight Intercept: 0

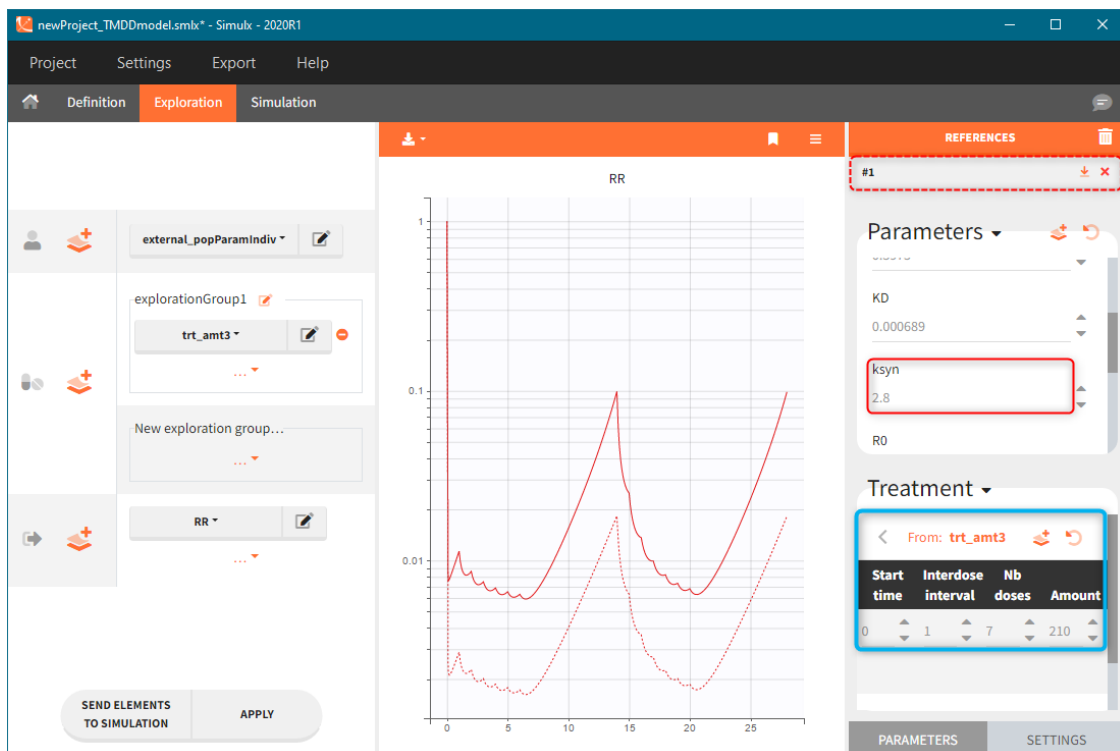
Non-compliance probability

CANCEL OK

- **Outputs** elements include the ratio between the free receptor and the baseline. This output on the regular grid is useful in the exploration. Simulation uses only the value at the end of the treatment. The latter choice speeds up the computations in case of many individuals and/or replicates.

Exploration

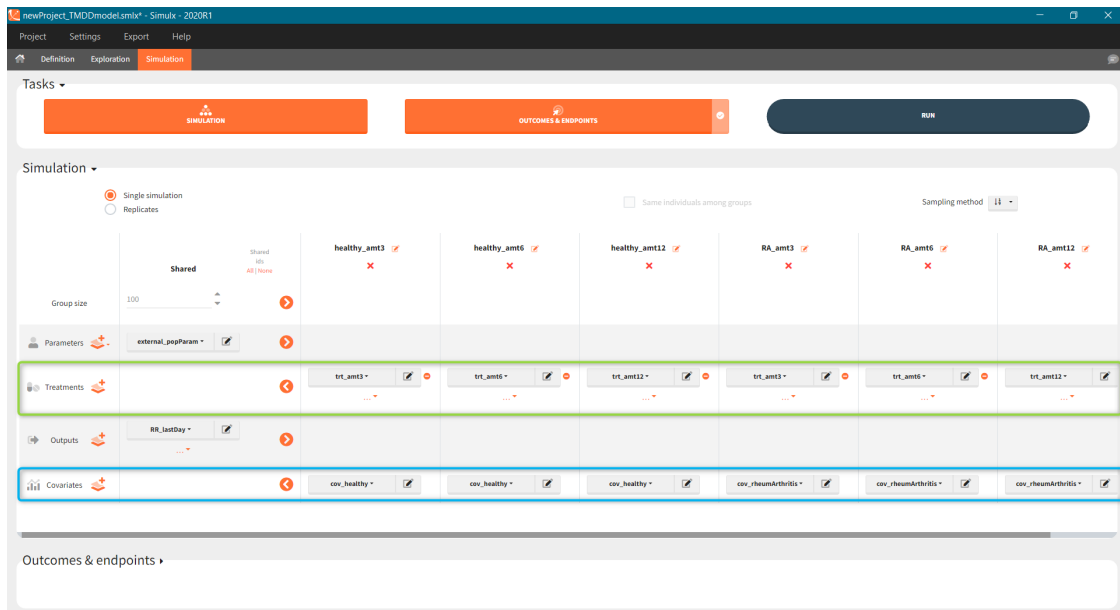
The model assumes that rheumatoid arthritis disease affects the synthesis of the free receptor (ksyn parameter). It means that for a certain dose level the efficacy target can be reached by healthy individuals (ksyn_pop = 0.789, dashed reference curve in the figure below), but not by patients with rheumatoid arthritis (ksyn_pop = 2.8, solid curve). In this exploration, the scaling of the amount 3mg/kg corresponds to typical individual with a weight 70kg (blue frame).



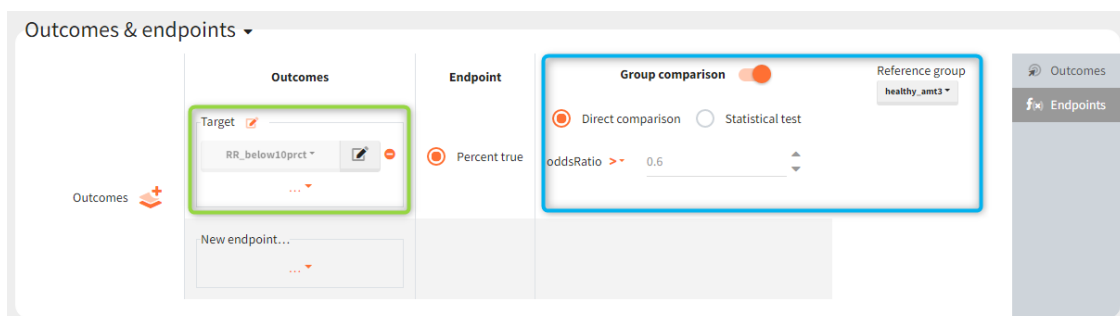
Simulation

The scenario consists of 6 groups for three dose levels and two populations, see [here](#) to learn about creating a simulation scenario.

- Treatment is group specific. Each group uses one of the three covariate dependent doses: 3mg/kg, 6mg/kg or 12mg/kg (green frame).
- Type of a population – healthy or with the rheumatoid arthritis – is in the group specific covariate (blue frame).
- Group size, population parameters and the output are the same for all groups.

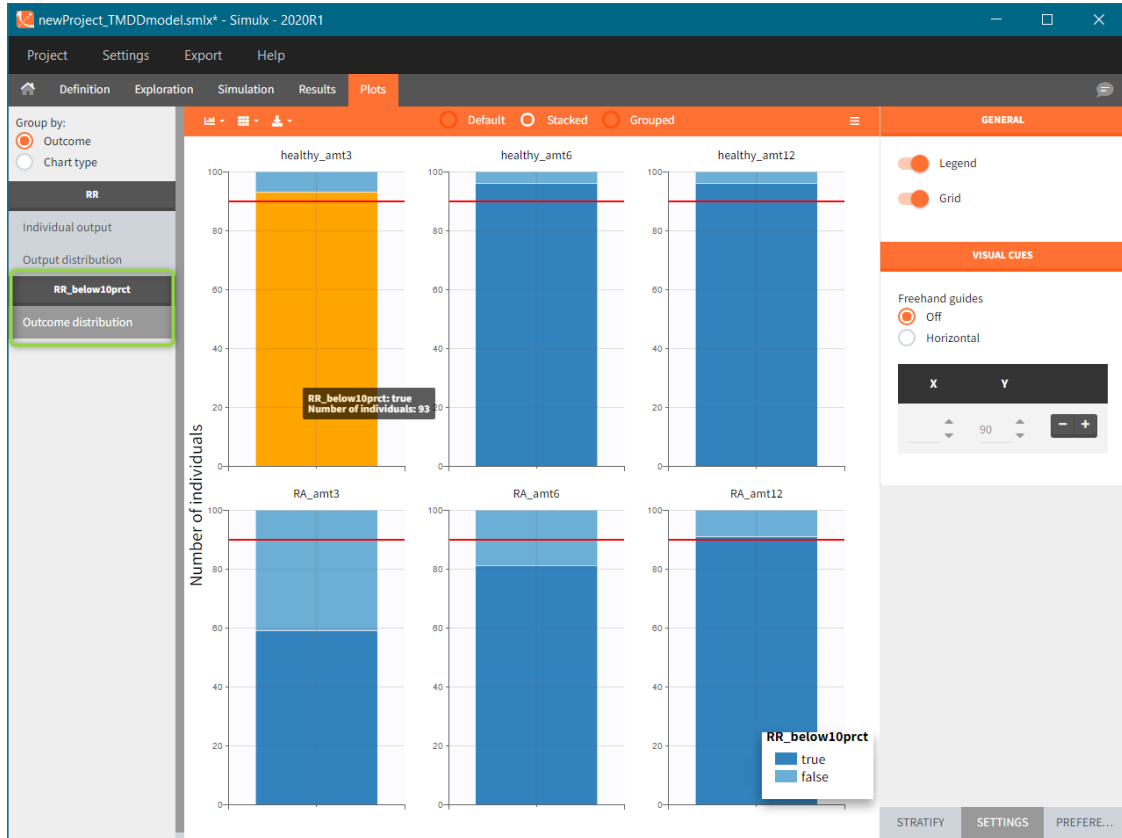


The **outcomes&endpoints** section defines post-processing of the simulation outputs for post-processing, which helps to compare the groups. The main endpoint is the percentage of individuals reaching the efficacy target in each group. It summarizes the “true” outcomes, which correspond to the situation when the ratio between the free receptor and the baseline at the end of the treatment is less than 10% (green frame). In addition, the “group comparison” (blue frame) compares the endpoints of all groups with respect to the reference group (healthy population with 3mg/kg dose level).



Results

To run the simulation and the calculation of the endpoints it is enough to select both tasks and click the button “Run”. Generation of the **results** and **plots** is automatic in two dedicated tabs. The outcome distribution shows that the efficacy of the treatment in healthy individuals stays above 90% (red line) for all dose levels. On the contrary, a group with rheumatoid arthritis achieves this level only at the highest tested dose.



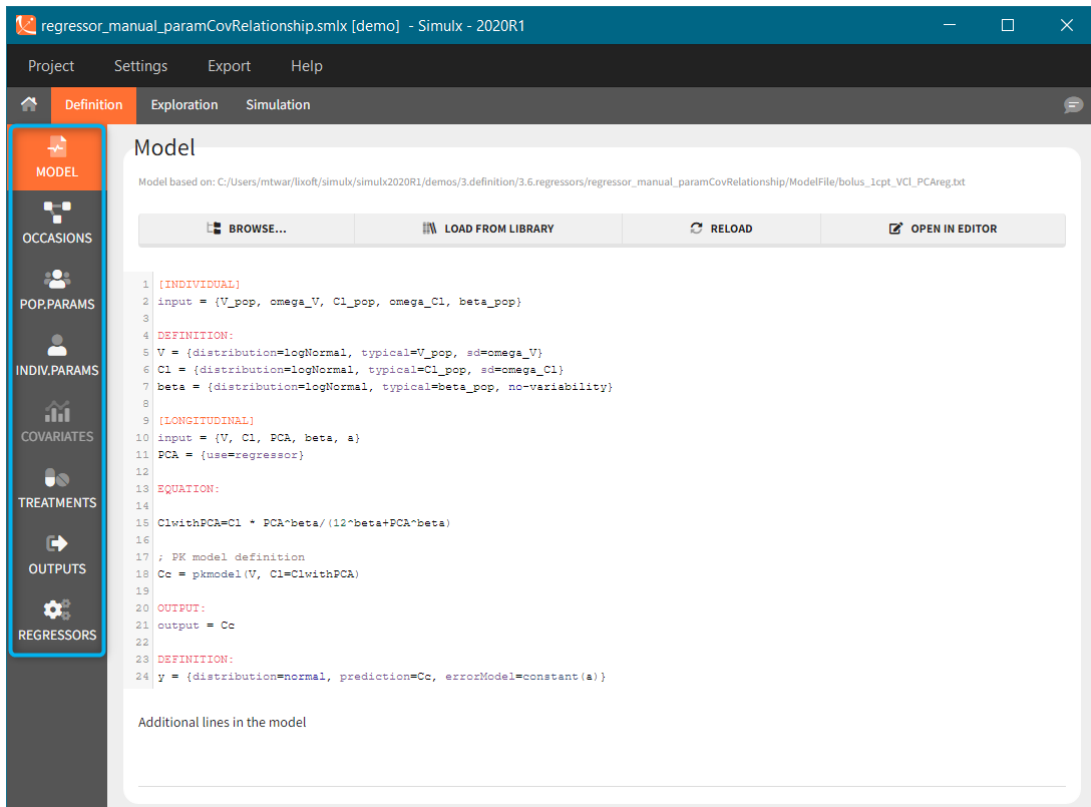
The response at the lowest dose level by healthy individuals is the reference. Group comparison considers a test group a “success” if the odds ratio of “percentage true” is larger than 0.6. Only the highest dose level satisfies this criterion for unhealthy patients.

REP	HEALTHY_AMT6		HEALTHY_AMT12		RA_AMT3		RA_AMT6		RA_AMT12	
	ODDSRATIO	SUCCESS	ODDSRATIO	SUCCESS	ODDSRATIO	SUCCESS	ODDSRATIO	SUCCESS	ODDSRATIO	SUCCESS
1	1.81	✓	1.81	✓	0.11	✗	0.32	✗	0.76	✓

2. Definition

Exploration or simulation scenarios are build from elements, such as model, parameters, treatments etc. Definition tab, as the name suggests, allows to create these simulation elements through user-friendly and flexible methods. Moreover, it displays them in dedicated sections, which gives a clear overview of the project status. Definition of the following simulation elements is available (click on any of them to see the full description):

- Model






The screenshot shows the Simulx software interface with the 'Definition' tab selected. The sidebar on the left contains navigation icons for MODEL, OCCASIONS, POP.PARAMS, INDIV.PARAMS, COVARIATES, TREATMENTS, OUTPUTS, and REGRESSORS. The main window displays the 'Model' definition for a file named 'regressor_manual_paramCovRelationship.smlx'. The model code is as follows:

```
1 [INDIVIDUAL]
2 input = {V_pop, omega_V, Cl_pop, omega_Cl, beta_pop}
3
4 DEFINITION:
5 V = {distribution=logNormal, typical=V_pop, sd=omega_V}
6 Cl = {distribution=logNormal, typical=Cl_pop, sd=omega_Cl}
7 beta = {distribution=logNormal, typical=beta_pop, no-variability}
8
9 [LONGITUDINAL]
10 input = {V, Cl, PCA, beta, a}
11 PCA = {use=regressor}
12
13 EQUATION:
14
15 ClwithPCA=Cl + PCA*beta/(12*beta+PCA*beta)
16
17 ; PK model definition
18 Cc = pkmodel(V, Cl=ClwithPCA)
19
20 OUTPUT:
21 output = Cc
22
23 DEFINITION:
24 y = {distribution=normal, prediction=Cc, errorModel=constant(a)}
```

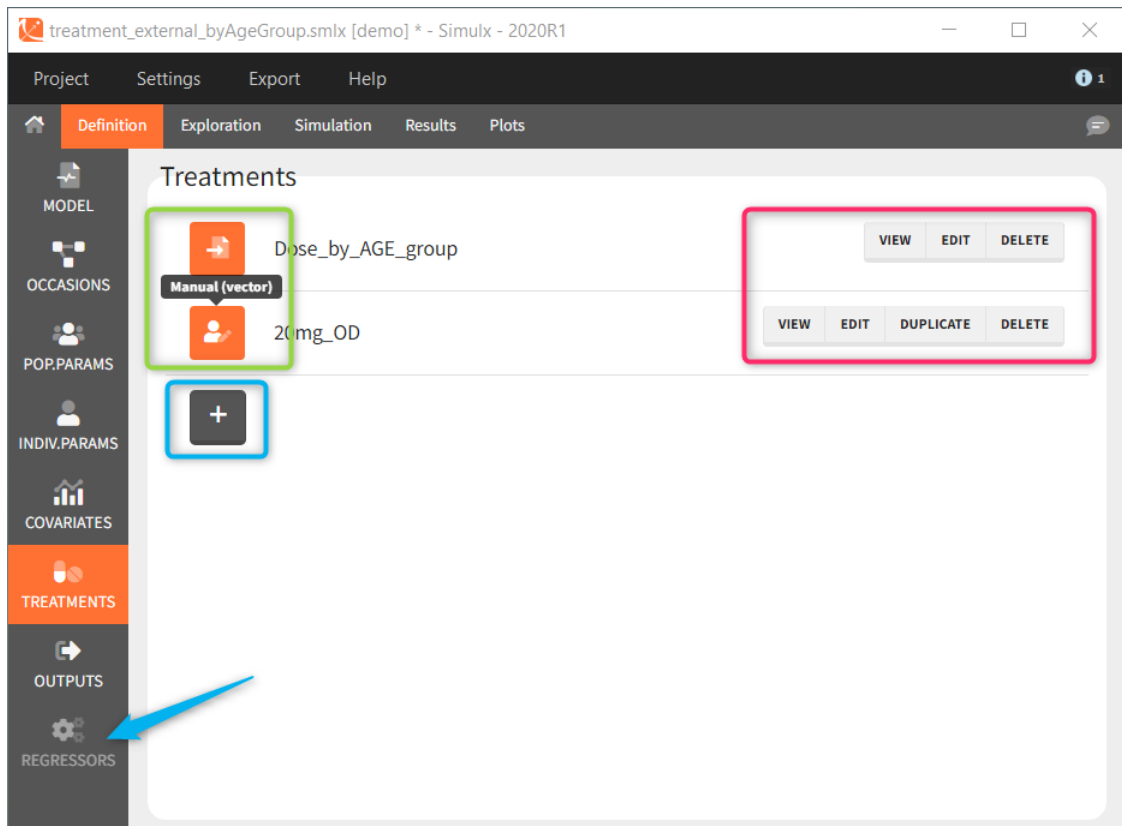
Additional lines in the model

- Occasions
- Population parameters
- Individual parameters
- Covariates
- Treatments
- Outputs
- Regressors

Each element, imported from **Monolix** or used-defined, is a table , vector  or a distribution . These icons next to the element name inform about the type when you hover on it with a cursor (green frame). Elements can be displayed, edited, duplicated (besides tables) or deleted (red frame). To create a new element, go to a dedicated section and click the “plus” button (blue frame).

Model is a mandatory element. It has to be loaded as first, because only elements present in the model can be defined. Otherwise, the section is greyed out and unavailable, for instance “regressors” (blue arrow at the bottom). Of course, more than one element of each category is possible.

After importing a project from Monolix, all elements are automatically created using Monolix model, results and dataset, see the list of elements [here](#). In a new project written from scratch, Simulx generates default elements based on the loaded model to guide building new ones.



2.1. Model

A single model must be defined in each Simulx project. This model is used both in [Exploration](#) and [Simulation](#). Loading or reloading a model deletes all definition elements to avoid incompatibilities between previously defined elements and the new model.

- **Model sections:**
 - [LONGITUDINAL]
 - [INDIVIDUAL]
 - [COVARIATE]
- [Loading or modifying a model](#)
- [Model imported from Monolix](#)
- [Additional lines](#)

Model sections

The model includes different sections:

- **[LONGITUDINAL]** (mandatory)

This section contains the structural model. All variables of the model can be defined as outputs of the simulation.

Details on the syntax to define the structural model is [here](#).

This section can include the definition of random variables in a block DEFINITION;, such as a random event or a variable with residual error.

- **[INDIVIDUAL]** (optional)

If the model considers the inter-individual variability, then definitions of models for [individual parameters](#) are in [LONGITUDINAL]. The inputs of [INDIVIDUAL] are [population parameters](#) and [covariates](#) involved in covariate effects. If the project is imported from Monolix, this section is created automatically. However, when building a project from scratch, the [INDIVIDUAL] section can be generated automatically starting with version 2024 by clicking "+ ADD INDIVIDUAL". This will add an [INDIVIDUAL] section with all parameters in the model with a lognormal distribution and random effects.

Example: Parameters Tlag, ka, V and Cl are described with lognormal distributions, effects of covariates SEX and logtAGE are included on Tlag and Cl, respectively, and there is a correlation between eta_V and eta_Cl:

```
[INDIVIDUAL]
input = {Tlag_pop, ka_pop, omega_ka, V_pop, omega_V, Cl_pop, omega_Cl,
corr_V_Cl, logtAGE, beta_Cl_logtAGE, SEX, beta_Tlag_SEX_M}
```

```
SEX = {type=categorical, categories={F, M}}
```

DEFINITION:

```
Tlag = {distribution=logNormal, typical=Tlag_pop, covariate=SEX, c
```

```
ka = {distribution=logNormal, typical=ka_pop, sd=omega_ka}
```

```
V = {distribution=logNormal, typical=V_pop, sd=omega_V}
```

```
Cl = {distribution=logNormal, typical=Cl_pop, covariate=logtAGE, c
```

```
F = {distribution=logitNormal, typical=F_pop, sd=omega_F}
```

```
correlation = {level=id, r(V, Cl)=corr_V_Cl}
```

More details on the syntax in this section is [here](#).

- **[COVARIATE]** (optional)

If covariates are used in the model of individual parameters, then they should be defined in a block [COVARIATE]. This block contains the list of inputs (covariates used in the model) and the definition of categories for categorical covariates. It can also include transformations of the covariates, in a section EQUATION: for continuous covariates and in a section DEFINITION: for categorical covariates. All the covariates defined in this block (transformed or not) can be then used in the block [INDIVIDUAL].

Example: There are 3 covariates AGE, RACE and SEX. The covariate AGE is transformed into logtAGE, RACE is transformed into tRACE.

```

[COVARIATE]
input = {AGE, RACE, SEX}

RACE = {type=categorical, categories={Asian, Black, White}}
SEX = {type=categorical, categories={F, M}}

EQUATION:
logtAGE = log(AGE/35)

DEFINITION:
tRACE =
{
transform = RACE,
categories = {
G_Asian = Asian,
G_Black_White = {Black, White} },
reference = G_Black_White
}

```

More details on the syntax in this section is [here](#).

Note that in Simulx it is not possible to define covariates as distributions within the block [COVARIATE] with a section DEFINITION:. However, the covariate distributions can be defined in the tab Definition in the interface of Simulx.

Loading or modifying a model

Several buttons are available on the top of the page to load or modify a model:

- Browse: to load a new model file by browsing if from the computer.
- Load from library: to load a model from the built-in libraries. Note that all models from the libraries contain only structural models ([LONGITUDINAL] block) with no individual models for the parameters or covariates.
- Reload: to reload the same model after modifying it using the built-in editor or with any text editor.
- Open in editor: to opens the current model in the built-in editor.

Loading or updating a model with the buttons “Browse”, “Load from library” or “Reload” deletes all definition elements to avoid incompatibilities with previously defined elements and the new model.





Model imported from Monolix

When **importing a model from Monolix**, the model that appears in the interface of Monolix comes from two sources:

- The structural model ([LONGITUDINAL] block) used in the Monolix project, copied in a new file stored in the result folder of the Simulx project, in a subfolder names ModelFile/.
- The [INDIVIDUAL] and (if there are covariates in the model) [COVARIATE] blocks, derived from the statistical model set up in Monolix, saved directly in the .smlx file.

Warning: When opening a model in the editor with “Open in editor”, only the new structural model file is opened for modifications. If this model is changed and reloaded, it will replace the two model sources mentioned above. It means that the [INDIVIDUAL] and [COVARIATE] blocks are removed. To keep these blocks in the modified model, they have to be pasted in the file together with the [LONGITUDINAL] block.

Additional lines

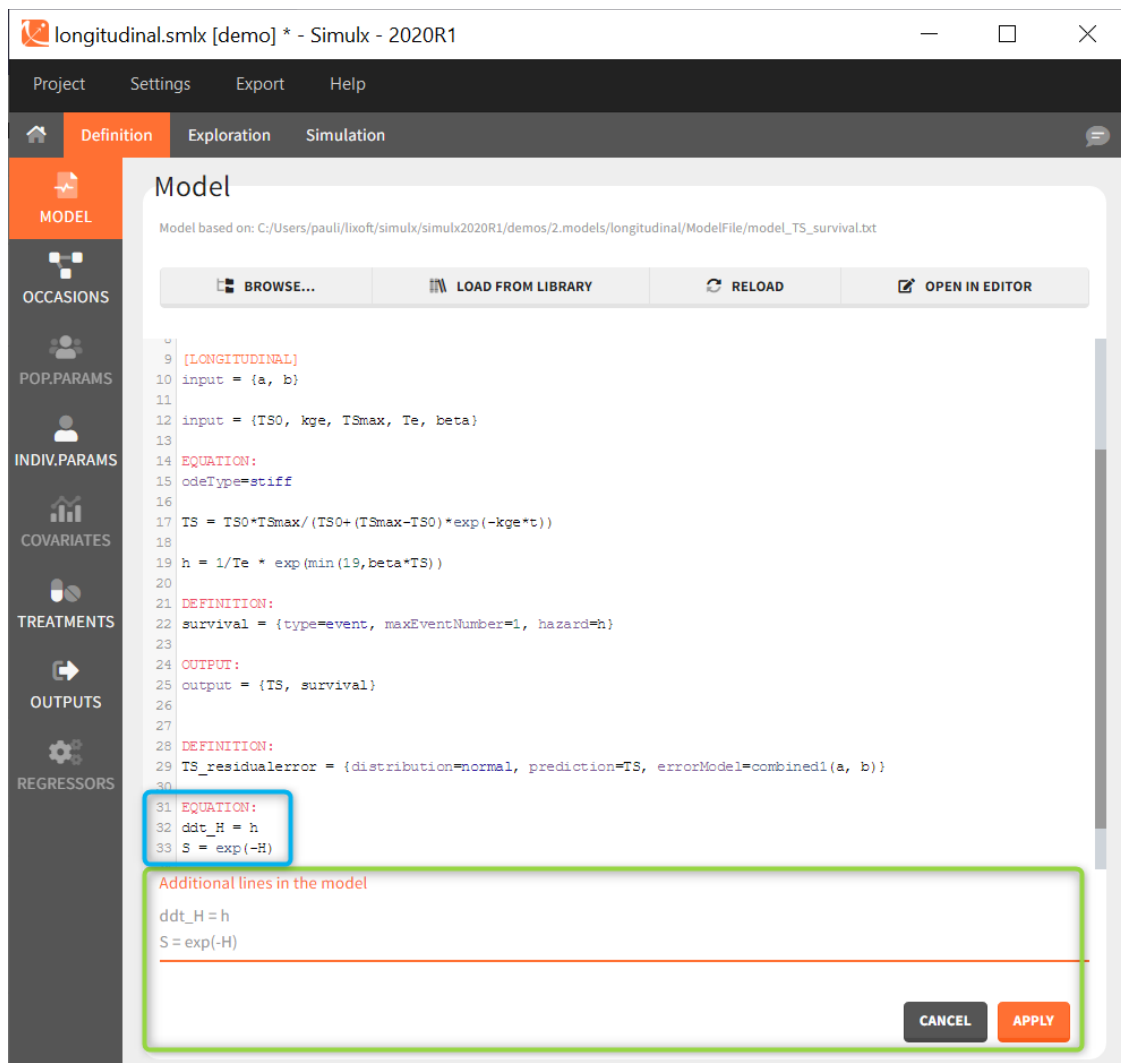
Below a model definition, there is a field to write additional lines, which are included “on the fly” in the model. Note that in version 2024 and later, this is accessed by clicking the button “+ ADD LINES”.

This is convenient for example to use variables for simulations that are not defined in the Mlxtran model code, such as the AUC, a change from baseline, or the survival function (see example below). The model file itself is not modified, the additional lines are saved in the .smlx file.

By default, the additional lines are added in a new section EQUATION: below the structural model. In order to add additional lines in the PK block, it is necessary to add "PK:" at the beginning of the additional lines. This is convenient to add a new route of administration, using the iv(), oral() or depot() macro for instance. If this new administration depends on a parameter (e.g the absorption rate ka), the parameter needs to be fixed in the additional lines. It cannot be added to the input parameter list. It is not possible to add additional lines in the [INDIVIDUAL] or [COVARIATE] blocks.

This function is available also with lixoftConnectors functions: setAddLines() and getAddLines(), see [here](#) for more details.

Warning: When using "import from Monolix", "additional lines" fields can (sometimes) be unavailable. In this case, open the Monolix project in Monolix and use "Export to Simulx". This problem has been fixed in the 2021R1 version.

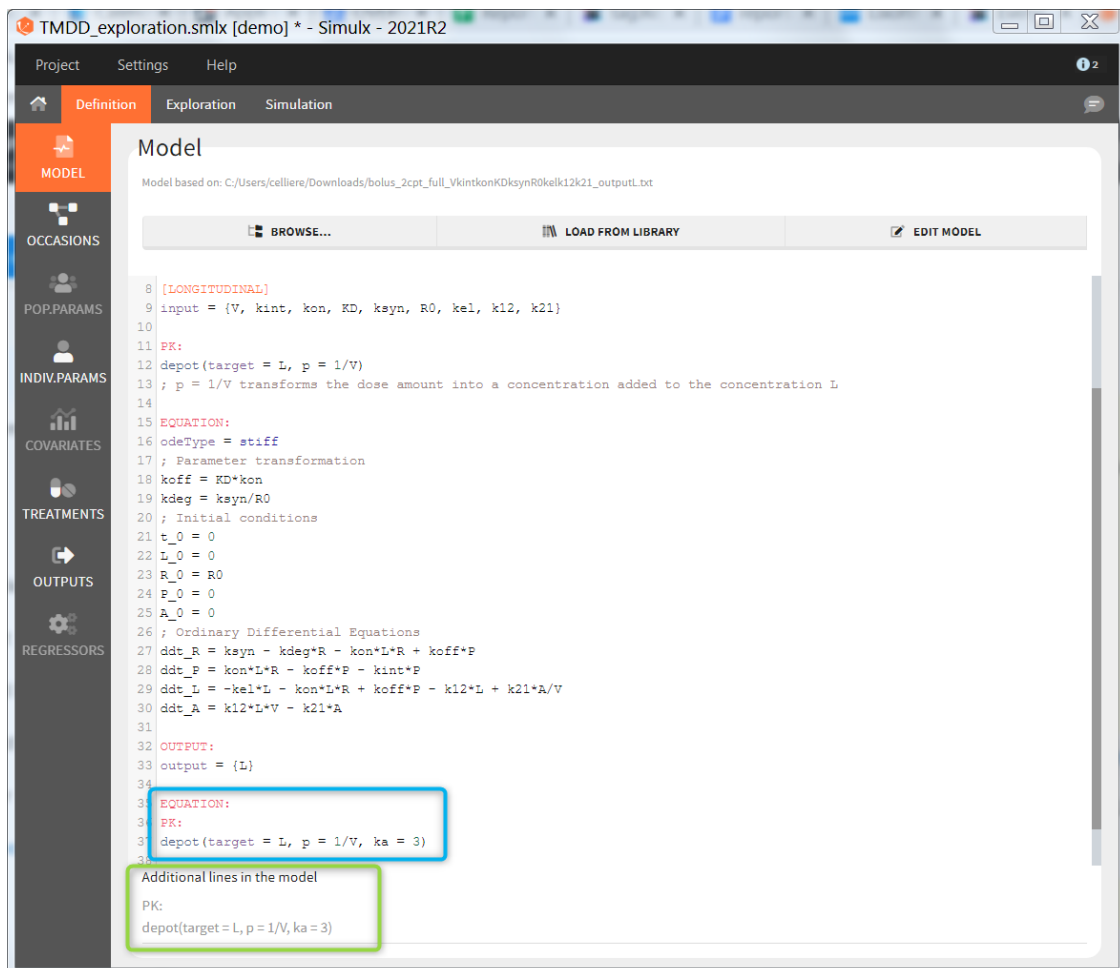


The screenshot shows the Simulx software interface for editing a model. The main window displays the model code in a text area, with the following content:

```
9 [LONGITUDINAL]
10 input = {a, b}
11
12 input = {TS0, kge, TSmax, Te, beta}
13
14 EQUATION:
15 odeType=stiff
16
17 TS = TS0*TSmax / (TS0 + (TSmax-TS0)*exp(-kge*t))
18
19 h = 1/Te * exp(min(19, beta*TS))
20
21 DEFINITION:
22 survival = {type=event, maxEventNumber=1, hazard=h}
23
24 OUTPUT:
25 output = {TS, survival}
26
27
28 DEFINITION:
29 TS_residualerror = {distribution=normal, prediction=TS, errorModel=combined1(a, b)}
30
31 EQUATION:
32 ddt_H = h
33 S = exp(-H)
```

A dialog box titled "Additional lines in the model" is open, showing the same code as the main window. The "APPLY" button is highlighted in orange.

In the example below, a oral absorption route is added to the demo TMDD_exploration.smlx, which originally had only iv:

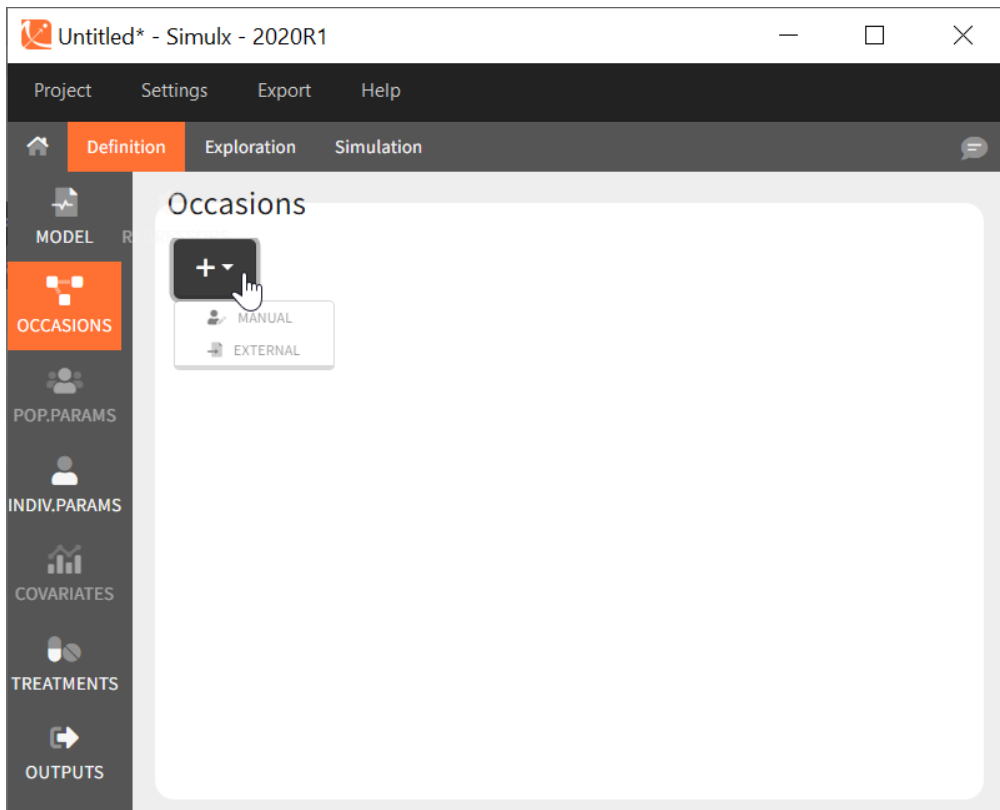


2.2. Occasions

Occasions definition. When the model to simulate contains **inter-occasion variability** (ie variability between different periods of measurements within the same individual), you should define a single element Occasions. Like for the model, and contrary to the other simulation elements, it is not possible to define several occasion elements to choose from for the simulation.

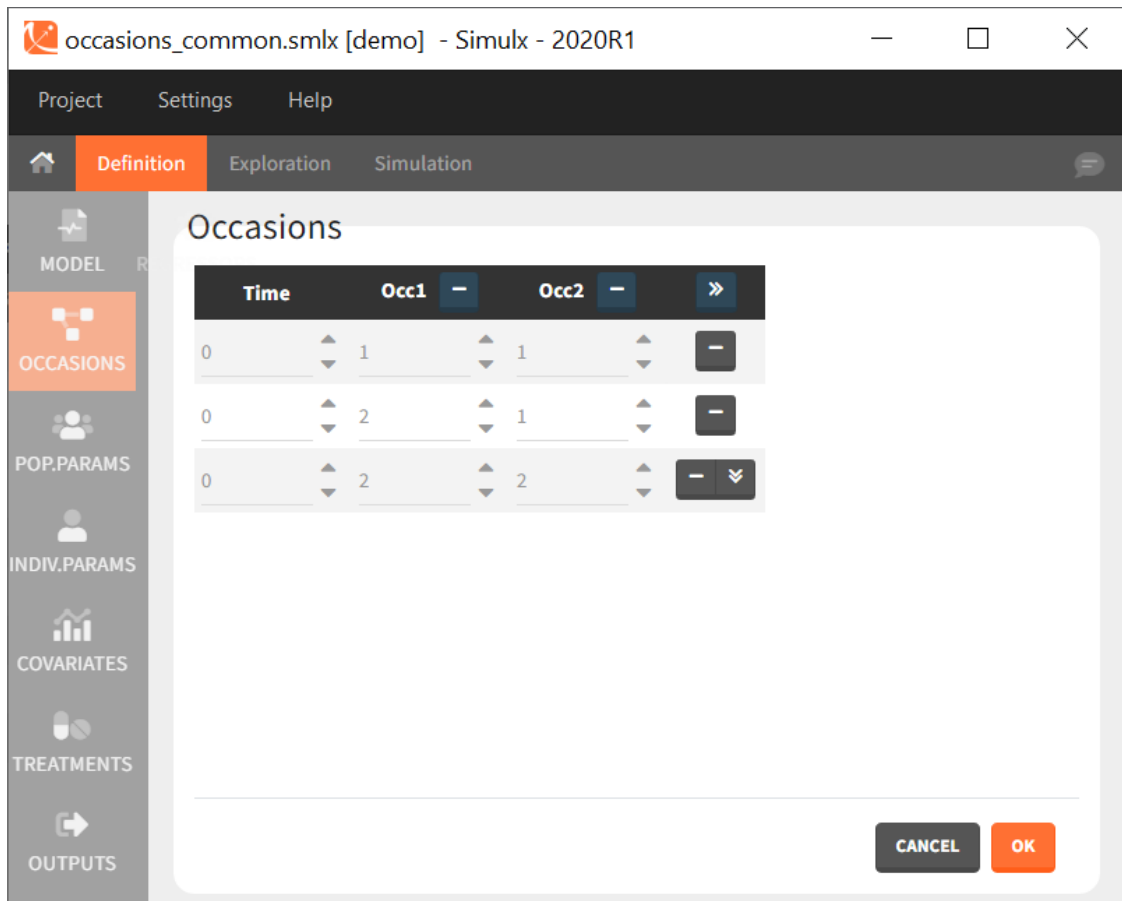
Demo projects: 3.7. occasions

All other defined elements must be compatible with the structure of occasions.



Common occasions for all subjects

In the interface, it is possible to define a common structure of occasions applied to all simulated subjects. This structure can contain one or several levels of occasions, and one or several occasions per subject.



After defining such common occasion structure, other common elements (parameters, covariates, treatments, outputs and regressors of type “manual” or “regular”) can be defined occasion-wise thanks to a dedicated toggle “occasion-wise values”, as on this example screenshot for covariates:

New manual covariates

Name
covManual1

Occasion-wise values

1#1

Covariate	Value
AGE	45
FOOD	Fed
SEX	F

2#1

Covariate	Value
AGE	45
FOOD	Fasted
SEX	F

2#2

Covariate	Value
AGE	45

CANCEL OK

Moreover, elements defined as external tables may contain one or several columns to assign different values to different subject-occasions.

Subject-specific occasions

To define subject-specific occasions use an external file containing a table with columns ID, time and one or several columns for the occasions. The header names for these columns are free and are used by Simulx. The external file separator can be tab, comma or semicolon. The possible file extensions are .csv or .txt.

Below is an example of the occasions element defined after loading the external table (on the left).

ID	OCC	TIME	
1	1	0	
1	2	24	
1	3	48	
2	1	0	
2	2	48	
3	1	0	
3	2	24	
4	1	0	
4	2	24	
4	3	48	
4	4	72	
5	1	0	
6	1	0	
6	2	48	
6	3	72	

Load external occasions

ID	time	OCC
1	0	1
1	24	2
1	48	3
2	0	1
2	48	2
3	0	1
3	24	2
4	0	1
4	24	2
4	48	3
4	72	4
5	0	1
6	0	1
6	48	2
6	72	3

After defining such subject-specific occasion structure, other elements (parameters, covariates, treatments, outputs and regressors) must be common over subjects (type “manual”, “regular” or “distribution”), or can be defined with occasion-wise values as external tables only, with the same occasion structure.

Occasions imported from Monolix

Upon import of a Monolix run, Simulx creates an occasion element with the individual structure of occasions from the Monolix data set.

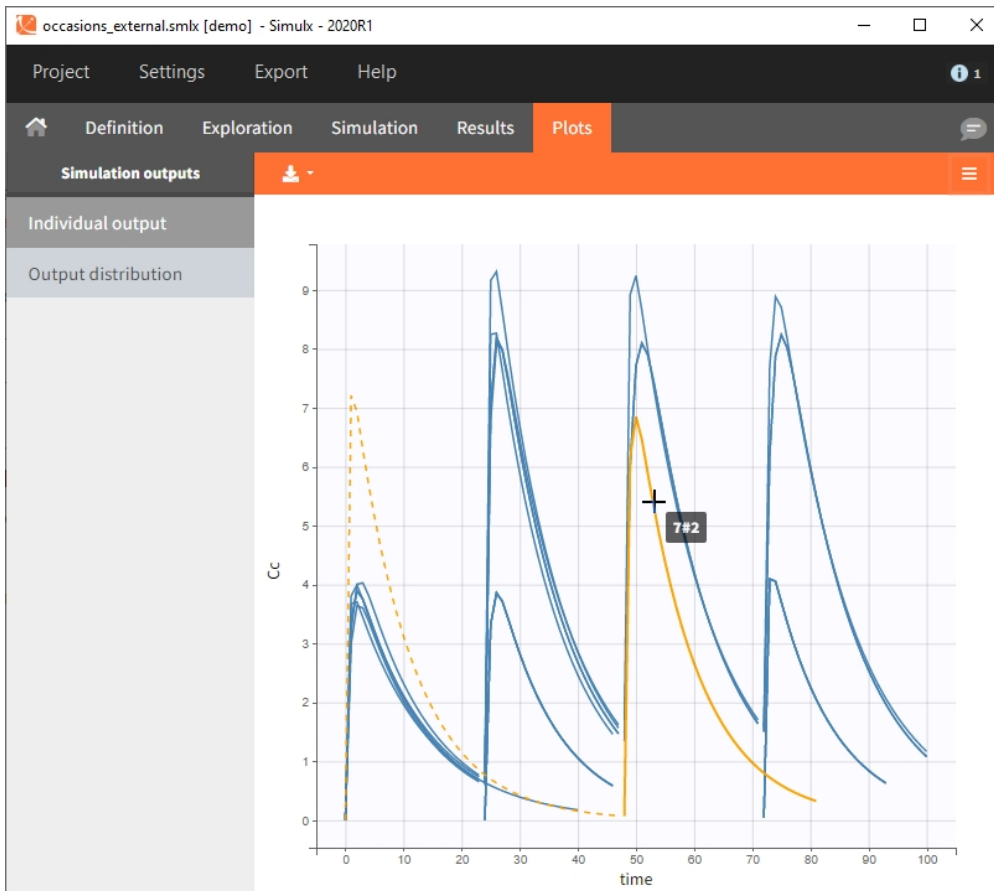
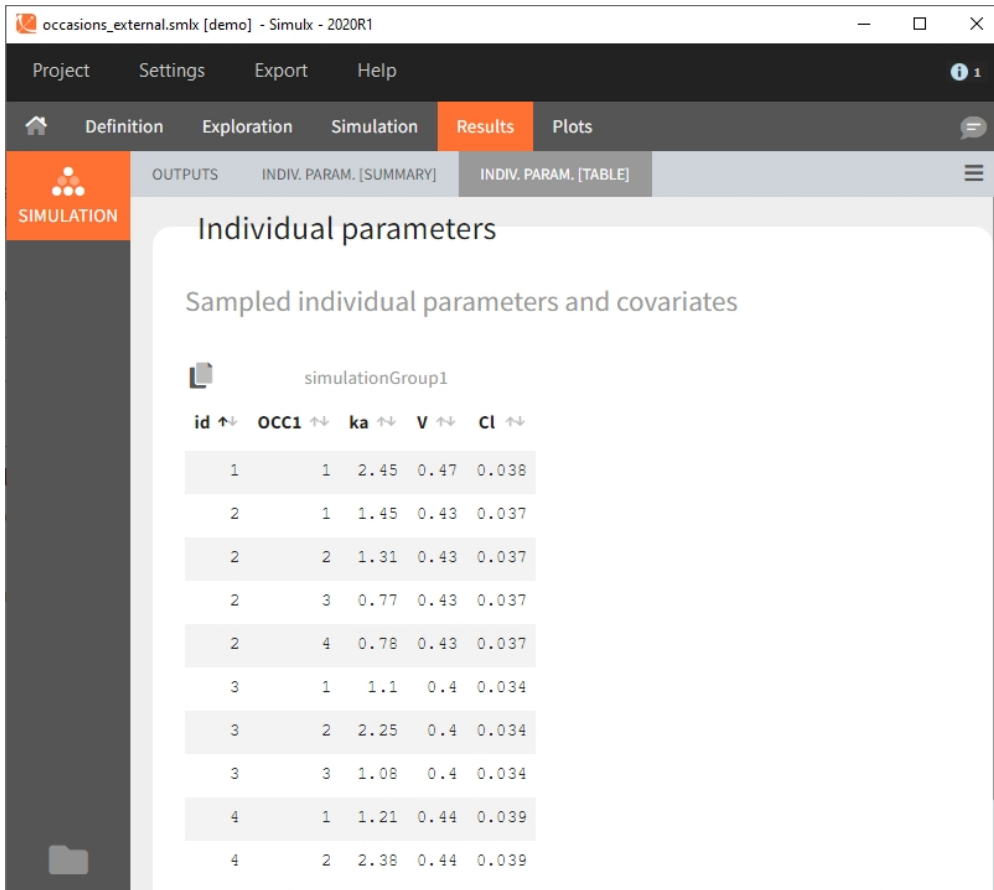
- **mlx_subjects**: a table containing the occasions and their starting times for each individual. The table has at least 3 columns: id, time, occ. Additional columns may be present in case of several levels of occasions.

Rules for simulating occasions

- Occasions are not simulated in Exploration: only the first occasion from the selected subject is simulated.
- If occasions are not defined, inter-occasion variability is not taken into account for the simulation of individual parameters.
- From the 2023 version on, the sampling of individual parameters takes into account the inter-occasion variability if one or several occasions are defined. In previous versions (2020 and 2021), if the structure of occasions contains a single occasion for some subject, then the simulation of individual parameters for this subject does not take into account inter-occasion variability.
- When an occasion element has been defined, in case of several replicates, the sampling method has to be 'Keep Order'.

Occasions in outcomes, results and plots

Occasions are indicated as additional columns in the table of individual parameters. In the plots of individual outputs, occasions are highlighted as dashed yellow lines on hover of one the occasions from the same subject. Furthermore, all plots of results can be stratified by occasions.



When occasions are defined, the definition of **outcomes** can be made by post-processing the simulation outputs by subject (one outcome value per individual) or by subject-occasion (one outcome value for each occasion of each individual). When the post-processing by subject-occasion is selected, the occasion column(s) appear(s) also in the corresponding outcome table.

outcome1

Name

outcome1

- calculate for each individual
 calculate for each occasion of each individual

Output

outputExternal1 ▾

relative to baseline (first value of output)

Output processing: average value per id ▾

Apply threshold

CANCEL

OK

2.3. Population parameters

Definition of population parameters as simulation elements allows to simulate individual parameters from **probability distributions**. In the mlxtran **model** they are:

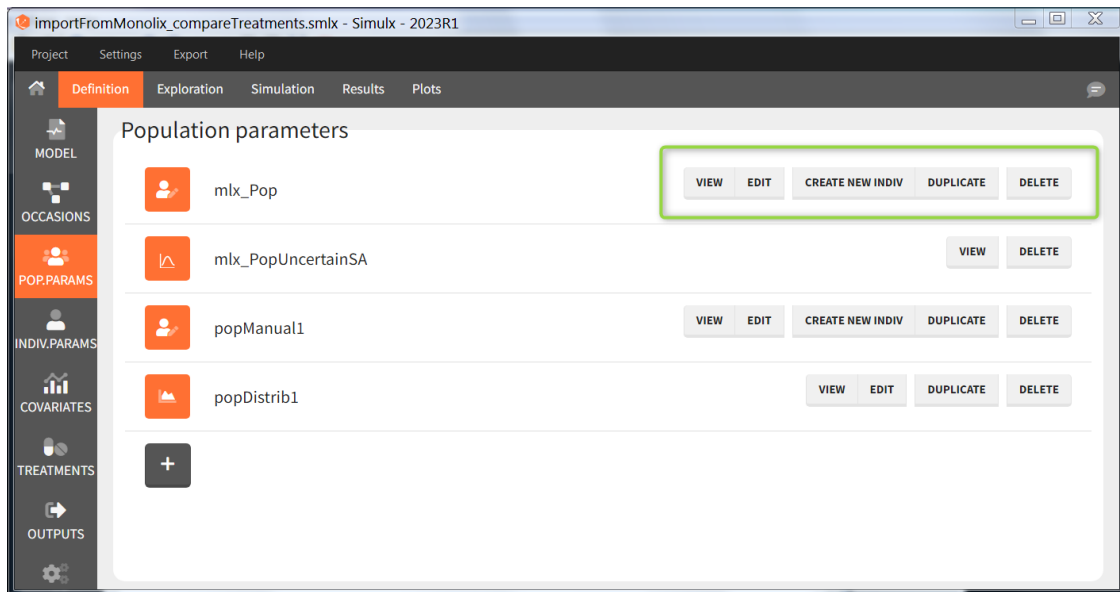
- [INDIVIDUAL] block input parameters that are neither inputs nor outputs of the block [COVARIATE].
- [LONGITUDINAL] block input parameters that are neither individual parameters nor regressors.

POP.PARAM section in the “Definition” tab is available only if the list of population parameters from the model is not empty.

Demo projects: 3.3. population parameters

Overview

In the “Population parameters” tab, elements created automatically after a “new project” or an “import from Monolix/PKanalix” and elements created by the user are shown.



The buttons on the right allow to do the following actions:

- VIEW: display the content of the element (the value of the population parameters)
- EDIT: modify the content of the element. *Note:* the elements “mlx_PopUncertainSA”, “mlx_PopUncertainLin”, “mlx_TypicalUncertainSA”, and “mlx_TypicalUncertainLin” represent multidimensional distributions and cannot be edited.
- DUPLICATE: open a pop-up window for a new element with the same content as the previous element. *Note:* the elements “mlx_PopUncertainSA”, “mlx_PopUncertainLin”, “mlx_TypicalUncertainSA”, and “mlx_TypicalUncertainLin” cannot be duplicated. Elements based on external files can also not be duplicated.
- DELETE: delete the element. When the deleted element is based on an external file located in <result folder>/ExternalFiles, the file itself is also deleted upon save.
- CREATE NEW INDIV: creates a new element “individual parameters” from the population parameters fixed effects (e.g $V = V_{pop}$). This option is available for population elements of type “manual” only.

New population parameters element

After loading a model, Simulx generates automatically a default population parameter element. It is a table with parameters names from the model and all entries equal to one. It helps to create a new element by just modifying the values. Clicking the button “plus” adds a new population parameter, which may be one of the three types.

- **Manual:** It is a vector and has one value for each parameter.

new manual population parameters

MANUAL DISTRIBUTION EXTERNAL

Name
manual_parameter

Parameter	Value
Tlag _{pop}	0.8
ω _{Tlag}	0.2
ka _{pop}	1
ω _{ka}	0.1
V _{pop}	8
ω _V	0.3
Cl _{pop}	2.5
ω _{Cl}	0

CANCEL OK

- Distribution:** Parameters are a distribution law – normal, lognormal (default), logitnormal, uniform – with a typical value and a standard deviation. If the distribution is lognormal or logitnormal, sd is the standard deviation of the distribution in the Gaussian space. The typical value is the median of the population parameter distribution. In the case of a lognormal distribution, in order to get the sd s_G of the distribution in the Gaussian space given a typical value of the lognormally distributed covariate μ , or given its mean m , and given its sd s , you can use the following formulae:

$$s_G = \sqrt{\ln\left(1 + \left(\frac{s}{m}\right)^2\right)} = \sqrt{\ln\left(1 + \sqrt{1 + 4\left(\frac{s}{\mu}\right)^2}\right) - \ln(2)}$$

Both formulae are equivalent if $\frac{s}{\mu} \ll 1$ (in that case $\mu \approx m$).

New population parameters from distribution

MANUAL DISTRIBUTION EXTERNAL

Name
distribution_parameter

Parameter	Distribution	Typical	sd
Tlag _{pop}	LOGNORMAL	0.8	0.1
ka _{pop}	LOGNORMAL	1	0.2
ω _{ka}	LOGNORMAL	0.1	0.1

Parameter configuration window showing three rows of LOGNORMAL distribution parameters:

- V_{pop} : 8, 0.1
- ω_V : 0.3, 0.1
- Cl_{pop} : 2.5, 0.01

Buttons: CANCEL, OK

- **External:** It is a file with a table that has each population parameter in a separate column.

New population parameters from external file

MANUAL DISTRIBUTION EXTERNAL

Name: external_parameter

File: ExternalFiles/simulated_popParam.csv SELECT

CANCEL OK

pop	Tlag_pop	ka_pop	V_pop	beta_V_logWt	Cl_pop	a	b
1	0.51	0.84	8.50	0.26	0.13	0.00	0.31
2	0.85	1.77	8.97	0.44	0.12	0.00	0.31
3	0.63	1.31	8.65	0.52	0.12	0.00	0.33

The external file can have only one column header different from the population parameters names. It indicates replicates (eg. from bootstrap or simpopmlx). All individual parameters of the same replicate come from the same population distribution.

Population parameters imported from Monolix

Demo projects: 1.overview/importFromMonolixsmlx

Importing a Monolix project generates automatically a population parameter element with values from the Monolix result folder.

- **mlx_Pop** contains population parameters estimated by Monolix if the **POP.PARAM** task results are available.
- **mlx_PopInit** contains initial estimates of the population parameters if the mlx_pop does not exist.
- **mlx_Typical** (NEW since 2023 version) is the same as mlx_Pop but with all standard deviations of random effects (omega parameters) set to 0. It enables to simulate a typical individual in terms of parameters (no unexplained variability), and still include non-typical covariates in the simulation. The variability in the sampled individual parameters will come only from sampling different covariate values. To see how this element can be used in practice, check the demo project 1.overview/

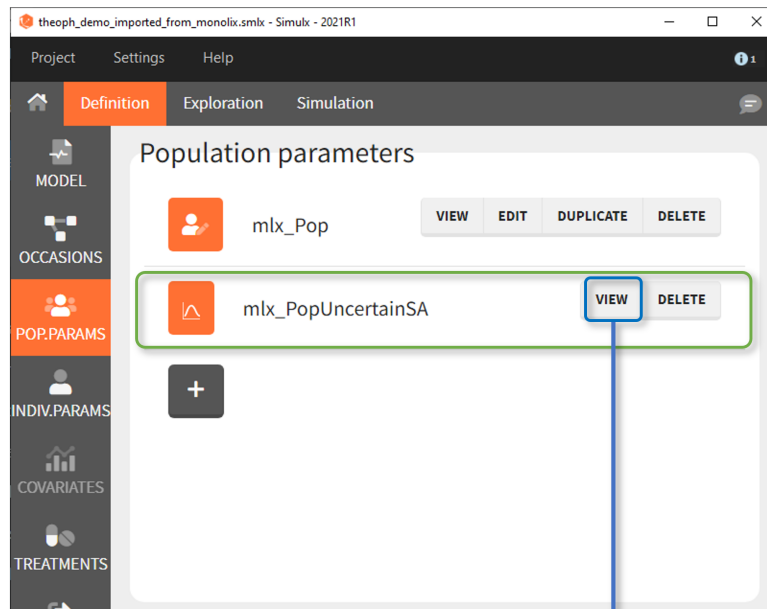
importFromMonolix_CovEffectOnTypical.smlx.

mlx_Pop, mlx_PopInit and mlx_Typical are vectors (single set of population parameters). If replicates are used in the Simulation tab, with these elements, the same set of population parameters will be used for all replicates.

- **mlx_PopUncertainSA** (resp. **mlx_PopUncertainLin**) enables to sample population parameters using the variance-covariance matrix of the estimates computed by Monolix if the **Standard Error task** (Estimation of the Fisher Information matrix) was performed by stochastic approximation (resp. by linearization). To sample several population parameter sets, this element needs to be used with replicates. In the interface, the element is displayed with a reminder of the estimated values and RSEs next to the **variance-covariance matrix** which is used to generate new samples.

The displayed variance-covariance matrix and the sampling is done in the gaussian space (i.e we sample $\{\log(V_{pop}), \log(CI_{pop}), \omega_V, \omega_{CI}, b\}$ from a multivariate normal distribution, if V and CI have a lognormal distribution). The sampled values are then converted to the non-gaussian space. For omega and error parameters, negative samples values are rejected, they are thus sampled from truncated normal distributions.

This element cannot be modified nor duplicated.



mlx_PopUncertainSA [read-only]

	Estimated value	R.S.E. (%)	Uncertainty variance-covariance matrix in Gaussian space							
ka_pop	1.53264	20.3	0.0412507							
V_pop	0.455504	4.48	0.00103617	0.00201065						
Cl_pop	0.0401724	8.40	-0.000608504	-0.000385393	0.00705529					
omega_Cl	0.270623	24.3	-0.000139782	-0.0000583648	-0.00001328	0.00431548				
omega_V	0.126281	27.8	0.000301576	0.000107252	-0.0000817639	-0.000245008	0.00123293			
omega_ka	0.671055	24.0	0.00151641	-0.000149856	0.00018779	0.0000541658	0.0000148019	0.0259983		
a	0.43328	34.8	0.00165727	-0.000268677	0.000255753	0.000276002	0.000407877	0.000369118	0.0227485	
b	0.0542595	55.1	-0.000384453	0.0000472563	-0.0000278134	-0.0000572644	-0.000101725	-0.000136792	-0.00422552	0.000992541

CLOSE

- mlx_TypicalUncertainSA** (resp. **mlx_PopUncertainLin**, NEW since 2023 version) is another population parameter element with the variance-covariance matrix of population parameter values estimated in Monolix. The only difference to the previous **mlx_PopUncertainSA/Lin** is that it has standard deviations of random effects (omega parameters) set to 0. It enables to simulate a typical individual with different typical parameter values for each replicate, such that the uncertainty of parameters is propagated to the predictions. To sample several parameter sets,

this element needs to be used with **replicates**. To see how this element can be used in practice, check the demo project `1.overview/importFromMonolix_UncertaintyOnTypical.smlx`.

In the following video we apply uncertainty on fixed effects only (with version 2021). With the 2023 version, the steps performed outside of the interface in this video can all be replaced by simply selecting the new element `mlx_TypicalUncertain`.

2.4. Individual parameters

In the **mlxtran** model, the following parameters are recognized as individual parameters:

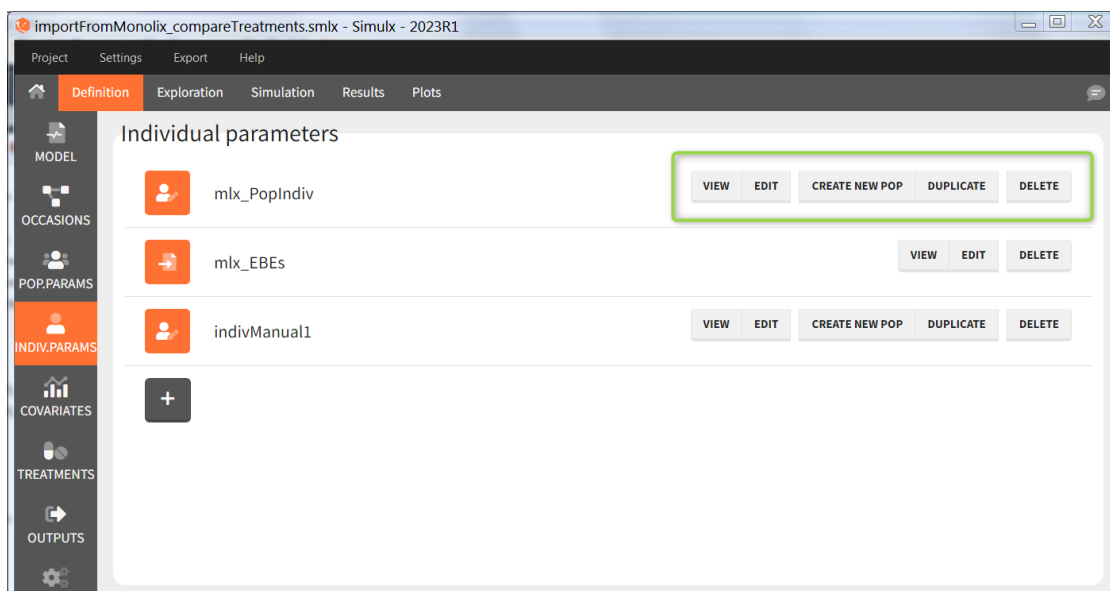
- Only the [LONGITUDINAL] block is present: all parameters of the input list, except those defined as regressors.
- Both the [LONGITUDINAL] and [INDIVIDUAL] blocks are present: parameters defined in the DEFINITION section of the [INDIVIDUAL] block.

Individual parameters can be used in the tab "**Exploration**" and "**Simulation**". An individual parameter definition is either a vector, which has one value per parameter, or a table, where each line correspond to a value of a different individual.

Demo projects: 3.4. individual parameters

Overview

In the "Individual parameters" tab, elements created automatically after a "new project" or an "import from Monolix/PKanalix" and elements created by the user are shown.



The buttons on the right allow to do the following actions:

- VIEW: display the content of the element (the value of the individual parameters)
- EDIT: modify the content of the element.
- DUPLICATE: open a pop-up window for a new element with the same content as the previous element. *Note:* elements based on external files cannot be duplicated.
- DELETE: delete the element. When the deleted element is based on an external file located in <result folder>/ExternalFiles, the file itself is also deleted upon save.
- CREATE NEW POP: creates a new element “population parameters” from the individual parameters (e.g $V_{pop} = V$). The betas (covariate effects), omegas (standard deviation of the random effects), correlation and error parameters are set to zero. This option is available for individual elements of type “manual” only, which contain a single set of individual parameters.

New individual parameters elements

After loading a model, Simulx generates automatically a default individual parameter element called “IndivParameters” with all values set to 1. This element can be modified (button EDIT). New elements can be created with the button “+”. Two different types of individual parameter elements are available:

- **Manual:** One value per parameter to type in. The required parameters are automatically listed.

New manual individual parameters

MANUAL EXTERNAL

Name
indivManual1

Parameter	Value
ka	1.4
V	3.6
Cl	0.25

CANCEL OK

- **External:** An external text file with either:

- A single row and one column per parameter. The headers must correspond to the parameter names.

ka	V	Cl
1.62	7.67	0.11

- Several rows, a first column with header "id", occasion columns (optional) and one column per parameter. The headers must correspond to the parameter names, and to the occasion names if applicable.

id	ka	V	Cl
1	1.62	7.67	0.11
2	0.87	8.34	0.13
3	0.96	8.47	0.11
4	1.51	5.09	0.072
5	1.83	11.52	0.18

The external file can be tab, comma or semicolon separated. Available file extension are: .csv or .txt.

New individual parameters from external file

MANUAL EXTERNAL

Name
indivExternal1

File
...S/ /external_indivparam.csv SELECT

CANCEL OK

Individual parameters elements imported from Monolix

Upon **import of a Monolix** run, several individual parameters elements are created automatically:

- **mlx_IndivInit:** [when POP.PARAM task has not run] a vector of individual parameters corresponding to the initial values of the population parameters in Monolix, i.e with

covariate betas and random effects set to zero. For instance, $V = V_{pop,ini}$

- **mlx_PopIndiv**: [when POP.PARAM task has run] a vector of individual parameters corresponding to the population parameters estimated by Monolix, i.e with covariate betas and random effects set to zero. For instance, $V = V_{pop,est}$
- **mlx_PopIndivCov**: [when POP.PARAM task has run and covariates are defined in the Monolix data set] a table of individual parameters corresponding to the population parameters and the impact of the covariates, but random effects set to zero. For instance, $V = V_{pop} \times \left(\frac{WT}{70}\right)^{\beta_{v,WT}}$
- **mlx_EBEs**: [when EBEs task has run] a table of individual parameters corresponding to the EBEs (conditional mode) estimated by Monolix (as displayed in Monolix Results > Indiv. param > Cond. mode.)
- **mlx_CondMean**: [when COND. DISTRIB. task has run] a table of individual parameters corresponding to the conditional mean estimated by Monolix (as displayed in in Monolix Results > Indiv. param > Cond. mean.)
- **mlx_CondDistSample**: [when COND. DISTRIB. task has run] a table of individual parameters corresponding to one sample of the conditional distribution (first replicate of the Monolix result file IndividualParameters/simulatedIndividualParameters.txt).

2.5. Covariates

Values to use for simulation should be defined for covariates used in the statistical model, if the simulation is based on population parameters. In this case, new individual parameters are simulated from the population distributions and the covariate values. Several covariate elements can be defined in **Definition**, but only one covariate element should be used per simulation group in **Simulation**. No covariate element can be used in **Exploration**, since only individual parameters are used in **Exploration**.

Demo projects: 3.2. covariates

New covariate element

Several types of covariate elements can be defined:

- **Manual**: It is a vector that has one or several dosing times with identical or different amounts. The + and – buttons allow to add and remove doses.

New manual covariates

MANUAL DISTRIBUTION EXTERNAL

Name
covTypical

Covariate	Value
AGE	35
RACE	Asian ▾
SEX	F ▾

CANCEL OK

- Distribution:** The covariates are described with distribution laws. When this type of element is used for simulation, new covariate values are simulated from the distributions. For continuous covariates, the distribution can be normal, lognormal or logitnormal with a mean and sd, or uniform with two interval limits. If the distribution is lognormal or logitnormal, sd is the standard deviation of the distribution in the Gaussian space. The typical value is the median of the covariate distribution..

New covariates from distribution

MANUAL DISTRIBUTION EXTERNAL

Name
covDistributionsAsian

Continuous covariates

Covariate	Distribution	Typical	sd
AGE	NORMAL ▾	45	10

Categorical covariates

Covariate	Category	Probability	Auto fill
RACE	Asian	1.0	<input type="checkbox"/>
	Black	0	<input type="checkbox"/>
	White	0	<input checked="" type="checkbox"/>
SEX	F	0.5	<input type="checkbox"/>
	M	0.5	<input checked="" type="checkbox"/>

CANCEL

OK

In the case of a lognormal distribution, in order to get the sd s_G of the distribution in the Gaussian space given a typical value of the lognormally distributed covariate μ , or given its mean m , and given its sd s , you can use the following formulas:

$$s_G = \sqrt{\ln\left(1 + \left(\frac{s}{m}\right)^2\right)} = \sqrt{\ln\left(1 + \sqrt{1 + 4\left(\frac{s}{\mu}\right)^2}\right) - \ln(2)}$$

Both formulas are equivalent if $\frac{s}{\mu} \ll 1$ (in that case $\mu \approx m$).

For categorical covariates, the probability for each category can be defined in $[0,1]$. The sum of probabilities over all categories must be 1, and an “auto fill” option allows to automatically fill one or several of the categories to satisfy this rule.

- **External:** An external text file with columns id (optional), occasions (optional), and one column per covariate (mandatory). The occasions headers must correspond to the occasion names defined in the occasion element. When id and occasion columns are present, then they must be the first columns. When the id column is not present, the covariate is considered ‘common’, i.e the same for all individuals. Categorical covariate values must correspond to the categories defined in the model (block [COVARIATE]). The external file can be tab, comma or semicolon separated. The possible file extensions are .csv or .txt.

Covariate elements imported from Monolix

Importing a **Monolix project** generates automatically a covariate element based on the Monolix data set.

- **mlx_Cov:** contains covariate information for each individual. The table is saved as external table in the result folder of the project. This table contains a column id, and one column per covariate. If there are occasions in the Monolix project, it also contains one or several columns for the occasion levels.
- **mlx_CovDist:** element of type “distribution” with typical values, standard deviations and probabilities calculated on the individuals of the Monolix project. No correlation between the covariates is assumed. In the 2020 version, all continuous covariates are assumed to follow a normal distribution. In the 2021 version, continuous covariates having only strictly positive values in the Monolix data set are

assumed to follow a log-normal distribution, and the others are set with a normal distribution.

2.6. Treatments

Treatments definition includes times and amounts of the doses applied to the model. Treatments are applied to the model via the macros `depot()`, `iv()` or `oral()` used in the `model`. Moreover, treatments definition has an administration id, which allows to choose to which macros used in the model the doses are applied.

When used in “**Exploration**” or “**Simulation**”, several treatment elements can be combined together. For instance, a loading dose of type manual followed by maintenance doses of type regular, or a oral dose with `adm=2` and an iv dose with `adm=1`.

The units of time and amounts must be consistent with the units of the Monolix data set or with the model definition when starting from scratch.

Demo projects: 3.1. treatments

New treatments element

Several types of elements can be defined:

- **Regular:** It is used for regularly spaced dosing times with a unique amount and a unique infusion duration or rate (optional). It requires the start time, inter-dose internal, number of doses, and amount.

New treatment with regular schedule

Start time	Interdose interval	Nb doses	Amount
0	0.1	1	100

Repeat

Non-compliance probability

CANCEL OK

- **Manual:** It is a vector that has one or several dosing times with identical or different amounts. The + and – buttons allow to add and remove doses.

New treatment with manual schedule

REGULAR MANUAL EXTERNAL

Name: trtManual1 Adm: 1

Values

Time	Amount	
0	10	-
24	5	- +

Repeat

Non-compliance probability

CANCEL OK

- **External:** An external text file with columns id (optional), occasions (optional), time (mandatory), amount (mandatory), tinf or rate (optional), washout (optional). The occasions headers must correspond to the occasion names defined in the occasion element. When id and occasion columns are present, then they must be the first columns. When the id column is not present, the treatment is considered 'common', i.e the same for all individuals. The washout column can contain zeros (no washout) or ones (washout).

id	time	amount	tinf
1	0	85.3	0.5
1	12	85.3	0.5
1	24	85.3	0.5
2	0	79	0.5
2	12	79	0.5
2	24	79	0.5

The external file can be tab, comma or semicolon separated. The possible file extensions are .csv or .txt.

New treatment from external file

REGULAR MANUAL EXTERNAL

Name Adm

trtExternal1 1

File

... /ExternalFiles/external_treatment_byAge.csv SELECT

Non-compliance probability

CANCEL OK

Additional treatments definition specifications:

- **adm:** *[all]* the administration id of the treatment element (integer). The doses of a treatment element with adm=2 apply to all model macros defined with adm=2, for instance `oral(adm=2, ka)`. Macros used in the model without adm are implicitly defined with adm=1.

New treatment with manual schedule

REGULAR MANUAL EXTERNAL

Name Adm

trtManual1 2

Values

Time	Amount
0	1

Repeat

Non-compliance probability

CANCEL OK

- **infusion duration or rate:** *[regular/manual]* the list button on the right allows to display the infusion column. The value is understood as either a duration or a rate, depending on the chosen option.

New treatment with regular schedule

REGULAR MANUAL EXTERNAL

Name Adm

trtRegular1 1

Infusion: Duration Rate

Values

Start time	Interdose interval	Nb doses	Amount	Infusion
0	0.1	1	100	1

Repeat

Non-compliance probability

CANCEL **OK**

- **washout:** *[manual]* the list button on the right allows to display the washout column. When the washout tick-box is selected, a washout is applied just before the dose. A washout means that all model variables are reset to their initial values.

New treatment with manual schedule

REGULAR
 MANUAL
 EXTERNAL

Name: trtManual1 Adm: 2

Values

Time	Amount	Washout
0	10	<input type="checkbox"/>
48	10	<input checked="" type="checkbox"/>

Repeat

Non-compliance probability

CANCEL **OK**

- **repeat:** *[regular/manual]* This option allows to repeat a base pattern of doses several times (number of repetitions), with a given periodicity (cycle duration). In the example below, the dose is given on the three first days of the week, for 4 weeks. The doses on the first week are defined using the type 'regular' with three doses and an inter-dose interval of 1 day. This base pattern is then repeated every 7 days (cycle duration), 3 times (number of repetitions) in addition to the first week.

trtRegular1*

regular

Name: Adm:

Values

Start time	Interdose interval	Nb doses	Amount
0	1	3	100

Repeat

Cycle duration: 7

Number of repetitions: 3

Non-compliance probability

CANCEL OK

- **scale amount by a covariate:** *[regular/manual]* This option allows to personalize the dose given to different individuals by scaling the dose amount with the selected covariate and with a specific intercept for the linear relationship: $\text{Scaled amount} = \text{Amount} \times \text{Selected covariate} + \text{Intercept}$.

New treatment with regular schedule

REGULAR MANUAL EXTERNAL

Name: trtRegular1 Adm: 1

Values

Start time	Interdose interval	Nb doses	Amount
0	0.1	1	100

Repeat

Scale amount by a covariate

Covariate: AGE Intercept: 0

Non-compliance probability

CANCEL OK

- **non-compliance probability:** [all] The doses defined are applied to the model with a given probability $(1-p)$, where p is the probability to miss a dose. This allows to simulate non-adherence to a treatment. Which doses are applied and which are not rely on the random number seed defined in the project settings to ensure reproducibility.

New treatment with regular schedule

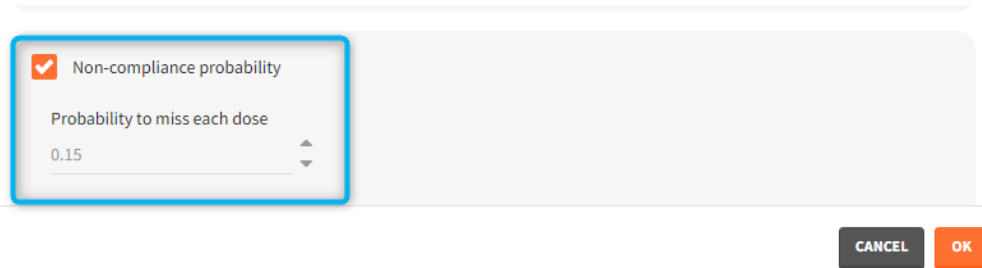
REGULAR MANUAL EXTERNAL

Name: trtRegular1 Adm: 1

Values

Start time	Interdose interval	Nb doses	Amount
0	1	28	100

Repeat



Treatment elements imported from Monolix

Importing a **Monolix project** generates automatically a treatment element for each administration id defined in the Monolix data set.

- **mlx_Admx**: contains treatment information for admID=X for each individual including: dosing times, amount, infusion duration or rates (optional) and washouts (optional, from EVID=3 or EVID=4 in Monolix). The Monolix data set column SS is replaced by additional dose lines as defined by the Monolix setting “number of steady-state doses”. The ADDL column is also replaced by additional dose lines.

2.7. Outputs

Outputs of simulations are variables from the **longitudinal model** of continuous, discrete or event type. There are two outputs groups:

- **Main outputs**
 - Model predictions defined in the structural model as the “output” (eg. Cc).
 - Observations (model predictions with an error model) defined in the observation model (eg. y1).
 - All variables defined in the structural model as “tables” (eg. AUC).
- **Intermediate outputs**
 - Variables computed in the structural model and not listed as an “output” or “table” in the OUTPUT block (eg. amount in the peripheral compartment, time varying clearance).
 - Regressors.
 - Variables defined with the “additional lines” in the model definition.

Loading a model automatically generates the “main outputs”. Model predictions and tables are on a default time grid. If a project is built from scratch, then model observations use also the default time grid. Otherwise, in case of project imported from Monolix, they are on a grid given by the measurement times from the Monolix dataset.

Projects: 3.5.outputs, 7.noncontinuous_outputs

New output element

Outputs correspond to **output variables**, which can be written or selected from the list (green frame).

New output

REGULAR MANUAL EXTERNAL

Name
outputRegular1

Output variable
y1 Main outputs ▾ Intermediate outputs ▾

Start time	Interval	Final time
0	1	100

CANCEL OK

It is possible to have different outputs of the same variable. For example, one output shows the time evolution over a treatment period and another gives a value at the end of treatment to define an endpoint. However, charts display these outputs on one plot with merged time grids.

There are three output types.

- **Regular:** It is a vector on a regular time grid (start time, step, final time, (occ)) common for all subjects.

New output

REGULAR MANUAL EXTERNAL

Name
outCc

Output variable
Cc Main outputs ▾ Intermediate outputs ▾

Start time	Interval	Final time
0	0.01	120

CANCEL OK

- **Manual:** It is a vector with manually defined list of time points (use “space” to separate them) common for all subjects.

New manual output

REGULAR MANUAL EXTERNAL

Name
outCc_manual

Output variable
Cc Main outputs Intermediate outputs

Times
0 x 2 x 4 x 6 x 8 x 12 x 24 x 48 x

CANCEL OK

- **External:** It is a file with a table that has: a mandatory column “time” for time points, an optional column “id” (in the absence of the “id” column the time grid is equal for all subjects), and an optional column “occ” for occasions.

outputExternal1 [read-only]

Output: Cc

ID	OCC	time
1	1	0
1	1	1
1	1	2
1	1	3
1	1	4
1	1	5
1	1	6
1	1	7
1	1	8
1	1	9
1	1	10
1	1	11
1	1	12
1	1	13
1	1	14
1	1	15
1	1	16

CLOSE

Occasions

By default, the regular and manual output types apply to all subject-occasions. If occasions are common among all individuals, then an option “**occasion-wise values**” is available. In this case each occasion has a separate time grid.

New output

REGULAR MANUAL EXTERNAL

Name
outputRegular1

Output variable
Y

Main outputs ▾

Occasion-wise values

1

Start time	Interval	Final time
0	1	100

2

Start time	Interval	Final time
100	1	200

CANCEL OK

Time-to-event outputs

Time limits (start time and final time) define the observation period during which the survival curve is computed. The final time thus defines the censoring time.

Intermediate times are ignored. The time grid must be the same for all individuals. If an external file is provided with different start and end values for each individual, the events will be simulated on the same observation period for all individuals, using the $\min(\text{start time})$ and $\max(\text{end time})$ over all individuals.

Outputs imported from Monolix

Importing a Monolix project generates automatically output elements for the “main output” variables.

- **mlx_observationName** (eg. `mlx_y1`, `mlx_y2`): For each output of the observation model it is a table with ids and observation times corresponding to the measurements read from the dataset.
- **mlx_predictionName** (eg. `mlx_Cc`, `mlx_R`): For each continuous output of the structural model it is a vector with a uniform time grid with 250 points on the same time interval as the observations.
- **mlx_TableName**: For each variable of the structural model defined as table in the

OUTPUT section it is a vector with a uniform time grid with 250 points on the same time interval as the observations.

2.8. Regressors

Definition of **regressors** elements is available only if the **mlxtran model** used in a Simulx project uses it. In this case, specification of regressors is mandatory. All regressors used in a model are displayed in one element as separate columns.

Regressors names

- If a **Monolix project was imported**, then regressors names come from a dataset – headers of columns tagged as regressor.
- When building a **project from scratch**, regressor names are equal to names used in a model.

Demo projects: 3.6.regressors

Create new regressors element

There are three types of regressor elements:

- **Manual:** It is a vector with manually defined time points and values, which are common for all individuals.

Example: parameter – covariate relationship with the covariate as a regressor

New manual regressor

MANUAL EXTERNAL

Name
PCA_REG

Time	PCA	
0	9	-
1	10	-
3	12	-
6	15	-
12	21	-
24	33	-
36	45	- +

CANCEL

OK

- **External:** Regressors definition is a table with the following mandatory columns: "time" for time points and "regressor_name" for regressor values. Optionally, it can have a column "id". In the absence of the "id" column, the time grid is equal for all individuals and occasions (if defined in the model).

Example: PK-PD model with PK measurements as a regressor

external_Cc [read-only]

ID	time	Cc
1	0.5	3.22
1	1	5.15
1	2	6.52
1	3	6.85
1	5	5.5
1	7	4.75
1	9	3.94
1	12	2.73
1	15	2.08
1	18	1.49
1	24	0.76
2	0.5	2.81
2	1	4.6
2	2	6.5
2	3	7.04
2	5	6.56
2	7	5.8
2	9	4.84

CLOSE

- **Distribution:** The regressors are described via distribution laws which define their value across individuals. When a distribution is used, the regressor is considered as constant over time. When used in the simulation, new regressor values are sampled from the defined distributions for each individual. The distribution for each regressor can be normal, lognormal or logitnormal with a mean and sd, or uniform with a lower and an upper and bound. If the distribution is lognormal or logitnormal, sd is the standard deviation of the distribution in the Gaussian space, i.e $V_{reg} = V_{typical} \times \exp(\eta)$ with $\eta \sim N(0, sd^2)$. The typical value is the median of the distribution.

Defining a regressor element as a distribution is convenient for sequential PD model where individual PK parameters have been defined as regressors. After exporting the project to Simulx, new individual PK parameters can be sampled by defining the regressor element for the individual PK parameters as a distribution. Note that correlation terms are not possible.

regDistrib_PKparameters*

DISTRIBUTION

Name
regDistrib_PKparameters

Regressors defined with distributions are constant over time

Regressor	Distribution	Typical	sd	Limits
Tlag	UNIFORM			0 1.6
ka	LOGNORMAL	1.28	0.45	
V	LOGNORMAL	7.91	0.21	
Cl	LOGNORMAL	0.13	0.29	

CANCEL OK

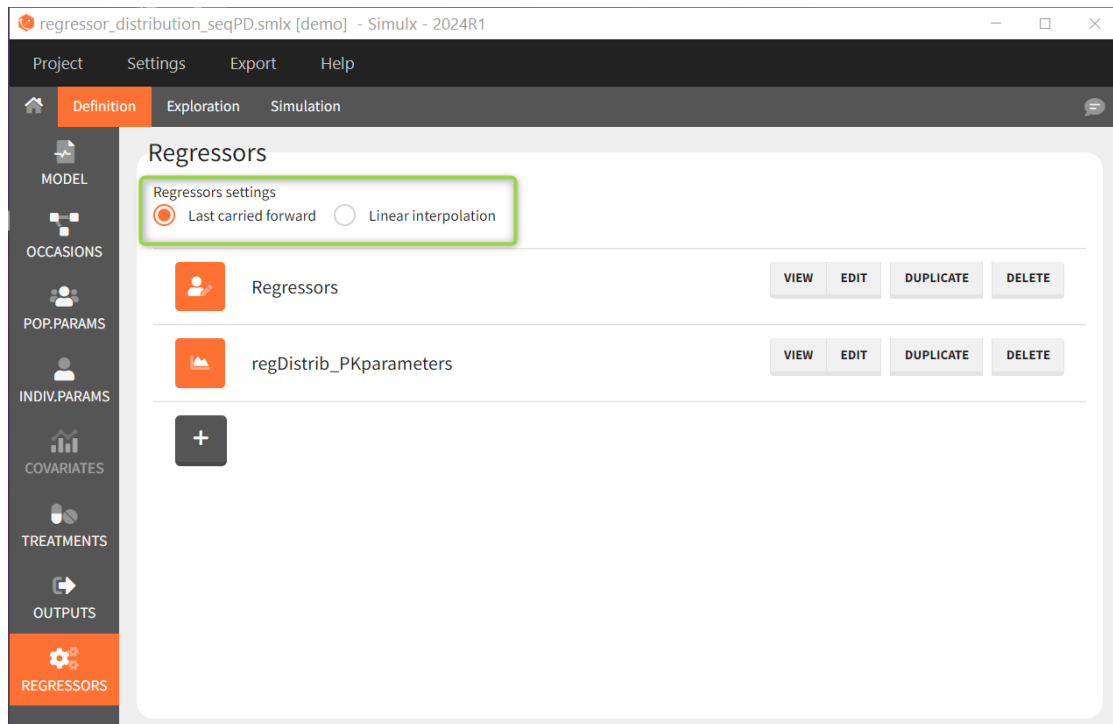
Interpolation between defined values

Regressors are defined over time for a finite number of time points. In between those time points, the regressor value can be interpolated in two different ways:

- **last value carried forward:** if we have defined reg_A at time t_A and reg_B at time t_B
 - for $t \leq t_A$, $reg(t) = reg_A$ [first defined value is used]
 - for $t_A \leq t < t_B$, $reg(t) = reg_A$ [previous value is used]
 - for $t > t_B$, $reg(t) = reg_B$ [previous value is used]
- **linear interpolation:**

- for $t \leq t_A$, $reg(t) = reg_A$ [first defined value is used]
- for $t_A \leq t < t_B$, $reg(t) = reg_A + (t - t_A) \frac{(reg_B - reg_A)}{(t_B - t_A)}$ [linear interpolation is used]
- for $t > t_B$, $reg(t) = reg_B$ [previous value is used]

This can be chosen in the DEFINITION > REGRESSORS tab and applies to all defined regressor elements.



Regressors imported from Monolix

Importing a Monolix project generates automatically regressors element with regressors names from the dataset:

- **mlx_Reg**: contains Ids, times and regressors values read from the dataset. Occasions are included if any.
- **mlx_RegDist**: (created only if all regressors of each subject and each occasion are constant over time) element of type "distribution" with typical values, standard deviations and probabilities calculated on the individuals of the Monolix project. No correlation between the regressors is assumed. Regressors having only strictly positive values in the Monolix dataset are assumed to follow a log-normal distribution, and the others are set with a normal distribution.

3. Exploration

The **Exploration** tab is the second tab in the interface of Simulx. It simulates a typical individual to explore a model by interactively changing treatments and model parameters.

- Simulate a single individual
- Compare treatments using exploration groups
- Display any non-random model variable as prediction output
- Interactivity: impact of parameters and dosing regimen on the model dynamics
- Define new parameter or treatment elements
- Add observed data in the plot
- Select elements for Simulation of clinical trials
- Plots settings
- Export the plots as images

Predicted individual

In Exploration, the predictions are based on only:

- one set of individual parameters (mandatory),
- one or several treatments (optional) for one individual,
- one or several outputs (mandatory) with a single vector of measurement times per output,
- one set of regressor values, only if regressors are present in the model (mandatory).

These elements are selected in the left panel.

Thus an element of type “**Individual parameters**” must be selected, and it is not possible to select an element of type “**Population parameters**”.

If a selected element is a table containing several sets of individual values, then the first row in the table is selected by default, and the “ID” field on top of the plot in the middle allows to select the row in the table corresponding to a particular ID.

[Demo: 1.overview/importFromMonolix_resimulateProject.smlx]

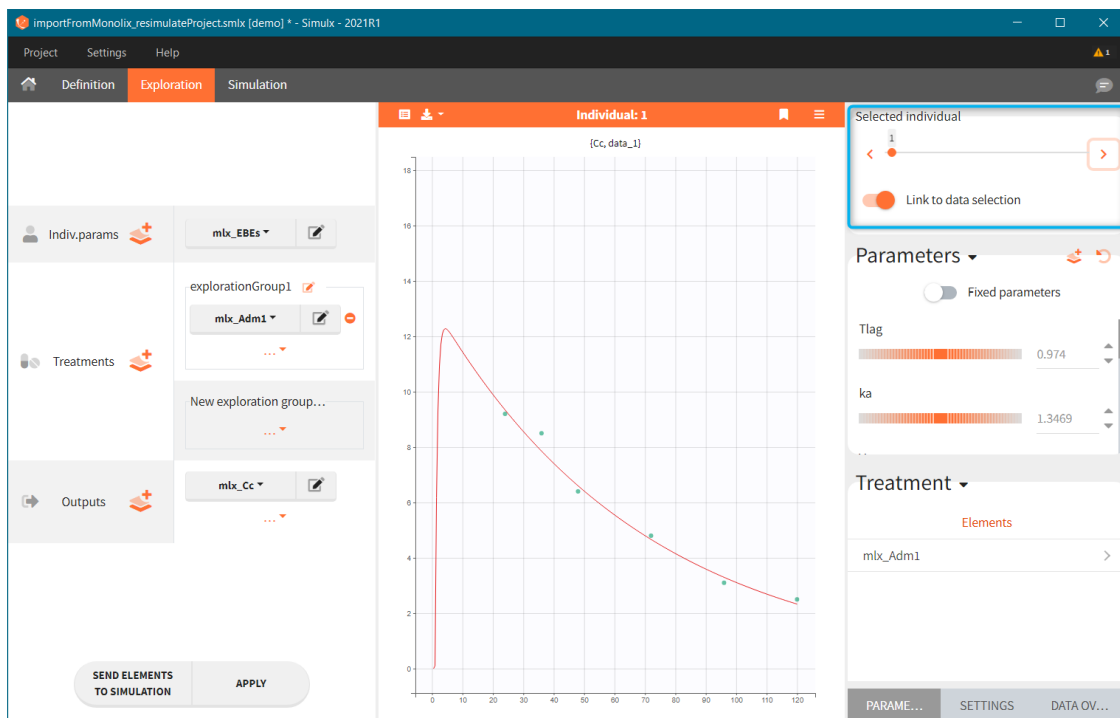
This field is highlighted on the figure below, corresponding to a project that comes from a **Monolix project imported into Simulx**:

- the individual parameters come from the table mlx_EBEs contains the EBEs

estimates,

- the treatment `mlx_Adm1` contains the individual doses imported from the dataset used in Monolix,
- the output `mlx_Cc` contains a single vector of regular measurement times for `Cc`, the structural model output.

Thus changing the “ID” value will change the row read from the two tables `mlx_EBEs` and `mlx_Adm1`.

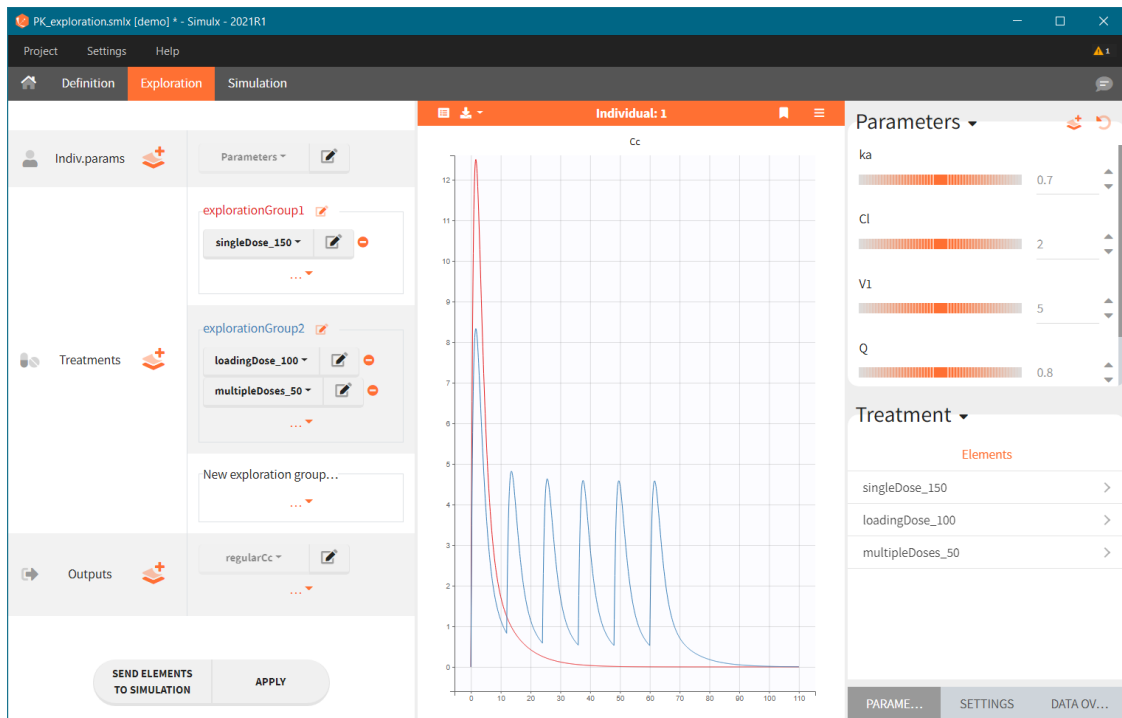


Exploration groups

• 3.exploration/PK_exploration.smlx

Treatment can contain several elements in different exploration groups (`explorationGroup1` and `explorationGroup2` below), or combined within an exploration group (as in the `explorationGroup2` below). Each exploration group produces a separate prediction. If an exploration group contains several treatments, then the prediction corresponds to the treatments given together to the same individual.

In the demo shown below, there are two exploration groups: the first corresponds to the treatment element `singleDose_150`, while the second corresponds to the combination of `loadingDose_100` and `multipleDoses_50`. Two prediction curves in different colors represent these two exploration groups in the plot.



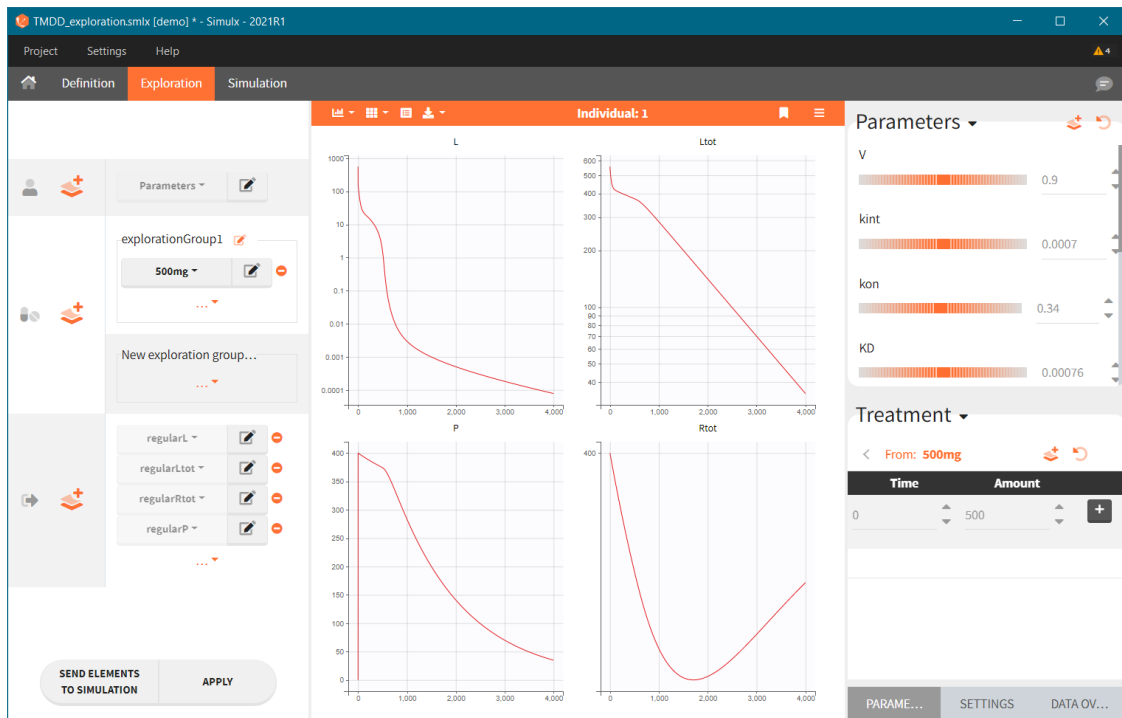
Exploration outputs

- **3.exploration/TMDD_exploration.smlx**

Only non-random element of type “**Output**” can be selected in Exploration, thus they should:

- correspond to continuous variables (consequently excluding time-to-event, categorical or count variables)
- not include a residual error model in their definition.

By default, there is a separate plot for each selected output, as shown below. This can be changed in the **exploration settings**, where it is possible to select variables to display at each chart. If the scenario contains several output elements based on the same variable (but on a different grid), then they are plotted on the same plot with merged grids.




Interactive modification of parameter values or dosing regimen


- **3.exploration/PK_exploration.smlx**

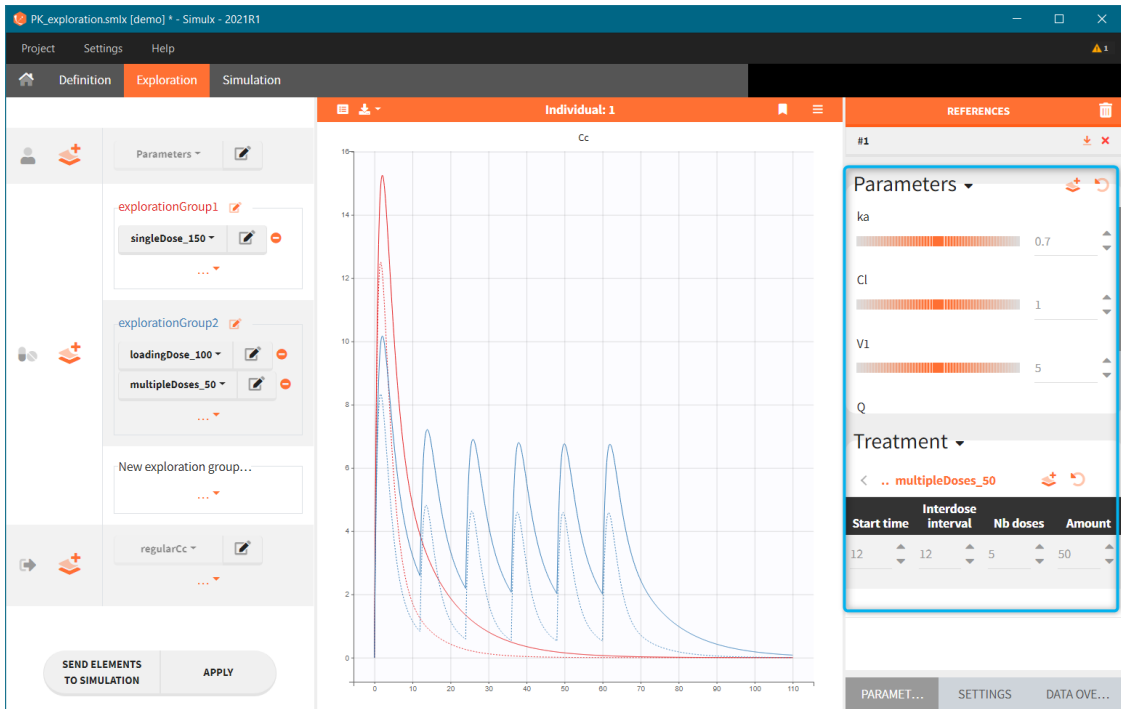
The right panel of the Exploration tab has three sub-tabs: PARAMETERS, SETTINGS and DATA OVERLAY. The parameter values and dosing regimens can be modified there to explore in real-time their impact on the model dynamics.


The figure below shows the PARAMETERS sub-tab with the model parameters (Parameters) and treatment parameters (Treatment) sections (highlighted in blue). The parameter values can be changed via the small arrows, sliders or by entering new

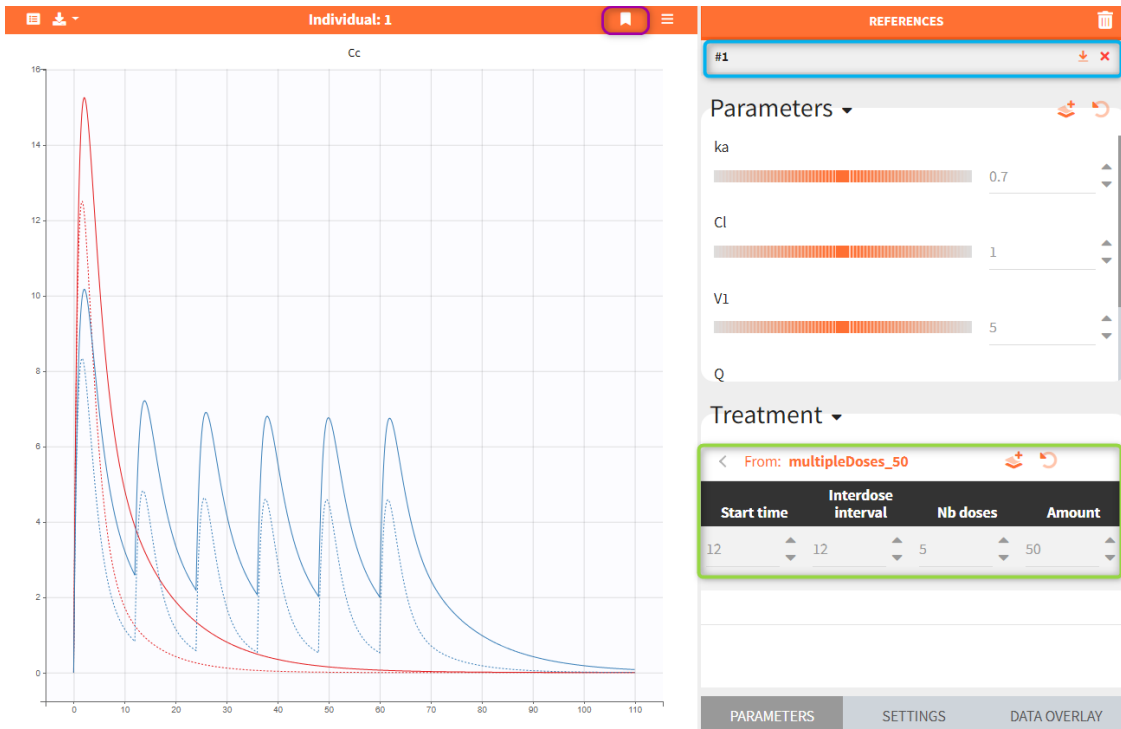
values with the keyboard. The “reset” button  resets parameters to their initial



values, while the “new” button  generates a new element (individual parameters element or treatment element) using the current values. On the plot, the solid curves correspond to the current predictions, while reference curves, added with the button

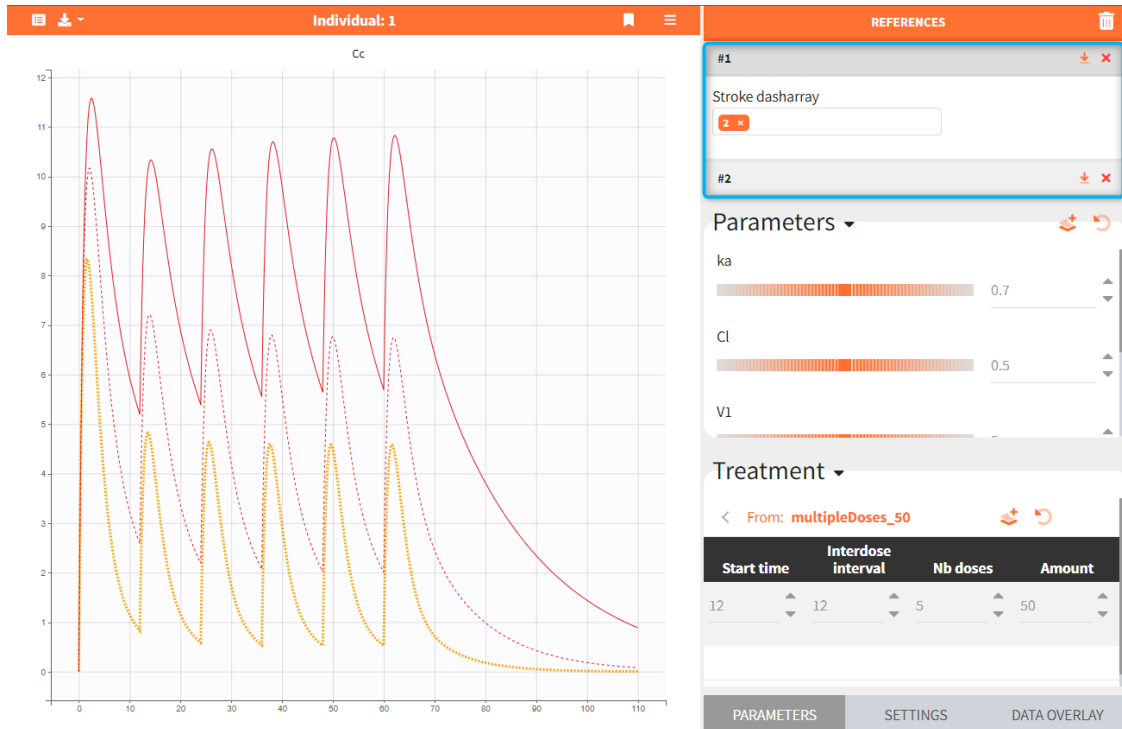
, are shown as dashed lines.



Reference curves. In the figure below, the section “Treatment” – highlighted in green – shows the elements from the treatment element: multipleDoses_50. All its parameters, such as start time, inter-dose interval, number of doses and dose amount can be modified. Current values can be set as a reference by clicking on the button  – highlighted in purple. The reference curves are listed on the right panel – highlighter in blue – and are represented by dashed lines in the plot.




Several reference curves can be created and each successive one is displayed with a corresponding number. The icon  restores the reference parameters values, and  removes the reference curve from the list and from the plot. Hovering on a reference line (#1 in the figure below) highlights in yellow the corresponding reference curve, as seen in the figure below. Stroke dash-array defines the pattern of dashes and gaps of a curve.

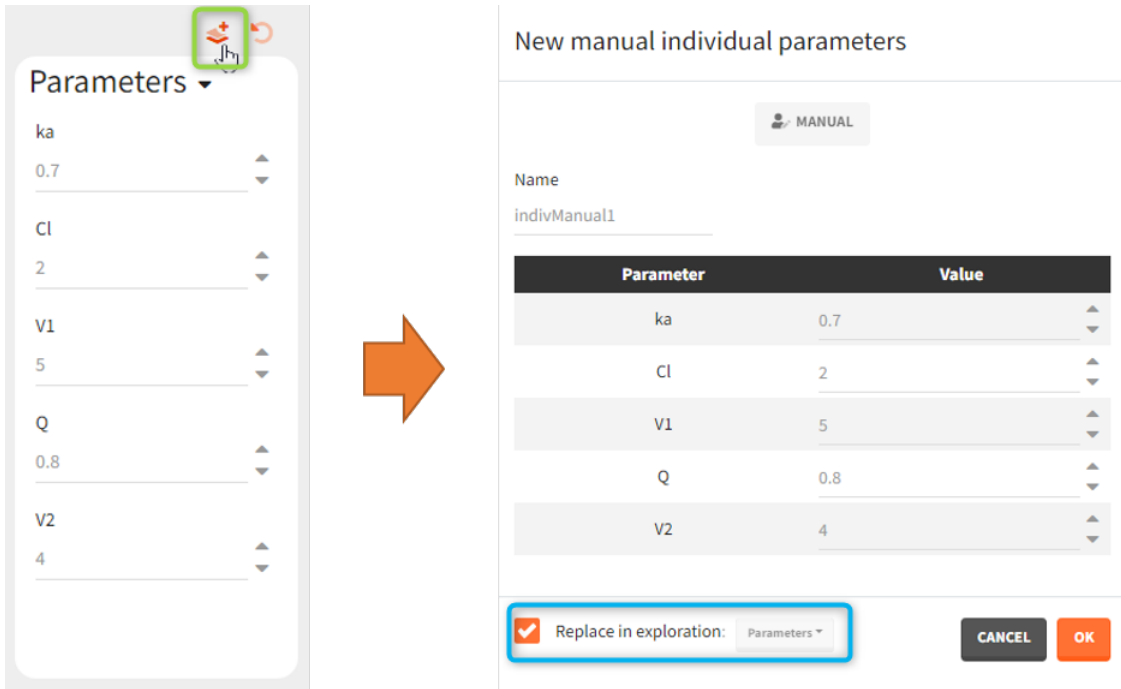


Defining new parameter or treatment elements

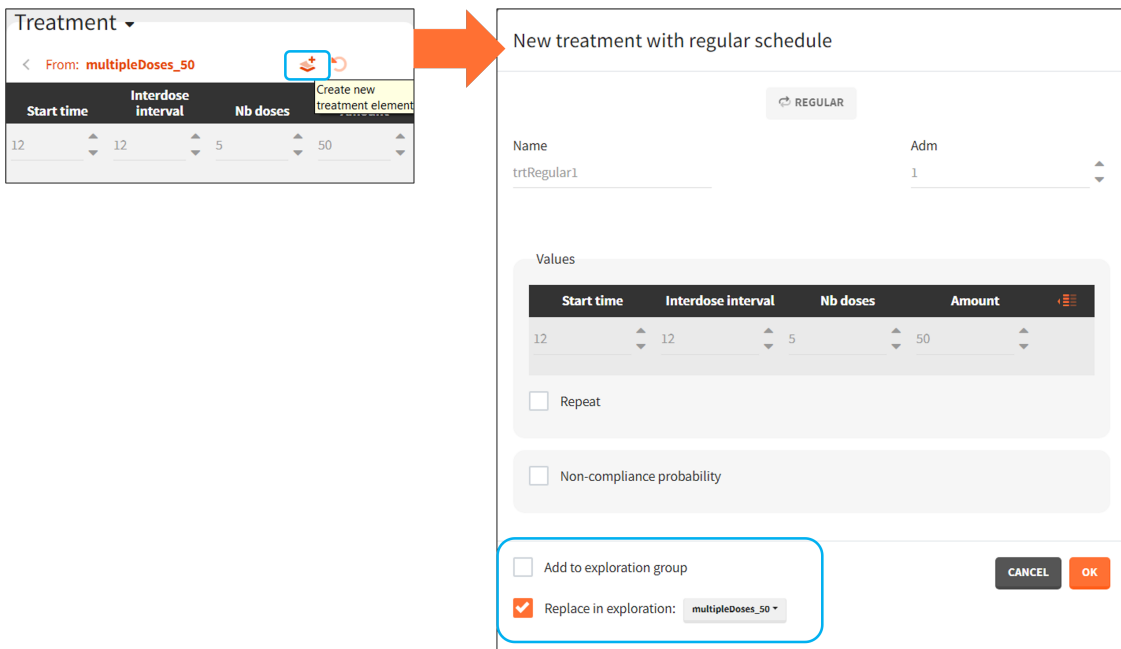
• 3.exploration/PK_exploration.smlx

Exploration allows to create new elements from the current values by clicking on the icon . In the parameter section it creates the **Individual parameters** element, while in the treatment section it creates the **Treatment** element.

A window to define a new element opens automatically. For the parameters, the individual parameter element is of type "manual" and by default replaces the previously selected element in the Exploration scenario. Untick the corresponding checkbox (blue mark in the figure below) to only define a new element – without the replacement.



A new treatment element can be set as a regular or manual type – depending on the definition in the exploration. By default it replaces the treatment element used in the exploration, but it can be also added as the new treatment to a new exploration group.

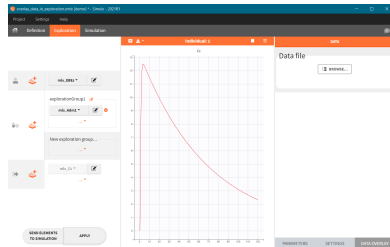


Overlay of the observed data

Continuous data can be overlaid on the prediction plots.

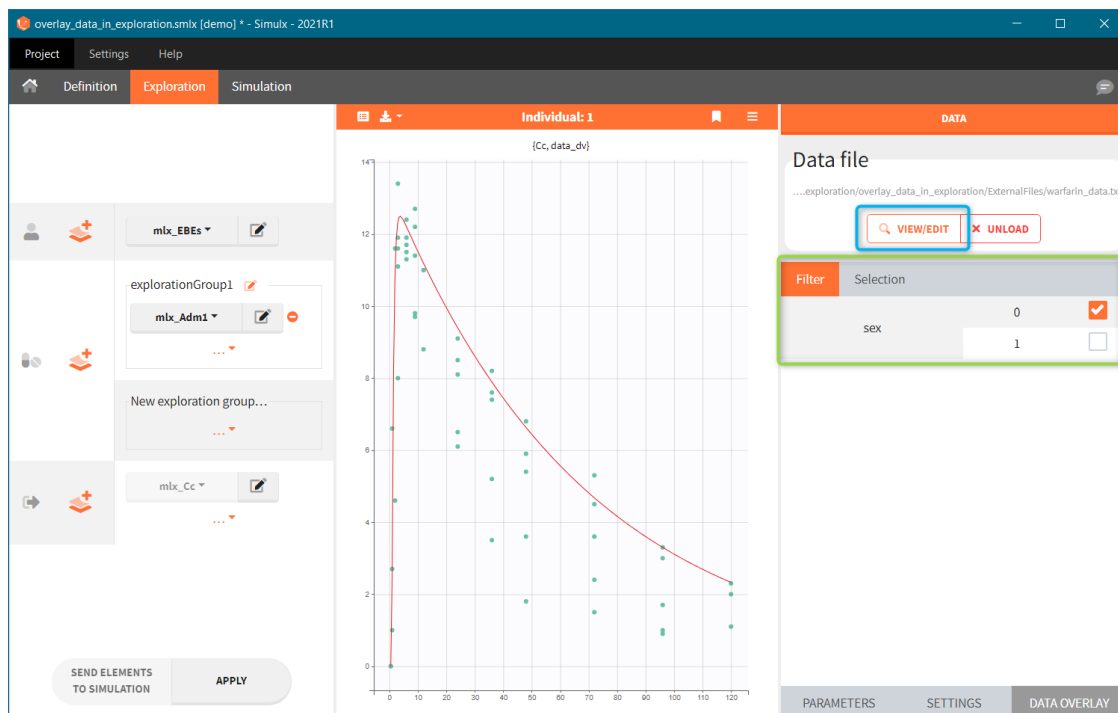
Load

Data overlay sub-tab in the Exploration tab allows to load a data file. Click on the BROWSE button and select a file. It has to be a Monolix compatible dataset and has to contain columns that correspond to: subject identifier (tag: ID), time of observations (tag: TIME) and observation values (tag: OBSERVATION). It is possible to have multiple observations to be displayed in different charts. In this case, the loaded datafile need to contain the column with observation id to distinguish different observations.



ID	TIME	OBSERVATION	COVARIATE 1	COVARIATE 2	COVARIATE 3	COVARIATE 4
1	100	0	0	0	0	0
4	100	0.5	0	0	0	0
4	100	1	1.0	0	0	0
4	100	2	0	0	0	0
4	100	3	0.5	0	0	0
4	100	4	0	0	0	0
4	100	5	0	0	0	0
4	100	6	0.5	0	0	0
4	100	7	0	0	0	0
4	100	8	0	0	0	0

After tagging columns, click on the ACCEPT button to load the data file. Displayed observations can be filtered (green frame) using categorical covariates present and correctly tagged in the loaded file. In the figure below, ticked box next to a value "0" means that in the plot there are only observations for individuals with SEX = 0. To change the tagging, click on the VIEW/EDIT button (blue frame).




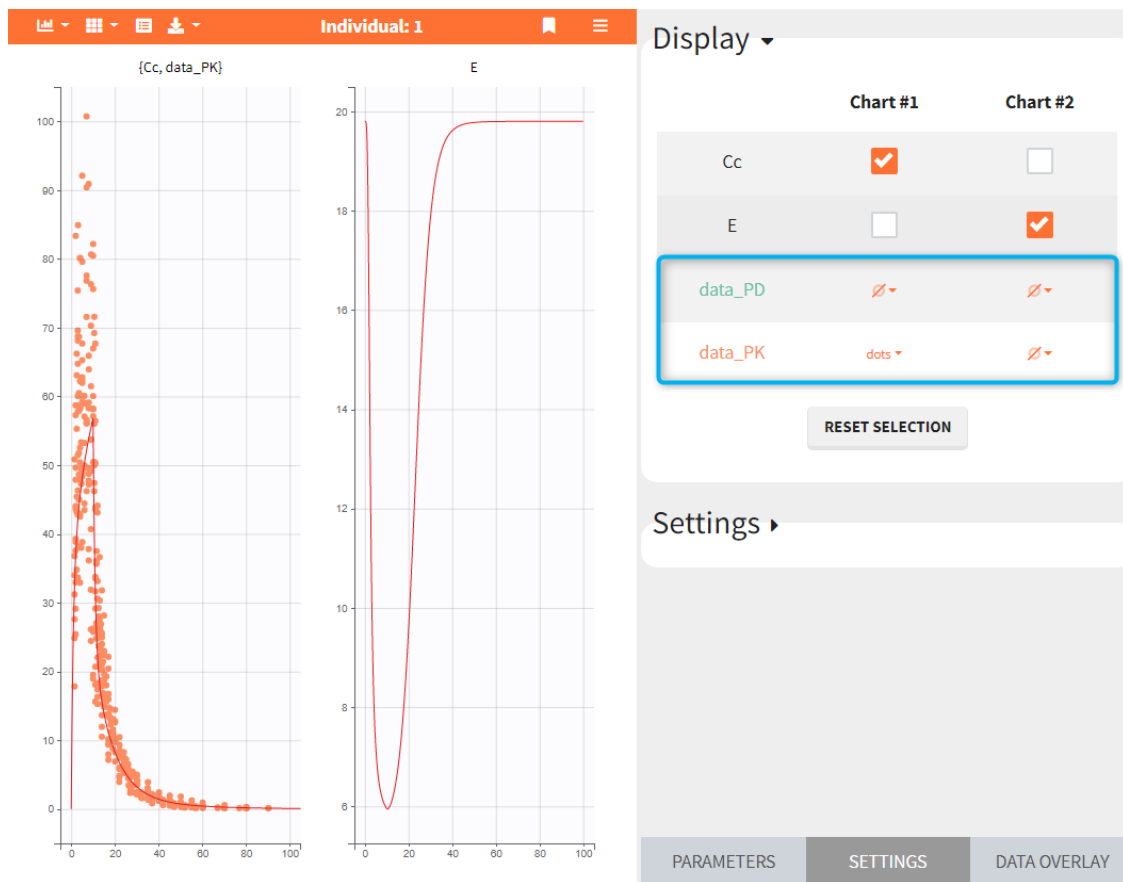
When a Simulx project is build by importing a Monolix project, then the dataset – used in the Monolix project – is set automatically in the DATA OVERLAY sub-tab.

Display

Observed data are displayed automatically as “dots” if there is only one output in the Exploration scenario and the data file contains only one observation type. Otherwise – in case of several different outputs or multiple observations types – data display has to be selected in the SETTINGS sub-tab.

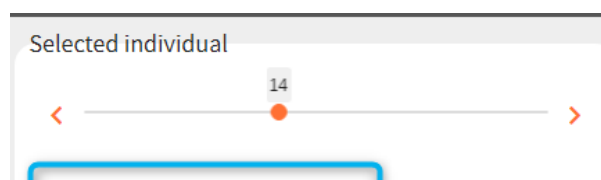
In the data – charts grid, see the figure below, observation data are listed in rows. Numerical order is applied if the observation identifiers in the data file are integers, eg. data_1, data_2, ...; or alphabetical order if the observation identifiers are strings, eg. data_PD, data_PK, as in the figure below. Charts are in columns. To display data

on a chart, click on the drop-down menu of  in the appropriate grid cell and select “dots” or “lines”.



Link to individual selection

If any of the exploration scenario elements contains a table with several individuals, then the simulated individual can be selected in the PARAMETERS sub-tab, see [individual prediction description](#) section for details. To link the observed data to the selected individual, enable the toggle “Link to individual selection”. Only data corresponding to the selected individual will be displayed on charts. The option “Link to individual selection” only appears if the list of ids found in the data set and in the elements selected for the simulation are the same.



Link to data selection

Parameters ↕ 🔄

Fixed parameters

Tlag 0.904

ka

Treatment ↕

Elements

mlx_Adml >

PARAMETERS SETTINGS DATA OVER...

It is also possible to display one or several individuals selected from the dataset, which may or may not be linked to the simulated individual. This option is available in the DATA OVERLAY sub-tab, in the "Selection" section. In the example below, selection contains a range of IDs between 15 and 42 and only observed data corresponding to these individuals are overlaid on the chart.

Individual: 1

[Cc, data_PK]

Data file

C:/Users/mtwar/Desktop/PKPD_data_remifentanil - strings.csv

VIEW/EDIT UNLOAD

Filter Selection

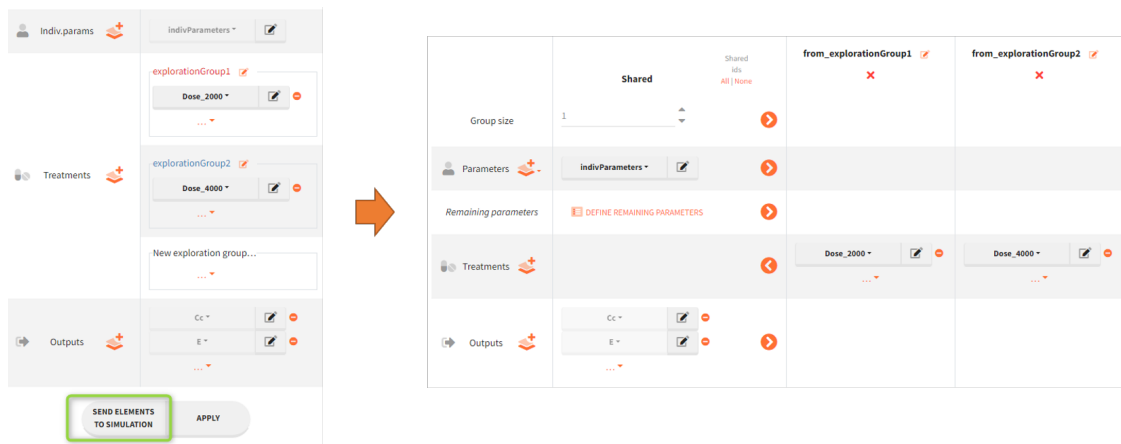
15 x 42 x

Range

PARAMETERS SETTINGS DATA OVERLAY

Sending elements to simulation

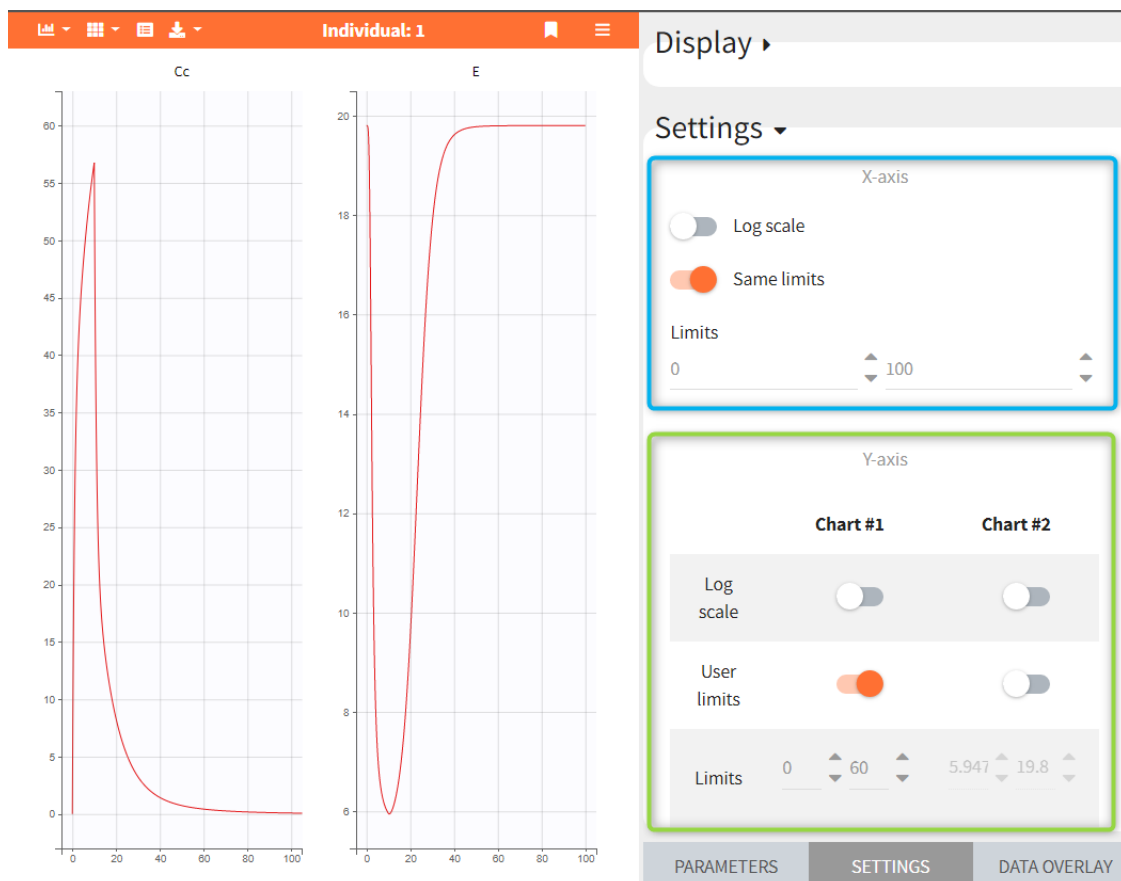
The button “Send elements to simulation” sends all elements selected in the Exploration to the scenario definition in the tab **Simulation**. Exploration groups are turned into simulation groups with a default size of 1 individual per group.



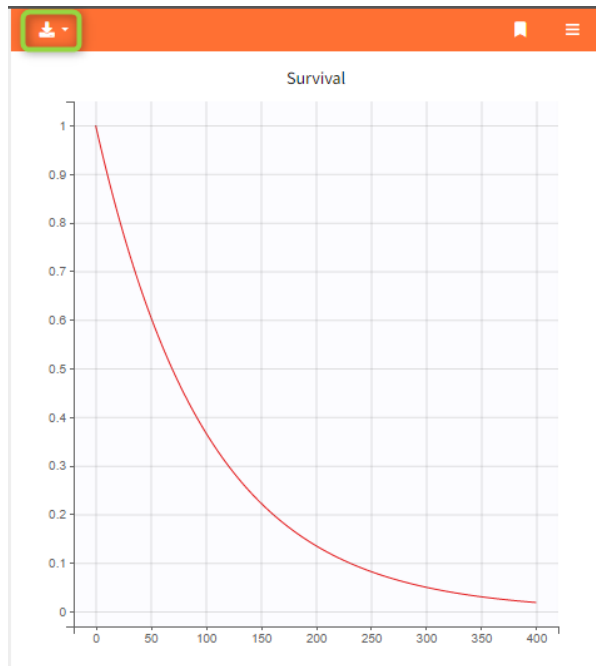
Plots settings

The panel “Settings” can be used to control plots features, such as the scale of the curves displayed on each plot and axis limits. The x-axis is common for all charts, while the y-axis can be specified independently for each chart.

In the following figure, two variables Cc and E are displayed on separate charts. They have the same x-axis limits – blue frame. The y-axis is specified for each chart: user limits 0-60 on the chart – 1, and automatic limits on the chart – 2 – green frame.



Exporting plots



The “export” button on top of the plots can be used to save the plots as an image in the result folder. Two image formats are available: PNG and SVG (Scalable Vector Graphics).

4. Simulation

The main goal of Simulx is to perform simulations of clinical trials. In other words, it simulates a population of individuals, in one or several groups, using different scenarios. It gives the last step of the modeling&simulation workflow, after model building in [Monolix](#) for instance.

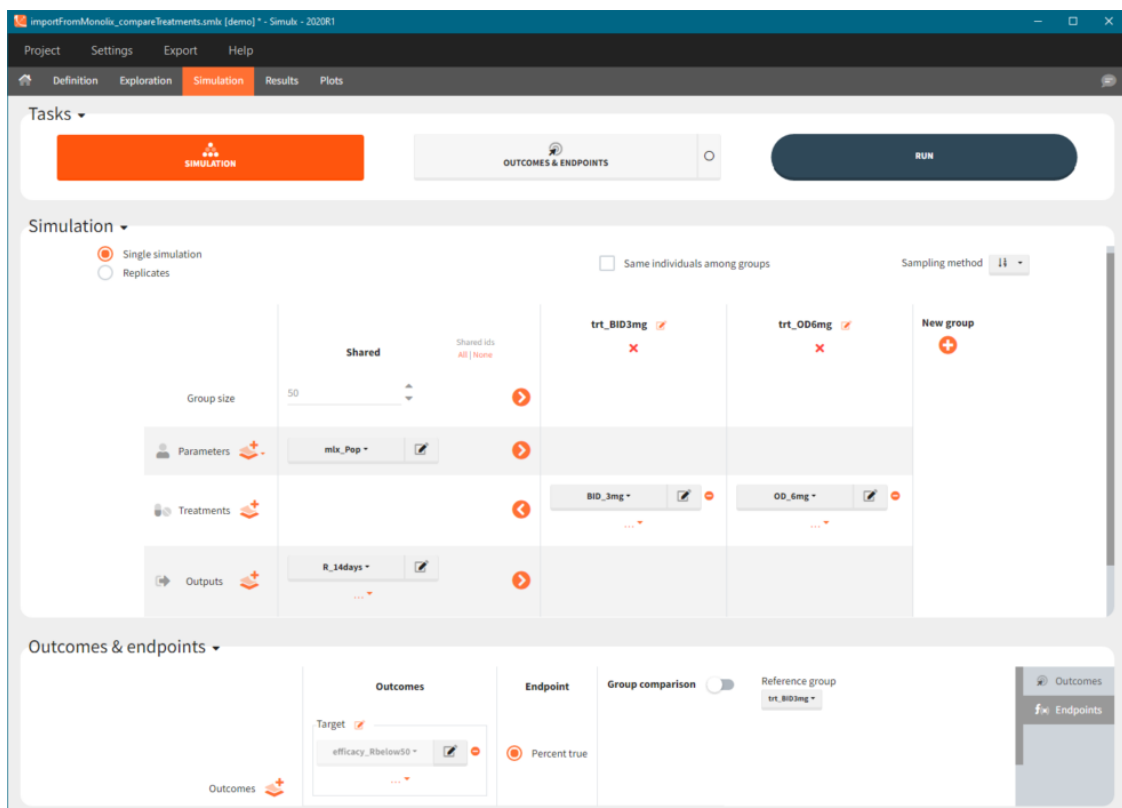
There are two tasks in the simulation tab: [SIMULATION](#) that generates outputs, and [OUTCOMES & ENDPOINTS](#) – a post-processing task – that uses the simulation outputs to calculate outcomes (one per individual) and endpoints (summary over all individuals of a group). In addition, running the tasks produces an immediate feedback thanks to the automatically generated [RESULTS](#) and [PLOTS](#).

Simulations of clinical trials include:

- [Simulation scenario](#) – Building a simulation scenario takes place in the dedicated section “Simulation” and includes selecting simulation elements, creating groups for comparison and choosing sampling options.
- [Outcomes and endpoints](#) – Outcomes & Endpoints are below the “Simulations”

section. Outcomes represent a post-processing of the simulation outputs for each individual. Endpoints summarize the outcome values over all individuals, for each simulation group and for each replicate. It is a separate task, so it does not require re-running the entire simulation after a modification.

- **Results** – Results are in a dedicated tab as copyable tables. They include individual and output values, outcome&endpoint analysis and statistical group comparison.
- **Plots** – All simulation outputs, defined outcomes and endpoints are displayed on separate plots as individual outputs and distributions. In addition, plot settings, stratification options and charts preferences are available at hand in the right panel of the interface.



Order of events

There are prioritization rules in place in case of various event types occurring at the same time. So, the order of row numbers in the data set is not important, and same is true for the order of administration and empty/reset macros in model files.

The sequence of events will always be the following:

1. regressors are updated,
2. reset done by EVID=3 or EVID=4 is performed,
3. dose is administered,
4. empty/reset done by macros is performed,

5. observation is made.

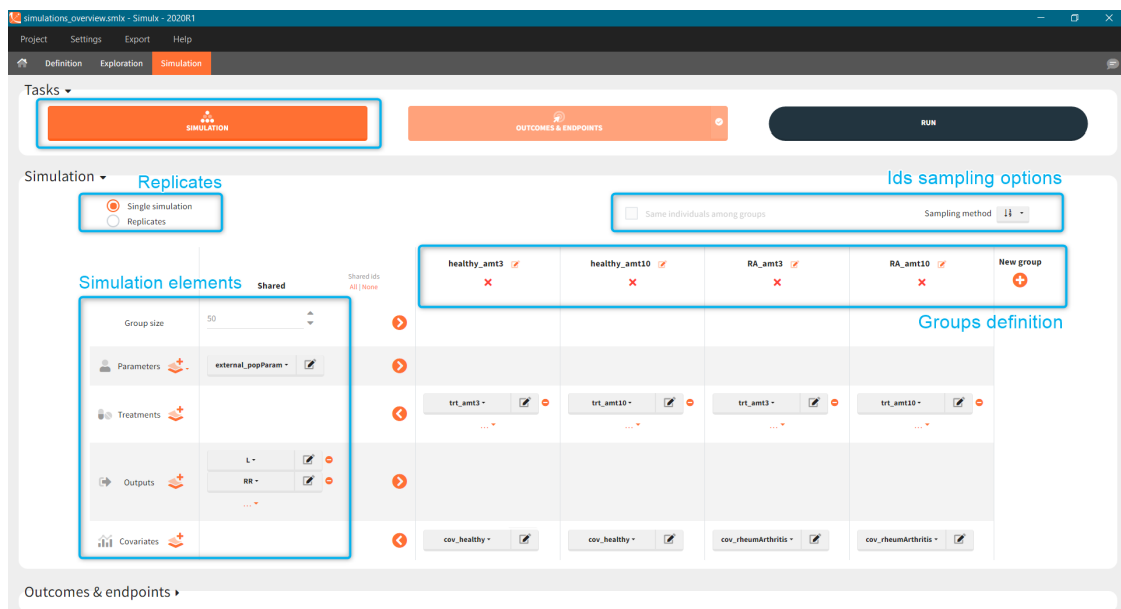
The behavior is consistent between Monolix and Simulx.


4.1. Building a simulation scenario

Simulx allows to **interactively build a simulation scenario** using simulation elements defined in the “Definition” tab and additional options available in the interface. It is done in the Simulations tab – Simulation section – which contains:

- List of **simulation elements**, which can be shared between groups or set as group specific.
- Flexible **group definition** with drop-down menus for selecting group specific elements and with editable group names.
- Option to run a single simulation or **replicates**.
- Different **sampling options** such as using subjects with the same individual parameters or drawing individuals from table type elements with different methods.



To run a scenario, just click on the button SIMULATION in the tasks section. When importing a Monolix project, Simulx automatically sets the scenario re-simulating the Monolix project.

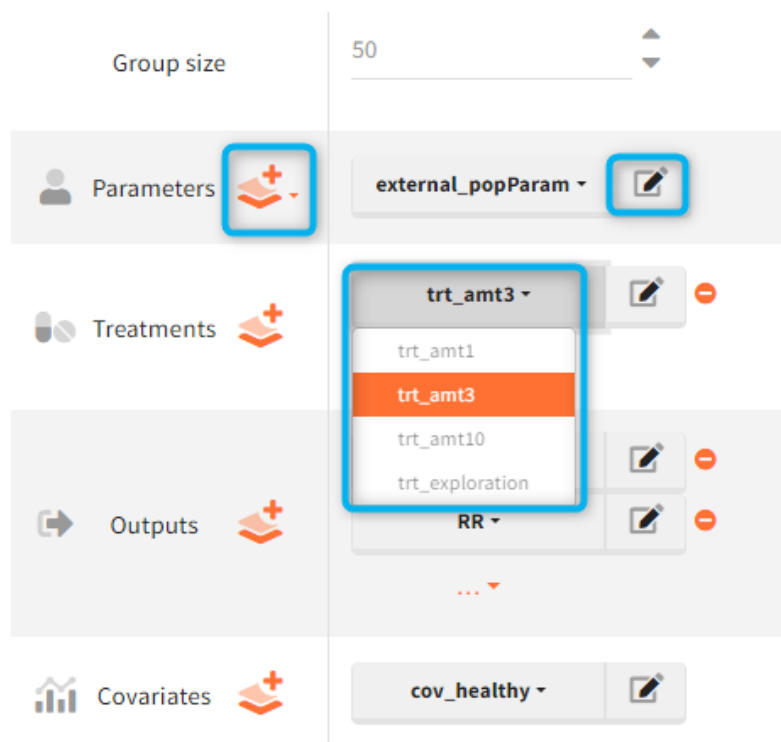


- Simulation elements
- Groups
- Replicates
- Sampling options and use of ids
 - Sampling options 

- Shared ids 
- Same individuals among groups 

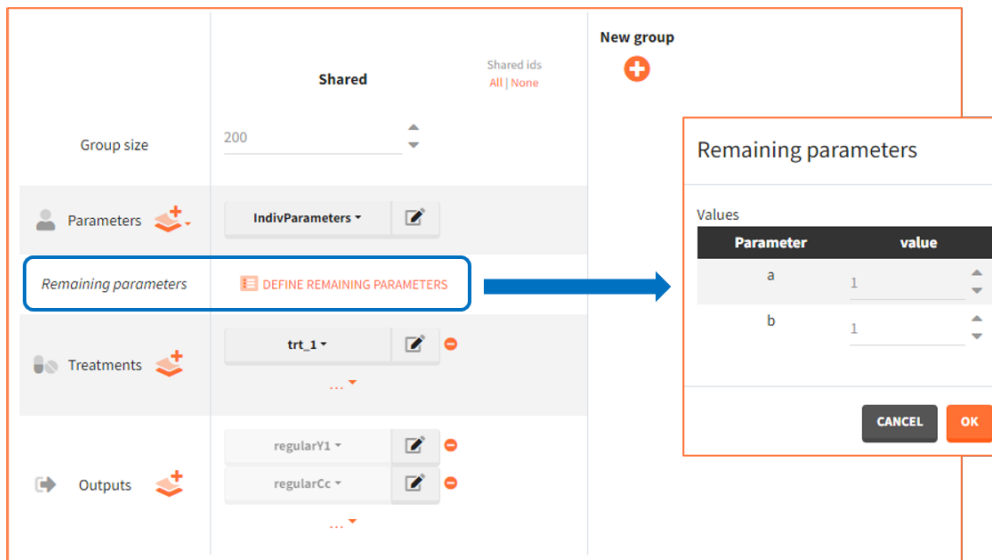
Simulation scenario elements

List of simulation elements contains: group size, parameters, treatments, outputs and, if used in a model, covariates and regressors. To **select an element** from already defined elements, use a drop-down menu available after clicking on the currently selected element. To **edit a selected element**, click on the icon . To **create a new element**, click on the “plus” icon  to open a “New element” pop-up window in the Definition tab.



Mandatory elements

- **Group size** corresponds to the number of individuals in a group.
- **Parameters** can be chosen from **POP. PARAM.** or **INDIV. PARAM.** elements in the Definition. When Individual parameters are selected, then the covariate element disappears and new section to set the error model parameters becomes available.



Note: Remaining parameters are all parameters that appear in the structural model (in the input line of [LONGITUDINAL]) and are neither individual parameters nor regressors. They are typically error model parameters. These error model parameters will impact the simulation only if a noisy observation (from the DEFINITION section of the [LONGITUDINAL] block) is set as output element (instead of a smooth prediction in OUTPUT or variable in EQUATION).

- **Outputs** correspond to one or more variables to be computed and displayed in plots and results. When importing a project from Monolix, by default all observation outputs (if defined in a model) are automatically selected, otherwise only the last defined model output.
- **Covariates** are available in two situations: selected treatments use a covariate scaling or individual parameters model uses covariates and population parameters are selected as an element in the Parameters.
- **Regressor** is available only if it is used in a model.

Optional elements

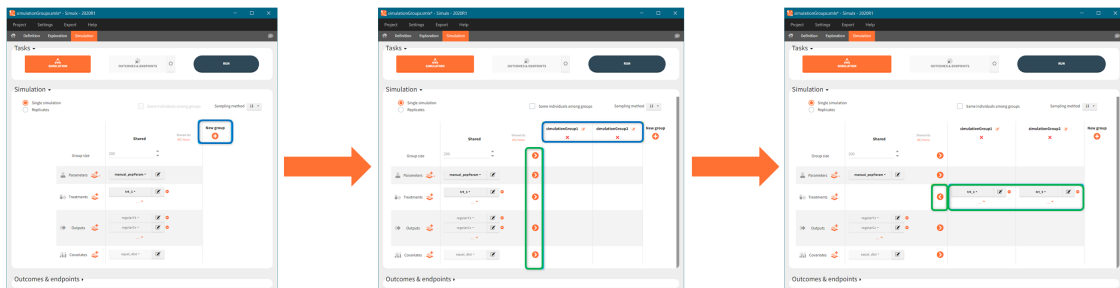
- **Treatment** can be one element or a combination of several treatment elements.
- **Occasions** are optional even if defined in a model, but are mandatory if any occasion-dependent element is selected in a scenario.


Simulx automatically assigns default elements: the last one from the definition list of each category or specific default "mlx_" types after importing a Monolix, [see here](#) for details. It is not possible to remove in the Definition tab an element currently used in a simulation. However, it might not be possible to run a simulation if any of the mandatory elements was removed. For instance, deleting occasions element removes automatically all occasion-dependent elements even if used in a simulation.

Simulation scenario with groups

Groups allows to build a scenario with population having different characteristics, such as parameters, regressors, covariates, different treatments and/or different simulation outputs. Moreover, groups can have different number of individuals to compare clinical trial of different sizes.

To *create a new group* click on the “plus” button, use the arrows to move elements from “shared” part to group section (or vice versa) and select specific element for each group. Each simulation element can be **shared between groups** or can be **group specific**.

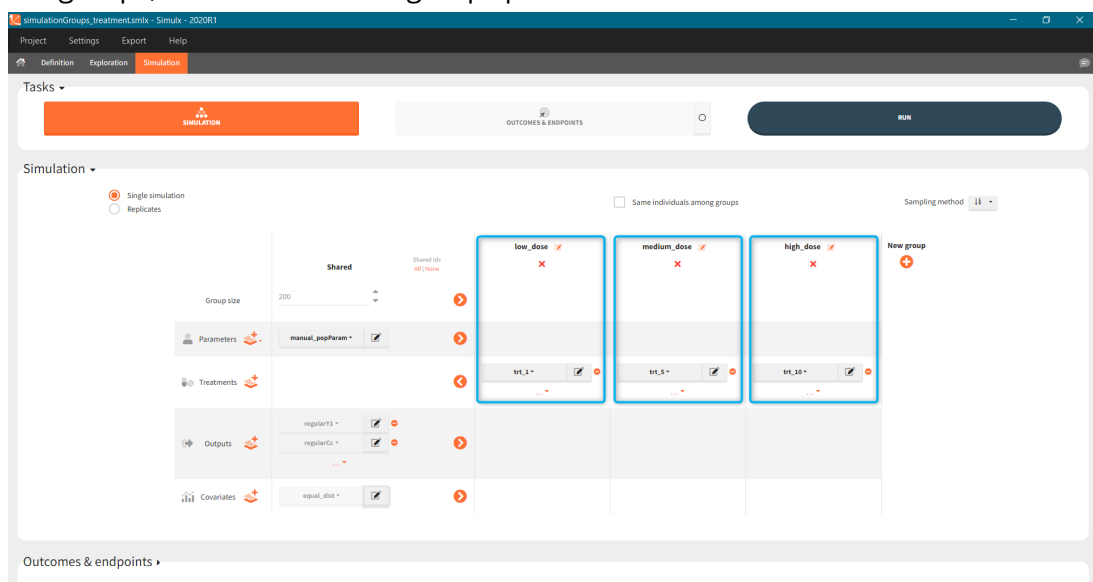


To **edit a group name** click  next to it. In the Plots tab, figures for different groups appear in the same order as they are defined in the Simulation tab, not in the alphabetical order. To **remove a group** click “x” under the group name.

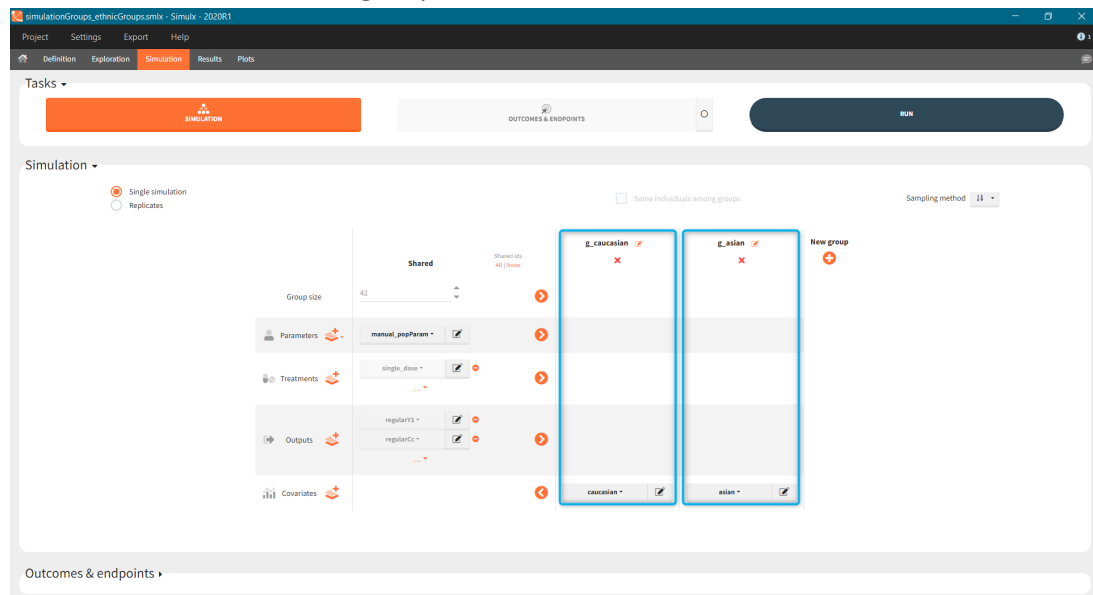
- Each new group is by default a copy of the previous one (eg. group 3 is a copy of group 2).
- Population and individual parameters can not be used in different groups simultaneously.

Examples:

1. [Demo 5.simulation: simulationGroups_treatment]: Simulation of three **groups with three different oral dose levels**: low (trt_1), medium (trt_5) and high dose (trt_10). Groups sizes, parameters, covariates and outputs are shared (the same) between the groups, while treatment is group specific.



2. [Demo 5.simulation: simulationGroups_ethnicGroups]: Simulation of two **groups with different ethnicity**, which affects the bioavailability. In this case, “covariates” simulation element is group specific with: “caucasian” modality for the first group and “asian” for the second group.

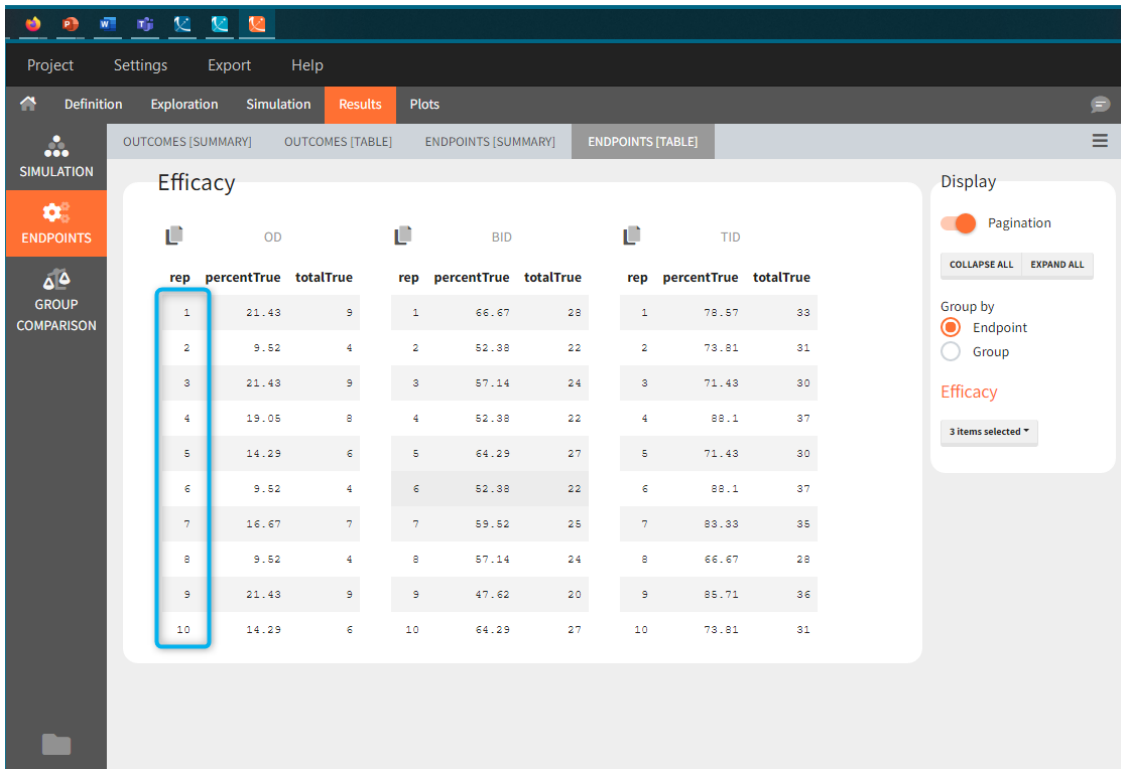


Simulation scenario with replicates

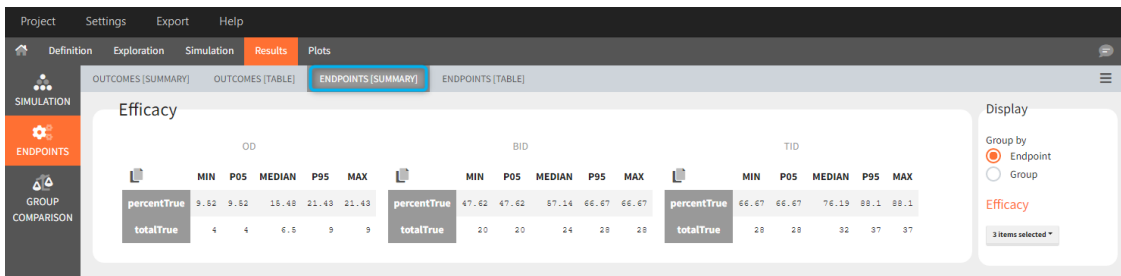
By default, Simulx simulates one clinical trial. When the option “replicates” is selected, then Simulx simulate the scenario several times.

- If population parameters are vectors (user-defined or imported from a Monolix project), then all replicates have the same population parameters.
- If population parameters are of a distribution type, then Simulx samples one set of population parameters for each replicate.
- If population parameters are given as an external table with several population parameters sets, for instance from the `simpopmlx` function, then each replicate uses one set of population parameters with the order of the appearance in the table.

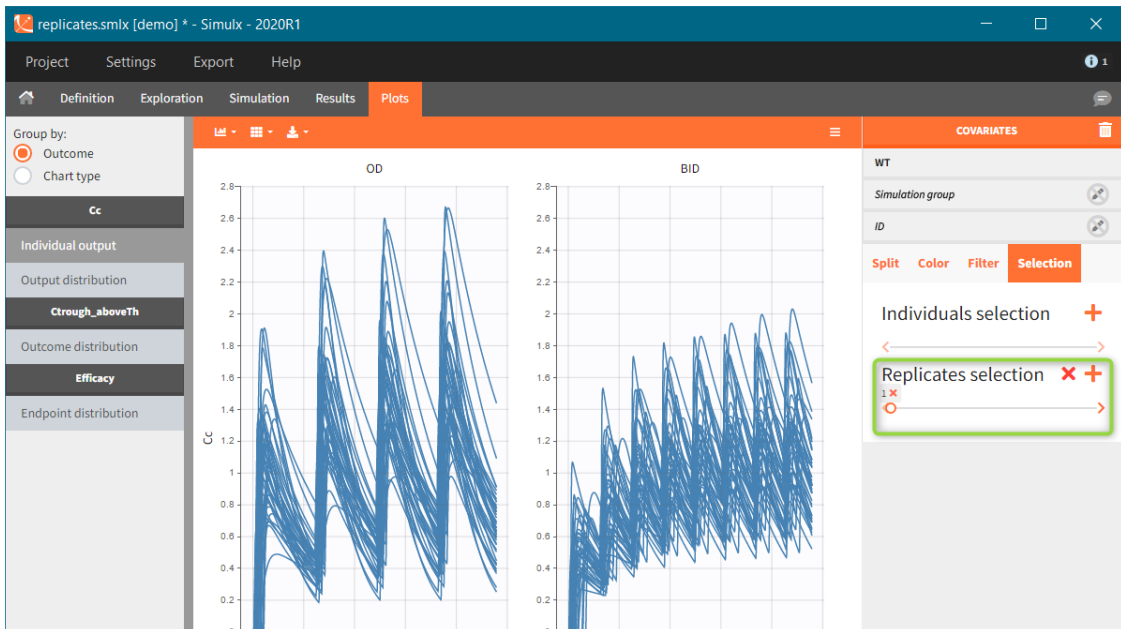
Tables in the results tab display information for all replicates.

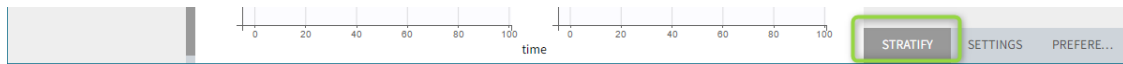


In “summaries” the values are summarized over all individuals and all replicates. For instance, endpoints result summarize outcomes over all replicates with the uncertainty of the result.

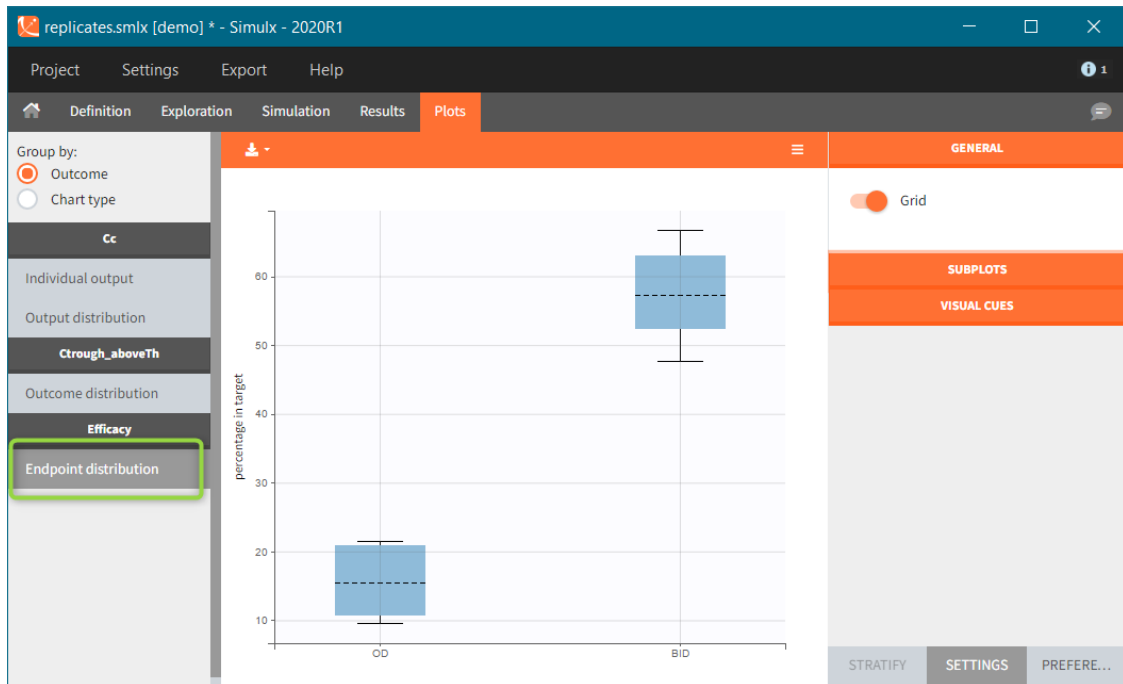


Similarly, plots contain information of all replicates. By default, individual outputs display only one replicate, however, stratification panel allows to select one or more replicate to plot (green frame).





Endpoint distribution in case of replicates is summarized in a box plot.



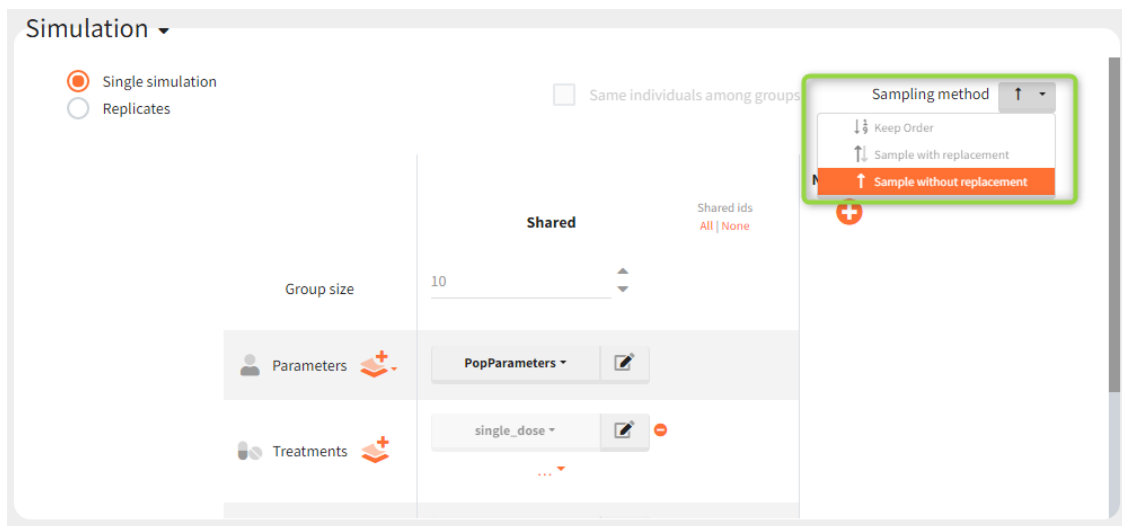
Sampling options

Individual sampling

Simulation elements of different types are subject to different individual sampling in a simulation. Elements defined as **vectors** provide common values for all individuals, so there is no sampling. When an element is a **distribution**, then sampling is done for each individual separately. **Tables**, from external files or from Monolix after importing a project from Monolix, can contain data for more than one individual distinguished with the column called "id". In this case, there are three sampling methods available in the Simulation tab.

- **Keep order** (default) – individual values are taken in the same order as they appear in a table.
- **With replacement** – individual values are samples from a table with replacement.
- **Without replacement** – individual values are samples from a table without replacement. This option is available only if tables contain at least the same number of individual values as a group size.

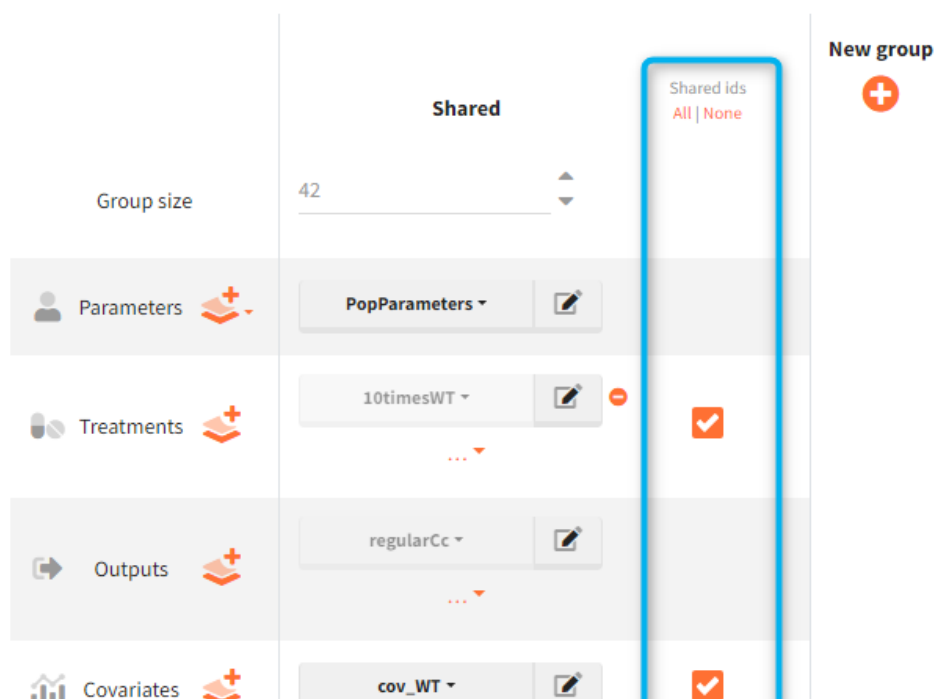
All of the above sampling methods are general and apply to all tables in a simulation scenario.



Shared ids

If there is a group of tables with individual values, an option “shared ids” allows to create an intersections of ids present in the considered tables. After that, ids from these intersection are sampled to create the data for simulation.

Example: Simulation uses a treatment with a dose amount depending on the patients weights. Treatment element and weights are external tables with columns (id, time, amount) for the treatment, and (id, weight) for the covariate. Moreover, an effect of the weight is added on the definition of the individual parameter (volume of the central compartment) in the model. In this case, it is important to match a sample from the treatment element with a sample from the covariate element. It is done by ticking two boxes in the “shared ids” column for both elements (blue frame).



Same individuals among groups

“Same individuals among groups” option allows to have **the same individual parameters in all groups** and is available if the following elements (required for the sampling) are the same for all groups: size of groups, parameters (population or individual) and covariates. The main aim is to make the comparison between groups easier. In particular, it is used to *compare different treatments on the same individuals* – subjects with the same individual parameters. Selecting same individuals among groups assures that the differences between groups are only due to the treatment itself. To obtain the same conclusion without this option enabled, simulation should be performed on a very large number of individuals to averaged out the individual differences.

Example: a simulation of two treatment groups with shared parameters, covariates and outputs. When the option “same individuals among groups” is enabled, then subjects in each group have the same individual parameters because the same Parameters (manual_popParam) and Covariates (race) are used in all groups (Demo 5.simulation: simulationGroups_sameldsInGroups).

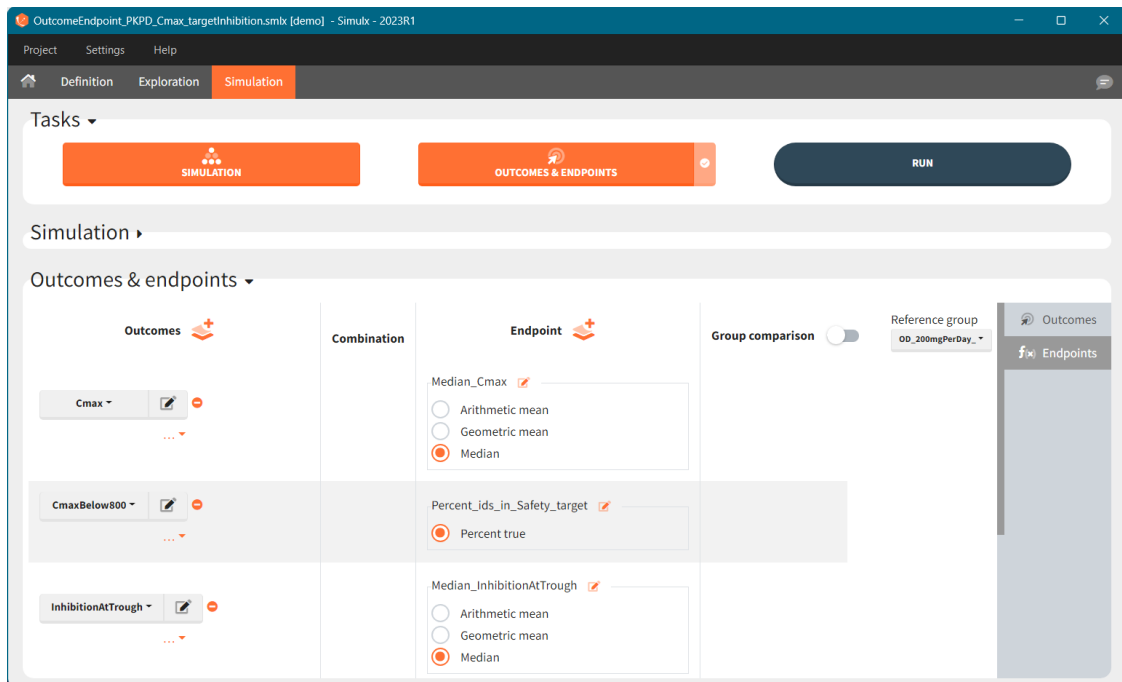
The screenshot displays the Simulx software interface for a simulation project. The main window is titled 'simulationGroups_sameldsInGroups.sim* - Simulx - 2020R1'. The 'Simulation' tab is active, showing a 'Definition' section with a checked option 'Same individuals among groups'. Below this, there are two groups: 'low_dose' and 'high_dose'. A 'Shared' section lists parameters and covariates: 'manual_popParam', 'race', 'regpara1+', 'regpara2+', and 'regpara3+'. A 'Sampling' button is visible. On the right, the 'Individual parameters' table is shown, displaying sampled individual parameters and covariates for 10 individuals across two groups. The table has columns for 'low_dose' and 'high_dose', each with sub-columns for 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'V29', 'V30', 'V31', 'V32', 'V33', 'V34', 'V35', 'V36', 'V37', 'V38', 'V39', 'V40', 'V41', 'V42', 'V43', 'V44', 'V45', 'V46', 'V47', 'V48', 'V49', 'V50', 'V51', 'V52', 'V53', 'V54', 'V55', 'V56', 'V57', 'V58', 'V59', 'V60', 'V61', 'V62', 'V63', 'V64', 'V65', 'V66', 'V67', 'V68', 'V69', 'V70', 'V71', 'V72', 'V73', 'V74', 'V75', 'V76', 'V77', 'V78', 'V79', 'V80', 'V81', 'V82', 'V83', 'V84', 'V85', 'V86', 'V87', 'V88', 'V89', 'V90', 'V91', 'V92', 'V93', 'V94', 'V95', 'V96', 'V97', 'V98', 'V99', 'V100'. The table shows values for each parameter and covariate for each individual in both groups.

4.2. Outcomes and endpoints

- Outcomes
- Endpoints
- Group comparison

- Statistical tests
- Endpoint “median” for value outcome
- Endpoint “arithmetic mean” for value outcome
- Endpoint “geometric mean” for value outcome
- Endpoint “percent true” for binary true/false outcome
- Endpoint “median survival” for time-to-event outcome

Outcomes, endpoints group comparison are in a dedicated section “Outcomes&endpoints” of the Simulation tab.



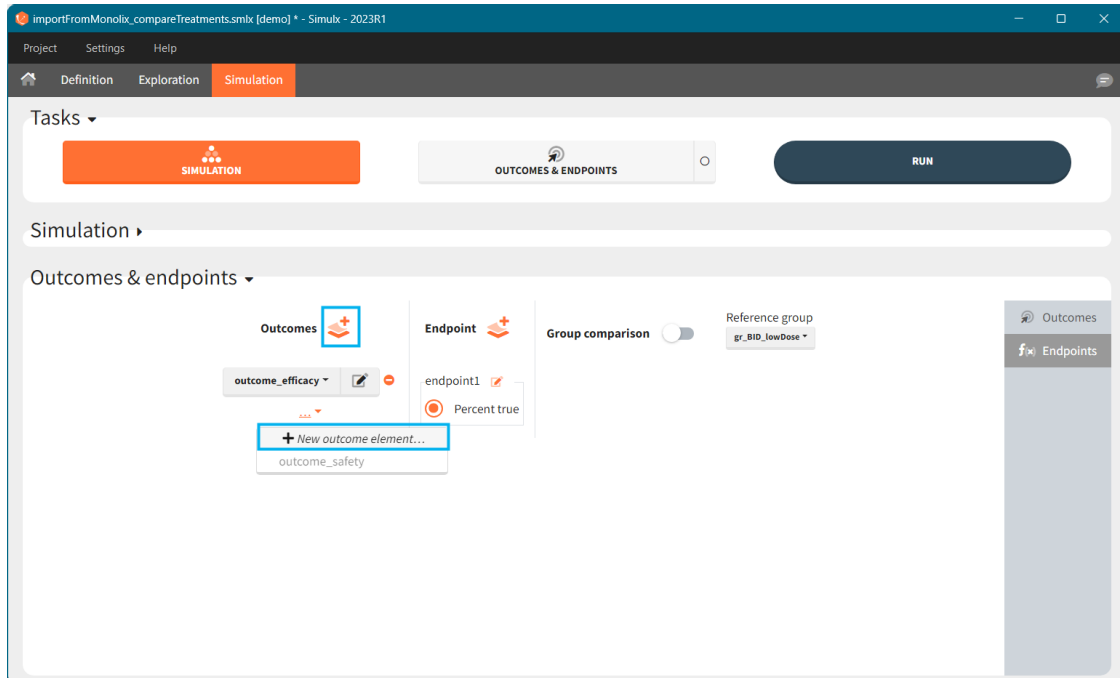
To calculate outcomes and endpoints click on the “*Outcomes&Endpoints*” button in the Tasks section (on top). The outcomes are a post-processing of the simulation outputs, so remember that you have to first run the “*Simulation*” task. You can also selected the task “*Outcomes&Endpoints*” in a scenario (circle radio button on the right hand side of the task button) and click the button RUN – Simulx will run both tasks automatically one by one.

The calculated values are displayed in the **Results** tab and **Plots** tab.

Outcomes

Outcomes represent a post-processing of the simulation outputs done for each individual. They correspond to the measure of interest per individual. Outcomes have different types: **values** (e.g Cmax), **binary true/false** (e.g true if Cmax < threshold), or **time-to-event** (e.g time to NADIR).

To create a new outcome, click on the “plus” button next to Outcomes header or on the “+ New outcome element” within the list of outcomes for an endpoint:



Each **outcome has a name** (default or user defined), which allows to select it in an endpoint definition. To define an outcome you have to **select an output element** for post – processing. You can only use output elements selected in the Simulation scenario. The specification of the outcome includes the following options to post-process the selected output.

For continuous, count and categorical outputs:

Relative to: none/ baseline/ maximal or minimal value/ maximal or minimal value up to current time/ custom value: it divides (“ratio”) or subtracts (“difference”) the output values by the specific value (i.e baseline – the first value of the output). For baseline option, because the first value of the transformed output is non-informative (ratio=1 or difference=0), it is removed.

New outcome

Name

outcome1

Output

TS_fineGrid_25weeks ▾

Relative to: maximal value up to current time ▾ ratio
 difference

Output process:

- none
- baseline (first value of output)
- maximal value
- minimum value
- maximal value up to current time**
- minimal value up to current time
- custom value

Apply the

Add to endpoint: [+ New endpoint task...](#)

CANCEL

OK

Output processing:

- **average value** – takes the average over all output values for each individual
- **first or last value** – takes a value at the first or last time point for each individual
- **min or max value** – takes the minimum or maximum over all output values for each individual. You can choose “*value of min/max*” to take the value of the min or max, or choose “*time of min/max*” to take the time of the min or max – as continuous time or time-to-event.
- **at custom point** – takes a value of an output at a specified time point. Only grid time points are allowed.
- **duration below or above a value or between values** – calculates time when the output values are below, above or between a given values. Strict inequalities (< or >) apply. You input threshold values manually below the output processing menu. As duration you can choose:
 - “cumulative time” – sum of time intervals between consecutive time points where output values satisfy the condition
 - “% of total time” – cumulative time divided by the final time (in percentage)
 - “number of observations” – number of time points where output values satisfy the condition
 - “time of the first occurrence” – a time point where the condition is satisfied for the first time (can be event or continuous)

New outcome

Name

outcome1

Output

noisyCc_2weeks

Relative to none

Output processing: duration between two values per id

- cumulative time
- % of total time
- number of observations
- time of the first occurrence (continuous)
- time of the first occurrence (event)

min 0 max 1

Apply threshold

Add to endpoint: + New endpoint task...

CANCEL

OK

Apply threshold: applies a logical test (usually comparing the outcome value to a threshold) to get a binary true/false outcome. You must set a threshold value and a comparison sign (=, !=, >=, <=, >, <).

New outcome

Name

outcome1

Output

Plasma_concentration

relative to baseline (first value of output) ratio difference

Output processing: max per id value of max time of max

Apply threshold \leq 800

Add to endpoint: + New endpoint task...

CANCEL

OK

For single time-to-event outputs:

- **Has event / has no event / time of event:** *“has event”* and *“has no event”* calculates if the individual had or didn't have an event during the observation period. This is a binary true/false outcome. *“time of event”* uses the TTE output as outcome directly. The outcome is then of type 'time-to-event'.

New outcome

Name
outcome1

Output

Death ▾

has event
 has no event
 time of event

Add to endpoint: + New endpoint task... CANCEL OK

For repeated time-to-event outputs:

- **Has at least one event / number of events per id / time of event #:** *“has at least one event”* leads to a binary true/false outcome depending if the individual has at least one event during the observation period or not. *“number of events per id”* count the total number of events for each individual. This is an outcome of type 'value'. *“Time of event # ”* generates a time-to-event outcome with a single event per individual. The index of the event to be considered can be typed-in by the user.

New outcome

Name
outcome1

Output

EventUntil250 ▾

has at least one event
 number of event(s) per id
 Time of event # ▾

Add to endpoint: + New endpoint task... CANCEL OK

Occasions

In case occasions, you can choose to compute the outcome for each individual, or for each occasion of each individual. Select a corresponding radio button in the outcome definition window

New outcome

Name

outcome1

- calculate for each individual
- calculate for each occasion of each individual

Output

regularCc ▾

Relative to none ▾

Output processing: min per id and per occasion ▾

- value of min
- time of min (continuous)
- time of min (event)

Apply threshold

Add to endpoint: + New endpoint task... ▾

CANCEL

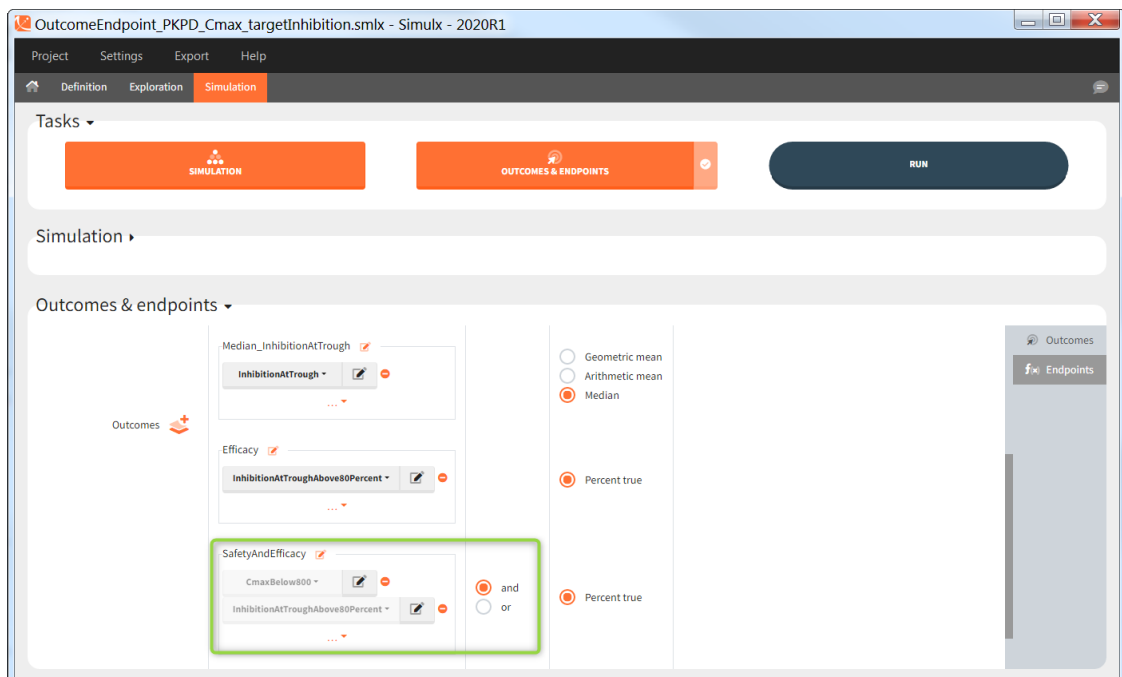
OK

Outcomes organization

Outcomes are building blocks for endpoints. When you create a new outcome, you can **add the outcome to a new or to an existing endpoint via the option “Add to endpoint”** at the bottom of the outcome definition window. The Outcomes & endpoints section displays only the outcomes used in endpoints. All defined outcomes, whether used or not in an endpoint, appear in the **list of outcomes**:

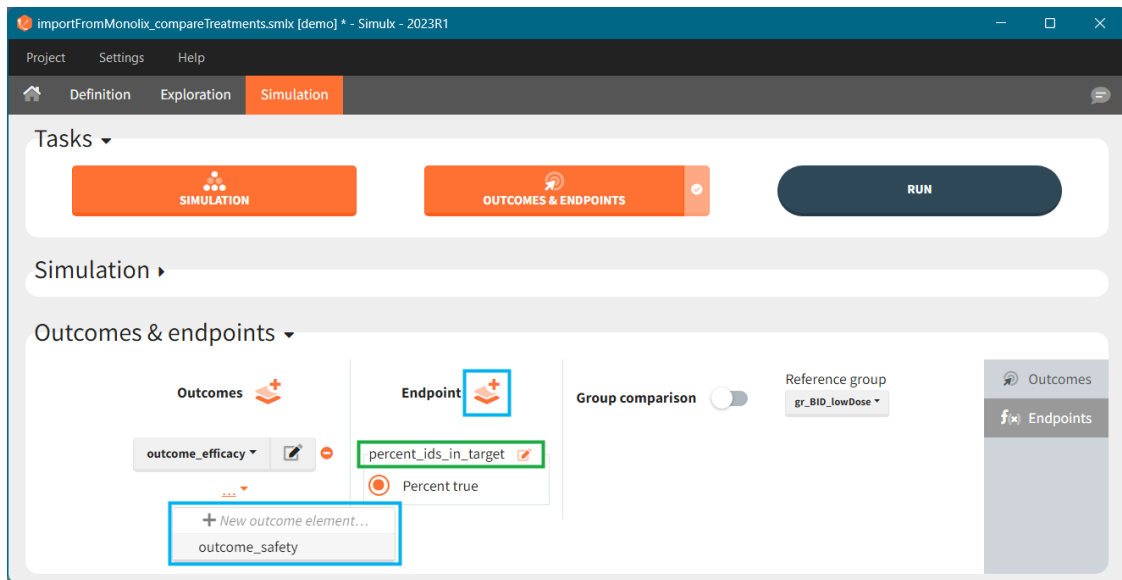
Name	Type	Output	Processing	Action
NADIR	numeric	PSA_measured_after0	min value	
Survival	tte	Death	timeOfEvents	
HasNoDeathEvent	boolean	Death	hasNoEvent	
TimeToNADIR	tte	PSA_measured_after0	min time	

Outcomes of the same type can be **combined together using AND/OR for binary true/false outcomes** (e.g $C_{max} < \text{threshold1}$ and $C_{trough} > \text{threshold2}$), or **MIN/MAX for double values and time-to-event** (e.g $\max(C_{maxParent}, C_{maxMetabolite})$).



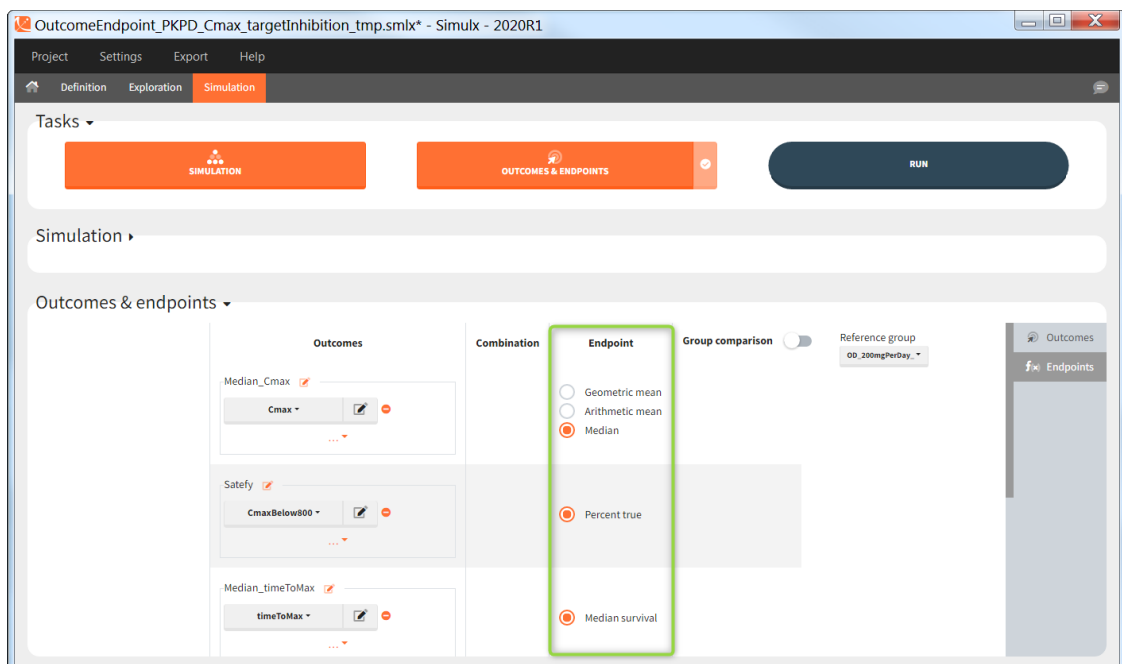
Endpoints

Endpoints summarize the outcome values over all individuals, for each simulation group and each replicate. To create a new endpoint, you can select one or several outcomes of the same type from the drop-down menu (highlighted below in blue) or click on the “+” icon next to the Endpoint header. Endpoints are also created automatically, when at the bottom of an outcome definition you select the option “Add to endpoint: New endpoint”. Once you create an endpoint, you can change its name (highlighted below in green).



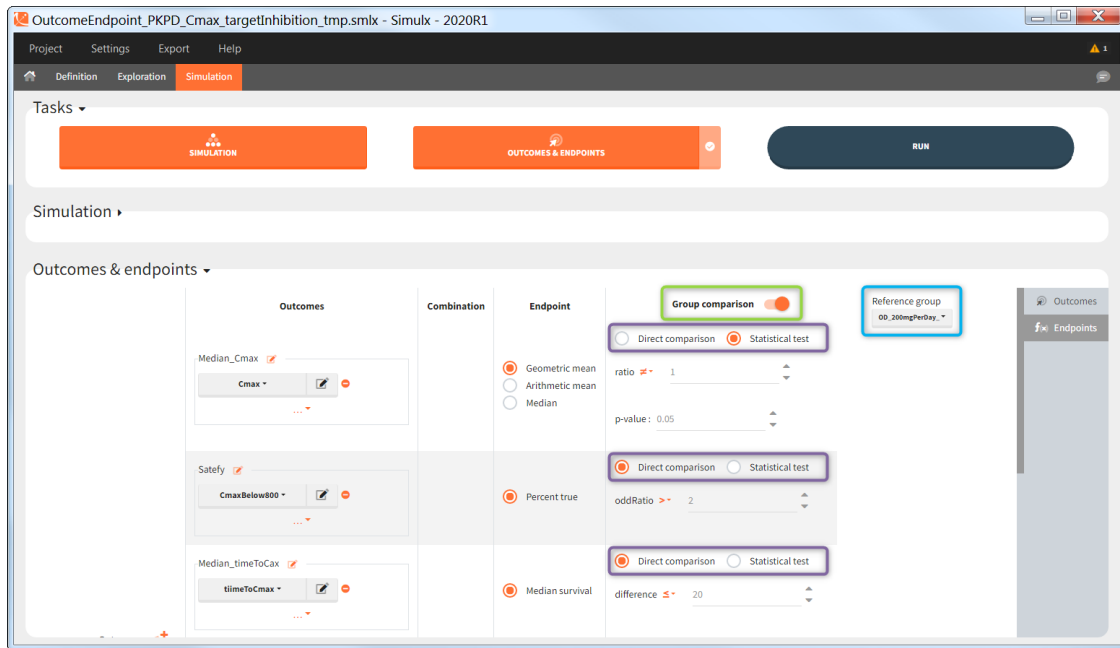
There are several options to summarize the individual outcomes into an endpoint. It depends on the type of outcomes:

- **value outcomes:** the endpoint can be
 - **geometric mean.** Includes calculation of the coefficient of variation.
 - **arithmetic mean.** Includes calculation of the standard deviation.
 - **median.** Includes calculation of the 5th and 95th percentiles.
- **binary true/false outcomes:** the endpoint will be the **percentage of true.** Includes calculation of the “total number true”.
- **time-to-event outcomes:** the endpoint will be the **median survival** (time at which the Kaplan Meier curve is equal to 0.5). Includes calculation of the lower bound (5th) and upper bound (95th) of the confidence interval of the median survival.



Group comparison

You can **compare endpoints across simulation groups** by activating the toggle “Group comparison” (green highlight). One of the simulation groups must be defined as the reference group (blue highlight) and **all other groups will be compared to this reference group**. For each endpoint, the group comparison can rely on a *direct comparison* of the endpoint values, or on a *statistical test* (purple highlight). The exact comparison or test applied depends on the type of the endpoint.



Statistical tests

This table gives an overview of the statistical tests performed in Group comparison depending on the type of outcome and endpoint. Next sections contain detailed information of each option.

Outcome	Endpoint	Metric	Statistical test	
			Same indiv = True	Same indiv = False
Continuous	Geometric mean	Ratio of means	Paired t-test on log-transformed	Unpaired t-test on log-transformed
	Arithmetic mean	Difference of means	Paired t-test	Unpaired t-test

	Median	Difference of medians	Wilcoxon signed rank test	Wilcoxon rank sum test
Binary true/false	Percent true	Odds ratio	McNemar's exact test	Fisher's exact test
Time-to-event	Median survival	Difference in median survival	Logrank test with variance correction	Logrank test

Endpoint “median” for value outcome

This test calculates the **difference between the median of the test group and of the reference group**: $\text{median}_{test} - \text{median}_{ref}$ with median_{test} and median_{ref} the median in the test and reference group respectively.

In case of a **direct comparison**, this difference is compared using operators =, !=, >, >=, <, <= to a user-defined value (default: 0). When this logical comparison is true, the trial simulation is considered as ‘*success*’.

Example: When the direct comparison “difference > 25” is true, it means that the median in the test group is larger than in the reference group by at least 25 units (e.g ng/mL if the outcome is a peak concentration).

In case of a **statistical test**, a **Wilcoxon rank sum test** (equivalent to the Mann-Whitney test) (“Same individuals among groups” not selected) or a **Wilcoxon signed rank test** (“Same individuals among groups” selected) is done to compare the median of the test and reference group. **“difference \neq 0” represents the alternative hypothesis H1**. By default, it performs a two-sided test (sign \neq) and checks if the medians significantly differ from each other. You can define single-sided tests by choosing “>” or “<”. The minimal or maximal (depending of the direction of the test) difference can also be specified (default: 0). The statistical test results into a p-value, which is compared to a user-defined threshold (default: 0.05). If the p-value is below the threshold, the trial simulation is considered as ‘*success*’.

Example: When the statistical test testing the alternative H1 hypothesis “difference > 25” results into a small p-value, it means that the medians of the test and reference groups differ by more than 25 units significantly (e.g ng/mL if the outcome is a peak concentration) with a larger value for test.

Endpoint “arithmetic mean” for value outcome

This test calculates the **difference between the mean of the test group and of the reference group**: $\text{mean}_{test} - \text{mean}_{ref}$ with mean_{test} and mean_{ref} the mean in the test and reference group respectively.

In case of a **direct comparison**, this difference is compared using operators =, !=, >, >=, <, <= to a user-defined value (default: 0). When this logical comparison is true, the trial simulation is considered as 'success'.

Example: When the direct comparison "difference > 14 " is true, it means that the mean in the test group is larger than in the reference group by at least 14 units (e.g ng/mL if the outcome is a peak concentration).

In case of a **statistical test**, it performs an **unpaired t-test** ("Same individuals among groups" not selected) or a **paired t-test** ("Same individuals among groups" selected) to compare the test and reference group. **"difference \neq 0" represents the alternative hypothesis H1**. By default, it performs a two-sided test (sign \neq) and checks if the two means significantly differ from each other. You can define single-sided tests by choosing ">" or "<". The minimal or maximal (depending of the direction of the test) difference can also be specified (default: 0). The statistical test results into a p-value, and compares it to a user-defined threshold (default: 0.05). If the p-value is below the threshold, the trial simulation is considered as 'success'.

Example: When the statistical test testing the alternative H1 hypothesis "difference > 14" results into a small p-value, it means that the means of the test and reference groups differ significantly by more than 14 units (e.g ng/mL if the outcome is a peak concentration) with a larger value for test.

Endpoint "geometric mean" for value outcome

This test calculates the **ratio of the test geometric mean divided by the reference geometric mean**: $\text{geoMean}_{test} / \text{geoMean}_{ref}$ with geoMean_{test} and geoMean_{ref} the geometric mean in the test and reference group respectively.

In case of a **direct comparison**, this ratio is compared using operators =, !=, >, >=, <, <= to a user-defined value (default: 1). When this logical comparison is true, the trial simulation is considered as 'success'.

Example: When the direct comparison "ratio > 2" is true, it means that the geometric mean in the test group is at least twice larger than in the reference group.

In case of a **statistical test**, it performs an **unpaired t-test** (“Same individuals among groups” not selected) or a **paired t-test** (“Same individuals among groups” selected) on the **log-transformed values** (which are assumed to follow a normal distribution) to compare the means of the test and reference group. **“ratio \neq 1” represents the alternative hypothesis H1**. By default, it performs a two-sided test (sign \neq) and checks if the geometric means significantly differ from each other. You can define single-sided tests by choosing “>” or “<”. The minimal or maximal (depending of the direction of the test) ratio can also be specified (default: 1). The statistical test results into a p-value, which is compared to a user-defined threshold (default: 0.05). If the p-value is below the threshold, the trial simulation is considered as ‘*success*’.

Example: When the statistical test testing the alternative H1 hypothesis “ratio > 2” results into a small p-value, it means that the geometric mean of the test group is significantly larger than twice the geometric mean of the reference group.

Endpoint “percent true” for binary true/false outcome

This test calculates the **odds ratio between the test group and the reference group**. The odds ratio definition is different depending if you use paired samples case or not.

“Same individuals among groups” not selected (unpaired samples)

The odd ratio is $\frac{p_{Test}}{1-p_{Test}} / \frac{p_{Ref}}{1-p_{Ref}}$ with p_{Test} and p_{Ref} the fraction of true outcomes in the test and reference group respectively. $1 - p_{Test}$ represents the fraction of false outcomes.

Equivalently, you can use the following contingency table to define the odd ratio.

UNPAIRED			
	Number of individuals		
	TRUE	FALSE	total
Treatment test	n11=95	n10=5	100
Treatment ref	n01=87	n00=13	100
$oddRatio=(n11/n10) / (n01/n00) = (95/5) / (87/13) = 2.83$			

“Same individuals among groups” selected (paired samples)

In case of identical individuals among groups, the individuals which have the same outcome value in both groups (so Ref=True and Test=True, or Ref=False and Test=False) are not counted. The odd ratio is defined as $\frac{n_{Test_T Ref_F}}{n_{Test_F Ref_T}}$, with $n_{Test_T Ref_F}$ the number of individuals with true in the test group and and false in the reference group. It corresponds to the contingency table below. The odds ratio can frequently be zero or infinity – in particular in the absence of measurement noise – an individual true in the reference group is also necessarily true in the test group (corresponding for instance to a higher dose).

PAIRED				
		Treatment Ref		
		TRUE	FALSE	total
Treatment test	TRUE	m11=85	m10=10	95
	FALSE	m01=2	m00=3	5
total		87	13	100
oddRatio=m10/m01 = 10 / 2 = 5				

In case of a **direct comparison**, this odds ratio is compared using operators =, !=, >, >=, <, <= to a user-defined value (default: 1). When this logical comparison is true, the trial simulation is considered as 'success'.

Example: When the direct comparison "odds ratio > 2" is true, it means that the odds in the test group are at least twice larger than in the reference group.

In case of a **statistical test**, it performs a **Fisher's exact test** ("Same individuals among groups" not selected) or a **McNemar's exact test** ("Same individuals among groups" selected) to compare the results of the test and reference group via the construction of a 2x2 contingency table (which contain more information than the endpoints p_{Test} and p_{Ref}). "**odds ratio \neq 1**" represents the **alternative hypothesis H1**. By default, it performs a two-sided test (sign \neq) and checks if the odds ratio significantly differs from 1. You can define a single-sided tests by choosing ">" or "<". The minimal or maximal (depending of the direction of the test) odds ratio can also be specified (default: 1). The statistical test results into a p-value, and compare it to a user-defined threshold (default: 0.05). If the p-value is below the threshold, the trial simulation is considered as 'success'.

Example: When the statistical test testing the alternative H1 hypothesis "odds ratio > 2" results into a small p-value, it means that the odds of the test group are significantly larger than twice the odds of the reference group.

Endpoint "median survival" for time-to-event outcome

This test calculates the **difference between the median survival of the test group and of the reference group**: $\text{medSurv}_{test} - \text{medSurv}_{ref}$ with medSurv_{test} and medSurv_{ref} the median survival (time at which the Kaplan-Meier estimates equals 0.5) in the test and reference group respectively.

In case of a **direct comparison**, this difference is compared using operators =, !=, >, >=, <, <= to a user-defined value (default: 1). When this logical comparison is true, the trial simulation is considered as 'success'.

Example: Direct comparison "difference > 60" corresponds to median survival in the test group larger than in the reference group by 60 time units (e.g days).

In case of a **statistical test**, it performs a **logrank test** to compare the survival Kaplan-Meier curves. Selecting "Same individuals among groups", applies a variance correction (see Jung 1999). "**difference \neq 0**" represents the **alternative hypothesis H1**. By default, it performs a two-sided test (sign \neq) and checks if the survival curves significantly differ. You can define single-sided tests by choosing ">" or "<". For the log rank test, it is not possible to define a "minimal difference". The statistical test gives a p-value, and compare it to a user-defined threshold (default: 0.05). If the p-value is below the threshold, the trial simulation is considered as 'success'.

Example: When the statistical test testing the alternative H1 hypothesis "difference \neq 0" results into a small p-value, it means that the survival curves from the two groups differ significantly.

4.3. Plots

Running the tasks in Simulx **automatically generates interactive plots** for simulation outputs, outcomes and endpoints. Interface provides many plots features such as:

- Different chart types depending on simulation outputs and post-processing methods: individual outputs or distributions.
- Plots settings: display, binning criteria, axes, visual cues.
- Stratification and customization options.
- Export: plots as images, charts data and charts settings.

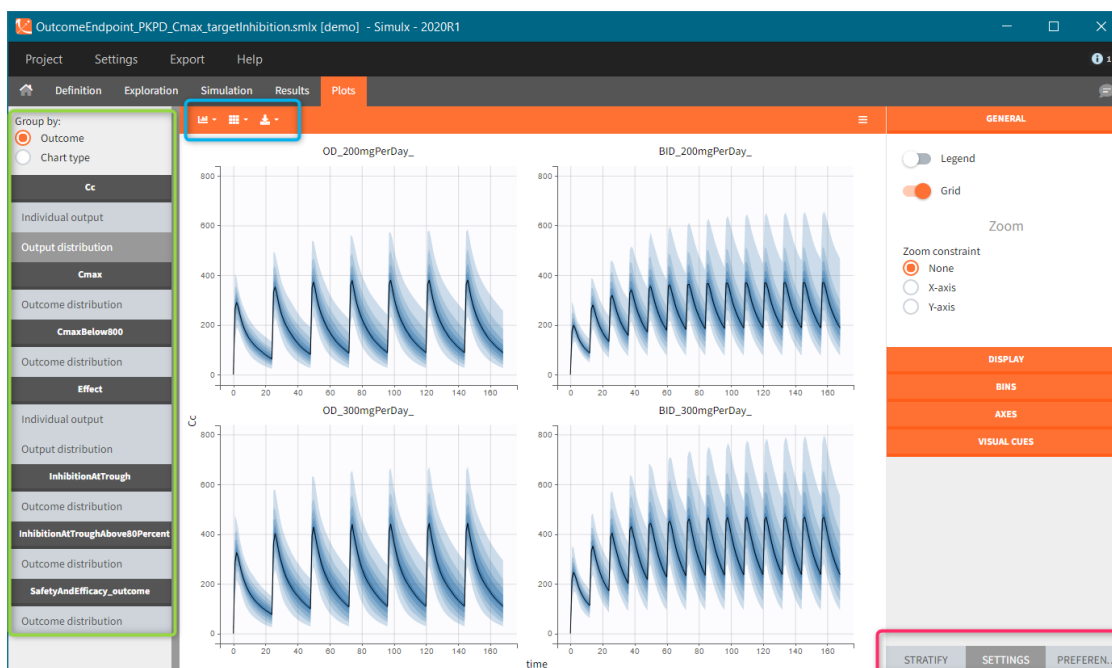
To learn more about plots in the exploration tab, click [here](#) – it is a special documentation page for the model Exploration feature.

-
- [The Plots tab](#)
 - [Continuous data](#)

- Individual output
- Output distribution
- **Non-continuous data**
 - Time-to-event
 - Discrete data
 - Individual output
 - Output distribution
- **Outcomes and endpoints**
 - Numeric outcome distribution
 - Binary outcome distribution
 - Endpoints distribution
- **Plots features**
 - Settings
 - Stratify
 - Preferences

The Plots tab

Plots tab structure and settings in Simulx is the same as in **Monolix** and **Pkanalix**. The left panel (green frame) contains the list of available plots grouped by outcome type or by chart type. Central region displays one selected plot, for instance output distribution of Cc. Each simulation group has its own subplot. On top of the plotting area, there are layout options and a button to export current plot as an image (blue frame). The right panel is for plots settings, such as display or axis options, definition of bins or visual cues. In addition, separate sub-tabs at the bottom of this panel (red frame) allow to stratify plots data and change preferences of the plotting area.



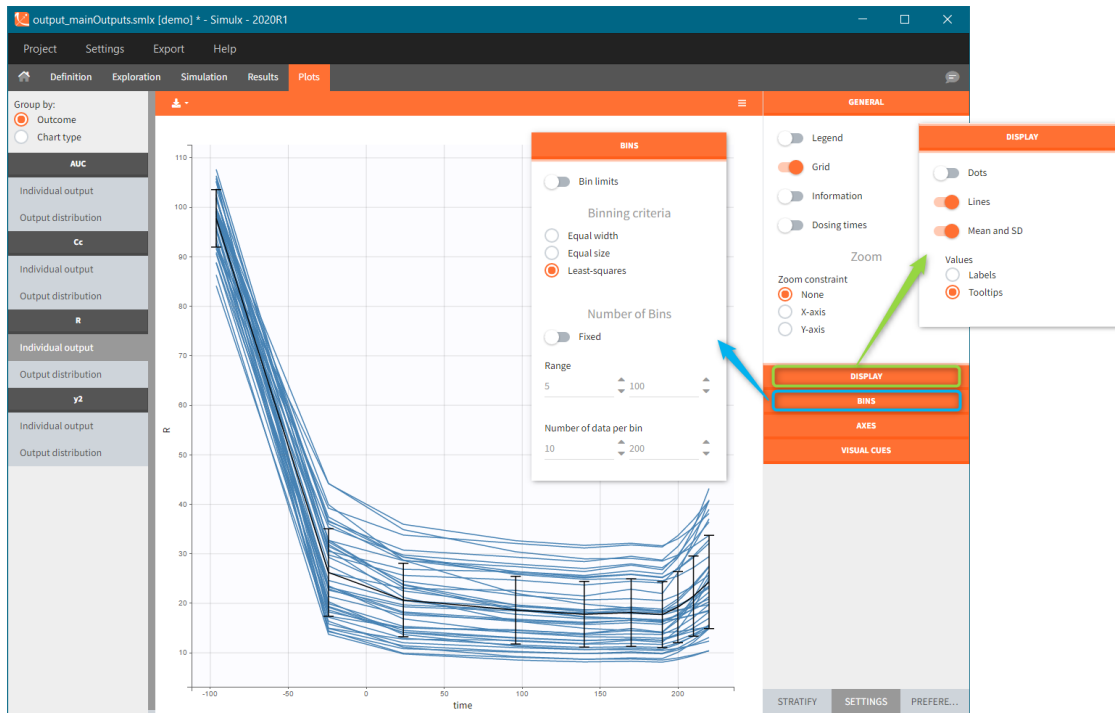
- **Simulation outputs.** Only output elements used in a simulation scenario generate plots. If more than one simulation output element uses the same model output, then all these simulation outputs are merged on one same plot. For instance, concentration during the whole treatment period on a fine grid and C_{trough} on the last output treatment day – both using model prediction C_c. Title name of the plot corresponds to the model output name.
- **Outcomes.** Outcomes of different endpoints have separate plots, for instance “C_{max}” and “C_{max}Below800”. If there is more than one outcome in an endpoint, then a plot displays the combination of them. Moreover, it appears in the plots list as “<endpointName>_outcome”, for instance “SafetyAndEfficacy_outcome”.
- **Endpoints** distribution is generated only for simulations with replicates.

Plots for continuous data

There are two types of plots for continuous outputs. Firstly, the individual output subplot, which contains individual curves. Secondly, the output distribution with the percentiles. Title of each outcome section in the plot list corresponds to the name of a model variable, for instance C_c for model predictions or y₁ for model observations. It is not the name of the output element selected in the simulation tab.

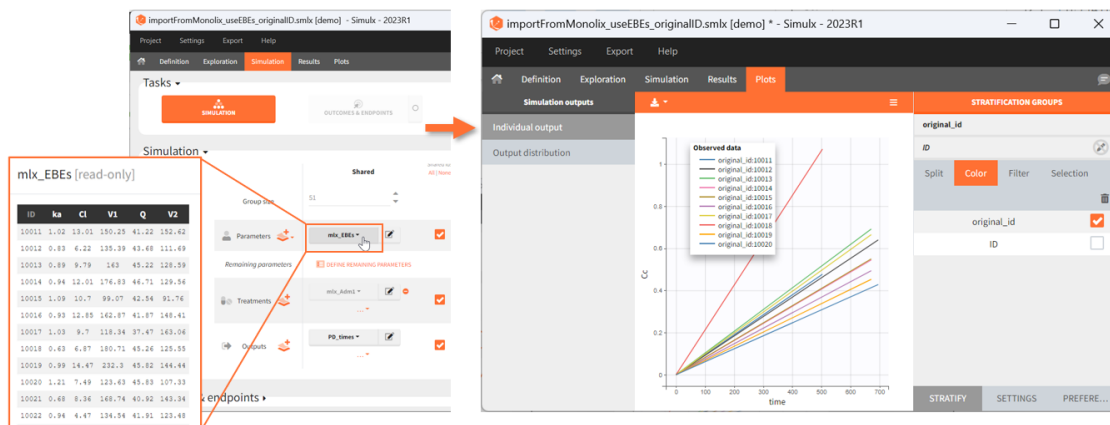
Individual output

Individual output plots represent individual curves computed for each id and each occasion (if present in the model). It is possible to DISPLAY lines, dots and median with standard deviations (green frame), or a combination of them. The statistics (median and standard deviations) use the binning options available in the BINS section of the settings (blue frame). If simulation uses replicates, then by default a plot shows only one replicate. However, you can select other replicates – one or several – in the DISPLAY sub-tab.



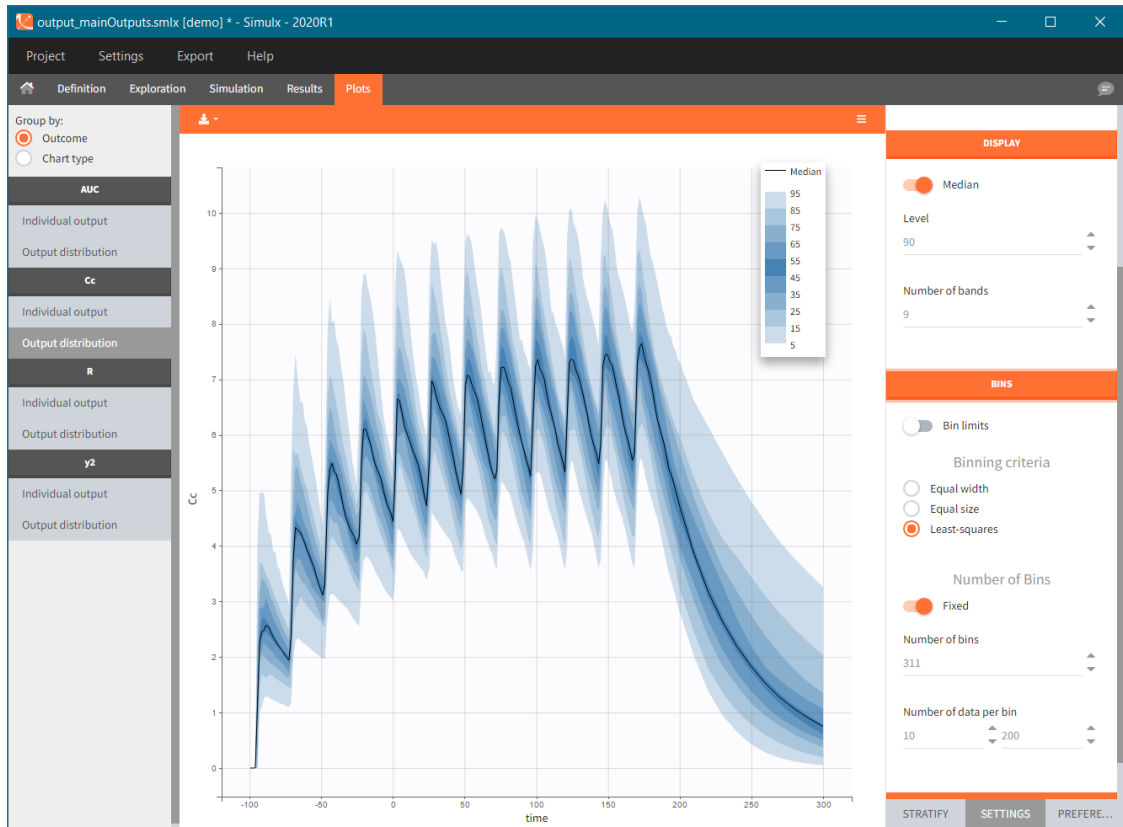
The ID used in plots stratification is the simulated ID, which is the id column in the result tables. ID i is the i-st parameter set sampled for simulation.

Since the version 2023R1, if a simulation uses elements with custom id lists, for example a table of individual treatments, or a table of individual parameters, the original ids in these tables are available to stratify the plots. For example in the demo project “importFromMonolix_useEBEs_originalID.smlx”, we use the element mlx_EBEs which is a table of individual parameters. The original IDs used in this table can be used to color or split the individual output trajectories.

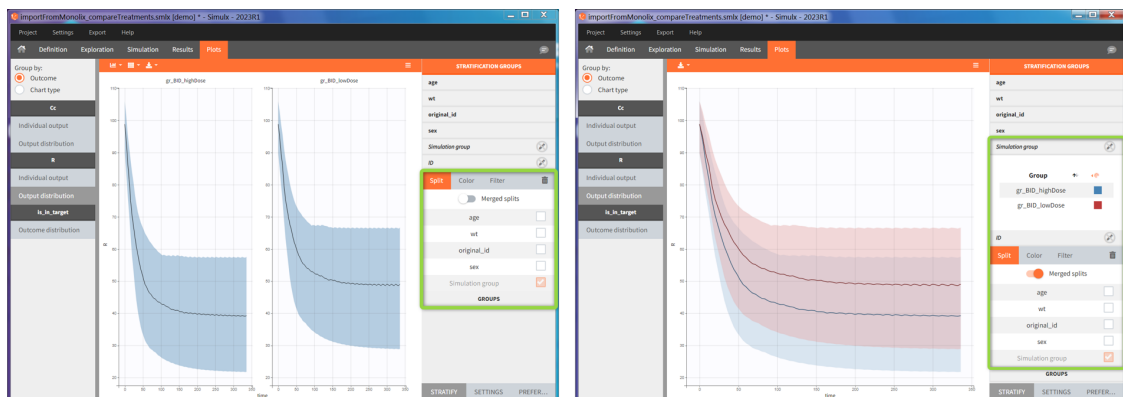


Output distribution

Output distribution plots show percentiles of the simulated data for all ids and all replicates computed. The BINS section of the settings contains the binning options. DISPLAY settings allow to change plot features, such as median, number of percentiles bands and levels.



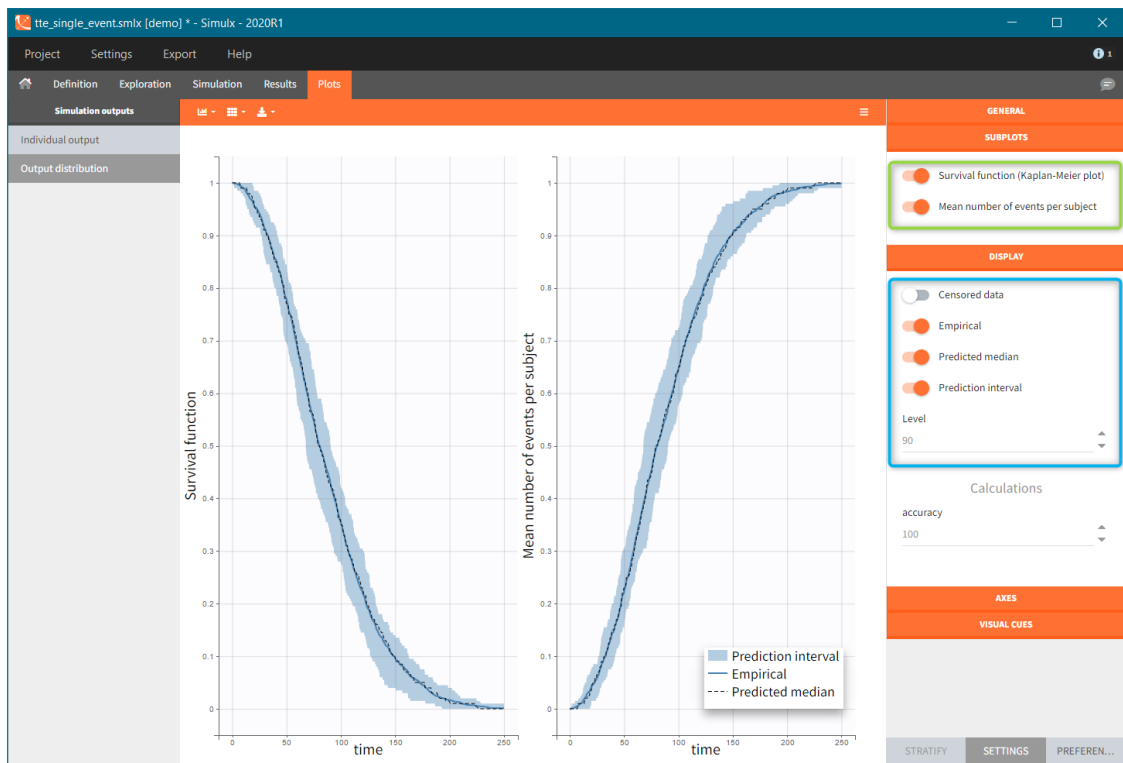
When the “Output distribution” plot is split (by simulation group or by a covariate), it is possible to overlay the prediction intervals on a single plot with different colors using the “merged splits” toggle. You can choose the colors in the “stratification groups”. This feature is available starting from the Simulx 2023 version.



Plots for non-continuous data

Time-to-event

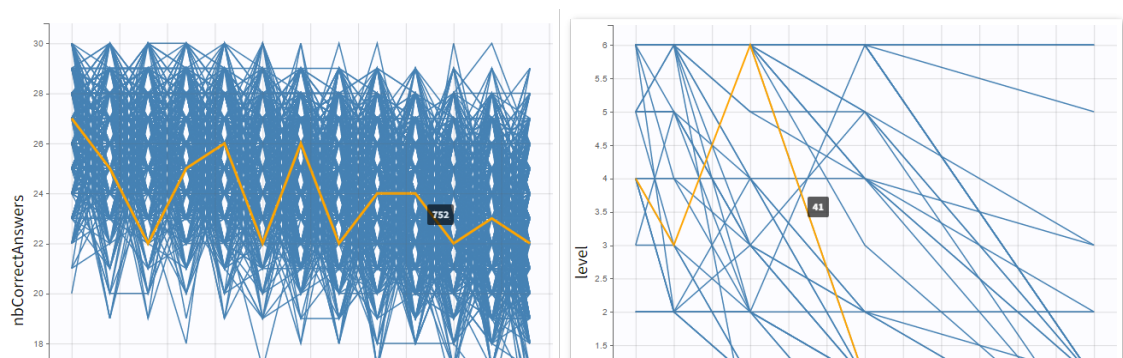
Individual output of time-to-event data is a single Kaplan-Meier curve – estimator of the survival function. In addition, for simulations with replicates, the Kaplan-Meier curves for each replicate are interpolated on one grid and displayed as a prediction interval at a specific level (blue frame). Mean number of subjects plot is available in the SUBPLOTS section (green frame). Click [here](#) for more information about time-to-event data plots.

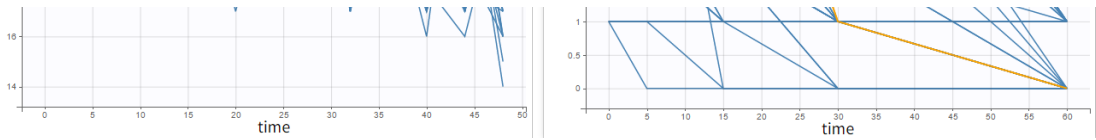


Discrete data (categorical and count)

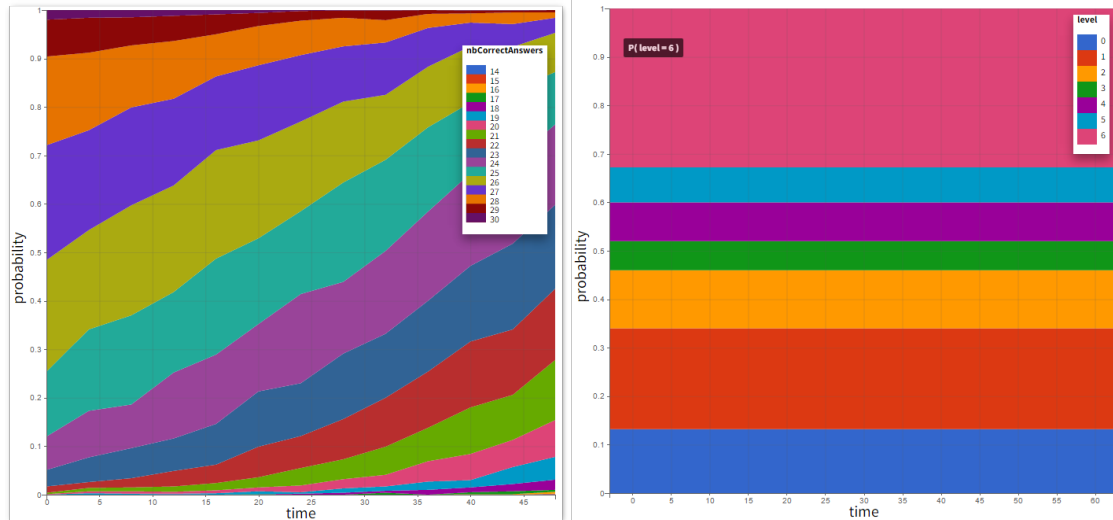
Individual outputs are functions of time for each individual and occasion.

- **Count data:** plots show the number of observed events in a specific period, for instance “number of good answers” (on the left).
- **Categorical data:** plots show nominal categories, for instance “respiratory status level” (on the right).





Output distribution shows the time evolution of probabilities of different categories. In case of count data, each number of “counts” constitutes a separate category.



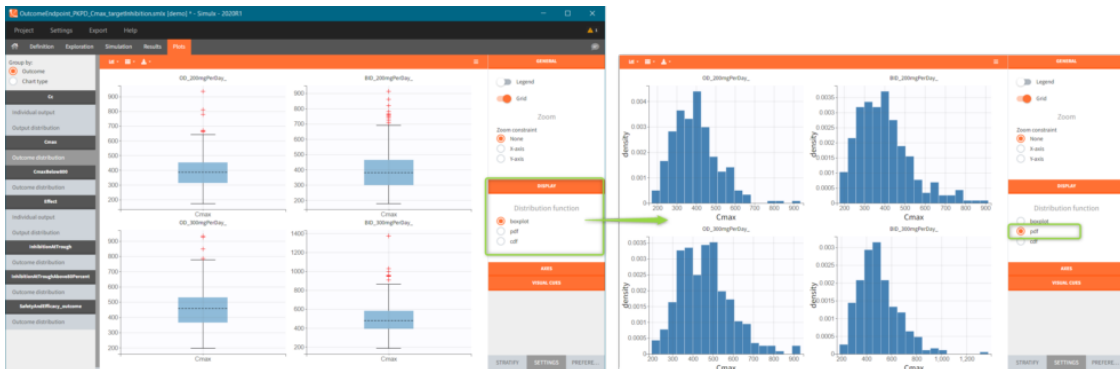
Plots for outcomes and endpoints

- Outcomes represent a post-processing of the simulation outputs done for each individual, so plots display their distribution within a population. Simulx displays only outcomes used in endpoints (Outcome&endpoint section in the simulation tab). If there are several outcomes in one endpoint, then the outcome distribution corresponds to of a combination of these outcomes, and not to separate outcomes.
- Depending on a type of an outcome, plots are histograms or box plots for numeric outcomes, and stacked histograms for binary outcomes.
- Endpoints summarize the outcome values over all individuals. In simulation with replicates, Simulx displays them as box plots.
- In all boxplots, the dashed line represents the median, the blue box represents the 25th and 75th percentiles (Q1 and Q3), and the whiskers extend to the most extreme data points, outliers excluded. Outliers are the points larger than $Q1 + w*(Q3 - Q1)$ or smaller than $Q1 - w*(Q3 - Q1)$ with $w=1.5$. Outliers are shown with red crosses.

[Demo: 6.1.outcome_endpoint/OutcomeEndpoint_PKPD_Cmax_targetinhibition]

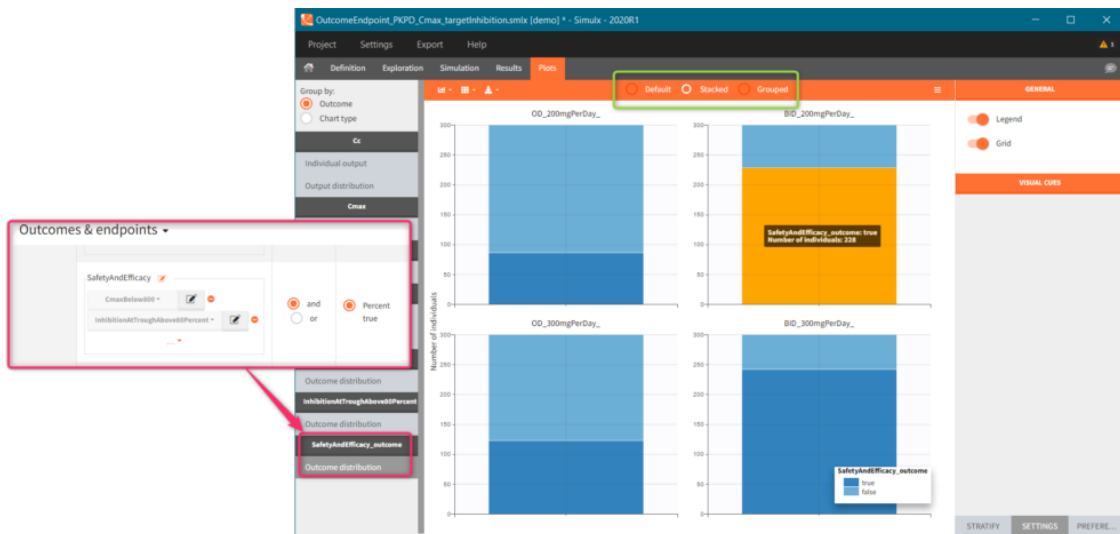
Numeric outcome distribution:

“Cmax” outcome gives for each individual the maximum of the output element “Plasma concentration”. In this case, the outcome distribution can be a box-plot or as a histogram (selection in the DISPLAY settings). Each group is on a separate subplot. This outcome is the only outcome in the endpoint, so the title name of the plot in the list coincides with the outcome name.



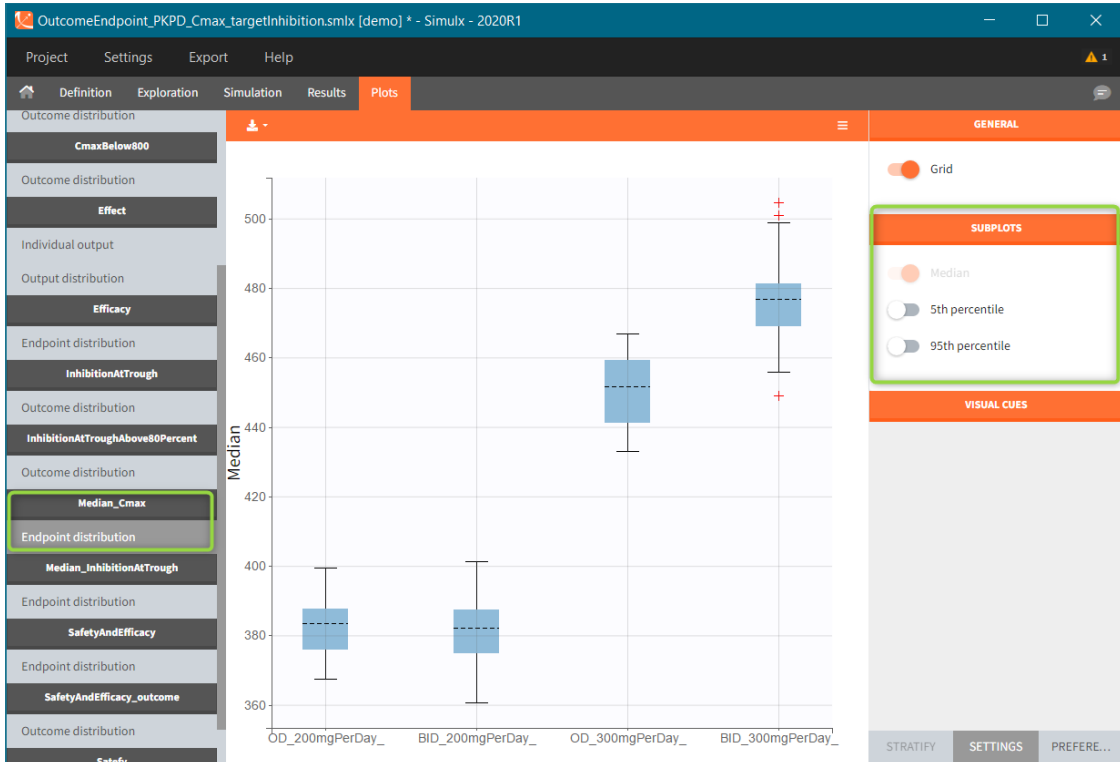
Binary outcome distribution

“CmaxBelow800” outcome is a of boolean type (true/false) and compares the maximal value of the “Plasma concentration” with a threshold of 800. If Cmax is below this value, then the outcome is true, otherwise it is false. Similarly, “inhibitionAtTroughAbove80Percent” outcome compares the average of the values in the output element “TargetInhibition_at168h” with a threshold of 0.8 (i.e 80%). These two outcomes belong to one endpoint “SafetyAndEfficacy”, so the plot shows the distribution of true/false values corresponding to the combination of the two conditions. In this case, the title name of the outcome plot is the name of the endpoint “_outcome” (red frame). Binary outcomes use stacked or grouped histograms (green frame). Hovering on a any part of the histogram highlights it, shows the outcome value and the corresponding number of individuals.

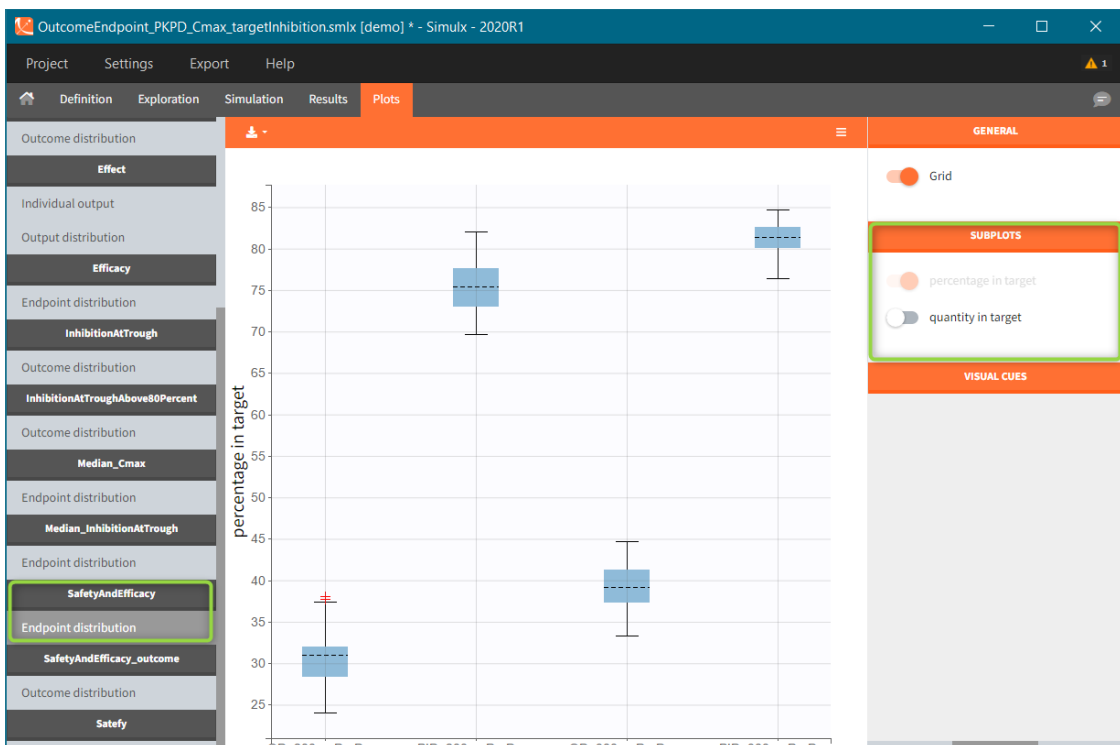


Endpoint distribution

If this simulation scenario is run several times – option “replicates” available in the Simulation section – then the endpoint distribution plots are added to the plots list. Endpoint median_Cmax is a a box-plot with median value and standard errors for groups listed on the x-axis. For endpoints using numerical outcomes, uncertainty of the 5th and 95th percentiles can be added from the settings – SUBPLOTS section (green frame).




Endpoints using binary outcomes, in this example the SafetyAndEfficacy endpoint, calculate the percentage of individuals with “true” outcomes values. As previously, the endpoint distribution is a box-plots with different groups on the x-axis. A subplot with the number of individuals is available from the settings.



Plots features

The right panel of the Plots tab has several sub-tabs – at the bottom of the interface window – to interact and customize the charts:

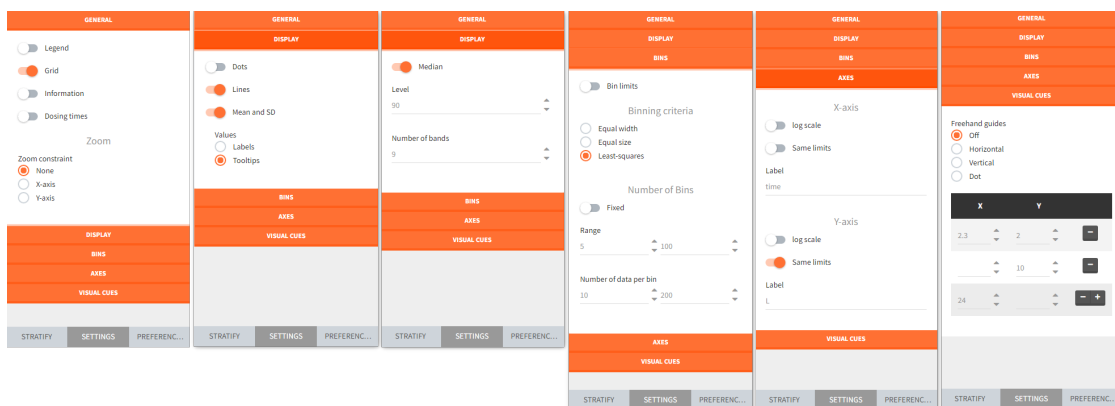
- “Settings” modify plots elements, such as: hide or display additional data, adjust binning criteria, change axis or add visual cues
- “Stratify” splits, filters or colors data points using covariates.
- “Preferences” customize graphical aspects of plots such as colors, font sizes, line width or margins.

Icons at the left-top angle of the plotting area –  – allow to select sub-plots for visualization, change the sub-plots layout and export the plots as a png or svg file (saved in the Chart folder in the Results). In addition, using the “Export” in the top menu saves all generated plots (Export plots), saves all data used to generate plots as txt files to re-plot them outside Simulx (Export charts data, see [here](#) for more information) and save charts settings so that the selected customization is applied to all Simulx projects (Export charts settings as default).

Simulx has the same architecture of plots features, preferences and export options as Monolix. [Here](#) is an **extensive documentation** about it, with links to **Features of the week videos** that explain how to use customization in MonolixSuite applications.

Settings

Settings sub tab has several sections, which may differ between plot.



- **General** settings include toggle buttons add legend, data information, dosing etc. Zoom is interactive by selecting with a cursor a region directly on a plot.
- **Display** section for “individual output” plots has toggle buttons to add lines, data dots and Mean and SD (at least one option needs to be selected). For “output distribution” it customizes the level of percentiles and the number of bands shown.
- **Bins** serve to specify the criteria for computing the mean&SD for the “individual

outputs” and percentiles for the “output distribution”. There are several strategies to segment the data: equal width, equal size and the least squares criterion. In addition, number of bins, range and number of data per bin are changeable.

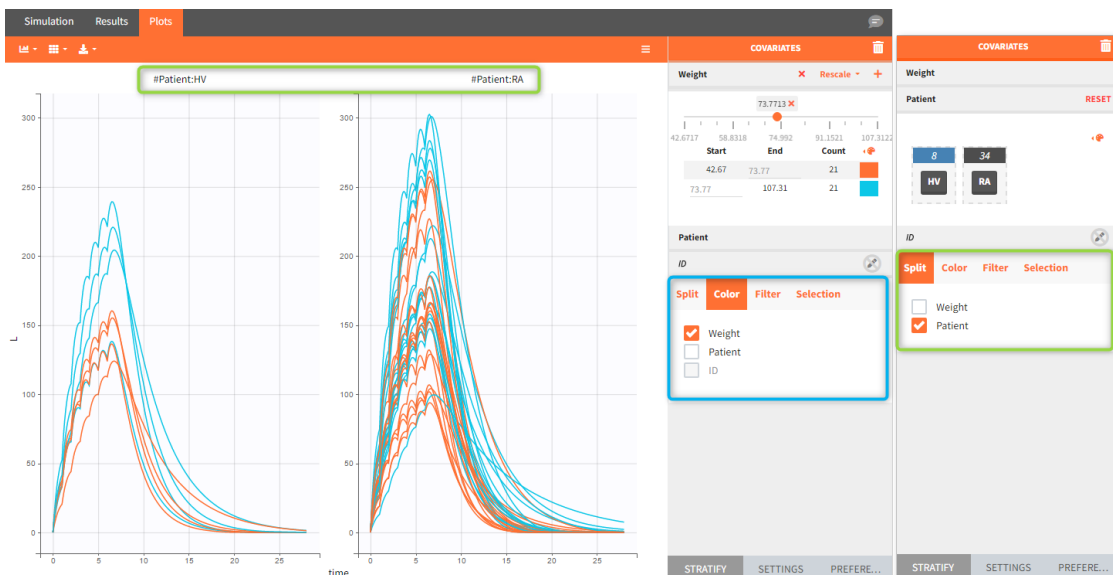
- **Axes** is for general X and Y axis settings, such as log scale, same limits across plots and main axis titles.
- **Visual cues** freehand guides option allow to place additional lines or points on plots directly with a cursor. Definition of a specific location is via (X,Y) coordinates – one entry filled gives a line, two filled entries specify a point.

Stratify

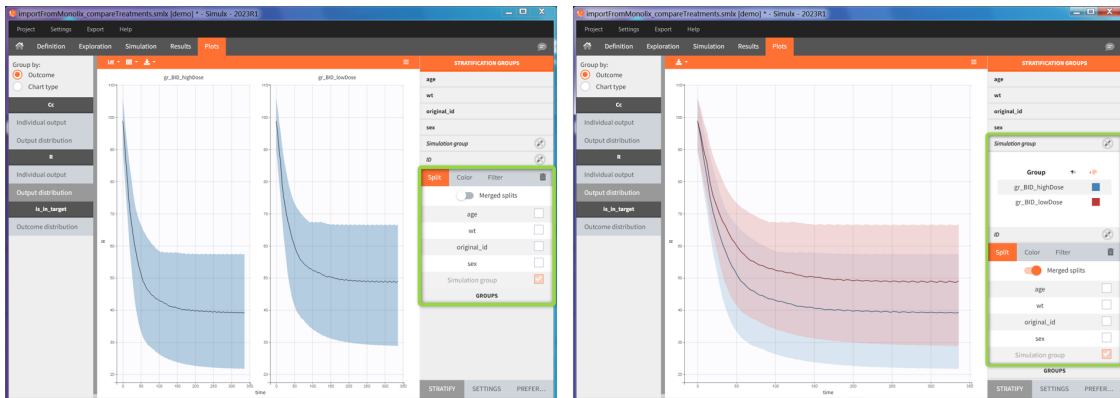
Stratification panel contains a list of covariates that can split, color or filter the data. Simulx automatically allocates covariates in groups – according to modalities for categorical covariates and in two “equal count” groups for continuous covariates. As a result stratification is immediately applied after ticking appropriate boxes.

Example:

- Color data according to two weight groups (blue frame)
- Split data between HV (healthy volunteers) and RA (rheumatoid arthritis) patients (green frame)



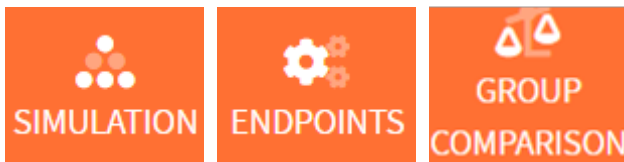
When the “Output distribution” plot is split (by simulation group or by a covariate), it is possible to overlay the prediction intervals on a single plot with different colors using the “merged splits” toggle. The colors can be chosen in the “stratification groups”. This feature is available starting from the 2023 version.



4.4. Results

The results tab displays the simulated values and the outcomes and endpoints as tables.

Result tables are organized into three sections:



- Simulation
 - Outputs
 - Individual Parameters Summary
 - Individual Parameters Table
- Endpoints
 - Outcomes Summary
 - Outcomes Table
 - Endpoints Summary
 - Endpoints Table
- Group comparison
 - Group Comparison Table
 - Group Comparison Summary
- Export simulations

- as output files at each run
- as a Monolix or PKanalix project
- as a formatted dataset
- export Simulx project as .zip
- Display Options

Simulation

The simulation tab displays the simulated output values and the sampled individual parameters.

Outputs

Because the simulated output values are usually a very large table, the results are presented as percentiles for continuous simulations (minimum, 5th percentile, median, 95th percentile and maximum) and as percentage of survival and average number of events for time-to-event simulations. These statistics are calculated over time bins calculated automatically. When replicates are used, the statistics are calculated over all replicates merged together. Each simulation group is displayed as a separate table.

The screenshot shows the 'Results' tab in the Simulx software. It displays simulation outputs for three dosing regimens: OD_200mgPerDay, BID_200mgPerDay, and OD_300mgPerDay. Each regimen has a table of statistics (TIME, MIN, P05, MEDIAN, P95, MAX) for various time points. The interface includes a sidebar with navigation options like SIMULATION, ENDPOINTS, and GROUP COMPARISON, and a right-hand panel for display settings like Pagnation and Group by.

TIME	MIN	P05	MEDIAN	P95	MAX
0.5	0	0	163.26	364.26	524.67
2	101.45	189.2	286.46	395.85	519.32
3.75	111.67	168.6	247.91	332.9	438.33
5.5	80.84	124.85	205.54	285.6	377.6
7.25	56.72	106.64	174.95	255.89	349.61
9	45.23	87.47	153.14	230.33	321.07
10.5	36.84	75.15	137.98	216.19	302.75
12.25	28.59	62.71	123.91	197.92	288.67
14	23.85	53.92	111.93	183.86	270.9
15.5	19.97	46.7	103.04	172.84	258.82

Individual parameters [Summary]

This tab displays the percentiles summarizing the distribution of individual parameters used for simulation for each of the groups. These parameters have been either directly defined by the user or sampled from the population parameters defined by the user. As for the simulated outputs, the statistics are calculated over all replicates merged together.

OutcomeEndpoint_PKPD_Cmax_targetInhibition.smk [demo] * - Simulx - 2020R1

Project Settings Export Help

Definition Exploration Simulation Results Plots

OUTPUS INDIV. PARAM. (SUMMARY) INDIV. PARAM. (TABLE)

SIMULATION

ENDPOINTS

GROUP COMPARISON

Summary

Display 4 items selected

OD_200mgPerDay_						BID_200mgPerDay_						OD_300mgPerDay_					
	MIN	Q1	MEDIAN	Q3	MAX		MIN	Q1	MEDIAN	Q3	MAX		MIN	Q1	MEDIAN	Q3	MAX
ka	0.2	0.34	0.38	0.44	0.68	ka	0.21	0.34	0.38	0.43	0.71	ka	0.23	0.34	0.38	0.44	0.73
Cl	5.26	13.97	18.11	23.79	54.31	Cl	5.72	13.66	17.82	22.82	55.9	Cl	5.7	13.33	18.24	24.12	57.74
V1	9.08	43.42	63.5	88.28	271.41	V1	14.43	43.29	62.61	90.76	277.52	V1	15.38	47.06	65.13	95.59	333.69
Q	48.1	48.1	48.1	48.1	48.1	Q	48.1	48.1	48.1	48.1	48.1	Q	48.1	48.1	48.1	48.1	48.1
V2	111.49	160.2	204.23	236.73	369.04	V2	109.12	177.01	205.74	238.46	370.35	V2	110.23	179.15	206.4	239.03	401.8
IC50	32	32	32	32	32	IC50	32	32	32	32	32	IC50	32	32	32	32	32
D50	95.79	137.18	150.64	164.13	209.4	D50	102.3	137.28	150.26	164.59	221.5	D50	101.37	136.65	149.44	164.24	230.85

BID_300mgPerDay_					
	MIN	Q1	MEDIAN	Q3	MAX
ka	0.2	0.33	0.37	0.43	0.65
Cl	5.13	13.9	17.68	23.46	63.67
V1	12.39	43.41	63.83	92.1	470.83
Q	48.1	48.1	48.1	48.1	48.1
V2	105.9	175.4	202.26	230.52	466.47
IC50	32	32	32	32	32
D50	92.37	138.62	151.65	164.55	219.27

Individual parameters [Table]

This tab displays the full tables of individual parameters for all replicates and for all groups. The content of these tables can also be found in a table gathering all groups in the result folder>Simulation>individualParameters.txt .

OutcomeEndpoint_PKPD_Cmax_targetInhibition.smk [demo] * - Simulx - 2020R1

Project Settings Export Help

Definition Exploration Simulation Results Plots

OUTPUS INDIV. PARAM. (SUMMARY) INDIV. PARAM. (TABLE)

SIMULATION

ENDPOINTS

GROUP COMPARISON

Individual parameters

Display

Pagination

COLLAPSE ALL EXPAND ALL

4 items selected

Sampled individual parameters and covariates

OD_200mgPerDay_											BID_200mgPerDay_										
rep	id	ka	Cl	V1	Q	V2	IC50	D50	rep	id	ka	Cl	V1	Q	V2	IC50	D50				
1	1	0.44	19.56	30.84	48.1	211.59	32	147.88	1	301	0.45	13.91	40.98	48.1	166.53	32	148.14				
1	2	0.44	24.22	33.49	48.1	222.2	32	121.95	1	302	0.4	13.01	94.64	48.1	195.51	32	140.96				
1	3	0.29	16.11	29.33	48.1	218.85	32	155.26	1	303	0.41	9.75	56.37	48.1	270.19	32	110.94				
1	4	0.4	33.59	59.84	48.1	161.43	32	125.34	1	304	0.35	18.28	63.16	48.1	183.14	32	132.65				
1	5	0.42	14.79	63.21	48.1	168.24	32	158.57	1	305	0.37	17.65	54.6	48.1	205.83	32	148.16				
1	6	0.29	18.27	9.08	48.1	200.12	32	139.28	1	306	0.45	9.9	31.78	48.1	163.04	32	135.36				
1	7	0.48	19.77	80.04	48.1	230.94	32	169.85	1	307	0.38	19.48	73.31	48.1	178.39	32	123.44				
1	8	0.29	19.89	61.31	48.1	282.1	32	127.93	1	308	0.57	26.68	144.14	48.1	249.84	32	168.09				
1	9	0.41	6.76	32.17	48.1	290.4	32	149.64	1	309	0.42	15.63	61.43	48.1	273.8	32	134.69				
1	10	0.31	12.93	84.06	48.1	308.2	32	163.63	1	310	0.4	18.56	76.73	48.1	195.28	32	152.32				

Records per page: 10 25 50 100 500 600

Records per page: 10 25 50 100 500 600

Page 1 of 60

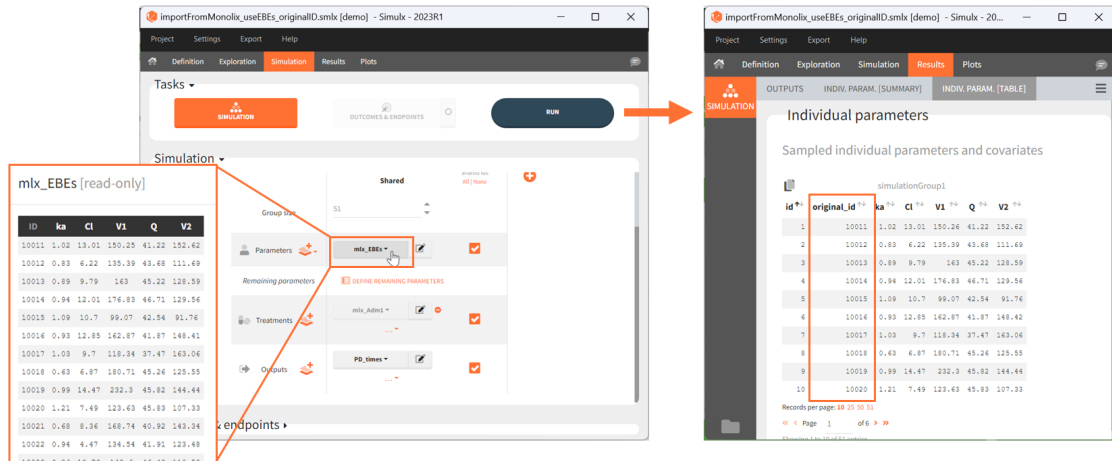
Showing 1 to 10 of 600 entries

Showing 1 to 10 of 600 entries

OD_300mgPerDay_											BID_300mgPerDay_										
rep	id	ka	Cl	V1	Q	V2	IC50	D50	rep	id	ka	Cl	V1	Q	V2	IC50	D50				
1	601	0.38	10.17	51.25	48.1	233.9	32	176.06	1	901	0.33	27.1	26.63	48.1	243.15	32	116.68				
1	602	0.47	11.97	76.71	48.1	329.06	32	185.64	1	902	0.42	24.29	71.22	48.1	191.87	32	134.47				
1	603	0.52	20.1	24.65	48.1	165.94	32	143.78	1	903	0.38	7.64	45.27	48.1	235.49	32	150.06				

The id column in these tables is the simulated id, meaning that id i is the i-st parameter set sampled for simulation.

Since the version 2023R1, if elements using original id lists have been used for simulation, for example a table of individual treatments, or a table of individual parameters, the original ids in these tables is reported in the individual parameter tables in the “original_id” column. For example in the demo project “importFromMonolix_useEBEs_originalID.smlx”, we use the element mlx_EBEs which is a table of individual parameters. The original IDs used in this table are reported in the sampled individual parameters table, together with the simulated id. The original id is also reported in the output files.



If a simulated id uses information coming from several original ids (e.g when a table a individual parameters and a table of treatment is used by the “shared ids” option is not selected), then the original id is not displayed in the GUI but it can be found in the result folder in the file *ID_mapping.txt*.

<https://www.youtube.com/watch?v=z8XB-2G5-Sw>

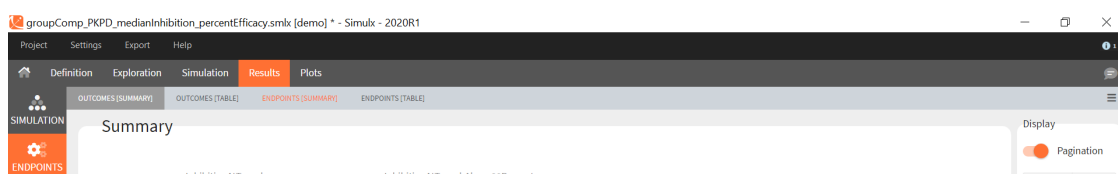


Endpoints

The endpoints section appears if an **outcome** has been defined in the simulation tab, to post-process the simulation results.

Outcomes [Summary]

The summary shows one table per outcome, displaying the percentiles over individuals and replicates for each group if the outcome is continuous, and the number of true and false individuals if the outcome is a boolean.



	MIN	P05	MEDIAN	P95	MAX		TRUE	FALSE
OD_200mgPerDay_	0.068	0.44	0.74	0.89	0.95	OD_200mgPerDay_	930	2070
BID_200mgPerDay_	0.37	0.69	0.86	0.93	0.97	BID_200mgPerDay_	2313	497
OD_300mgPerDay_	0.16	0.49	0.77	0.9	0.96	OD_300mgPerDay_	1205	1795
BID_300mgPerDay_	0.42	0.73	0.88	0.94	0.97	BID_300mgPerDay_	2561	439

Outcomes [Table]

The outcomes table shows the outcome values for all individuals. A table of all values for each outcome also appears the result folder/Endpoints.

The screenshot shows the Simulx software interface with the 'Results' tab selected. The 'OUTCOMES (TABLE)' view is active, displaying two tables of simulation results. The first table is for 'OD_200mgPerDay_' and the second is for 'BID_200mgPerDay_'. Each table has columns for 'rep ID', 'InhibitionAtTroughAbove80Percent', and 'InhibitionAtTrough'. The 'OD_200mgPerDay_' table shows 10 replicates, with the last one (rep 10) having a true value of 0.86. The 'BID_200mgPerDay_' table shows 3 replicates, with true values of 0.9, 0.89, and 0.81. The interface also includes a sidebar with 'ENDPOINTS' and 'GROUP COMPARISON' options, and a 'Display' panel on the right with 'Pagination' and 'Group by' settings.

rep ID	InhibitionAtTroughAbove80Percent	rep ID	InhibitionAtTrough
1 1	false	1 1	0.68
1 2	false	1 2	0.59
1 3	false	1 3	0.77
1 4	false	1 4	0.37
1 5	false	1 5	0.79
1 6	false	1 6	0.7
1 7	false	1 7	0.73
1 8	false	1 8	0.72
1 9	true	1 9	0.92
1 10	true	1 10	0.86

rep ID	InhibitionAtTroughAbove80Percent	rep ID	InhibitionAtTrough
1 31	true	1 31	0.9
1 32	true	1 32	0.89
1 33	true	1 33	0.81

Endpoints [Summary]

Endpoints summarize the outcome values over all individuals for a specific group and replicate. The endpoints summary shows percentiles for the calculated endpoint over **replicates**. These percentiles give the uncertainty of the endpoint value over several clinical trial simulations.

The screenshot shows the Simulx software interface with the 'Results' tab selected. The 'ENDPOINTS (SUMMARY)' view is active, displaying summary statistics for the endpoints 'InhibitionAtTroughAbove80Percent' and 'InhibitionAtTrough' for both 'OD_200mgPerDay_' and 'BID_200mgPerDay_'. The interface includes a sidebar with 'ENDPOINTS' and 'GROUP COMPARISON' options, and a 'Display' panel on the right with 'Pagination' and 'Group by' settings.

OD_200mgPerDay_

Efficacy						median_inhibitionAtTrough					
	MIN	P05	MEDIAN	P95	MAX		MIN	P05	MEDIAN	P95	MAX
percentTrue	13.33	20	30	46.67	56.67	median	0.64	0.68	0.74	0.79	0.82
totalTrue	4	6	9	14	17	lowerInterquartile	0.1	0.3	0.46	0.56	0.61
						upperInterquartile	0.83	0.85	0.89	0.91	0.92

BID_200mgPerDay_

Efficacy						median_inhibitionAtTrough					
	MIN	P05	MEDIAN	P95	MAX		MIN	P05	MEDIAN	P95	MAX
percentTrue	60	66.67	76.67	88.33	96.67	median	0.82	0.84	0.86	0.87	0.88
totalTrue	18	20	23	26.5	29	lowerInterquartile	0.59	0.59	0.69	0.77	0.81
						upperInterquartile	0.9	0.91	0.93	0.95	0.95

OD_300mgPerDay_

Efficacy						median_inhibitionAtTrough					
	MIN	P05	MEDIAN	P95	MAX		MIN	P05	MEDIAN	P95	MAX
percentTrue	20	23.33	40	56.67	63.33	median	0.72	0.73	0.77	0.81	0.84
totalTrue	6	7	12	17	19	lowerInterquartile	0.3	0.36	0.51	0.61	0.66
						upperInterquartile	0.84	0.87	0.9	0.92	0.94

Endpoints [Table]

The endpoints table shows the value of the endpoint defined in the simulation tab for each group and each replicate. A table for each endpoint is also saved in the result folder/Endpoints.

Efficacy

rep	percentTrue	totalTrue	rep	percentTrue	totalTrue	rep	percentTrue	totalTrue	rep	percentTrue	totalTrue
1	33.33	10	1	76.67	23	1	56.67	17	1	76.67	23
2	36.67	11	2	80	24	2	40	12	2	90	27
3	43.33	13	3	76.67	23	3	43.33	13	3	83.33	25
4	26.67	8	4	83.33	25	4	50	18	4	76.67	23
5	26.67	8	5	73.33	22	5	36.67	11	5	86.67	26
6	23.33	7	6	86.67	26	6	40	12	6	73.33	22
7	30	9	7	76.67	23	7	53.33	16	7	80	24
8	36.67	11	8	70	21	8	43.33	13	8	83.33	28
9	40	12	9	76.67	23	9	40	12	9	76.67	23
10	26.67	8	10	83.33	25	10	36.67	11	10	80	24

median_inhibitionAtTrough

rep	median	5thPercentile	95thPercentile	rep	median	5thPercentile	95thPercentile	rep	median	5thPercentile	95thPercentile
1	0.74	0.5	0.9	1	0.85	0.7	0.93	1	0.81	0.5	0.89
2	0.75	0.45	0.89	2	0.84	0.65	0.91	2	0.77	0.57	0.91
3	0.72	0.37	0.92	3	0.86	0.74	0.93	3	0.78	0.46	0.87



Group Comparison

If several simulation groups are used, and if group comparison is checked in the simulation tab, the group comparison section appears in the results.

Group comparison Table

For each replicate study, a test comparing the groups is performed as defined in the group comparison section of the simulation tab. The statistics of the test, p-value and the resulting decision (success or failure) according to the defined criteria are reported for each group (compared to the reference group) and for each replicate. A table is also saved for each statistical test and for each decision criterion in the result folder/Endpoints.

Efficacy										median_inhibitionATrough						
REP	BID_200MGPERDAY_			OD_300MGPERDAY_			BID_300MGPERDAY_			REP	BID_200MGPERDAY_		OD_300MGPERDAY_		BID_300MGPERDAY_	
	ODDSRATIO	P-VALUE	SUCCESS	ODDSRATIO	P-VALUE	SUCCESS	ODDSRATIO	P-VALUE	SUCCESS		DIFFERENCE	SUCCESS	DIFFERENCE	SUCCESS	DIFFERENCE	SUCCESS
1	6.57	3.42e-2	✓	2.62	4.18e-1	✗	6.57	3.42e-2	✓	1	0.12	✓	0.07	✗	0.12	✓
2	6.91	3.07e-2	✓	1.15	9.09e-1	✗	10.55	1.94e-2	✓	2	0.086	✗	0.016	✗	0.19	✓
3	4.3	1.43e-1	✗	1	9.47e-1	✗	6.54	4.43e-2	✓	3	0.14	✓	0.039	✗	0.13	✓
4	13.75	1.59e-3	✓	2.75	3.91e-1	✗	9.04	9.36e-3	✓	4	0.18	✓	0.1	✓	0.18	✓
5	7.56	1.92e-2	✓	1.59	7.43e-1	✗	17.68	5.33e-4	✓	5	0.11	✓	0.046	✗	0.15	✓
6	21.36	2.03e-4	✓	2.19	5.62e-1	✗	9.04	9.36e-3	✓	6	0.16	✓	0.049	✗	0.15	✓
7	7.47	1.86e-2	✓	2.67	4.07e-1	✗	9.33	8.68e-3	✓	7	0.11	✓	0.066	✗	0.14	✓
8	4.03	1.6e-1	✗	1.32	8.59e-1	✗	24.18	5.07e-4	✓	8	0.13	✓	0.068	✗	0.17	✓
9	4.93	9.4e-2	✗	1	9.46e-1	✗	4.93	9.4e-2	✗	9	0.083	✗	0.026	✗	0.13	✓
10	13.75	1.59e-3	✓	1.59	7.43e-1	✗	11	4.11e-3	✓	10	0.14	✓	0.046	✗	0.15	✓

Group comparison Summary

The summary of group comparison shows for each group the percentage of replicates that led to a successful test.

ENDPOINT	BID_200MGPERDAY_			OD_300MGPERDAY_			BID_300MGPERDAY_		
	SUCCESS%	CI 95% LOWER	CI 95% UPPER	SUCCESS%	CI 95% LOWER	CI 95% UPPER	SUCCESS%	CI 95% LOWER	CI 95% UPPER
Efficacy	65	55.25	73.64	0	0	3.7	92	85	95.89
median_inhibitionATrough	65	55.25	73.64	4	1.57	9.84	85	76.72	90.69

Starting from version 2024R1, the success rate across the replicates is now complemented by the corresponding confidence interval. The formula is based on the Wilson interval formula given by:

$$CI_W = \frac{X + \kappa^2/2}{n + \kappa^2} \pm \frac{\kappa \sqrt{n}}{n + \kappa^2} \sqrt{\hat{p}\hat{q} + \kappa^2/(4n)},$$

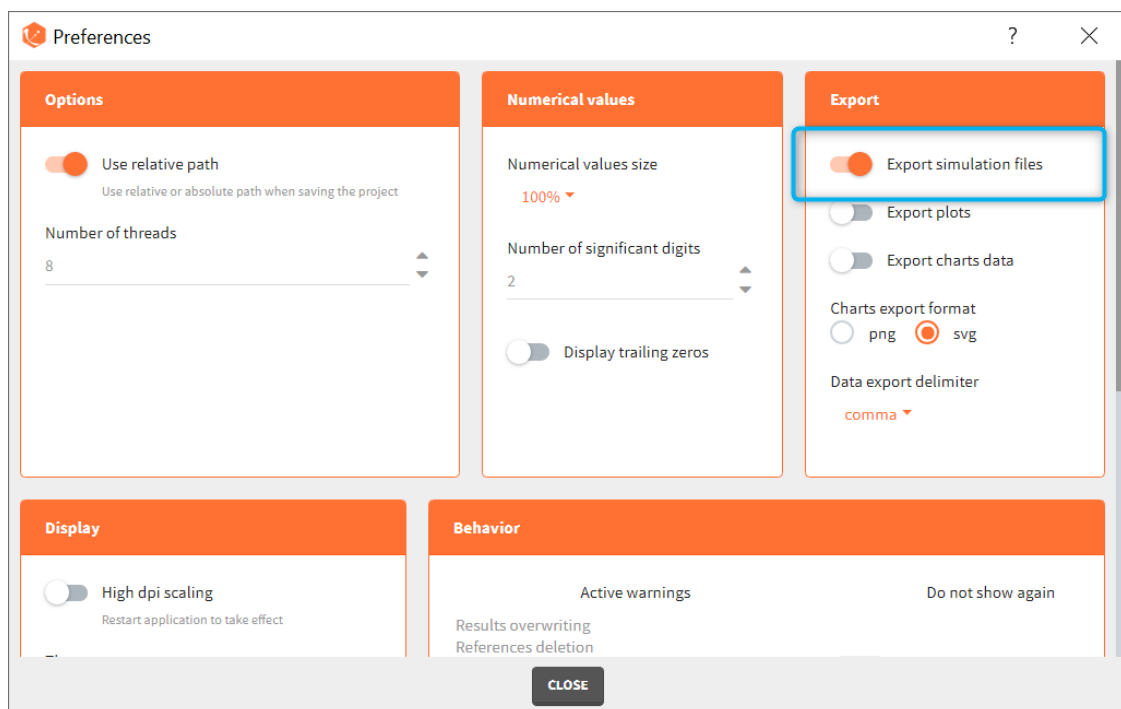
where n is the number of replicates, X the number of successful trials over the replicates, $\hat{p} = \frac{X}{n}$ corresponds to the rate of successful trials, $\hat{q} = (1 - \hat{p})$ to the rate of unsuccessful trials. κ corresponds to the α level's z -score, where $\alpha = 0.05$ for a 95% confidence interval.



Export Simulations

Export simulations as output files at each run

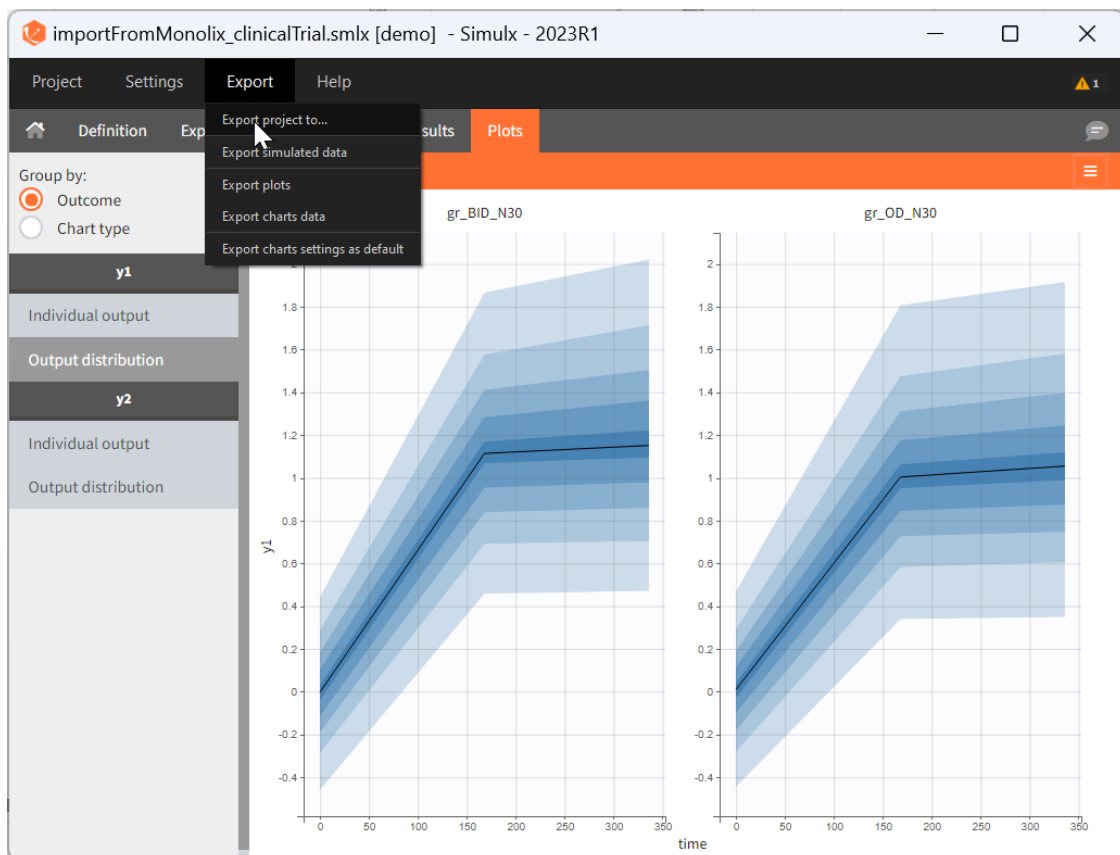
The full table of simulated values can be generated at each run if the preference setting "Export simulation files" is true. It will then appear in the result folder > Simulation > simulation_xxx.txt with xxx the output variable name.



Export simulations as a Monolix or PKanalix project

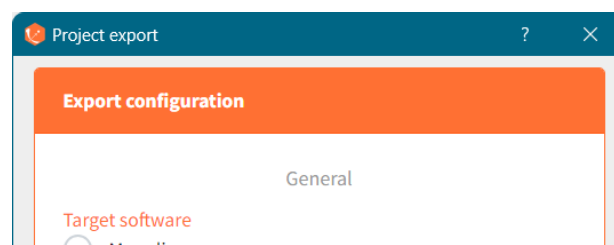
Note that this option not available in Simulx versions prior to 2023.

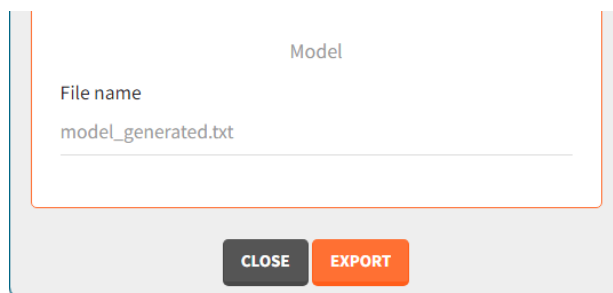
After running simulations, it is possible to generate a new Monolix or PKanalix project using the simulated values as dataset. This can be done from the menu Export > Export project to:



In the pop-up window, you can choose:

- to which application you want to export the simulated dataset: **PKanalix** or **Monolix**.
- names to the generated dataset (based on simulations) and generated model file (if the model is not part of the library).





Click the “EXPORT” button at the bottom to confirm. The target application (Monolix or PKanalix) will open automatically with a predefined project called “untitled”. The simulated dataset and model files (if not from the library) will be copied next to the new new project – first in the temporary folder, when the project is “untitled”, then to the final destination, where you save the project.

The dataset in the new project will be set in the following way:

- A column ID contains the identifiers for the individuals simulated in the Simulx project. It is tagged as ID if no replicates have been used in Simulx.
- If replicates have been simulated, a UID column containing a unique identifier combining replicate and ids (eg 1_rep1) is tagged as ID, and a rep column indicates the replicate (it is possible to use filters to select a single replicate if necessary).
- If external elements have been used in Simulx, original IDs coming from these elements are indicated in a column original_id which is ignored by default, but can be tagged by the user as categorical covariate or ID, depending on the situation.
- Simulation groups are given in a column group tagged as stratification categorical covariate.
- If several model outputs are simulated, the output names are indicated in an obsid column tagged as observation ID.
- Pre-defined tagging is used for time, observation, amount, and simulated covariates.

LINE NUMBER	rep	ID	UID	original_id	TIME	obs	obsid	AMOUNT	age	wt	sex	group
1	1		1_rep1						31	66.7	1	gr_BID_N30
2	1	1	1_rep1	2	0			4	31	66.7	1	gr_BID_N30
30	1	1	1_rep1	2	0	-0.157348	y1		31	66.7	1	gr_BID_N30
33	1	1	1_rep1	2	0	108.048	y2		31	66.7	1	gr_BID_N30
3	1	1	1_rep1	2	12			4	31	66.7	1	gr_BID_N30



For an export to PKanalix, the new project will be set in the following way:

- the data is set as described above.
- NCA settings: default PKanalix settings for administration type, integral method, treatment of BLQ values, parameters
- NCA settings: “observation ID to use” set to the first one alphabetically (if obsid are strings) or numerically (if obsid are integers).
- Acceptance criteria: not selected.
- Bioequivalence: default PKanalix settings
- the structural model is set to the generated model file attached to the new project, or a model from the library if the Simulx project used a library model.
 - The generated structural model only includes the [LONGITUDINAL] block of the Simulx model, without the error model.
 - If a statistical model was used in simulx (observation or individual model), it is not included in the generated model file for PKanalix.
 - The mapping automatically links the simulated observations to the model outputs.
 - If additional lines were used in the Simulx project, they are added at the end of the generated structural model file.
 - If output elements are defined based on intermediate variables of the model in the Simulx project, they are added to the output section of the model.
- The mapping automatically links the simulated observations to the model outputs. If a statistical model was used in simulx (observation or individual model), it is not included in the generated model file for PKanalix. The generated model only includes the [LONGITUDINAL] block without DEFINITION section.
- CA initial parameter values: set to default
- CA parameters constraints: none for normally distributed parameters, positive for log-normally distributed parameters; bounded with limits imported from the Monolix project for logit-normally and probit-normally distributed parameters.
- CA calculations settings: default PKanalix settings

For an export to Monolix, the project will be set in the following way:

- the data is set as described above.
- the structural model is set to the generated model file attached to the new project, or a model from the library if the Simulx project used a library model.
 - The generated structural model only includes the [LONGITUDINAL] block of the

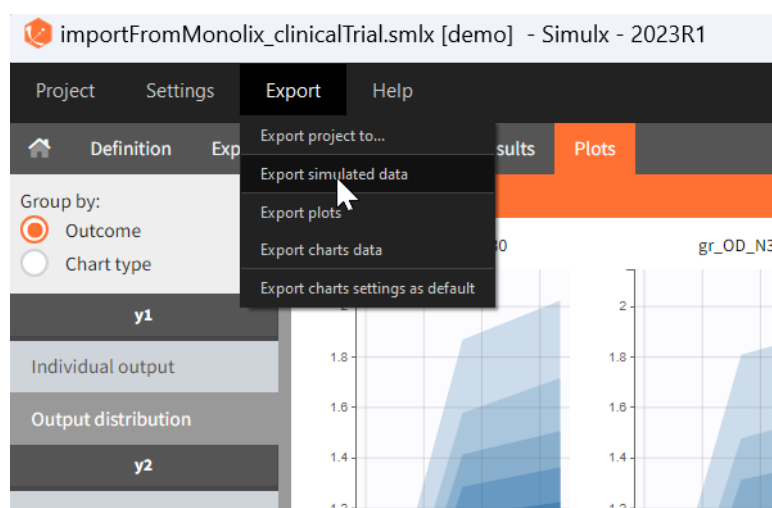
Simulx model, without the error model.

- The mapping automatically links the simulated observations to the model outputs.
- If additional lines were used in the Simulx project, they are added at the end of the generated structural model file.
- If output elements are defined based on intermediate variables of the model in the Simulx project, they are added to the output section of the model.
- Initial values are set to default.
- the observation model and individual model in Monolix statistical model tab is set based on the Simulx project, if a statistical model was **defined in the model** of the Simulx project (INDIVIDUAL, COVARIATE blocks or error model in the LONGITUDINAL block). Note that the statistical model is not included in the generated model file for Monolix, but it is taken into account in the statistical model tab. Only the **Monolix style** syntax for the INDIVIDUAL block can be imported to the Monolix project, not the **flexible style**. When modifications are done to the model for compatibility reasons, Simulx warns you at export to double check the model interpretation in the Monolix project.
- if some part of the statistical model is not defined in the Simulx project, it is set to default in the statistical model tab.

Export simulations as a formatted dataset

Note that this option not available in Simulx versions prior to 2021.

To export simulations as a formatted dataset for further investigations outside the Monolixsuite, it is possible to use the menu Export > Export simulated data. This data set contains the dose, occasions, covariate, simulation groups, observation id informations in addition to the id, time and simulated values. The exported simulated data file is located in the result folder > Simulation > simulatedData.txt.








Share a Simulx project

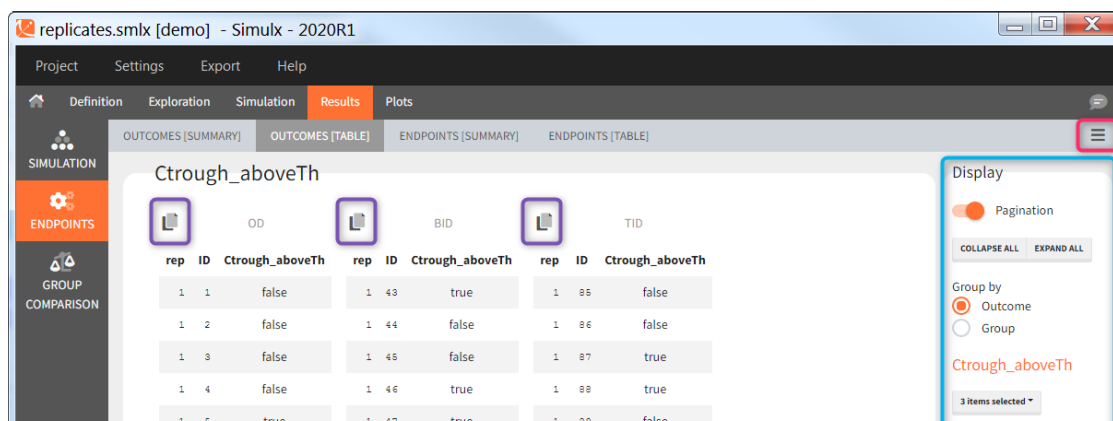
The 2024 version of MonolixSuite introduces a highly convenient method for sharing projects. Simply click on “Share Project” in the export menu, and a zip folder is generated containing all the required files to re-open the project (dataset if applicable, model if applicable, external files if applicable, smlx file, result folder). By default, the suggested location for the zipped shared project matches that of the original, but this can be easily modified to any other location on the computer.



General display options

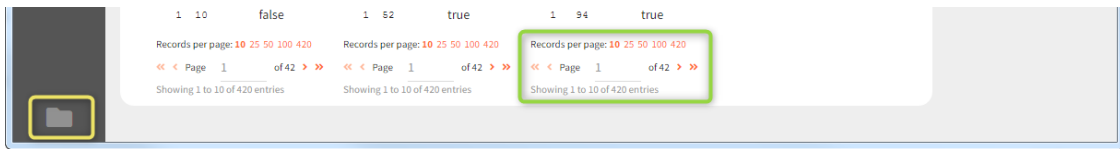
The tables generally contain many rows and are by default paginated. The page displayed and the number of records per page can be changed at the bottom (green highlight). The **Display** panel on the right (blue highlight) allows to switch the pagination off in order to show the entire table. The tables can also be all expanded and collapsed. When several groups and several output variables/outcomes/endpoints have been defined, the tables can be grouped by either simulation group or output/outcome/endpoint to easily compare the tables of interest. Finally it is possible to hide some of the simulation groups using the drop-down selection menu.

The display panel can be hidden with the button  at the top right. The button  on the bottom left opens the result folder which contains the simulated values and their post-processing as text files. All tables can be copied to the clipboard using the button  at the top left. Only the displayed page is copied. To copy the entire table, expand it first.



The screenshot shows the Simulx software interface. The main window displays a table of results for the variable 'Ctrough_aboveTh'. The table has three columns for 'rep ID' and 'Ctrough_aboveTh' values. The 'Display' panel on the right is highlighted in blue and contains the following options:

- Pagination
- COLLAPSE ALL EXPAND ALL
- Group by:
 - Outcome
 - Group
- Ctrough_aboveTh
- 3 items selected



5. Simulx API

On the use of the R-functions

We now propose to use Simulx via R-functions. The package **lixoftConnectors** provides access to the project exactly in the same way as you would do with the interface. All the installation guidelines and initialization procedure can be found [here](#). All the functions are described below.

- Installation guidelines and initialization procedure
- Page with typical examples
 - Load a project and run the simulation
 - Import a Monolix project and simulate a new output variable with the EBEs
 - Create a Simulx project from scratch and simulate covariate-dependent treatments
 - Import a Monolix project and simulate with new output times
 - Run a simulation with different sample sizes and plot the power of the study based on an endpoint
 - Handling of warning/error/info messages

Minimal examples based on demo projects are also included in the documentation of the functions below.

- List of the R functions
 - Description of the functions concerning the element definition
 - Description of the functions concerning the group comparison
 - Description of the functions concerning the initialization and path to demo projects

- Description of the functions concerning the model extension
- Description of the functions concerning the project management
- Description of the functions concerning the project settings and preferences
- Description of the functions concerning the results
- Description of the functions concerning the scenario

List of the R functions

Description of the functions concerning the element definition

- `defineCovariateElement`: Define or edit a covariate element.
- `defineEndpoint`: Define or edit an endpoint.
- `defineIndividualElement`: Define or edit an element of individual parameters.
- `defineOccasionElement`: Define the occasion structure of the project.
- `defineOutcome`: Define or edit an outcome.
- `defineOutputElement`: Define or edit an output element.
- `definePopulationElement`: Define or edit an element of population parameters.
- `defineRegressorElement`: Define or edit a regressor element.
- `defineTreatmentElement`: Define or edit a treatment element.
- `deleteElement`: Delete an element of any type.
- `deleteEndpoint`: Delete an endpoint.
- `deleteOccasionElement`: Delete the occasion element.
- `deleteOutcome`: Delete an outcome.
- `getCovariateElements`: Get the list of all available covariate elements in the loaded project.
- `getEndpoints`: Get the list of all endpoints.
- `getIndividualElements`: Get the list of all available individual parameters elements for the simulation.
- `getOccasionElements`: Get the content of the occasion element that will be used for the simulation.
- `getOutcomes`: Get the list of all available outcomes.
- `getOutputElements`: Get the list of all available output elements for simulation.
- `getPopulationElements`: Get the list of all available population parameters elements for the simulation.
- `getRegressorElements`: Get the list of all available regressor elements for simulation.
- `getRegressorsSettings`: Get the regressors global settings.
- `getTreatmentElements`: Get the list of all available treatments elements for the

simulation.

- **setRegressorsSettings**: Edit the regressors global settings.

Description of the functions concerning the group comparison

- **getGroupComparisonSettings**: Set settings related to the comparison of endpoints across groups.
- **setGroupComparisonSettings**: Set settings related to the comparison of endpoints across groups.

Description of the functions concerning the initialization and path to demo projects

- **initializeLixoftConnectors**: Initialize lixoftConnectors API for a given software.
- **addIndividualBlock**: Add an INDIVIDUAL block to a structural model.
- **getDemoPath**: Get the path to the demo projects.

Description of the functions concerning the model extension

- **getAddLines**: Get the lines that were added to a model with `setAddLines` (or in the GUI).
- **setAddLines**: Add equations to the structural model.

Description of the functions concerning the project management

- **exportProject**: Export the current project to another application of the MonolixSuite, and load the exported project.
- **getLibraryModelContent**: Get the content of a library model.
- **getLibraryModelName**: Get the name of a library model given a list of library filters.
- **getStructuralModel**: Get the model file for the structural model used in the current project.
- **importProject**: Import a Monolix or a PKanalix project into the currently running application initialized in the connectors.
- **isProjectLoaded**: Get a logical saying if a project is currently loaded.
- **loadProject**: Load a project in the currently running application initialized in the connectors.
- **newProject**: Create a new project.
- **saveProject**: Save the current project as a file that can be reloaded in the connectors or in the GUI.
- **shareProject**: Create a zip archive file from current project and its results.

Description of the functions concerning the project settings and preferences

- **getConsoleMode**: Get console mode, ie volume of output after running estimation tasks.
- **getPreferences**: Get a summary of the project preferences.
- **getProjectSettings**: Get a summary of the project settings.
- **setConsoleMode**: Set console mode, ie volume of output after running estimation tasks.
- **setPreferences**: Set the value of one or several of the project preferences.
- **setProjectSettings**: Set the value of one or several of the settings of the project.

Description of the functions concerning the results

- **exportSimulatedData**: Export the simulated dataset into a MonolixSuite compatible format.
- **getEndpointsResults**: Get the results of the outcomes & endpoints task.
- **getSimulationResults**: Get the results of the simulation.

Description of the functions concerning the scenario

- **computeChartsData**: Compute (if needed) and export the charts data of a given plot or, if not specified, all the available project plots.
- **getScenario**: Get the list of tasks that will be run at the next call to `runScenario`.
- **runScenario**: Run the scenario that has been set with `setScenario`.
- **setScenario**: Clear the current scenario and build a new one from a given list of

tasks.

- **addGroup**: Add a new simulation group.
- **getGroupRemaining**: Get the values of the remaining parameters (typically the error model parameters) for a group.
- **getGroups**: Get the list of simulation groups and elements in each group.
- **getNbReplicates**: Get the number of replicates of the simulation.
- **getSameIndividualsAmongGroups**: Get the information if the same individuals are simulated among all groups.
- **getSamplingMethod**: Get which sampling method is used for the simulation.
- **getSharedIds**: Get the element types that will share the same individuals in the simulation.
- **removeGroup**: Remove a simulation group.
- **removeGroupElement**: Remove an element from a simulation group.
- **renameGroup**: Rename a simulation group.
- **runEndpoints**: Run the endpoints task.
- **runSimulation**: Run the simulation task.
- **setGroupElement**: Set the new element of a specific group.
- **setGroupRemaining**: Set the values of the remaining parameters (typically the error model parameters) for a group.
- **setGroupSize**: Define the size of a simulation group.
- **setNbReplicates**: Define the number of replicates of the simulation.
- **setSameIndividualsAmongGroups**: Define if the same individuals will be simulated among all groups.
- **setSamplingMethod**: Define which sampling method is used for the simulation.
- **setSharedIds**: Select the element types that will share the same individuals in the simulation.
- **printOutcomesEndpoints**: Get all the outcomes and endpoints defined in the Simulx project.
- **printSimulationSetup**: Get all the elements used in simulation setup: content of simulation groups and simulation settings.

[Simulx] Define covariate element

Description

Define or edit a covariate element.

Covariate elements are defined and used for simulation [as in Simulx GUI](#). As for other elements, the covariate elements can be defined or imported, and they are saved with the Simulx project if calling `saveProject`. Once a covariate element is defined, it needs to be added to a simulation group with `setGroupElement` to be used in simulation.

Usage

```
defineCovariateElement(name, element)
```

Arguments

<code>name</code>	<i>(character)</i> Element name.
<code>element</code>	<i>(character or dataframe or list)</i> Element definition from external file path or data frame with covariates as columns, or list to select the sheet of an excel file: <ul style="list-style-type: none">• <code>file</code> <i>(character)</i> Path to the population file.• <code>sheet</code> <i>(character)</i> Name of the sheet in xlsx/xls file.

Details

Covariate elements can be defined only if the model used in the Simulx project contains [a block \[COVARIATE\]](#).

A covariate element can be defined as an external file (csv, xlsx, xlsx, sas7bdat, xpt or txt) or as a data frame.

In any case, it can contain columns occasions (optional), and it should contain one column per covariate (mandatory). Covariate names and categorical covariate values must correspond to covariates and categories defined in the model (block [\[COVARIATE\]](#)). The occasion headers must correspond to the occasion names defined in the occasion element.

A data frame can be used only to define covariate elements of type 'common', i.e the same for all individuals (potentially occasion-wise). If you want to define subject-specific covariates, use an external file with an "id" column. Covariate definition with distributions is only possible in the GUI (in R, please sample from the desired distribution to generate an external file).

An external file can be used in all cases (common or subject-specific). It can contain a column id (optional) in addition to occasions (optional), and should contain one column per covariate (mandatory). When id and occasion columns are present, then they must be the first columns. When the id column is not present, the covariate is considered common.

If the project has a subject-specific occasion structure (defined with an external file with an ID column (see [defineOccasionElement](#))), occasion-wise common elements are not allowed. Covariate elements must be either common over all subjects and all occasions, or can be defined with subject-specific occasion-wise values as an external table, with the same occasion structure.

See Also

[getCovariateElements](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Define endpoint element

Description

Define or edit an endpoint. Endpoints summarize the outcome values over all individuals, for each simulation group and each replicate. Endpoints are defined [as in Simulx GUI](#).

Usage

```
defineEndpoint(name, element)
```

Arguments

name	(<i>character</i>) (required) Endpoint name.
element	(<i>list</i>) (required) List with the endpoint settings: <ul style="list-style-type: none">• <code>outcome</code> (<i>character</i> or <i>list</i>) (required) – Outcome on which the endpoint is based on. If one outcome, use a string containing its name. Use list to combine outcomes with:<ul style="list-style-type: none">• <code>names</code> (<i>vector</i> of <i>character</i>) – Vector of outcome names• <code>groupName</code> (<i>character</i>) – Name you want to give to the outcome combination

- `operator` (*character*) – Way in which output should be combined. One of "and"/"or" (in case of boolean outcomes) or "min"/"max" (in other cases)
- `metric` (*character*) (optional) – Calculation method for the endpoint. One of "**arithmeticMean**" (default), "geometricMean" or "median" if value-based outcome. In case of event-based outcomes, "kaplanMeier" (median survival) will be used and in case of boolean outcomes, "percentTrue" will be used by default.
- `groupComparison` (optional) (*list*) – Group comparison settings. List of:
 - `type` (*character*) (optional) – one of "directComparison", "**statisticalTest**" (default).
 - `h1` (*list*) (optional) – a list containing hypothesis information:
 - `operator` (*character*) – one of "!=" (default), ">" or "<",
 - `threshold` (*double*) – a real number indicating the threshold for difference/oddsRatio (0 by default)
 - `pvalue` (*double*) – a real number indicating the p-value (if `type` is "statisticalTest", 0.05 by default)

Details

To compute the defined endpoints, use `runEndpoints` and get the results with `getEndpointsResults`.

To specify if endpoints should be compared across simulation groups, use `setGroupComparisonSettings`. If group comparison is relevant, the way the comparison will be done for each endpoint (eg if statistical test and which p-value) should be defined in the endpoint element.

See Also

[getEndpoints](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Define individual parameters element

Description

Define or edit an element of individual parameters. Individual parameter elements are defined and used for simulation [as in Simulx GUI](#). As for other elements, the individual parameters elements can be defined or imported, and they are saved with the Simulx project if calling `saveProject`. Once an individual parameters element is defined, it needs to be added to a simulation group with `setGroupElement` to be used in simulation.

Usage

```
defineIndividualElement(name, element)
```

Arguments

name	(<i>character</i>) Element name.
element	(<i>character</i> or <i>dataFrame</i> or <i>list</i>) Element definition from external file path or data frame with individual parameters as columns, or list to select the sheet of an excel file: <ul style="list-style-type: none">• <code>file</code> (<i>character</i>) Path to the individual parameters file.• <code>sheet</code> (<i>character</i>) Name of the sheet in xlsx/xls file.

Details

Individual parameters to be defined depend on the model of the simulx project. If only the [LONGITUDINAL] block is present in the model: all parameters of the input list, except those defined as regressors. If both the [LONGITUDINAL] and [INDIVIDUAL] blocks are present: parameters defined in the DEFINITION section of the [INDIVIDUAL] block.

An individual parameters element can be defined as an external file (csv, xlsx, xls, sas7bdat, xpt or txt) or as a data frame. In any case, it can contain columns occasions (optional), and it should contain one column per individual parameter (mandatory). The parameter headers must correspond to the parameter names used in the model. The occasion headers must correspond to the occasion names defined in the occasion element.

A data frame can be used only to define individual parameter elements of type 'common', i.e the same for all individuals (but potentially occasion-wise). If you want to define subject-specific individual parameters, use an external file with an "id" column.

An external file can be used in all cases (common or subject-specific). It can contain a column id (optional) in addition to occasions (optional), and should contain one column per parameter (mandatory). When id and occasion columns are present, then they must be the first columns. When the id column is not present, the parameter element is considered common.

If the project has a subject-specific occasion structure (defined with an external file with an ID column (see [defineOccasionElement](#))), occasion-wise common elements are not allowed. In this case, individual parameters elements have to be either common over all subjects and all occasions, or can be defined with subject-specific occasion-wise values as an external table, with the same occasion structure.

See Also

[getIndividualElements](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Define occasion element

Description

Define the occasion structure of the project.

Usage

```
defineOccasionElement(element)
```

Arguments

`element` (*character* or *dataFrame* or *list*) Element definition from external file path or data frame with time and occasion levels as columns, or list to select the sheet of an excel file:

- `file` (*character*) Path to the occasion file.
- `sheet` (*character*) Name of the sheet in xlsx/xls file.

Details

The occasion structure of a project is defined and used for simulation [as in Simulx GUI](#).

The occasion element impacts the definition of other elements and the simulation. As for other elements, the occasion element can be defined or imported, and it is saved with the Simulx project if calling [saveProject](#).

It can be defined as an external file (csv, xls, xlsx, sas7bdat, xpt or txt) or as a data.frame.

In any case, it should contain a column time and one column per occasion level. The header names for these occasion levels are free and used by Simulx.

A data frame can be used only to define a common structure of occasions applied to all simulated subjects.

An external file can be used in all cases (common or subject-specific structure). It can contain a column id (optional) in addition to other columns. When the id column is not present, the occasion structure is considered common.

After defining a subject-specific occasion structure, other elements (parameters, covariates, treatments, outputs and regressors) must be either common over all subjects and all occasions, or can be defined with subject-specific occasion-wise values as an external table, with the same occasion structure.

See Also

[getOccasionElements](#)

Click here to see examples	
--	--

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Define outcome element

Description

Define or edit an outcome. Outcomes represent a post-processing of the simulation outputs done for each individual.

Outcomes are defined [as in Simulx GUI](#).

Outcomes can only be defined by the user (no outcome is imported), and they are saved with the Simulx project if calling [saveProject](#).

Contrary to the GUI, defining an outcome with the connectors does not automatically generate an endpoint.

To compute the defined outcomes, use them in endpoints with [defineEndpoint](#), run [runEndpoints](#) and get the results with [getEndpointsResults](#).

Usage

```
defineOutcome(name, element)
```

Arguments

name	(character)	Outcome name.
(list) List with the outcome settings:		
output	(character)	Name of the output element on which the outcome is based.
perOccasion	(logical)	(optional) If occasions are present in the simulation, it indicates if the outcome is calculated for each id or each occasion of each id. TRUE by default.
element		Then, if the outcome is based on a continuous or categorical output:
relativeTo	(list)	(optional) List of elements that define reference settings: <ul style="list-style-type: none">• <code>reference</code> - one of "baseline", "min", "max", "minCurrentTime", "maxCurrentTime" or "customValue",• <code>type</code> - "ratio" or "difference",• <code>value</code> - if <i>reference</i> is "customValue".

(required) List of elements that define how the output will be processed:

- `operator` (required) – one of "avg", "min", "max", "first", "last", "durationBelow", "durationAbove", "durationBetween", "timePoint", or "none" (if the output has a single time point and does not require processing).
- `type` (optional)
 - if `operator` is "min" or "max", one of "**value**" (default), "timeContinuous" or "timeEvent",
 - if `operator` is "durationBelow", "durationAbove" or "durationBetween", one of "**cumulativeTime**" (default), "percentTime", "nbObs", "firstOccurrenceContinuous" or "firstOccurrenceEvent",
- `value` (optional) – vector of boundaries if `operator` is "durationBelow", "durationAbove" or "durationBetween", or time point if `operator` is "timePoint" (0 by default).

`processing` (list)

(optional) List of elements:

- `operator` – one of "==" , "!=" , ">=" , ">" , "<=" or "<" ,
- `value` – a real number indicating the threshold value.

`applyThreshold` (list)

Or if the outcome is based on a **TTE** output:

(required) List of arguments that define event settings:

`event` (list)

-

- `type` (required) – one of "timeOfEvents" (in case of single and repeated TTE), "hasAnEvent", "hasNoEvent" (in case of single TTE) or "numberOf" (in case of repeated TTE),
- `rank` (optional) – rank of the event of which time is the outcome (if `type` is "timeOfEvents" and repeated TTE).

See Also

[getOutcomes](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Define output element

Description

Define or edit an output element.

Usage

```
defineOutputElement(name, element)
```

Arguments

<code>name</code>	<i>(character)</i> Element name.
-------------------	----------------------------------

List with:

- `data` (*character* or *dataFrame* or *list*): data frame or path to external file (csv, xlsx, xls, sas7bdat, xpt or txt), or list to select the sheet of an excel file:

<code>element</code>	<ul style="list-style-type: none">• <code>file</code> (<i>character</i>) Path to the output file.• <code>sheet</code> (<i>character</i>) Name of the sheet in xlsx/xls file.•
----------------------	---

output (*character*): name of any variable from the [LONGITUDINAL] block of the model (variable in EQUATION, smooth prediction listed under OUTPUT or noisy observation listed under DEFINITION).

Details

Output elements are defined and used for simulation [as in Simulx GUI](#). As for other elements, the output elements can be defined or imported, and they are saved with the Simulx project if calling `saveProject`. Once an output element is defined, it needs to be added to a simulation group with `setGroupElement` to be used in simulation.

To define an output element, in addition to the element name, you need to provide the time grid for simulation and the output to simulate.

The field data can be specified with a data frame or an external file (csv, xlsx, xlsx, sas7bdat, xpt or txt).

To define a regular output, common to all individuals, you can use a data frame, with column headers start, interval and final. All time points from "start" to "final" by steps of "interval" will be used for simulation. If the project has a common occasion structure, this data frame can contain a column occasion and several lines to define the regular treatment occasion-wise.

To define an output by giving a specific list of times, both data frames and external files (csv, xlsx, xlsx, sas7bdat, xpt or txt) can be used, with a column time. They can contain columns occasions (optional). The occasion headers must correspond to the occasion names defined in the occasion element.

Data frames can be used only to define output elements of type 'common', i.e the same for all individuals (potentially occasion-wise). If you want to define subject-specific output elements, you have to use an external file with an "id" column.

An external file can be used in all cases (common or subject-specific). It can contain a column id (optional) in addition to occasions (optional), and should contain one column time (mandatory). When id and occasion columns are present, then they must be the first columns. When the id column is not present, the covariate is considered common.

If the project has a subject-specific occasion structure (defined with an external file with an ID column (see [defineOccasionElement](#))), occasion-wise common elements are not allowed. Output elements must be either common over all subjects and all occasions, or can be defined with subject-specific occasion-wise values as an external table, with the same occasion structure.

See Also

[getOutputElements](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Define population element

Description

Define or edit an element of population parameters. Population parameter elements are defined and used for simulation [as in Simulx GUI](#). As for other elements, the population parameters elements can be defined or imported, and they are saved with the Simulx project if calling [saveProject](#). Once a population parameters element is defined, it needs to be added to a simulation group with [setGroupElement](#) to be used in simulation.

Usage

```
definePopulationElement(name, element)
```

Arguments

<code>name</code>	<i>(character)</i> Element name.
<code>element</code>	<i>(character or dataframe or list)</i> Element definition from external file path or data frame with population parameters as columns, or list to select the sheet of an excel file: <ul style="list-style-type: none"><code>file</code> <i>(character)</i> Path to the population file.<code>sheet</code> <i>(character)</i> Name of the sheet in xlsx/xls file.

Details

Definition of population parameters as simulation elements allows to simulate individual parameters from probability distributions. It is possible only if the model includes [a block \[INDIVIDUAL\]](#) to consider the inter-individual variability.

A population parameters element can be defined as an external file (csv, xlsx, xls, sas7bdat, xpt or txt) or as a data frame. In any case, it should contain one column per population parameter (mandatory).

To check exactly which column names to use, you can use [getPopulationElements](#) and view the population parameters element that was automatically created after defining the model (if the model has an [INDIVIDUAL] block).

To define a population parameters element with several lines, with several sets to be used in different replicate simulations, an external file is required. In this case, you should also set the number of replicates for your simulation with [setNbReplicates](#), otherwise only the first set of population parameters will be taken into account. Each replicate uses one set of population parameters with the order of the appearance in the table.

Note: It is not possible to define population elements with distributions with the connectors as in the GUI. To do this, please sample values from distributions in R and create the element with different rows as in the last example below.

See Also

[getPopulationElements](#), [setNbReplicates](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Define regressor element

Description

Define or edit a regressor element.

Regressor elements are defined and used for simulation [as in Simulx GUI](#). Once a regressor element is defined, it needs to be added to a simulation group with [setGroupElement](#) to be used in simulation. Regressor elements can be defined only if regressors are defined in the model loaded in the Simulx project.

Usage

```
defineRegressorElement(name, element)
```

Arguments

name	(<i>character</i>) Element name.
element	(<i>character</i> or <i>dataFrame</i> or <i>list</i>) Element definition from external file path or data frame with time and regressors as columns, or list to select the sheet of an excel file:

- `file` (*character*) Path to the regressors file.
- `sheet` (*character*) Name of the sheet in xlsx/xls file.

Details

To define a regressor element, in addition to the element name, you need to provide in the field data the time points and values of the regressor at each time point. To simulate the model for points outside of this grid, last value carried forward interpolation is used.

The field data can be specified with a data frame or an external file (csv, xlsx, xls, sas7bdat, xpt or txt). They should contain a column time and a column for each regressor variable. They can contain columns occasions (optional). The occasion headers must correspond to the occasion names defined in the occasion element.

Data frames can be used only to define regressor elements of type 'common', i.e the same for all individuals (potentially occasion-wise). If you want to define subject-specific regressor elements, you have to use an external file with an additional "id" column.

An external file can be used in all cases (common or subject-specific). It can contain a column id (optional) in addition to occasions (optional), time (mandatory) and one column per regressor defined in the model (mandatory). When id and occasion columns are present, then they must be the first columns. When the id column is not present, the covariate is considered common.

If the project has a subject-specific occasion structure (defined with an external file with an ID column (see [defineOccasionElement](#))), occasion-wise common elements are not allowed. In this case, regressors must be either common over all subjects and all occasions, or defined with subject-specific occasion-wise values as an external table, with the same occasion structure.

Note: It is not possible to define regressor elements with distributions with the connectors as in the GUI. To do this, please sample values from distributions in R and create the element with different rows as in the examples below.

See Also

[getRegressorElements](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Define treatment element

Description

Define or edit a treatment element.

Usage

```
defineTreatmentElement(name, element)
```

Arguments

name	(<i>character</i>) Element name.						
element	<p>(<i>list</i>) List with the treatment settings:</p> <table border="1"><tr><td>data (mandatory)</td><td>(data frame OR path to external file (csv, xlsx, xlsx, sas7bdat, xpt or txt) OR list to select the sheet of an excel file: list(file = path_to_excel, sheet = sheet_name)</td></tr><tr><td colspan="2">Column headers:</td></tr><tr><td colspan="2"><ul style="list-style-type: none">• [only data frame] for a regular treatment common to all ids: <code>occ</code> (optional, if common occasion structure, same header as in the occasion element), <code>start</code>, <code>interval</code>, <code>nbDoses</code>, <code>amount</code>• [only data frame] for a manual treatment common to all ids: <code>occ</code> (optional, if common occasion structure, same header as in the occasion element), <code>time</code>, <code>amount</code>, <code>washout</code> (optional, to add a washout just before the dose, otherwise <code>washout = FALSE</code> by default)• [only external file] for a manual treatment common or specific to each id: <code>id</code> (optional), <code>occ</code> (optional), <code>time</code>, <code>amount</code>, <code>washout</code> (optional)• [data frame or external] in case of infusion: <code>tInf</code> (duration) OR <code>rate</code> (mutually exclusive).</td></tr></table>	data (mandatory)	(data frame OR path to external file (csv, xlsx, xlsx, sas7bdat, xpt or txt) OR list to select the sheet of an excel file: list(file = path_to_excel, sheet = sheet_name)	Column headers:		<ul style="list-style-type: none">• [only data frame] for a regular treatment common to all ids: <code>occ</code> (optional, if common occasion structure, same header as in the occasion element), <code>start</code>, <code>interval</code>, <code>nbDoses</code>, <code>amount</code>• [only data frame] for a manual treatment common to all ids: <code>occ</code> (optional, if common occasion structure, same header as in the occasion element), <code>time</code>, <code>amount</code>, <code>washout</code> (optional, to add a washout just before the dose, otherwise <code>washout = FALSE</code> by default)• [only external file] for a manual treatment common or specific to each id: <code>id</code> (optional), <code>occ</code> (optional), <code>time</code>, <code>amount</code>, <code>washout</code> (optional)• [data frame or external] in case of infusion: <code>tInf</code> (duration) OR <code>rate</code> (mutually exclusive).	
data (mandatory)	(data frame OR path to external file (csv, xlsx, xlsx, sas7bdat, xpt or txt) OR list to select the sheet of an excel file: list(file = path_to_excel, sheet = sheet_name)						
Column headers:							
<ul style="list-style-type: none">• [only data frame] for a regular treatment common to all ids: <code>occ</code> (optional, if common occasion structure, same header as in the occasion element), <code>start</code>, <code>interval</code>, <code>nbDoses</code>, <code>amount</code>• [only data frame] for a manual treatment common to all ids: <code>occ</code> (optional, if common occasion structure, same header as in the occasion element), <code>time</code>, <code>amount</code>, <code>washout</code> (optional, to add a washout just before the dose, otherwise <code>washout = FALSE</code> by default)• [only external file] for a manual treatment common or specific to each id: <code>id</code> (optional), <code>occ</code> (optional), <code>time</code>, <code>amount</code>, <code>washout</code> (optional)• [data frame or external] in case of infusion: <code>tInf</code> (duration) OR <code>rate</code> (mutually exclusive).							

admID (optional)	(integer) same as integer in the model as administration type
probaMissDose (optional)	(double) probability to miss a dose (number in [0 1])
repeats (optional)	(vector) to repeat the specified treatment after a specific duration.
	Elements:
	<p>cycleDuration duration after which the treatment will be repeated (can be longer than the treatment duration)</p> <p>NumberOfRepetitions number of times the treatment will be repeated</p>
scale (optional)	(list) to scale the dose amount by a covariate. The scaled amount will be administered instead of amount.
	<p>covariate (character) covariate to use for scaling (same name as in the [COVARIATE] block of the model)</p> <p>intercept (double, required for continuous covariate): intercept to use in the scaling formula: $\text{scaledAmount} = \text{amount} * \text{cov} + \text{intercept}$</p> <p>modalities (list, required for categorical covariate): list of lists with, for each modality, the name of the modality (eg "male"), the factor, and the intercept to use in the scaling formula: $\text{scaledAmount} = [\text{cov} = \text{modality}] * \text{factor} * \text{amt} + \text{intercept}$ (no scaling if factor =1 and intercept = 0)</p> <p>scaleDuration</p>

(optional, logical) if TRUE (default), infusion duration will be scaled, otherwise it will be rate

Details

Treatment elements are defined and used for simulation [as in Simulx GUI](#). As for other elements, the treatment elements can be defined or created at import, and they are saved with the Simulx project if calling `saveProject`. Once an output element is defined, it needs to be added to a simulation group with `setGroupElement` to be used in simulation. Several treatment elements can be added to the same simulation group and they will be both administered to every individual in the group.

To define a treatment element, in addition to the element name, you need to provide a list with at list one field "data" containing the dose information. The field data can be specified with a data frame or an external file (csv, xlsx, xlsx, sas7bdat, xpt or txt).

To define a regular treatment, common to all individuals, you can use a data frame, with column headers start, interval, nbDoses and amount. You can include an optional column tInf or rate to define an infusion. If the project has a common occasion structure (i.e. same occasions for all individuals), this data frame can contain a column occasion to define the regular treatment occasion-wise. The occasion headers must correspond to the occasion names defined in the occasion element.

To define a treatment by giving a specific list of times and amounts, both data frames and external files (csv, xlsx, xlsx, sas7bdat, xpt or txt) can be used, with a column time. They can contain columns id and occasions (optional). The occasion headers must correspond to the occasion names defined in the occasion element.

Data frames can be used only to define output elements of type 'common', i.e the same for all individuals (potentially occasion-wise). If you want to define subject-specific treatment elements, you have to use an external file with an "id" column.

An external file can be used in all cases (common or subject-specific). It can contain a column id (optional) in addition to occasions (optional), and should contain one column time (mandatory) and one column amount (mandatory). When id and occasion columns are present, then they must be the first columns. When the id column is not present, the covariate is considered common.

Note

To define a regular schedule, it is advised to use a regular treatment without repeats, rather than a manual treatment with repeats. Repeats are useful to create more complex schedules in addition to a manual or regular definition, such as dosing regimen 3 weeks ON, 1 week OFF.

To see the impact of a treatment until the end of a dosing regimen, you should set an output element that spans the duration of the treatment to the same simulation group.

See Also

[getTreatmentElements](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Delete element

Description

Delete an element of any type.

Usage

```
deleteElement (name)
```

Arguments

name	(<i>character</i>) Element name.
------	------------------------------------

Details

Elements defined are created in the background and saved with the Simulx project if calling [saveProject](#).

To check which elements of a certain type have been defined so far, please use one of the "get..Elements" connectors: [getCovariateElements](#), [getPopulationElements](#), [getIndividualElements](#), [getTreatmentElements](#), [getOccasionElements](#), [getRegressorElements](#).

Elements cannot be deleted if they are used for the simulation. To remove an element from the simulation, use [removeGroupElement](#).

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Delete an endpoint

Description

Delete an endpoint.

Usage

```
deleteEndpoint (name)
```

Arguments

name	(<i>character</i>) Endpoint name
------	------------------------------------

Details

Endpoints defined are created in the background and saved with the Simulx project if calling [saveProject](#).

To check which endpoints have been defined, please use [getEndpoints](#).

See Also

[deleteOutcome](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Delete occasion element

Description

Delete the occasion element.

Usage

```
deleteOccasionElement ()
```

Details

The occasion element impacts the definition of other elements and the simulation. As for other elements, the occasion element can be defined or imported, and it is saved with the Simulx project if calling [saveProject](#).

To check if an occasion element has been defined, please use [getOccasionElements](#). The occasion element may impact the definition of other elements. When deleting the occasion element, all other elements that depend on occasions are also deleted.

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Delete an outcome

Description

Delete an outcome.

Usage

```
deleteOutcome (name)
```

Arguments

name	(<i>character</i>) Outcome name
------	-----------------------------------

Details

Outcomes defined are created in the background and saved with the Simulx project if calling [saveProject](#).

To check which outcomes have been defined, please use [getOutcomes](#).

An outcome used in an endpoint cannot be deleted. The related endpoint must be deleted first with [deleteEndpoint](#).

See Also

[deleteEndpoint](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get covariate elements

Description

Get the list of all available covariate elements in the loaded project.

To use one of these elements in simulation, please add it to a simulation group with `setGroupElement`.

Usage

```
getCovariateElements()
```

Details

Covariate elements can be defined with `defineCovariateElement`, or created by importing a Monolix project with `importProject`. Elements defined are created in the background and saved with the Simulx project if calling `saveProject`. They can be deleted with `deleteElement`.

Each element is a list of

"inputType"	(character)	Type of input definition: can be "manual", "distribution" or "external".
"file"	(list)	if the inputType is external, list with path to the file and sheet in the excel (if relevant) . NULL else wise.
"data"	(data.frame)	Values of the element.

Note that:

- if the project was created from a model file, a covariate element Covariates is created with all values equal 1.
- if the project was created by importing a Monolix project,
 - a covariate element `mlx_Cov` is created with the values corresponding to the covariates values from the dataset of the Monolix project.
 - a covariate element `mlx_CovDist` is created with the values corresponding to the estimation of the distribution of covariates in the dataset of the Monolix project.

The "distribution" type of covariate elements can only be created in the GUI. The "manual" type corresponds to elements created manually in the GUI, or with a data frame in connectors. The external type corresponds to elements created from an external file in the GUI or with the connectors.

See Also

[defineCovariateElement](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get endpoint elements

Description

Get the list of all endpoints.

Usage

```
getEndpoints()
```

Details

Endpoints summarize the outcome values over all individuals, for each simulation group and each replicate. Endpoints are defined and calculated [as in Simulx GUI](#).

To compute the defined endpoints, use `runEndpoints` and get the results with `getEndpointsResults`.

To check if endpoints will be compared across simulation groups, use [getGroupComparisonSettings](#). If group comparison is relevant, the way the comparison will be done for each endpoint (eg if statistical test and which p-value) should be defined in the endpoint element.

Each element is a list of

- `outcome` (*character* or *list*) – Outcome on which the endpoint is based on. If one outcome, a string containing its name. If combined outcomes were used, a list with:
 - `names` (*vector* of *character*) – Vector of outcome names
 - `groupName` (*character*) – Name of the outcome combination
 - `operator` (*character*) – Way in which output should be combined. One of "and"/"or" (in case of boolean outcomes) or "min"/"max" (in other cases)
-

`metric` (*character*) – Calculation method for the endpoint. One of "arithmeticMean", "geometricMean" or "median" if value-based outcome. In case of event-based outcomes, "kaplanMeier" (median survival) and in case of boolean outcomes, "percentTrue".

- `groupComparison` (*list*) – Group comparison settings containing:
 - `type` (*character*) – One of "directComparison", "statisticalTest"
 - `operator` (*character*) – One of "!=", ">" or "<",
 - `threshold` (*double*) – A real number indicating the threshold for difference/oddsRatio
 - `pvalue` (*double*) – A real number indicating the p-value (if `type` is "statisticalTest")

See Also

[defineEndpoint](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get individual parameters elements

Description

Get the list of all available individual parameters elements for the simulation. To use one of these elements in simulation, please add it to a simulation group with [setGroupElement](#).

Usage

```
getIndividualElements()
```

Details

Individual parameters elements can be defined with `defineIndividualElement`, or created by importing a Monolix or PKanalix project. Elements defined are created in the background and saved with the Simulx project if calling `saveProject`. They can be deleted with `deleteElement`.

Each element is a list of

"inputType"	(<i>character</i>)	Type of input definition: can be "manual" or "external".
"file"	(<i>list</i>)	if the inputType is external, list with path to the file and sheet in the excel (if relevant) . NULL else wise.
"data"	(<i>data.frame</i>)	Values of the element.

If the Simulx project was created from a model file, an individual element IndivParameters is created with all values equal 1.

If the Simulx project was created by importing or exporting a Monolix project, the following individual elements are created:

- mlx_IndivInit is created with a vector of initial population parameters if the population parameters were not estimated.
- mlx_PopIndiv is created with a vector of estimated population parameters (without the covariate(s) impact(s)) if the population parameters were estimated.
- mlx_PopIndivCov is created with a table of estimated population parameters with covariate impact if the population parameters were estimated.
- mlx_EBEs is created with a table of estimated EBEs if the EBEs were estimated.
- mlx_CondMean is created with a table of estimated conditional mean for each individual if the conditional distribution task was run.
- mlx_CondDistSample is created with a table including the first sample from the conditional distribution for each individual if the conditional distribution task was run.

If the Simulx project was created by importing or exporting a PKanalix project, the following individual elements are created:

- an individual element pkx_IndivInit is created with a vector of initial parameters if the CA task has not run.
- an individual element pkx_Indiv is created with a table of individual parameters estimated in PKanalix if the CA task has run with the method "individual fit".
- an individual element pkx_IndivPooled is created with a vector of parameters estimated in PKanalix if the CA task has run with the method "pooled fit".
- an individual element pkx_IndivGeoMean is created with a vector of individual parameters corresponding to the geometric mean of individual parameters estimated in PKanalix if the CA task has run with the method "individual fit".

See Also

[defineIndividualElement](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get occasion elements

Description

Get the content of the occasion element that will be used for the simulation. The element will be automatically used for simulation and does not need to be added to a simulation group.

Usage

```
getOccasionElements()
```

Details

The occasion element can be defined with `defineOccasionElement`, or created during the import of a Monolix or PKanalix project with `importProject`. It can be deleted with `deleteOccasionElement`.

The element is a list of

"id"	(<i>character</i>)	Ids of the occasions
"names"	(<i>character</i>)	Name of the occasions. If empty, no occasions will be used in the project.
"times"	(<i>data.frame</i>)	Times of the occasions.
"occasions"	(<i>data.frame</i>)	Values of the element. If empty, no occasions will be used in the project.

The structure of the occasion elements impacts the definition of all other elements in the Simulx project. If the occasion element is subject-specific, other elements (parameters, covariates, treatments, outputs and regressors) must be either common over all subjects and all occasions, or they must be defined with subject-specific occasion-wise values as an external table (with id and occ columns), with the same occasion structure.

See Also

[defineOccasionElement](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get outcome elements

Description

Get the list of all available outcomes.

The function returns a list containing the following elements:

output	<i>(character)</i>	Name of the output element on which the outcome is based.
perOccasion	<i>(logical)</i>	If occasions are present in the simulation, it indicates if the outcome is calculated for each id or each occasion of each id.

Then, if the outcome is based on a continuous or categorical output:

relativeTo	<i>(list)</i>	List of elements that define reference settings: <ul style="list-style-type: none">• reference – one of "baseline", "min", "max", "minCurrentTime", "maxCurrentTime" or "customValue",• type – "ratio" or "difference",• value – if <i>reference</i> is "customValue".
processing	<i>(list)</i>	List of elements that define how the output will be processed:

- operator – one of "avg", "min", "max", "first", "last", "durationBelow", "durationAbove", "durationBetween" or "timePoint",
- type
 - if *operator* is "min" or "max", one of "value", "timeContinuous" or "timeEvent",
 - if *operator* is "durationBelow", "durationAbove" or "durationBetween", one of "cumulativeTime", "percentTime", "nbObs", "firstOccurrenceContinuous" or "firstOccurrenceEvent",
- value – vector of boundaries if *operator* is "durationBelow", "durationAbove" or "durationBetween", or time point if *operator* is "timePoint".

applyThreshold (*list*)

List of elements:

- operator – one of "==" , "!=" , ">=" , ">" , "<=" or "<" ,
- value – a real number indicating the threshold value.

Or if the outcome is based on a TTE output:

event (*list*)

List of arguments that define event settings:

- type – one of "timeOfEvents" (in case of repeated TTE), "hasAnEvent", "hasNoEvent" (in case of single TTE) or "numberOf" (in case of repeated TTE),
- rank – rank of the event of which time is the outcome (if *type* is "timeOfEvents").

Usage

```
getOutcomes ()
```

See Also

[defineOutcome](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get output elements

Description

Get the list of all available output elements for simulation.

Usage

```
getOutputElements()
```

Details

Output elements can be defined with `defineOutputElement`, or created by importing a Monolix or a PKanalix project. Elements defined are created in the background and saved with the Simulx project if calling `saveProject`. They can be deleted with `deleteElement`.

Each element is a list of

"output"	(character)	Output name.
"inputType"	(character)	Type of input definition: can be "manual" or "external".
"file"	(list)	if the inputType is external, list with path to the file and sheet in the excel (if relevant) . NULL else wise.
"data"	(data.frame)	Values of the element.

Importantly, all the variables in the model file can be used as an output. The user is not constrained to the ones defined in the OUTPUT section of the model.

If the project was created from a model file, an output element is created for each output of the section OUTPUT of the model, with regular grid from 0 to 100 by steps of 1.

If the project was created by importing a Monolix project, with for example Cc as a prediction and y as measurement,

-

an output element `mlx_y1` is created as an external file with the times of the output for each id in the original dataset of the Monolix project.

- an output element `mlx_Cc` is created with a regular grid starting from the first time to the final time measured in the original dataset.

If the project was created by importing a PKanalix project, with for example `Cc` as a prediction,

#

- an output element `pkx_Cc_OriginalTimes` is created as an external file with the times of the output for each id in the original dataset of the Monolix project.
- an output element `pkx_Cc_FineGrid` is created with a regular grid starting from the first time to the final time measured in the original dataset.

See Also

[defineOutputElement](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get population parameters elements

Description

Get the list of all available population parameters elements for the simulation. To use one of these elements in simulation, please add it to a simulation group with [setGroupElement](#).

Usage

```
getPopulationElements()
```

Details

Population parameters elements can be defined with `definePopulationElement`, or created at the import of a Monolix project with `importProject`. Elements defined are created in the background and saved with the Simulx project if calling `saveProject`. They can be deleted with `deleteElement`.

Each element is a list of

"inputType"	(<i>character</i>)	Type of input definition: can be "manual", "distribution" or "external".
"file"	(<i>list</i>)	if the inputType is external, list with path to the file and sheet in the excel (if relevant) . NULL else wise.
"data"	(<i>data.frame</i>)	Values of the element.

Notice that:

– if the project was created from a model file, a population element PopParameters is created with all values equal 1.

– if the project was created using by importing a Monolix project (with `importProject`),

- a population element `mlx_Pop` is created with the population parameters estimated in the Monolix project.
- if the parameters were not estimated in the Monolix project, a population element `mlx_PopInit` is created instead of `mlx_Pop`, with the initial values of the population parameters.
- a population element `mlx_PopUncertainSA` (resp. `mlx_PopUncertainLin`) is created which enables to sample population parameters using the covariance matrix of the estimates computed by Monolix if the Standard Error task (Estimation of the Fisher Information matrix) was performed by stochastic approximation (resp. by linearization). To sample several population parameter sets, this element needs to be used with replicates (`usetNbReplicates`).
- a population element `mlx_Typical` is created with the population parameters estimated in the Monolix projects and all omega parameters set to zero. It is useful to simulate a typical individual with different covariate values than the reference in the model.
- a population element `mlx_TypicalUncertainSA` (resp. `mlx_TypicalUncertainLin`) is created which is the same as `mlx_PopUncertain` but with all omegas set to zero to remove the inter-individual variability. It is useful to propagate the uncertainty of population parameters to the prediction of a typical individual.

See Also

[definePopulationElement](#)

[Click here to see examples](#)

[Simulx] Get regressor elements

Description

Get the list of all available regressor elements for simulation. To use one of these elements in simulation, please add it to a simulation group with `setGroupElement`. To simulate the model for times outside of the specified grid, last value carried forward interpolation is used.

Usage

```
getRegressorElements()
```

Details

Regressor elements can be defined with `defineRegressorElement`, or created by importing a Monolix or a PKanalix project with `importProject`. Elements defined are created in the background and saved with the Simulx project if calling `saveProject`. They can be deleted with `deleteElement`.

Each element is a list of

"inputType"	(<i>character</i>)	Type of input definition: can be "manual" or "external".
"file"	(<i>list</i>)	if the inputType is external, list with path to the file and sheet in the excel (if relevant) . NULL else wise.
"data"	(<i>data.frame</i>)	Values of the element.

Note that:

- if the project was created from a model file with regressors, a regressor element Regressors is created with one time point at 0 and all regressors equal 1.
- if the project was created by importing a Monolix or a PKanalix project with regressors, a regressor element `mlx_Reg` is created based on an external file with ids, times and regressor values and names read from the dataset of the Monolix project.

See Also

defineRegressorElement

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get regressors settings

Description

Get the regressors global settings.

- interpolation method: either "lastCarriedForward" or "linearInterpolation".

Usage

```
getRegressorsSettings ()
```

[Simulx] Get treatment elements

Description

Get the list of all available treatments elements for the simulation. To use one or several of these elements in simulation, please add it to a simulation group with [setGroupElement](#).

Usage

```
getTreatmentElements ()
```

Details

Treatment elements can be defined with `defineTreatmentElement`, or created by importing a Monolix or a PKanalix project with `importProject`. Elements defined are created in the background and saved with the Simulx project if calling `saveProject`. They can be deleted with `deleteElement`.

Each element is a list of

"inputType"	<i>(character)</i>	Type of input definition: can be "manual" or "external".
"file"	<i>(list)</i>	if the inputType is external, list with path to the file and sheet in the excel (if relevant) . NULL else wise.

"data"	(<i>data.frame</i>)	Values of the element as defined in defineTreatmentElement .
"admID"	(<i>integer</i>)	Administration id as defined in defineTreatmentElement .
"scale"	(<i>integer</i>)	information on scaling by covariates if defined in defineTreatmentElement .

If the project was created from a model file, no treatment element is added.

If the project was created by importing a Monolix or a PKanalix project, for each administration type present in the original project (dataset column tagged as administration id), an individual element `mlx_adm` is created as an external file with the dosing times and amounts from the original dataset in the Monolix or PKanalix project.

See Also

[defineTreatmentElement](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Edit regressors settings

Description

Edit the regressors global settings.

Usage

```
setRegressorsSettings (method)
```

Arguments

<code>method</code>	(<i>character</i>): interpolation method for the regressors. Possible values: "lastCarriedForward" (default), "linearInterpolation".
---------------------	--

[Simulx] Get group comparison settings

Description

Set settings related to the comparison of endpoints across groups.

Usage

```
getGroupComparisonSettings()
```

Details

Endpoints summarize the outcome values over all individuals, for each simulation group and each replicate. Endpoints are defined with `defineEndpoint` and can be compared across groups [as in Simulx GUI](#).

`getGroupComparisonSettings` enables to check if endpoints will be compared across simulation groups and which group will be used as a reference. Group comparison is performed during the Endpoints task.

See Also

[setGroupComparisonSettings](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Set group comparison settings

Description

Set settings related to the comparison of endpoints across groups.

Usage

```
setGroupComparisonSettings(referenceGroup = NULL, enable = TRUE)
```

Arguments

<code>referenceGroup</code>	<i>(character)</i> (optional) Group to use as reference.
<code>enable</code>	<i>(logical)</i> (optional) Enable group comparison, TRUE by default.

Details

Endpoints summarize the outcome values over all individuals, for each simulation group and each replicate. Endpoints are defined with `defineEndpoint` and can be compared across groups [as in Simulx GUI](#).

`setGroupComparisonSettings` enables to specify if endpoints should be compared across simulation groups and which group to use as a reference. Group comparison is performed during the Endpoints task.

See Also

[getGroupComparisonSettings](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Initialize lixoftConnectors API

Description

Initialize lixoftConnectors API for a given software.

Usage

```
initializeLixoftConnectors(software = "monolix", path = "", force
```

Arguments

<code>software</code>	<i>(character)</i> [optional] Name of the software to be loaded. By default, "monolix" software is used.
<code>path</code>	<i>(character)</i> [optional] Path to installation directory of the Lixoft suite. If lixoftConnectors library is not already loaded and no path is given, the directory written in the lixoft.ini file is used for initialization.
<code>force</code>	<i>(logical)</i> [optional] Should software switch security be overpassed or not. Equals FALSE by default.

Value

A logical equaling TRUE if the initialization has been successful and FALSE if not.

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Add an INDIVIDUAL block to a structural model

Description

Add an INDIVIDUAL block to a structural model.

The structural model can be either a txt file or a library model, and the resulting model is written to another a file.

Usage

```
addIndividualBlock(sourceModelFile, targetModelFile, force = FALSE)
```

Arguments

<code>sourceModelFile</code>	<i>(character)</i> Path to the model file. Can be absolute or relative. Alternatively, the name of a library model (starting with "lib:"); see getLibraryModelName for a list of available models.
<code>targetModelFile</code>	<i>(character)</i> Path to the resulting file. Can be absolute or relative.
<code>force</code>	<i>(logical)</i> [optional] Should the resulting file be overwritten if already existing? Default: FALSE.

Details

Adding an **INDIVIDUAL block** enables to add inter-individual variability to all parameters of a structural model.

For every parameter `theta` available in the structural model file, population parameters `theta_pop` (typical value) and `omega_theta` (standard deviation of random effects) are added to the model, and lognormal distributions are used by default.

If the structural model is a library model, the distributions are set to normal, lognormal or logitnormal depending on the meaning of the parameter.

The resulting file can then be used as a starting point to remove IIV from some parameters, change parameter distributions, or add correlations if needed, and the final model can then be used to create a new Simulx project with [newProject](#).

More details on the syntax of the INDIVIDUAL block can be found [here](#).

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Get Lixoft demos path

Description

Get the path to the demo projects. The path depends on the software used to initialize the connectors with [initializeLixoftConnectors](#).

Usage

```
getDemoPath()
```

Value

The Lixoft demos path corresponding to the currently active software.

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get lines added to the model.

Description

Get the lines that were added to a model with [setAddLines](#) (or in the GUI).

Usage

```
getAddLines()
```

Details

Additional equations can be added to the model file [as in Simulx GUI](#).

It is useful in case of import from Monolix or PKanalix, in order to add equations to the model, eg to compute an additional variable, without modifying the model file used for estimation and without impacting elements already defined.

See Also

setAddLines

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Add lines to the model.

Description

Add equations to the structural model.

Usage

```
setAddLines (lines)
```

Arguments

<code>lines</code>	(<i>character</i>) Additional lines to define.
--------------------	--

Details

Additional equations can be added to the model file [as in Simulx GUI](#).

It is useful in case of import from Monolix or PKanalix, in order to add equations to the model, eg to compute an additional variable, without modifying the model file used for estimation.

All variables defined in the add lines will be available as an output.

See Also

[getAddLines](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Export current project to Monolix, PKanalix or Simulx

Description

Export the current project to another application of the MonolixSuite, and load the exported project.

NOTE: This action switches the current session to the target software. Current unsaved modifications will be lost.

The extensions are .mlxtran for Monolix, .pkx for PKanalix, .smlx for Simulx and .dpx for Datxplore.

WARNING: R is sensitive between ‘\’ and ‘/’, only ‘/’ can be used.

Usage

```
exportProject(settings, force = F)
```

Arguments

	<p>(<i>character</i>) Export settings:</p> <ul style="list-style-type: none">• targetSoftware (<i>character</i>) Target software ("monolix" "simulx" "pkanalix")• filesNextToProject (<i>logical</i>) [optional][Monolix – PKanalix] Save data and/or structural model file next to exported project ([TRUE] FALSE). Forced to TRUE for Simulx.• dataFilePath (<i>character</i>) [optional][Monolix – Simulx] Path (filesNextToProject == FALSE) or name (filesNextToProject == TRUE) of the exported data file. Available only for generated datasets in Monolix (vpc, individual fits)
settings	<ul style="list-style-type: none">• dataFileType (<i>character</i>) [optional][Monolix] Dataset used in the exported project ("original" "vpc" "individualFits")• modelFileName (<i>character</i>) [optional][Simulx] Name of the exported model file.• exportedUnusedCovariates (<i>list</i>) [optional][Monolix – PKanalix => Simulx] Covariates not used in the individual model (if present) to be exported to Simulx. A list with:<ul style="list-style-type: none">• all (<i>logical</i>) Export all unused covariates. FALSE by default.• name (<i>vector<character></i>) List of exported unused covariate names. Required if all == FALSE, ignored if not.
force	(<i>logical</i>) [optional] Should software switch security be overpassed or not. Equals FALSE by default.

Details

At export, a new project is created in a temporary folder. By default, the file is created with a project setting `filesNextToProject = TRUE`, which means that file dependencies such as data and model files are copied and kept next to the new project (or in the result folder for Simulx). This new project can be saved to the desired location with `saveProject`.

Exporting a Monolix or a PKanalix project to Simulx automatically creates elements that can be used for simulation, [exactly as in the GUI](#).

To see which elements of some type have been created in the new project, you can use the `get..Element` functions: `getOccasionElements`, `getPopulationElements`, `getIndividualElements`, `getCovariateElements`, `getTreatmentElements`, `getOutputElements`, `getRegressorElements`.

See Also

[newProject](#), [loadProject](#), [importProject](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Get a library model's content.

Description

Get the content of a library model.

Usage

```
getLibraryModelContent(filename, print = TRUE)
```

Arguments

<code>filename</code>	(<i>character</i>) The filename of the requested model. Can start with "lib:", end with ".txt", but neither are mandatory.
-----------------------	--

<code>print</code>	(<i>logical</i>) If TRUE (default), model's content is printed with human-readable line breaks (alongside regular output with "\n").
--------------------	--

Value

The model's content as a single string.

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Get the name of a library model given a list of library filters.

Description

Get the name of a library model given a list of library filters.

Usage

```
getLibraryModelName(library, filters = list())
```

Arguments

library (*character*) One of the MonolixSuite library of models. Possible values are "pk", "pd", "pkpd", "pkdoubleabs", "pm", "tmdd", "tte", "count" and "tgi".

filters (*list(name = character)*) Named list of filters (optional), format: list(filterKey = "filterValue", ...). Default empty list. Since available filters are not in any particular order, filterKey should always be stated.

Details

Models can be loaded from a library based on a selection of filters [as in PKanalix, Monolix and Simulx GUI](#). For a complete description of each model library, and guidelines on how to select models, please visit <https://mlxtran.lixoft.com/model-libraries/>.

getLibraryModelName enables to get the name of the model to be loaded. You can then use it in [setStructuralModel](#) or [newProject](#) to load the model in an existing or in a new project.

All possible keys and values for each of the libraries are listed below.

PK library

key	values
-----	--------

administration	bolus, infusion, oral, oralBolus
delay	noDelay, lagTime, transitCompartments
absorption	zeroOrder, firstOrder
distribution	1compartment, 2compartments, 3compartments
elimination	linear, MichaelisMenten
parametrization	rate, clearance, hybridConstants
bioavailability	true, false

PD library

key	values
response	immediate, turnover
drugAction	linear, logarithmic, quadratic, Emax, I _{max} , productionInhibition, degradationInhibition, degradationStimulation, productionStimulation
baseline	const, 1-exp, exp, linear, null
inhibition	partialInhibition, fullInhibition
sigmoidicity	true, false

PKPD library

key	values
administration	bolus, infusion, oral, oralBolus
delay	noDelay, lagTime, transitCompartments

absorption	zeroOrder, firstOrder
distribution	1compartment, 2compartments, 3compartments
elimination	linear, MichaelisMenten
parametrization	rate, clearance
bioavailability	true, false
response	direct, effectCompartment, turnover
drugAction	E _{max} , I _{max} , productionInhibition, degradationInhibition, degradationStimulation, productionStimulation
baseline	const, null
inhibition	partialInhibition, fullInhibition
sigmoidicity	true, false

PK double absorption library

key	values
firstAbsorption	zeroOrder, firstOrder
firstDelay	noDelay, lagTime, transitCompartments
secondAbsorption	zeroOrder, firstOrder
secondDelay	noDelay, lagTime, transitCompartments
absorptionOrder	simultaneous, sequential
forceLongerDelay	true, false
distribution	1compartment, 2compartments, 3compartments

elimination	linear, MichaelisMenten
parametrization	rate, clearance

Parent-metabolite library

key	values
administration	bolus, infusion, oral, oralBolus
firstPassEffect	noFirstPassEffect, withDoseApportionment, withoutDoseApportionment
delay	noDelay, lagTime, transitCompartments
absorption	zeroOrder, firstOrder
transformation	unidirectional, bidirectional
parametrization	rate, clearance
parentDistribution	1compartment, 2compartments, 3compartments
parentElimination	linear, MichaelisMenten
metaboliteDistribution	1compartment, 2compartments, 3compartments
metaboliteElimination	linear, MichaelisMenten

TMDD library

key	values
administration	bolus, infusion, oral, oralBolus
delay	noDelay, lagTime, transitCompartments
absorption	zeroOrder, firstOrder

distribution	1compartment, 2compartments, 3compartments
tmddApproximation	MichaelisMenten, QE, QSS, full, Wagner, constantRtot, constantRtotIB, irreversibleBinding
output	totalLigandLtot, freeLigandL
parametrization	rate, clearance

TTE library

key	values
tteModel	exponential, Weibull, Gompertz, loglogistic, uniform, gamma, generalizedGamma
delay	true, false
numberOfEvents	singleEvent, repeatedEvents
typeOfEvent	intervalCensored, exact
dummyParameter	true, false

Count library

key	values
countDistribution	Poisson, binomial, negativeBinomial, betaBinomial, generalizedPoisson, geometric, hypergeometric, logarithmic, Bernoulli
zeroInflation	true, false
timeEvolution	constant, linear, exponential, Emax, Hill

parametrization	probabilityOfSuccess, averageNumberOfCounts
-----------------	---

TGI library

key	values
shortcut	ClaretExponential, Simeoni, Stein, Wang, Bonate, Ribba, twoPopulation
initialTumorSize	asParameter, asRegressor
kinetics	true, false
model	linear, quadratic, exponential, generalizedExponential, exponentialLinear, Simeoni, Koch, logistic, generalizedLogistic, SimeoniLogisticHybrid, Gompertz, exponentialGompertz, vonBertalanffy, generalizedVonBertalanffy
additionalFeature	none, angiogenesis, immuneDynamics
treatment	none, pkModel, exposureAsRegressor, startAtZero, startTimeAsRegressor, armAsRegressor
killingHypothesis	logKill, NortonSimon
dynamics	firstOrder, MichaelisMenten, MichaelisMentenHill, exponentialKill, constant
resistance	ClaretExponential, resistantCells, none
delay	signalDistribution, cellDistribution, none

additionalTreatmentEffect

none, angiogenesisInhibition,
immuneEffectorDecay

Value

Name of the filtered model, or vector of names of the available models if not all filters were selected. Names start with "lib:".

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Get structural model file

Description

Get the model file for the structural model used in the current project.

Usage

```
getStructuralModel()
```

Details

For Simulx, this function will return the path to the structural model only if the project was imported from Monolix, and the path to the full custom model otherwise.

Note that a custom model in Simulx may include also a statistical part.

For Simulx, there is no associated function `getStructuralModel()` because setting a new model is equivalent to creating a new project. Use [newProject](#) instead.

If a model was loaded from the libraries, the returned character is not a path, but the name of the library model, such as "lib:model_name.txt". To see the content of a library model, use [getLibraryModelContent](#).

Value

The path to the structural model file.

See Also

For Monolix and PKanalix only: [setStructuralModel](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Import project from Datxplore, Monolix or PKanalix

Description

Import a Monolix or a PKanalix project into the currently running application initialized in the connectors.

The extensions are .mlxtran for Monolix, .pkx for PKanalix, .smlx for Simulx and .dpx for Datxplore.

WARNING: R is sensitive between ‘\’ and ‘/’, only ‘/’ can be used.

Allowed import sources are:

CURRENT SOFTWARE	ALLOWED IMPORTS
Monolix	PKanalix
PKanalix	Monolix, Datxplore
Simulx	Monolix, PKanalix.

Usage

```
importProject(projectFile)
```

Arguments

<code>projectFile</code>	(<i>character</i>) Path to the project file. Can be absolute or relative to the current working directory.
--------------------------	--

Details

At import, a new project is created in a temporary folder with a project setting `filesNextToProject = TRUE`, which means that file dependencies such as data and model files are copied and kept next to the new project (or in the result folder for Simulx). This new project can be saved to the desired location with `saveProject`.

Simulx projects can only be exported, not imported. To export a Simulx project to another application, please load the Simulx project with the Simulx connectors and use `exportProject`.

Importing a Monolix or a PKanalix project into Simulx automatically creates elements that can be used for simulation, [exactly as in the GUI](#).

To see which elements of some type have been created in the new project, you can use the `get..Element` functions:

```
getOccasionElements, getPopulationElements, getPopulationElements,  
getIndividualElements,  
getCovariateElements, getTreatmentElements, getOutputElements,  
getRegressorElements.
```

See Also

[saveProject](#), [exportProject](#)

Click here to see examples	
--	--

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Get current project load status.

Description

Get a logical saying if a project is currently loaded.

Usage

```
isProjectLoaded()
```

Value

TRUE if a project is currently loaded, FALSE otherwise

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Load project from file

Description

Load a project in the currently running application initialized in the connectors. The extensions are .mlxtran for Monolix, .pkx for PKanalix, and .smlx for Simulx.

WARNING: R is sensitive between ‘\’ and ‘/’, only ‘/’ can be used.

Usage

```
loadProject(projectFile)
```

Arguments

<code>projectFile</code>	(<i>character</i>) Path to the project file. Can be absolute or relative to the current working directory.
--------------------------	--

See Also

[saveProject](#), [importProject](#), [exportProject](#), [newProject](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Create a new project

Description

Create a new project. New projects can be created in the connectors as in [PKanalix](#), [Monolix](#) or [Simulx](#) GUI. The creation of a new project requires a dataset in PKanalix, a dataset and a model in Monolix, and a model in Simulx.

Usage

```
newProject(modelFile = NULL, data = NULL, mapping = NULL)
```

Arguments

modelFile	<p>(<i>character</i>) Path to the model file. Mandatory for Monolix and Simulx, optional for PKanalix (used only for the CA part). Can be absolute or relative to the current working directory.</p> <p>To use a model from the libraries, you can find the model name with <code>getLibraryModelName</code> and set <code>modelFile = "lib:modelName.txt"</code> with the name obtained.</p> <p>To simulate inter-individual variability in Simulx with a new project, the model file has to include the statistical model, contrary to Monolix and PKanalix for which the model file only contains the structural model. Check here in detail how to write such a model from scratch.</p>
data	<p>(<i>list</i>) Structure describing the data. Mandatory for Monolix and PKanalix.</p> <ul style="list-style-type: none">• <code>dataFile</code> (<i>character</i>): Path to the data file (csv, xls, xlsx, sas7bdat, xpt or txt). Can be absolute or relative to the current working directory.• <code>sheet</code> [optional] (<i>character</i>): Name of the sheet in xlsx/xls file. If not provided, the first sheet is used.• <code>headerTypes</code> (<i>vector<character></i>): A vector of header types. The possible header types are: "ignore", "id", "time", "observation", "amount", "contcov", "catcov", "occ", "evid", "mdv", "obsid", "cens", "limit", "regressor", "nominaltime", "admid", "rate", "tinf", "ss", "ii", "addl", "date". Notice that these are not exactly the types displayed in the interface, they are shortcuts.• <code>observationTypes</code> [optional] (<i>list</i>): A list giving the type of each observation present in the data file. If there is only one y-type, the corresponding observation name can be omitted. The possible observation types are "continuous", "discrete", and "event".• <code>nbSSDoses</code> (<i>integer</i>): Number of doses (if there is a SS column for steady-state).• <code>regressorsSettings</code> [optional] (<i>character</i>): Regressors interpolation method. Either 'lastCarriedForward' (default) or 'linearInterpolation'.
mapping	<p>[optional](<i>list</i>): A list of lists representing a link between observation types and model outputs. Each list contains:</p> <ul style="list-style-type: none">•

obsId (*character*) Name of observation id present in the dataset. It corresponds to the content of column tagged as "obsid" in case of several obs ids, or to the header of the column tagged as "observation" otherwise

- observationName (*character*) Name of the model prediction listed in the output= line of the structural model
- modelOutput [Monolix] (*character*) Name of the observation, e.g "y1" (for continuous observations only)

Details

Note: instead of creating a project from scratch, it is also possible in Monolix and PKanalix to load an existing project with `loadProject` or `importProject` and change the dataset or the model with `setData` or `setStructuralModel`.

See Also

`newProject` `saveProject`

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Save current project

Description

Save the current project as a file that can be reloaded in the connectors or in the GUI.

Usage

```
saveProject(projectFile = "")
```

Arguments

<code>projectFile</code>	[optional](<i>character</i>) Path where to save a copy of the current mlxtran model. Can be absolute or relative to the current working directory. If no path is given, the file used to build the current configuration is updated.
--------------------------	---

Details

The extensions are .mlxtran for Monolix, .pkx for PKanalix, and .smlx for Simulx.

WARNING: R is sensitive between ‘\’ and ‘/’, only ‘/’ can be used.

If the project setting "userfilesnexttoproject" is set to TRUE with `setProjectSettings`, all file dependencies such as model, data or external files are saved next to the project for Monolix and PKanalix, and in the result folder for Simulx.

See Also

[newProject](#) [loadProject](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Share project.

Description

Create a zip archive file from current project and its results.

Usage

```
shareProject (archiveFile)
```

Arguments

`archiveFile` (character) Path to the .zip archive file to create.

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Get console mode

Description

Get console mode, ie volume of output after running estimation tasks. Possible verbosity levels are:

"none"	no output
"basic"	at the end of each algorithm, associated results are displayed
"complete"	each algorithm iteration and/or status is displayed

Usage

```
getConsoleMode()
```

Value

A string corresponding to current console mode

See Also

[setConsoleMode](#)

[Monolix – PKanalix – Simulx] Get project preferences

Description

Get a summary of the project preferences. Preferences are:

"relativepath"	<i>(logical)</i>	Use relative path for save/load operations.
"threads"	<i>(integer > 0)</i>	Number of threads.
"temporarydirectory"	<i>(character)</i>	Path to the directory used to save temporary files.

"usebinarydataset"	(logical)	Save dataset as binary file to speed project reload up.
"timestamping"	(logical)	Create an archive containing result files after each run.
"delimiter"	(character)	Character use as delimiter in exported result files.
"reportingrenamings"	(list("label" = "alias">))	For each label, an alias to be use at report generation (using <code>generateReport</code>).
"exportchartsdata"	(logical)	Should charts data be exported.
"savebinarychartsdata"	(logical)	[Monolix] Save charts simulations as binray file to speed charts creation up.
"exportchartsdatasets"	(logical)	[Monolix] Should charts datasets be exported if possible.
"exportvpcsimulations"	(logical)	[Monolix] Should vpc simulations be exported if possible.
"exportsimulationfiles"	(logical)	[Simulx] Should simulation results files be exported.
"headeraliases"	(list("header" = vector<character>))	For each header, the list of the recognized aliases.
"ncaparameters"	(vector<character>)	[PKanalix] Defaulty computed NCA parameters.
"units"	(list("type" = character))	[PKanalix] Time, amount and/or volume units.

Usage

```
getPreferences(...)
```

Arguments

... [optional] (character) Name of the preference whose value should be displayed. If no argument is provided, all the preferences are returned.

Value

A list with each preference name mapped to its current value.

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Get project settings

Description

Get a summary of the project settings.

Associated settings for Monolix projects are:

"directory"	(character)	Path to the folder where simulation results will be saved. It should be a writable directory.
"exportResults"	(logical)	Should results be exported.
"seed"	(0 < integer < 2147483647)	Seed used by random generators.
"grid"	(integer)	Number of points for the continuous simulation grid.
"nbSimulations"	(integer)	Number of simulations.
"dataandmodelnexttoproject"	(logical)	Should data and model files be saved next to project.
"project"	(character)	Path to the Monolix project.

Associated settings for PKanalix projects are:

"directory"	<i>(character)</i>	Path to the folder where simulation results will be saved. It should be a writable directory.
"seed"	<i>(0 < integer < 2147483647)</i>	Seed used by random generators.
"datanexttoproject"	<i>(logical)</i>	Should data and model (in case of CA) files be saved next to project.

Associated settings for Simulx projects are:

"directory"	<i>(character)</i>	Path to the folder where simulation results will be saved. It should be a writable directory.
"seed"	<i>(0 < integer < 2147483647)</i>	Seed used by random generators.
"userfilesnexttoproject"	<i>(logical)</i>	Should user files be saved next to project.

Usage

```
getProjectSettings(...)
```

Arguments

... [optional] (character) Name of the settings whose value should be displayed.
If no argument is provided, all the settings are returned.

Value

A list with each setting name mapped to its current value.

See Also

[setProjectSettings](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Set console mode

Description

Set console mode, ie volume of output after running estimation tasks. Possible verbosity levels are:

"none"	no output
"basic"	for each algorithm, display current iteration then associated results at algorithm end
"complete"	display all iterations then associated results at algorithm end

Usage

```
setConsoleMode(mode)
```

Arguments

mode (*character*) Accepted values are: "none" [default], "basic", "complete"

See Also

[getConsoleMode](#)

[Monolix – PKanalix – Simulx] Set preferences Description

Set the value of one or several of the project preferences. Preferences are:

"relativepath"	(logical)	Use relative path for save/load operations.
"threads"	(integer > 0)	Number of threads.
"temporarydirectory"	(character)	Path to the directory used to save temporary files.
"usebinarydataset"	(logical)	Save dataset as binary file to speed project reload up.
"timestamping"	(logical)	Create an archive containing result files after each run.
"delimiter"	(character)	Character use as delimiter in exported result files.
"reportingrenamings"	(list("label" = "alias"))	For each label, an alias to be use at report generation (using generateReport).
"exportchartsdata"	(logical)	Should charts data be exported.
"savebinarychartsdata"	(logical)	[Monolix] Save charts simulations as binary file to speed charts creation up.
"exportchartsdatasets"	(logical)	[Monolix] Should charts datasets be exported if possible.
"exportvpcsimulations"	(logical)	[Monolix] Should vpc simulations be exported if possible.

"exportsimulationfiles"	(<i>logical</i>)	[Simulx] Should simulation results files be exported.
"headeraliases"	(<i>list("header" = vector<character>)</i>)	For each header, the list of the recognized aliases.
"ncaparameters"	(<i>vector<character></i>)	[PKanalix] Defaultly computed NCA parameters.
"units"	(<i>list("type" = character)</i>)	[PKanalix] Time, amount and/or volume units.

Usage

```
setPreferences(...)
```

Arguments

... A collection of comma-separated pairs {preferenceName = settingValue}.

See Also

[getPreferences](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Set project settings

Description

Set the value of one or several of the settings of the project.

Associated settings for Monolix projects are:

"directory"	(<i>character</i>)	Path to the folder where simulation results will be saved. It should be a writable directory.
-------------	----------------------	---

"exportResults"	(logical)	Should results be exported.
"seed"	(0 < integer < 2147483647)	Seed used by random generators.
"grid"	(integer)	Number of points for the continuous simulation grid.
"nbSimulations"	(integer)	Number of simulations.
"dataandmodelnexttoproject"	(logical)	Should data and model files be saved next to project.

Associated settings for PKanalix projects are:

"directory"	(character)	Path to the folder where simulation results will be saved. It should be a writable directory.
"dataNextToProject"	(logical)	Should data and model (in case of CA) files be saved next to project.
"seed"	(0 < integer < 2147483647)	Seed used by random generators.

Associated settings for Simulx projects are:

"directory"	(character)	Path to the folder where simulation results will be saved. It should be a writable directory.
"seed"	(0 < integer < 2147483647)	Seed used by random generators.
"userfilesnexttoproject"	(logical)	Should user files be saved next to project.

Usage

```
setProjectSettings(...)
```

Arguments

... A collection of comma-separated pairs {settingName = settingValue}.

See Also

[getProjectSettings](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Export simulated data

Description

Export the simulated dataset into a MonolixSuite compatible format.

It contains treatment information and simulation results and can be generated only when simulation results are available.

Usage

```
exportSimulatedData(filePath = "")
```

Arguments

`filePath` [optional](*character*) Custom path for the exported file. By default, it is written in the results folder of the current project, next to simulation results files. The default file name is simulatedData.csv.

Details

The generated dataset can then be loaded in Datxplorer, PKanalix or Monolix.

Note: to export the simulated data and load it into another application in a single line, you can also use [exportProject](#).

See Also

[exportProject](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get endpoints results

Description

Get the results of the outcomes & endpoints task. Outcomes, endpoints and group comparisons are calculated [as in Simulx GUI](#) with the task `runEndpoints`. The output is a list with outcomes, endpoints and comparison results if they have been computed.

Usage

```
getEndpointsResults()
```

See Also

[runEndpoints](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get simulation results

Description

Get the results of the simulation.

The output is a list of four elements: `res`, `individualParameters`, `populationParameters` and `doses`, and optionally `id_mapping` for external id tables.

- `res` corresponds to the list of the output(s) of the simulation. It includes a data frame for each output with the columns `id`, `original_id` (if an external element with `id` column is used in the simulation), `time`, `outputName`, and `group` (corresponding to the group name if there are several groups).
- `individualParameters` corresponds to a list of data frames, one for each simulation group, with the individual parameters sampled in the group.
- `populationParameters` corresponds to a list of data frames, one for each simulation group, with the population parameters sampled in the group (can be several sets only in case of replicates).
-

doses corresponds to a list of data frames, one for each simulation group, with the dose amounts administered to each individual in the group.

- `id_mapping` maps the original ids used in the external tables to the simulated ids.

Usage

```
getSimulationResults(id = NULL, rep = NULL)
```

Arguments

`id` [optional](*character*) If provided, results are retrieved only for this id.

`rep` [optional](*integer*) If provided, results are retrieved only for this replicate.

See Also

[runSimulation](#)

[Click here to see examples](#)

[Back to the list, PKanalix API, Monolix API, Simulx API.](#)

[Monolix – PKanalix – Simulx] Compute the charts data

Description

Compute (if needed) and export the charts data of a given plot or, if not specified, all the available project plots.

Usage

```
computeChartsData(plot = NULL, output = NULL, exportVPCsimulations = FALSE)
```

Arguments

`plot`

	<p>(<i>character</i>) [optional][Monolix] Plot type. If not specified, all the available project plots will be considered. Available plots: bivariatedataviewer, covariateviewer, outputplot, indfits, obspred, residualsscatter, residualsdistribution, vpc, npc, predictiondistribution, parameterdistribution, randomeffects, covariancemodeldiagnosis, covariatemodeldiagnosis, likelihoodcontribution, fisher, saemresults, condmeanresults, likelihoodresults.</p>
output	<p>(<i>character</i>) [optional][Monolix] Plotted output (depending on the software, it can represent an observation, a simulation output, ...). By default, all available outputs are considered.</p>
exportVPCsimulations	<p>(<i>logical</i>) [optional][Monolix] Should VPC simulations be exported if available. Equals FALSE by default. NOTE: If 'plot' argument is not provided, 'output' and 'task' arguments are ignored.</p>

Details

computeChartsData can be used to compute and export the charts data for plots available in the graphical user interface as in [Monolix](#), [PKanalix](#) or [Simulx](#), when you export > export charts data.

The exported charts data is saved as txt files in the result folder, in the ChartsData subfolder.

Notice that it does not impact the current scenario.

To get a ggplot equivalent to the plot in the GUI, but customizable in R with the ggplot2 library, better use one of the plot... functions available in the connectors for Monolix and PKanalix (not available for Simulx). To get the charts data for one of these plot functions as a dataframe, you can use [getChartsData](#).

See Also

[getChartsData](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Get current scenario

Description

Get the list of tasks that will be run at the next call to `runScenario`. For Monolix, get in addition the associated method (linearization true or false), and the associated list of plots.

Usage

```
getScenario()
```

Details

For Monolix, `getScenario` returns a given list of tasks, the linearization option and the list of plots.

Every task in the list is associated to a logical

NOTE: Within a MONOLIX scenario, the order according to which the different algorithms are run is fixed:

Algorithm	Algorithm Keyword
Population Parameter Estimation	"populationParameterEstimation"
Conditional Mode Estimation (EBEs)	"conditionalModeEstimation"
Sampling from the Conditional Distribution	"conditionalDistributionSampling"
Standard Error and Fisher Information Matrix Estimation	"standardErrorEstimation"
LogLikelihood Estimation	"logLikelihoodEstimation"
Plots	"plots"

For PKanalix, `getScenario` returns a given list of tasks.

Every task in the list is associated to a logical

NOTE: Within a PKanalix scenario, the order according to which the different algorithms are run is fixed:

Algorithm	Algorithm keyword
Non Compartmental Analysis	"nca"
Bioequivalence estimation	"be"

For Simulx, `setScenario` returns a given list of tasks.

Every task in the list is associated to a logical

NOTE: Within a Simulx scenario, the order according to which the different algorithms are run is fixed:

Algorithm	Algorithm keyword
Simulation	"simulation"
Outcomes and endpoints	"endpoints"

Note: every task can also be run separately with a specific function, such as `runSimulation` in Simulx, `runEstimation` in Monolix. The CA task in PKanalix cannot be part of a scenario, it must be run with `runCAEstimation`.

Value

The list of tasks that corresponds to the current scenario, indexed by task names.

See Also

[setScenario](#)

Click here to see examples	
----------------------------	--

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Run scenario Description

Run the scenario that has been set with `setScenario`.

Usage

```
runScenario()
```

Details

A scenario is a list of tasks to be run. Setting the scenario is equivalent to selecting tasks in [Monolix](#), [PKanalix](#) or [Simulx](#) GUI that will be performed when clicking on RUN.

If `exportchartsdata` preference is set to TRUE with [setPreferences](#), `runscenario` generates the charts data in the result folder.

Every task can also be run separately with a specific function, such as `runSimulation` in [Simulx](#), `runEstimation` in [Monolix](#). The CA task in [PKanalix](#) cannot be part of a scenario, it must be run with `runCAEstimation`.

See Also

[setScenario](#) [getScenario](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Monolix – PKanalix – Simulx] Set scenario

Description

Clear the current scenario and build a new one from a given list of tasks.

Usage

```
setScenario(...)
```

Arguments

... A list of tasks as previously defined

Details

A scenario is a list of tasks to be run by `runScenario`. Setting the scenario is equivalent to selecting tasks in [Monolix](#), [PKanalix](#) or [Simulx](#) GUI that will be performed when clicking on RUN.

For Monolix, setScenario requires a given list of tasks, the linearization option and the list of plots.

Every task in the list should be associated to a logical

NOTE: by default the logical is FALSE, thus, the user can only state what will run during the scenario.

NOTE: Within a MONOLIX scenario, the order according to which the different algorithms are run is fixed:

Algorithm	Algorithm Keyword
Population Parameter Estimation	"populationParameterEstimation"
Conditional Mode Estimation (EBEs)	"conditionalModeEstimation"
Sampling from the Conditional Distribution	"conditionalDistributionSampling"
Standard Error and Fisher Information Matrix Estimation	"standardErrorEstimation"
LogLikelihood Estimation	"logLikelihoodEstimation"
Plots	"plots"

For PKanalix, setScenario requires a given list of tasks.

Every task in the list should be associated to a logical

NOTE: By default the logical is FALSE, thus, the user can only state what will run during the scenario.

NOTE: Within a PKanalix scenario, the order according to which the different algorithms are run is fixed:

Algorithm	Algorithm keyword
Non Compartmental Analysis	"nca"
Bioequivalence estimation	"be"

For Simulx, setScenario requires a given list of tasks.

Every task in the list should be associated to a logical

NOTE: By default the logical is FALSE, thus, the user can only state what will run during the scenario.

NOTE: Within a Simulx scenario, the order according to which the different algorithms are run is fixed:

Algorithm	Algorithm keyword
Simulation	"simulation"
Outcomes and endpoints	"endpoints"

Note: every task can also be run separately with a specific function, such as runSimulation in Simulx, runEstimation in Monolix. The CA task in PKanalix cannot be part of a scenario, it must be run with runCAEstimation.

See Also

[getScenario](#).

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Add simulation group

Description

Add a new simulation group.

Usage

```
addGroup (group)
```

Arguments

group	(<i>character</i>) Name of the group to add.
-------	--

Details

Simulation groups can be added to the simulation [as in Simulx GUI](#). By default, the elements of the newly added group are the same as the first simulation group. To check which elements have been set for this group, please use [getGroups](#). To change a group element, use [setGroupElement](#).

Note: when a Simulx project is created, a first group is created by default with the name "simulationGroup1".

See Also

[getGroups](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get remaining parameters for a simulation group

Description

Get the values of the remaining parameters (typically the error model parameters) for a group.

Usage

```
getGroupRemaining(group)
```

Arguments

<code>group</code>	<i>(character)</i> Group name
--------------------	-------------------------------

Details

Remaining parameters are all parameters that appear in the structural model (in the input line of [LONGITUDINAL]) and are neither individual parameters nor regressors. They are typically error model parameters.

If an individual parameters element is selected for simulation, and the model includes remaining parameters, it is possible to set their values with [setGroupRemaining](#).

It typically enables to make a simulation with measurement noise, with an individual element.

These error model parameters will impact the simulation only if a noisy observation (from the DEFINITION section of the [LONGITUDINAL] block) is set as output element (instead of a smooth prediction in OUTPUT or variable in EQUATION).

If a population parameters element is selected, it is not possible to set remaining parameters because these parameters are already part of the population element.

See Also

[setGroupRemaining](#)

Click here to see examples	
--	--

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get simulation groups

Description

Get the list of simulation groups and elements in each group.

Usage

```
getGroups ()
```

Details

Simulation groups are used for simulation [as in Simulx GUI](#).

At the creation of a Simulx project, a first group is created by default with the name "simulationGroup1".

Use [getGroups](#) to check which groups have already been defined, and which elements are set in each group.

To add a simulation group, use [addGroup](#).

To remove a simulation group, use `removeGroup`. To add or change a group element, use [setGroupElement](#).

To define new elements, use one of the `define...Element` functions.

See Also

[addGroup](#)

Click here to see examples	
--	--

[Simulx] Get number of replicates

Description

Get the number of replicates of the simulation.

Usage

```
getNbReplicates()
```

Details

The number of replicates is the number of times that Simulx will simulate a given study.

It should not be mixed up with the group size defined by [setGroupSize](#).

To simulate one study, Simulx samples a number of individuals for each group, defined by [setGroupSize](#) (let's say NidsPerGroup).

To simulate replicate studies, it will sample for each replicate NidsPerGroup other individuals for each group (it is like changing the seed).

If the parameter element is an individual element or a population parameter defined with a vector, replicates will always sample individuals using the same population parameters. In this case, they are useful to check the effect of changing the seed, to get for example the uncertainty of an endpoint due to limited sampling.

If the parameter element is a population element defined with a table containing several lines, or an imported element such as `mlx_PopUncertainSA` or `mlx_TypicalUncertainSA`, each replicate will use a different population parameter to simulate the study. In this case, it is possible to see the effect of changing the population parameters on the prediction (in addition to uncertainty due to limited sampling).

See Also

[setNbReplicates](#)

Click here to see examples	
--	--

[Simulx] Get same individuals among groups

Description

Get the information if the same individuals are simulated among all groups.

Usage

```
getSameIndividualsAmongGroups ()
```

Details

`setSameIndividualsAmongGroups(value = TRUE)` allows to have the same individual parameters in all groups.

It is available if the following elements (required for the sampling) are the same for all groups: size of groups, parameters (population or individual) and covariates.

The main goal is to make the comparison between groups easier.

In particular, it is used to compare different treatments on the same individuals – subjects with the same individual parameters.

Selecting same individuals among groups ensures that the differences between groups are only due to the treatment itself.

To obtain the same conclusion without this option enabled, simulation should be performed on a very large number of individuals to averaged out the individual differences.

All options of the Simulx scenario are the same as in Simulx GUI. Check [the online doc of Simulx](#) to get more guidance on how to use them.

See Also

[setSameIndividualsAmongGroups](#)

Click here to see examples	
--	--

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get sampling method

Description

Get which sampling method is used for the simulation. The possibilities are:

- `keepOrder` (default): individual values are taken in the same order as they appear in a table.

- `withReplacement`: individual values are sampled from a table with replacement.
- `withoutReplacement`: individual values are sampled from a table without replacement. This option is available only if tables contain at least the same number of individual values as a group size.

All of the above sampling methods are general and apply to all tables in a simulation scenario.

Usage

```
getSamplingMethod()
```

Details

All options of the Simulx scenario are the same as in Simulx GUI. Check [the online doc of Simulx](#) to get more guidance on how to use them.

See Also

[setSamplingMethod](#)

Click here to see examples	
--	--

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Get element types sharing individuals

Description

Get the element types that will share the same individuals in the simulation.

Usage

```
getSharedIds()
```

Details

If several elements are defined with tables of individual values and set to some simulation groups, the option "shared ids" allows to create an intersection of ids present in these tables. After that, ids from this intersection are sampled to create the data for simulation.

All options of the Simulx scenario are the same as in Simulx GUI. Check [the online doc of Simulx](#) to get more guidance on how to use them.

See Also

[setSharedIds](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Remove simulation group

Description

Remove a simulation group.

Usage

```
removeGroup (group)
```

Arguments

<code>group</code>	(<i>character</i>) Name of the group to remove.
--------------------	---

Details

Simulation groups are used for simulation [as in Simulx GUI](#).

At the creation of a Simulx project, a first group is created by default with the name "simulationGroup1".

Use [getGroups](#) to check which groups have already been defined, and which elements are set in each group.

To add a simulation group, use [addGroup](#).

To remove a simulation group, use `removeGroup`. To add or change a group element, use [setGroupElement](#).

To define new elements, use one of the `define...Element` functions.

See Also

[getGroups](#), [addGroup](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Remove element from simulation group

Description

Remove an element from a simulation group.

Usage

```
removeGroupElement(group, element)
```

Arguments

<code>group</code>	<i>(character)</i> Group name
<code>element</code>	<i>(character)</i> Element to remove

Details

Simulation groups are used for simulation [as in Simulx GUI](#).

At the creation of a Simulx project, a first group is created by default with the name "simulationGroup1".

Use [getGroups](#) to check which groups have already been defined, and which elements are set in each group.

To add a simulation group, use [addGroup](#).

To remove a simulation group, use `removeGroup`. To add or change a group element, use [setGroupElement](#).

To define new elements, use one of the `define...Element` functions.

Note: Removing an output element used in an outcome will delete the corresponding outcome and remove the outcome from the endpoints using it.

See Also

[setGroupElement](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Rename simulation group

Description

Rename a simulation group.

Usage

```
renameGroup(currentGroupName, newGroupName)
```

Arguments

<code>currentGroupName</code>	(<i>character</i>) Name of the current group name.
<code>newGroupName</code>	(<i>character</i>) Name of the new group name.

Details

Note: At the creation of a Simulx project, a first group is created by default with the name "simulationGroup1". It is possible to rename this group.

See Also

`addGroup` `getGroups`

[Click here to see examples](#)

[Back to the list, PKanalix API, Monolix API, Simulx API.](#)

[Simulx] Run endpoints task

Description

Run the endpoints task.

Usage

```
runEndpoints()
```

See Also

`getEndpointsResults`

[Click here to see examples](#)

[Back to the list, PKanalix API, Monolix API, Simulx API.](#)

[Simulx] Run simulation

Description

Run the simulation task.

Usage

```
runSimulation()
```

Details

As when clicking on SIMULATION in [Simulx GUI](#), simulated values are saved as text files in a result folder which is located next to the project file when calling [saveProject](#).

To get the sampled parameters and simulated outputs as data frames, use [getSimulationResults](#).

To post-process the results by computing outcomes and endpoints, use [runEndpoints](#).

To run both the simulation and endpoint tasks, use [setScenario](#) and [runScenario](#).

To get some output in the console showing the status of the simulation, use [setConsoleMode](#).

`runSimulation` does not generate charts data in the result folder even if the preference `exportchartsdata` is `TRUE`.

To generate charts data in the result folder at run, use [runScenario](#) instead.

See Also

[getSimulationResults](#)

Click here to see examples	
--	--

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Set elements to a simulation group

Description

Set the new element of a specific group. If an element of the same type is already set, [setGroupElement](#) will replace it.

For treatments and outputs, it is possible to set several elements at the same time by using a vector.

Usage

```
setGroupElement(group, elements)
```

Arguments

<code>group</code>	<i>(character)</i> Group name (when creating a new Simulx project, the default group name is "simulationGroup1").
<code>elements</code>	<i>(character)</i> Vector of elements that are already defined

Details

Simulation groups are used for simulation [as in Simulx GUI](#).

The same rules apply as in the GUI to set group elements. For example, a covariate element can be set only if the parameter element is a population parameter.

At the creation of a Simulx project, a first group is created by default with the name "simulationGroup1".

Use [getGroups](#) to check which groups have already been defined, and which elements are set in each group.

To add a simulation group, use [addGroup](#).

To remove a simulation group, use `removeGroup`. To add or change a group element, use [setGroupElement](#).

To remove an element from a group, use [removeGroupElement](#).

To define new elements, use one of the `define...Element` functions.

See Also

[getGroups](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Set remaining parameters for a simulation group

Description

Set the values of the remaining parameters (typically the error model parameters) for a group.

Usage

```
setGroupRemaining(group, remaining)
```

Arguments

group	(<i>character</i>) Group name
remaining	(<i>vector</i>) list of the remaining variables

Details

Remaining parameters are all parameters that appear in the structural model (in the input line of [LONGITUDINAL]) and are neither individual parameters nor regressors. They are typically error model parameters.

If an individual parameters element is selected for simulation, and the model includes remaining parameters, it is possible to set their values with `setGroupRemaining`.

It typically enables to make a simulation with measurement noise, with an individual element.

These error model parameters will impact the simulation only if a noisy observation (from the DEFINITION section of the [LONGITUDINAL] block) is set as output element (instead of a smooth prediction in OUTPUT or variable in EQUATION).

If a population parameters element is selected, it is not possible to set remaining parameters because these parameters are already part of the population element.

See Also

[getGroupRemaining](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Set simulation group size

Description

Define the size of a simulation group.

Usage

```
setGroupSize(group, size)
```

Arguments

<code>group</code>	(<i>character</i>) Name of the group where the size will be changed.
--------------------	--

<code>size</code>	(<i>integer</i>) Size of the new group.
-------------------	---

Details

Group size is the number of individuals (ie sets of individual parameters and covariate values) that will be sampled by Simulx for a given group.

It should not be mixed up with the number of replicates that can be set with [setNbReplicates](#).

To get the size of a group, please use [getGroups](#).

All options of the Simulx scenario are the same as in Simulx GUI. Check [the online doc of Simulx](#) to get more guidance on how to use them.

See Also

[getGroups](#)

Click here to see examples	
--	--

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Set number of replicates

Description

Define the number of replicates of the simulation.

Usage

```
setNbReplicates(nb)
```

Arguments

<code>nb</code>	(<i>integer</i>) Number of replicates.
-----------------	--

Details

The number of replicates is the number of times that Simulx will simulate a given study.

It should not be mixed up with the group size defined by [setGroupSize](#).

To simulate one study, Simulx samples a number of individuals for each group, defined by `setGroupSize` (let's say `NidsPerGroup`).

To simulate replicate studies, it will sample for each replicate `NidsPerGroup` other individuals for each group (it is like changing the seed).

If the parameter element is an individual element or a population parameter defined with a vector, replicates will always sample individuals using the same population parameters. In this case, they are useful to check the effect of changing the seed, to get for example the uncertainty of an endpoint due to limited sampling.

If the parameter element is a population element defined with a table containing several lines, or an imported element such as `mlx_PopUncertainSA` or `mlx_TypicalUncertainSA`, each replicate will use a different population parameter to simulate the study. In this case, it is possible to see the effect of changing the population parameters on the prediction (in addition to uncertainty due to limited sampling).

All options of the Simulx scenario are the same as in Simulx GUI. Check [the online doc of Simulx](#) to get more guidance on how to use them.

See Also

[getNbReplicates](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Set same individuals among groups

Description

Define if the same individuals will be simulated among all groups.

Usage

```
setSameIndividualsAmongGroups (value)
```

Arguments

<code>value</code>	<i>(logical)</i> logical to define if the same individuals will be the same for all groups.
--------------------	---

Details

`setSameIndividualsAmongGroups(value = TRUE)` allows to have the same individual parameters in all groups.

It is available if the following elements (required for the sampling) are the same for all groups: size of groups, parameters (population or individual) and covariates.

The main goal is to make the comparison between groups easier.

In particular, it is used to compare different treatments on the same individuals – subjects with the same individual parameters.

Selecting same individuals among groups ensures that the differences between groups are only due to the treatment itself.

To obtain the same conclusion without this option enabled, simulation should be performed on a very large number of individuals to averaged out the individual differences.

All options of the Simulx scenario are the same as in Simulx GUI. Check [the online doc of Simulx](#) to get more guidance on how to use them.

See Also

[getSameIndividualsAmongGroups](#)

Click here to see examples	
--	--

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Set sampling method

Description

Define which sampling method is used for the simulation. The possibilities are:

- `keepOrder` (default): individual values are taken in the same order as they appear in a table.
- `withReplacement`: individual values are sampled from a table with replacement.
- `withoutReplacement`: individual values are sampled from a table without replacement. This option is available only if tables contain at least the same number of individual values as a group size.

All of the above sampling methods are general and apply to all tables in a simulation scenario.

Usage

`setSamplingMethod (method)`

Arguments

`method` (*character*) `keepOrder`, `withReplacement`, `withoutReplacement`

Details

All options of the Simulx scenario are the same as in Simulx GUI. Check [the online doc of Simulx](#) to get more guidance on how to use them.

See Also

[getSamplingMethod](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Set element types sharing individuals

Description

Select the element types that will share the same individuals in the simulation.

Usage

`setSharedIds (sharedIds)`

Arguments

`sharedIds` (*vector<character>*) List of element types. The available types are: `covariate`, `output`, `treatment`, `regressor`, `population`, `individual`

Details

If several elements are defined with tables of individual values and set to some simulation groups, the option "shared ids" allows to create an intersection of ids present in these tables. After that, ids from this intersection are sampled to create the data for simulation.

All options of the Simulx scenario are the same as in Simulx GUI. Check [the online doc of Simulx](#) to get more guidance on how to use them.

See Also

[getSharedIds](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Print outcomes and endpoints

Description

Get all the outcomes and endpoints defined in the Simulx project.

The output is a list of:

- `outcomes` is a table of all defined simulation outcomes. To define an outcome, use `defineOutcome()`.
- `endpoints` is a table of all defined simulation endpoints. To define an endpoint, use `defineEndpoint()`.
- `groupComparison` is a Boolean and indicates whether group comparison is enabled. To change this setting, use `setGroupComparisonSettings()`.
- `groupComparisonReferenceGroup` corresponds to the reference simulation group for group comparison if enabled. To change this setting, use `setGroupComparisonSettings()`.

#

Usage

```
printOutcomesEndpoints()
```

See Also

[printSimulationSetup](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

[Simulx] Print simulation setup

Description

Get all the elements used in simulation setup: content of simulation groups and simulation settings.

The output is a list of:

- `simulationGroups` is a table of the elements selected in the simulation groups, with one simulation group per column. To add a simulation group, use `addGroup`. To remove a simulation group, use `removeGroup`. To add or change a group element, use `setGroupElement`. To change the number of simulated individuals in a group, use `setGroupSize`.
- `nbReplicates` is the number of simulation replicates. To change the number of replicates, use `setNbReplicates`.
- `samplingMethod` is the sampling method used for the simulation (`keepOrder`, `withReplacement` or `withoutReplacement`). To change the sampling method, use `setSamplingMethod`.
- `sharedIds` corresponds to element types that will share the same individuals in the simulation. To change this, use `setSharedIds`.
- `sameIndividualsAmongGroups` is a Boolean. If TRUE, the same individual parameters are sampled in all groups. TO change this setting, use `setSameIndividualsAmongGroups`.

#'

Usage

```
printSimulationSetup()
```

See Also

[printOutcomesEndpoints](#)

[Click here to see examples](#)

[Back to the list](#), [PKanalix API](#), [Monolix API](#), [Simulx API](#).

5.1. R-package installation and initialization

In this page, we present the installation procedure of the R-package that allow to run Simulx from R.

Installation

The R package **lixoftConnectors** is located in the installation directory as tar.gz ball. It can be installed directly using Rstudio (Tools > Install packages > from package archive file) or with the following R command:

```
install.packages(packagePath, repos = NULL, type="source", INSTALL
```

with the packagePath = '<installDirectory>/connectors/lixoftConnectors.tar.gz' where <installDirectory> is the MonolixSuite installation directory.

With the default installation directory, the command is:

```
# for Windows OS
install.packages("C:/ProgramData/Lixoft/MonolixSuite2023R1/connect
                repos = NULL, type="source", INSTALL_opts = "--no-

# for Mac OS
install.packages("/Applications/MonolixSuite2023R1.app/Contents/R
                monolixSuite/connectors/lixoftConnectors.tar.gz'
                repos = NULL, type="source", INSTALL_opts = "--no-
```

The lixoftConnectors package depends on the **RJSONIO** and **ggplot2** packages that may need to be installed from CRAN first using:

```
install.packages('RJSONIO')
install.packages('ggplot2')
```

R version 3.0.0 or higher is required to install lixoftConnectors.

Initializing

When starting a new R session, you need to load the library and initialize the connectors with the following commands

```
library(lixoftConnectors)
initializeLixoftConnectors(software = "simulx")
```

In some cases, it may be necessary to specify the path to the installation directory of the Lixoft suite. If no path is given, the one written in the <user home>/lixoft/lixoft.ini file is used (usually "C:/ProgramData/Lixoft/MonolixSuiteXXXX" for Windows). where XXXX corresponds to the version of MonolixSuite

```
library(lixoftConnectors)
initializeLixoftConnectors(software = "simulx", path = "/path/to/1
```

Making sure the installation is ok

To test if the installation is ok, you can load and run a project from the demos as on the following:

```
1. # load and initialize the API
2. library(lixoftConnectors)
3. initializeLixoftConnectors(software="simulx")
4.
5. # Get the project. <UserName> is the user's home folder
   (on windows C:/Users/toto if toto is your username).
6. demoPath = '<UserName>/lixoft/simulx/simulx2020R1/demos/'
7. project <- paste0(demoPath, "2.models/longitudinal.smlx")
8. loadProject(projectFile = project)
9.
10. # Run the simulation
11. runSimulation()
12.
13. # The results are accessible through the function
   getSimulationResults()
14. # The results for the output is the list res and TS is
   one of the outputs
15. head(getSimulationResults()$res$TS)
16. id time TS
17. 1 1 0 10.00000
18. 2 1 1 11.04939
19. 3 1 2 12.20862
20. 4 1 3 13.48915
21. 5 1 4 14.90359
22. 6 1 5 16.46585
```

5.2. Examples using R functions

This page provides pieces of code to perform some procedures with the R functions for Simulx. The code has to be adapted to use your projects and use the results as you wish in R.

- [Load a project and run the simulation](#)
- [Import a Monolix project and simulate a new output variable with the EBEs](#)
- [Create a Simulx project from scratch and simulate covariate-dependent treatments](#)
- [Import a Monolix project and simulate with new output times](#)
- [Run a simulation with different sample sizes and plot the power of the study based on an endpoint](#)
- [Hide warning/error/info messages and force software switch](#)

Minimal examples based on demo projects are also included in the [documentation of all Simulx functions](#).

Load a project and run the simulation

Simple example to load an existing project, run the simulation and look at the results :

```
1. # load and initialize the API
2. library(lixoftConnectors)
3. initializeLixoftConnectors(software="simulx")
4.
5. # Get the project
6. project <- paste0(getDemoPath(), "/2.models/
longitudinal.smlx")
7. loadProject(projectFile = project)
8.
9. # Run the simulation
10. runSimulation()
11.
12. # The results are accessible through the function
getSimulationResults()
13. # The results for the output is the list res and TS is
one of the outputs
14. head(getSimulationResults()$res$TS)
```

	id	time	TS
1	1	0	10.00000
2	1	1	11.04939
3	1	2	12.20862
4	1	3	13.48915
5	1	4	14.90359
6	1	5	16.46585

Import a Monolix project and simulate a new output variable with the EBEs

In this example, the Monolix demo theophylline_project.mlxtran is imported to Simulx. A new variable AUC is added to the model, and it is simulated on a regular grid using the EBEs estimated in the Monolix project. This requires that the tasks "Population parameters" and "EBEs" have run in the Monolix project.

```
1. # Import a Monolix project and simulate a new output
variable with the EBEs =====
2. # Load and initialize library =====
```

```

3.   library(lixoftConnectors)
4.   initializeLixoftConnectors(software = "monolix")
5.
6.   #Load project from Monolix Demos and run it to get EBEs
estimated =====
7.   MonolixProject <- paste0(getDemoPath(), "/"
1.creating_and_using_models/1.1.libraries_of_models/
theophylline_project.mlxtran")
8.   loadProject(MonolixProject)
9.   runScenario()
10.  initializeLixoftConnectors(software = "simulx", force =
TRUE)
11.
12.  #IMPORT Monolix project and check that mlx_EBEs element
has been created =====
13.  importProject(MonolixProject) getIndividualElements()
14.
15.  #SET ADDITIONAL formula in the model to compute the AUC
=====
16.  setAddLines("ddt_AUC = Cc") # define a new element output
with a regular grid for the variable "AUC"
17.  defineOutputElement(name="AUCregular", element = list(
data = data.frame( start = 0, interval = 1, final = 120),
output = "AUC"))
18.  #CHECK the simulation group that already exists =====
19.  getGroups()
20.  # => we currently have one group, called
"simulationGroup1", which uses the following elements:
21.  # - mlx_Pop (population parameter values) => need to be
replaced by mlx_EBEs
22.  # - mlx_Adml (treatment from monolix data set) => OK
23.  # - mlx_CONC (output CONC with times as in monolix data
set) => need to be replaced by AUC with new grid
24.  # - size = 12 (same number of in original data set) => OK
25.
26.  #MODIFY the existing simulation group such that it uses
EBEs and the new output element=====
27.  setGroupElement(group = "simulationGroup1",
28.                  elements = c("mlx_EBEs", "AUCregular"))
29.
30.  # check the new simulation setup
31.  getGroups()
32.
33.  #Set shared IDs to say the individuals in the treatment
element and in the parameter element are the same and
should always be sampled together =====
34.  setSharedIds(c("treatment", "individual"))
35.
36.  # check the simulation setup
37.  getGroups()
38.  getSharedIds()
39.
40.  # - mlx_EBEs => ok
41.  # - remaining parameters (error model parameters if
residual error is simulated) => not needed as we output
model predictions
42.  # - mlx_Adml => ok
43.  # - AUCregular => ok

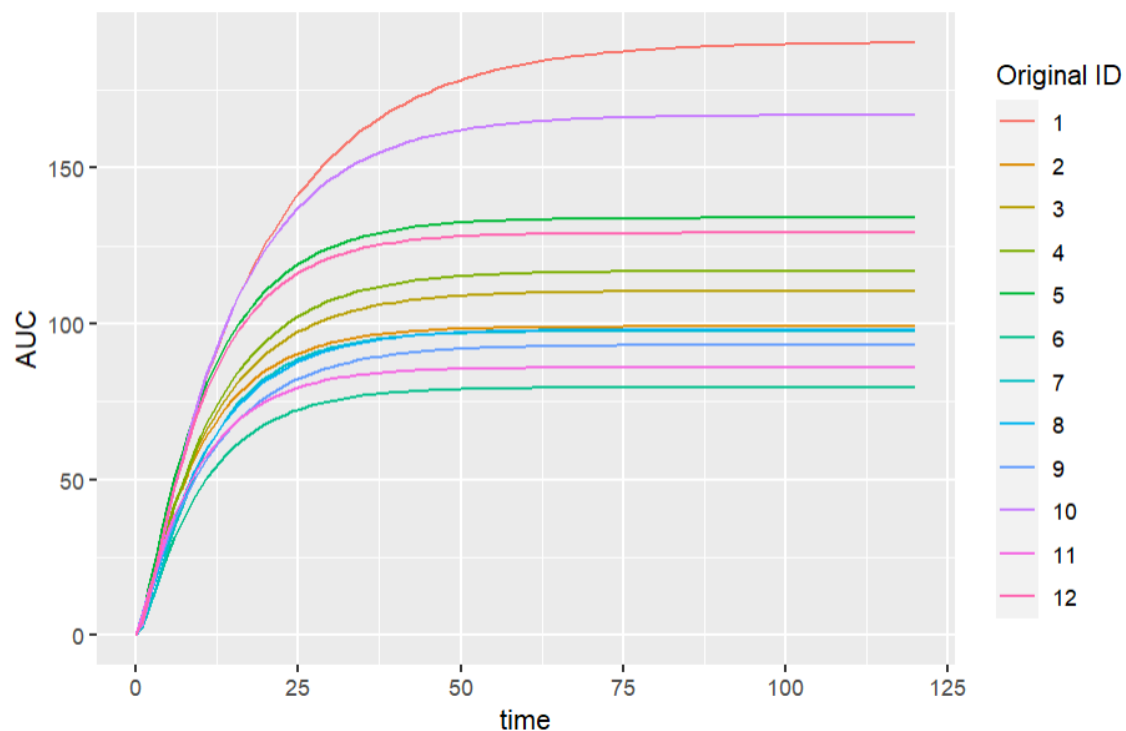
```

```

44. # - size=12 => ok
45.
46.
47. #SAVE and RUN project =====
48. saveProject("simulx_api_example.smlx")
49.
50. # run simulation
51. runSimulation()
52.
53.
54. #GET the results: simulated values for AUC and sampled
  individual parameters =====
55. simulatedParam <- getSimulationResults()
  $IndividualParameters$simulationGroup1
56. simulatedOutput <- getSimulationResults()$res$AUC
57.
58.
59. #PLOT the results =====
60. library(ggplot2)
61. ggplot(data = simulatedOutput, aes(x=time, y=AUC))+
62.   geom_line() + aes(color =
  factor(as.double(original_id)))+
63.   labs(colour="Original ID")

```

We get the following simulations for the new output variable. "Original_id" is the ID used in the original dataset loaded in the Monolix project.



Create a Simulx project from scratch and simulate covariate-dependent treatments

This script creates a project similar to the demo project *treatment_weight_and_genotype_based.smlx*, available in the folder 3.1.treatments of Simulx demos. The model file *TMDDmodel.txt* can be downloaded [here](#).

While in the interface of Simulx it is possible to directly define distribution laws for covariates and covariate-dependent treatments that are computed at the simulation step, in R it is necessary to sample covariates from distributions and derive the corresponding doses before defining the elements for the simulation.

```
1. #####
2. # Create a Simulx project from scratch and simulate covari-
   dependent treatments
3. #####
4.
5. # load and initialize the API
6. library(lixoftConnectors)
7. initializeLixoftConnectors(software = "simulx")
8.
9. # Create a new project based on the model
10. newProject(modelFile = "TMDDmodel.txt")
11.
12. # Define vector of population parameters
13. definePopulationElement(name="PopParameters",
14.                         element = data.frame(V_pop=3, omega
,Cl_pop=0.1, omega_Cl=0.1, Q_pop=1, omega_Q=0.2,
15.                                              V2_pop=3,
16. omega_V2=0.2, beta_V_logtWeight=1, KD_pop=0.01, omega_KD=0.
17.                                              R0_pop=0.2,
18. omega_R0=0.3, kint_pop=50, omega_kint=0.2, kon_pop=10,
19.                                              omega_kon=0.0
20. ksyn_pop=10, omega_ksyn=0.2, beta_Cl_logtWeight=0.75,
21. beta_Q_logtWeight=0.75, beta_V2_logtWeight=0.75,
22. beta_KD_Genotype_Heterozygous=1.3))
23.
24. # Define covariates: in the interface it is possible to de
   distribution laws,
25. # but in R we have to sample from the distributions prior
   simulation.
26. # Here we define 3 tables for the 3 simulation groups
27. for (i in 1:3){
28.   covTable <- data.frame(id=1:250,
29.                          Weight = rlnorm(n=250, mean=log(7
30. sd=0.3),
31.                          Genotype=c("Homozygous", "Heteroz
32. [sample(1:2, 250, replace=T)])
33.   write.csv(covTable, file = paste0("covTable",i,".csv"),
34.             quote=F, row.names=F)
35.   defineCovariateElement(name=paste0("covTable",i),
36.                          element = paste0("covTable",i,".c
37. }
38.
39. # Define common treatment
40. defineTreatmentElement(name="1000nmol", element=list(admID:
41. data=data.frame(time=seq(0,by=21,length.out = 5),
```

```

35. amount=1000, tInf=0.208)))
36.
37. # Define weight-based treatment: in the interface it can be
   directly with a scaling formula,
38. # but in R we have to define a table of individual doses per
   the simulation.
39. # We use covTable2.csv for simulation group 2.
40. covTable <- read.csv("covTable1.csv")
41. trt_14nmolPerKg <- data.frame(id=rep(1:250, each=5),
42.                               time=rep(seq(0,by=21,length.out=
   5), times=250),
43.                               amount=14*covTable$Weight, t
   0.208)
44. write.csv(trt_14nmolPerKg, file="trtTableWeight.csv", quote=
   row.names=F)
45. defineTreatmentElement(name="14nmolPerKg", element=list(adm
   data="trtTableWeight.csv"))
46.
47. # Define genotype-based treatment: in the interface it can
   done directly with a scaling formula,
48. # but in R we have to define a table of individual doses per
   the simulation.
49. # We use covTable3.csv for simulation group 3.
50. covTable <- read.csv("covTable3.csv")
51. trt_genotype <- data.frame(id=rep(1:250, each=5),
52.                             time=rep(seq(0,by=21,length.out=
   times=250),
53.
   amount=ifelse(covTable$Genotype=="Heterozygous", 1000, 800)
   = 0.208)
54. write.csv(trt_genotype, file="trtTableGenotype.csv", quote=
   row.names=F)
55. defineTreatmentElement(name="1000nmolHomo_800nmolHetero",
   element=list(admID=1, data="trtTableGenotype.csv"))
56.
57. # Define outputs
58. defineOutputElement(name="Concentration", element =
   list(output="L", data=data.frame(time=seq(0,96,by=1))))
59. defineOutputElement(name="TargetOccupancy", element =
   list(output="TO", data=data.frame(time=seq(0,96,by=1))))
60.
61. # By default there is a single simulation group named
   simulationGroup1,
62. # change its name and set all elements for the simulation
63. setGroupSize("simulationGroup1", 250)
64. setGroupElement("simulationGroup1", elements =
   c("PopParameters", "14nmolPerKg", "Concentration", "TargetOccu
   covTable1"))
65. renameGroup("simulationGroup1", "Weight_based")
66.
67. # Define the two other simulation groups with different
   covariates and treatments.
68. # By default the other elements are the same as the first
69. addGroup("Flat_dose")
70. setGroupElement("Flat_dose", elements =
   c("1000nmol", "covTable2"))
71. addGroup("Genotype_based")

```

```

72.   setGroupElement("Genotype_based", elements =
      c("1000nmolHomo_800nmolHetero", "covTable3"))
73.
74.   # Save project and run the simulation
75.   saveProject(projectFile =
      "treatment_weight_and_genotype_based.smlx")
76.   runSimulation()
77.
78.   # Get simulation results: a table of individual parameters
      each group, and a table of simulated values for each output
79.   sim <- getSimulationResults()
80.   names(sim$IndividualParameters) # "Weight_based", "Flat_do
      "Genotype_based"
81.   names(sim$res) # "L", "TO"

```

Import a Monolix project and simulate with new output times

Monolix simulates the original dataset with replicates to generate the VPC. It can be useful to perform the same simulations in Simulx with modified observation times, for example to correct the bias in VPC due to dropout. The example below shows this step, and uses the project [PDTTE_dropout](#). The full example to correct a VPC for dropout is detailed [on this page](#).

In this example the Monolix project with the biased VPC is named PD_TTE.mlxtran, the new regular measurement times are similar to the measurements in the original dataset, but are not impacted by dropout, and the number of replicates is 500 (as the default number of simulations for the VPC in Monolix). The random variable capturing dropouts in the model used in Monolix is called survival.

```

1.   #####
2.   # Import a Monolix project and simulate with new output times
3.   #####
4.
5.   library(lixoftConnectors)
6.   initializeLixoftConnectors(software = "simulx")
7.
8.   importProject(projectFile = "PDTTE_dropout.mlxtran")
9.   defineOutputElement(name="y1_alltimes", element =
      list(output="y1", data=data.frame(time=seq(-100,1500,by=100)
10.   setGroupElement("simulationGroup1", elements =
      c("y1_alltimes", "mlx_survival"))
11.   setNbReplicates(500)
12.   saveProject(projectFile = "PD_TTE_alltimes.smlx")
13.   setPreferences(exportsimulationfiles=T)
14.   runSimulation()

```

Run a simulation with different sample sizes and plot the power of the study based on an endpoint

This piece of code uses the demo project OutcomeEndpoint_PKPD_changeFromBaseline.smlx (demo folder 6.1), with the goal of comparing two treatment arms (400mg Q4W and 800mg Q8W) with a range of different sample sizes (number of simulated individuals per arm). The power of the clinical trial is defined as the chance of having a significant difference in the mean change from baseline between the two arms. This chance is computed over 200 replicates for each sample size. The script then plots the power of the clinical trial with respect to the sample size, with a target power of 85% displayed as a horizontal line.

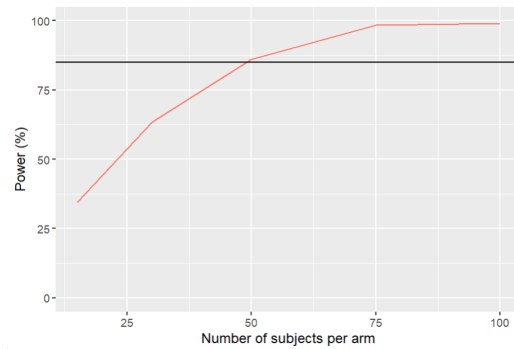
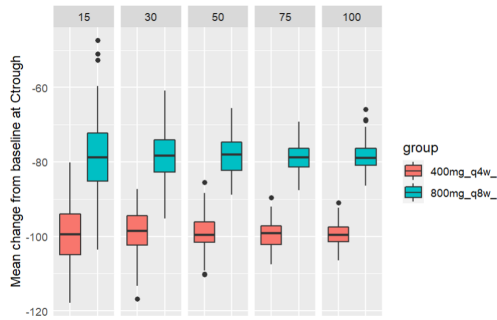
Note that Simulx versions prior to 2023 did not provide connectors to define the outcomes and endpoints like in the interface.

```
1. library(ggplot2)
2.
3. #initialize LixoftConnectors with Simulx
4. library(lixoftConnectors)
5. initializeLixoftConnectors(software="simulx")
6.
7. # load Simulx demo project comparing 3 treatment arms
8. project <- paste0(getDemoPath(), "/6.outcome_endpoints/
6.1.outcome_endpoints/OutcomeEndpoint_PKPD_changeFromBaseli
loadProject(projectFile = project) # check the groups defin
project. # We are interested comparing the two groups 400mg
800mg_q8w_ for the output "Cholesterol_measured_atCtrough"
remove other groups and outputs
9. getGroups()
10. removeGroup("400mg_q8w_")
11. removeGroupElement("400mg_q4w_", "Cholesterol_prediction")
12. removeGroupElement("800mg_q8w_", "Cholesterol_prediction")
13. removeGroupElement("400mg_q4w_", "mAbTotal_prediction")
14. removeGroupElement("800mg_q8w_", "mAbTotal_prediction")
15.
16. # check the endpoints defined in the project. We are inter
the endpoint mean_changeFromBaseline
17. getEndpoints()
18. deleteEndpoint("percentChangeFromBaseline")
19.
20.
21. #enable group comparison since it is disabled in the demo
22. setGroupComparisonSettings(referenceGroup = "400mg_q4w_", e
TRUE)
23.
24.
```

```

25. # gather endpoints for trials of several sample sizes
26. Nrep <- 200
27. sample_sizes <- c(15, 30, 50, 75, 100)
28. PowerDF =
data.frame(sampleSize=sample_sizes, Power=rep(0, length(sample
29.
30.
31. meanChFromBL <-NULL
32. for(indexN in 1:length(sample_sizes)){
33.
34.     N <- sample_sizes[indexN]
35.     # change the number of individuals for each arm
36.     setGroupSize("400mg_q4w_", N)
37.     setGroupSize("800mg_q8w_", N)
38.     setNbReplicates(Nrep)
39.
40.     # run the simulation and endpoints
41.     runScenario()
42.
43.     # optional - gathering endpoints to plot them for each g
sample size
44.     meanChFromBL_N <- getEndpointsResults()
$endpoints$mean_changeFromBaseline
45.     meanChFromBL <- rbind(meanChFromBL, cbind(meanChFromBL_N
sampleSize=N))
46.
47.     # gathering success for each replicate
48.     successForAllRep <- as.logical(getEndpointsResults()
$groupComparison$mean_changeFromBaseline$success)
49.     PowerDF$Power[indexN]=sum(successForAllRep)/Nrep*100
50.
51. }
52.
53. # to plot the results (no connectors available yet for Sim
54.
55. library(ggplot2)
56.
57. ggplot(meanChFromBL, aes(group, arithmeticMean)) +
58.     geom_boxplot(aes(fill = group)) +
59.     facet_grid(. ~ sampleSize) +
60.     ylab("Mean change from baseline at Ctrough")+
61.     xlab("")+
62.     theme(axis.text.x = element_blank(), axis.ticks =
element_blank())
63.
64. ggplot(PowerDF)+geom_line(aes(x=sampleSize,y=Power, color=
65.     ylab("Power (%)")+xlab("Number of subjects per arm")
+geom_hline(yintercept = 85)+
66.     ylim(c(0,100))+ theme(legend.position = "none")

```



Handling of warning/error/info messages

Error, warning and info messages from Simulx are displayed in the R console when performing actions on a Simulx project. They can be hidden via the R options. Set `lixoft_notificationOptions$errors`, `lixoft_notificationOptions$warnings` and `lixoft_notificationOptions$info` to 1 or 0 to respectively hide or show the messages.

Example

```
op = options()
op$lixoft_notificationOptions$warnings = 1 #hide the warning messages
options(op)
```

Force software switch

By default, function `initializeLixoftConnectors()` prompts users to confirm that they want to proceed with the software switch, in order to avoid losing unsaved changes in the currently loaded project. To override this behavior, `force` argument can be set to `TRUE` when calling the function. However, the default behavior can be changed globally as well.

Example

```
op = options()
op$lixoft_lixoftConnectors_forceSoftwareSwitch <- TRUE
options(op)
```

6. RsSimulx

RsSimulx is an additional R package which offers two main functionalities:

- provide a **wrapper around the lixoftConnectors for Simulx**, with a more

- compact syntax than the `lixoftConnectors`
- provide a function similar to the `simulx()` function of the old package `mlxR` and compatible with the `MonolixSuite` versions 2020 and 2021, to ease the transition.

- Installation
- Versions and change log
- `RsSimulx` usage

Installation

`RsSimulx` is available on CRAN but it requires the `lixoftConnectors` package which is provided within the `MonolixSuite` installation. In addition `RJSONIO` (available on CRAN) is required.

Installation of RJSONIO (from CRAN)

```
install.packages("RJSONIO")
```

Installation of the lixoftConnectors

The `lixoftConnectors` are not available on CRAN. The installation package is located in the `MonolixSuite` installation directory. The default path to the installation directory differs between operating systems. You may have to adapt the path if you have chosen another installation directory.

```
# for Windows
install.packages("C:/ProgramData/Lixoft/MonolixSuite2023R1/connectors",
                 repos = NULL, type="source", INSTALL_opts = "--no-r")

# for MAC OS
install.packages("/Applications/MonolixSuite2023R1.app/Contents/Resources/connectors",
                 repos = NULL, type="source", INSTALL_opts = "--no-r")

# for Linux
install.packages("/home/<your username>/Lixoft/MonolixSuite2023R1/connectors",
                 repos = NULL, type="source", INSTALL_opts = "--no-r")
```

Installation of RsSimulx

```
install.packages("RsSimulx", INSTALL_opts = "--no-multiarch")
```

Change log

Version 2023.1 (released on CRAN on April 20th 2023)

This version is compatible with MonolixSuite version 2023. The lixoftConnectors 2023 should be installed first.

- In case of simulations using elements defined via external files, output will contain a column "original_id" indicating id of individuals in external files.
- Versioning was changed to be consistent with lixoftConnectors and MonolixSuite versioning.

Version 2.0.0 (released on CRAN on February 18th 2022)

This version is compatible with MonolixSuite version 2021. The lixoftConnectors 2021 should be installed first.

In this version, new functionalities already available in the GUI were added and the syntax has been slightly changed to be consistent with the display in the GUI.

The main changes in the simulx() function are summarized below:

- **parameter:**
 - accept only pop param and indiv param (and 'remaining parameters'). No covariates anymore, use the "covariate" argument instead.
 - accept strings corresponding to the elements automatically generated after an import: "mlx_Pop", "mlx_PopUncertainSA", "mlx_PopUncertainLin", "mlx_PopIndiv", "mlx_PopIndivCov", "mlx_CondMean", "mlx_EBEs", "mlx_CondDistSample"
 - deprecated: "mode" and "mean" (use the keyword above instead)
- **covariate:**
 - accepts covariates elements as lists (same value for all individuals), data frames or path to text file
 - accept strings corresponding to the elements automatically generated after an import: "mlx_Cov" and "mlx_CovDist"
- **treatment:**
 - accept string corresponding to the elements automatically generated after an import: "mlx_admXXX"
 - new options "repeats" and "probaMissDose"
- **varlevel:**
 - deprecated and now called "occasion"
- **nrep:**
 - now samples with or without uncertainty depending which element is given as "parameter"
- **stat.f, data, result_folder et les settings load.design, data.in, disp.iter:**
 - have been removed

- **npop and fim:**
 - deprecated: use nrep and "mlx_PopUncertainSA" or "mlx_PopUncertainLin" instead
- **settings:**
 - "kw.max": deprecated
 - "replacement": has been replaced by "samplingMethod" with values "keepOrder", "withReplacement", "withoutReplacement"
 - new option "sameIndividualsAmongGroups": true/false
 - new option "sharedIds": vector of strings among covariate, output, treatment, regressor, population, individual.
 - new option "exportData": true/false
- **saveSmlxProject:**
 - save the .smlx project to the specified path.

Known limitations:

- when just resimulating a Monolix project, the individuals having no observations for some obs ids (in case of several observation identifiers) or no dose will be missing from the simulation.
- it is not possible to retrieve the original IDs when using data frames with id column as input

Version 1.0.1 (released on CRAN on April 8th 2021)

This version corrects several bugs and is compatible with MonolixSuite version 2020.

- The initialization of the lixoftConnectors (i.e linking to the MonolixSuite installation folder) is now done on the first call to simulx() instead of the loading of the RsSimulx package.
- The new function **initRsSimulx(path=...)** allows to indicate a path to the MonolixSuite installation folder when the <home>/lixoft/lixoft.ini file is missing or needs to be bypassed.
- Messages, warnings and errors raised by the lixoftConnectors are now propagated to RsSimulx such that the user can see them.
- Columns "type" or "adm" for treatments defined as data.frames are now properly recognized (they were previously ignored).
- When both population parameters and covariates are given in the "parameter=" argument of simulx(), the covariates are now properly taken into account (they were previously ignored).
- The seed given to simulx() now propagates to simpopmlx, which is sampling the population parameters when using the argument "npop" (previously the

reproducibility was not ensured when using npop).

- The use of a named vector with only characters (typically in the case of categorical-only covariates defined as strings) does not generate an error any more.
- When population parameters are defined as a data.frame with a column "pop", the dataframe is now used with the correct number of rows (it was previously cut or replicated to match the npop argument).
- When npop is used but no treatment is present, no error is raised anymore.
- When the "parameter" argument is defined as a named vector with both characters (for categorical covariates for instance) and numerical values, no error is raised anymore.

Known issues remaining in version 1.0.1:

- regressor values are not outputted from simulx()
- when just resimulating a Monolix project, in case of both continuous and non-continuous outputs, only the continuous outputs are outputted.

Version 1.0.0 (released on CRAN on January 21st 2021)

RsSimulx is compatible with the MonolixSuite version 2020. It provides retro-compatibility of old mlxR scripts and functions to complement the use of Simulx-GUI (writeData(), prctilemlx() and simpopmlx()).

6.1. RsSimulx usage

- **simulx()** to run the simulation
- **prctilemlx()** to plot prediction intervals (with overlay of the groups on a single plot)
- writeData() to write the simulation results to a MonolixSuite compatible format – **This feature is now available directly in the interface and in the API with `exportSimulatedData`.**

prctilemlx(): prediction interval plots

Description

Compute and plot percentiles of the empirical distribution of longitudinal data. When several groups are present, the groups can be plotted as subplots or as different colors on one plot. The input can be a R object (parameter `r`) or a monolix project and a variable name (arguments `project` and `outputVariableName`).

Usage

```
prctilemlx(  
  r = NULL,  
  col = NULL,  
  project = NULL,  
  outputVariableName = NULL,  
  number = 8,  
  level = 80,  
  plot = TRUE,  
  color = NULL,  
  group = NULL,  
  facet = TRUE,  
  labels = NULL,  
  band = NULL  
)
```

Arguments

<code>r</code>	a data frame with a column <code>id</code> , a column <code>time</code> and a column with values. The times should be the same for all individuals.
<code>col</code>	a vector with the three column indexes for <code>id</code> , <code>time</code> and <code>y</code> . Default = <code>c(1,2,3)</code> .
<code>project</code>	simulx project filename (with extension ".smlx")
<code>outputVariableName</code>	name of the output to consider. By default the first output will be consider. You must define either a 'r' dataframe (and the associated 'col' argument) or a simulx project and the name of the output 'outputVariableName'
<code>number</code>	the number of intervals (i.e. the number of percentiles minus 1).

<code>level</code>	the largest interval (i.e. the difference between the lowest and the highest percentile).
<code>plot</code>	if <code>TRUE</code> the empirical distribution is displayed, if <code>FALSE</code> the values are returned
<code>color</code>	colors to be used for the plots In case of one group or <code>facet = TRUE</code> , only the first color will be used
<code>group</code>	variable to be used for defining groups (by default, 'group' is used when it exists)
<code>facet</code>	makes subplots for different groups if <code>TRUE</code>
<code>labels</code>	vector of strings
<code>band</code>	is deprecated (use <code>number</code> and <code>level</code> instead) ; a list with two fields <ul style="list-style-type: none"> • <code>number</code> the number of intervals (i.e. the number of percentiles minus 1). • <code>level</code> the largest interval (i.e. the difference between the lowest and the highest percentile).

Details

You must define either a dataframe `x` (and if needed the associated `col` argument) or a `simulx` project and the name of the output `outputVariableName`. The `outputVariableName` corresponds to the name of the model variable, not to the output element name.

Value

a `ggplot` object if `plot=TRUE` ; otherwise, a list with fields:

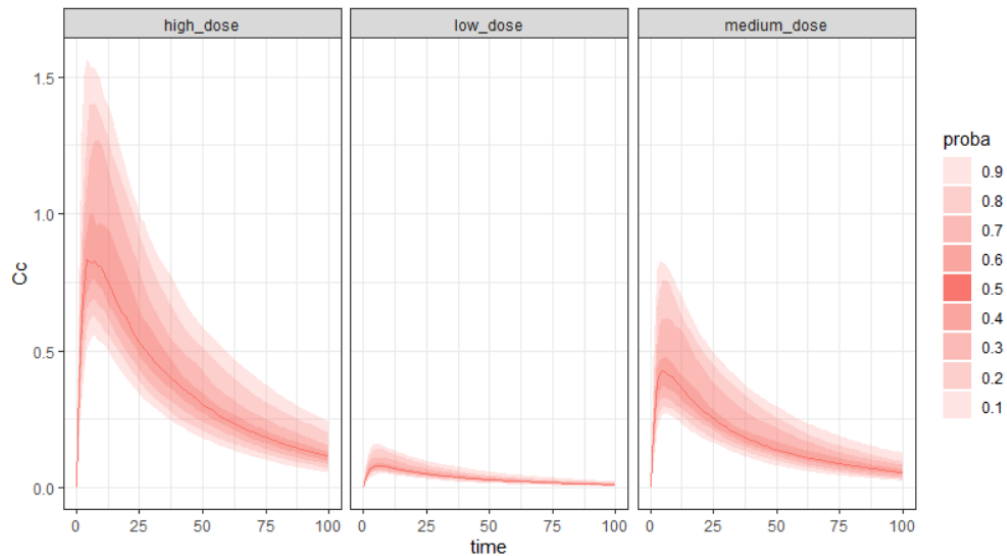
- `proba`: a vector of probabilities of length `band$number+1`
- `color`: a vector of colors used for the plot of length `band$number`
- `y`: a data frame with the values of the empirical percentiles computed at each time point

Examples

This simulx-GUI demo project contains 3 simulation groups, with a low, medium or high dose. In the Simulation tab, the output element 'regularCc' is selected, to output the variable 'Cc' on a regular time grid.

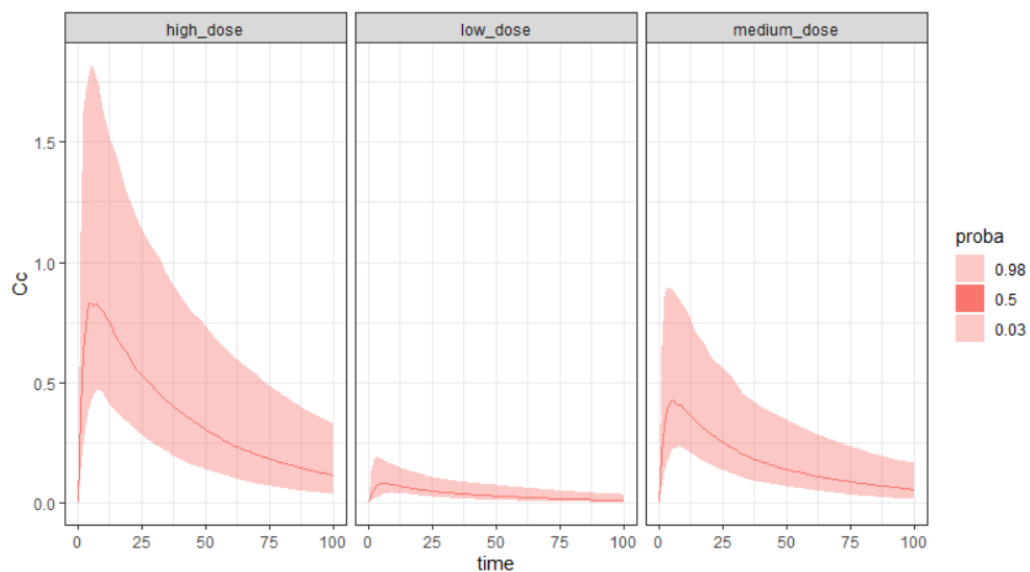
With `facet=T` (default), each group is displayed on a separate subplot with the different shades representing different percentiles. The largest band represents the 90% prediction interval.

```
project.file <- "~/../lixoft/simulx/simulx2020R1/demos/5.simulatic  
prctilemlx(project=project.file, outputVariableName = "Cc")
```



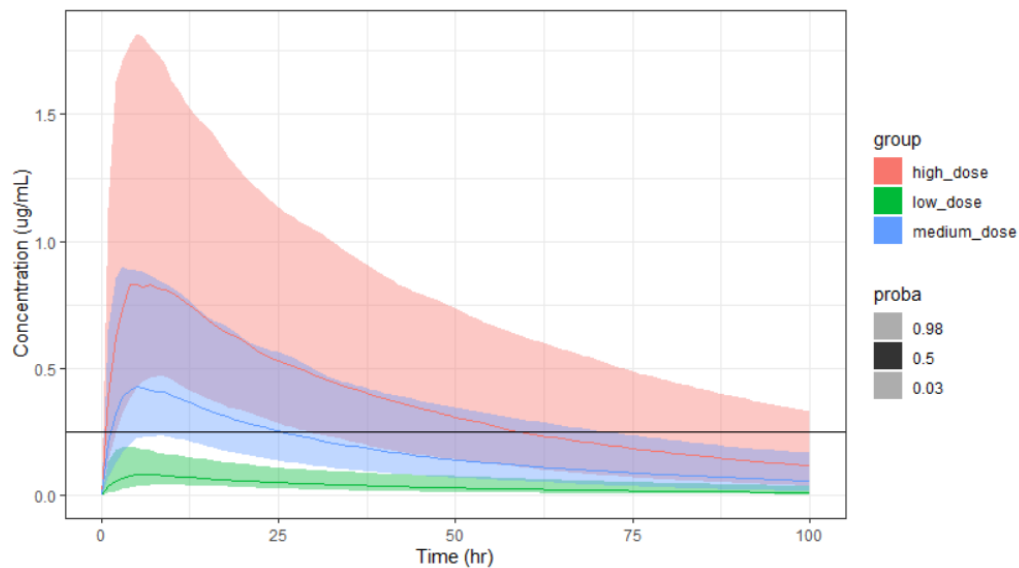
With `number=2` and `level=95`, we can select to display only the median and the 95% prediction interval.

```
project.file <- "~/../lixoft/simulx/simulx2020R1/demos/5.simulatic  
prctilemlx(project=project.file, outputVariableName = "Cc", number
```



To display the prediction intervals on top of each other, use `facet=F`. As a ggplot object is returned, additional ggplot functions can be used to overlay additional features or modify the legend, etc.

```
prctilemlx(project=project.file, outputVariableName = "Cc", facet=
```



See also <http://simulx.webpopix.org/mlxr/prctilemlx/> for more examples.

- `simulx()` to run the simulation [documentation under construction]
- `writeData()` to write the simulation results to a MonolixSuite compatible format
- `prctilemlx()` to plot prediction intervals (with overlay of the groups on a single plot)

writeData(): write simulations to MonolixSuite data set format

Description

Format outputs of simulx simulations and write datasets in monolix and pkanalix project format.

Usage

```
writeData(  
  project = NULL,  
  filename = "simulated_dataset.csv",  
  sep = ",",  
  ext = "csv",  
  nbdigits = 5  
)
```

Arguments

1	24	9.49323	1	.	50	66.7	1
1	36	21.64449	2	.	50	66.7	1
1	36	6.36576	1	.	50	66.7	1
1	48	13.97912	2	.	50	66.7	1

prctilemlx(): prediction interval plots

Description

Compute and plot percentiles of the empirical distribution of longitudinal data. When several groups are present, the groups can be plotted as subplots or as different colors on one plot. The input can be a R object (parameter `r`) or a monolix project and a variable name (arguments `project` and `outputVariableName`).

Usage

```
prctilemlx(
  r = NULL,
  col = NULL,
  project = NULL,
  outputVariableName = NULL,
  number = 8,
  level = 80,
  plot = TRUE,
  color = NULL,
  group = NULL,
  facet = TRUE,
  labels = NULL,
  band = NULL
)
```

Arguments

<code>r</code>	a data frame with a column <code>id</code> , a column <code>time</code> and a column with values. The times should be the same for all individuals.
<code>col</code>	a vector with the three column indexes for <code>id</code> , <code>time</code> and <code>y</code> . Default = <code>c(1,2,3)</code> .
<code>project</code>	simulx project filename (with extension ".smlx")

<code>outputVariableName</code>	name of the output to consider. By default the first output will be consider. You must define either a 'r' dataframe (and the associated 'col' argument) or a simltx project and the name of the output 'outputVariableName'
<code>number</code>	the number of intervals (i.e. the number of percentiles minus 1).
<code>level</code>	the largest interval (i.e. the difference between the lowest and the highest percentile).
<code>plot</code>	if <code>TRUE</code> the empirical distribution is displayed, if <code>FALSE</code> the values are returned
<code>color</code>	colors to be used for the plots In case of one group or facet = <code>TRUE</code> , only the first color will be used
<code>group</code>	variable to be used for defining groups (by default, 'group' is used when it exists)
<code>facet</code>	makes subplots for different groups if <code>TRUE</code>
<code>labels</code>	vector of strings
<code>band</code>	is deprecated (use number and level instead) ; a list with two fields <ul style="list-style-type: none"> • <code>number</code> the number of intervals (i.e. the number of percentiles minus 1). • <code>level</code> the largest interval (i.e. the difference between the lowest and the highest percentile).

Details

You must define either a dataframe `r` (and if needed the associated `col` argument) or a simltx `project` and the name of the output `outputVariableName`. The `outputVariableName` corresponds to the name of the model variable, not to the output element name.

Value

a ggplot object if `plot=TRUE` ; otherwise, a list with fields:

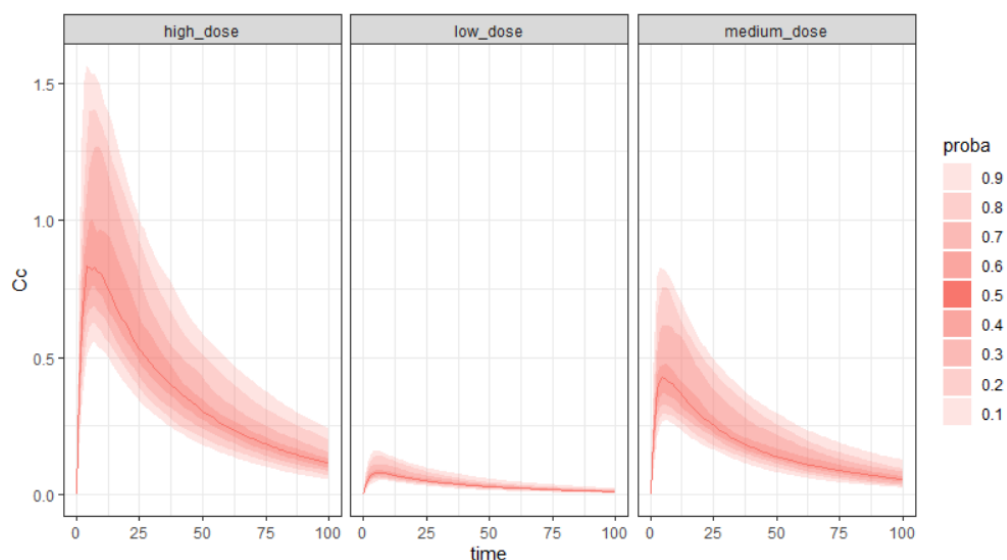
- `proba`: a vector of probabilities of length `band$number+1`
- `color`: a vector of colors used for the plot of length `band$number`
- `y`: a data frame with the values of the empirical percentiles computed at each time point

Examples

This simulx-GUI demo project contains 3 simulation groups, with a low, medium or high dose. In the Simulation tab, the output element 'regularCc' is selected, to output the variable 'Cc' on a regular time grid.

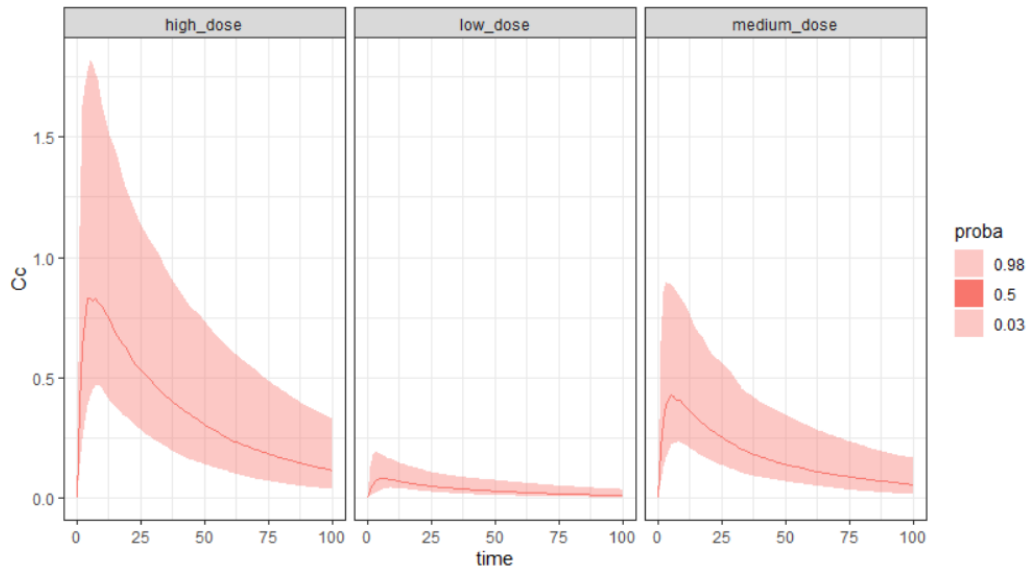
With `facet=T` (default), each group is displayed on a separate subplot with the different shades representing different percentiles. The largest band represents the 90% prediction interval.

```
project.file <- "~/../lixoft/simulx/simulx2020R1/demos/5.simulatic  
prctilemlx(project=project.file, outputVariableName = "Cc")
```



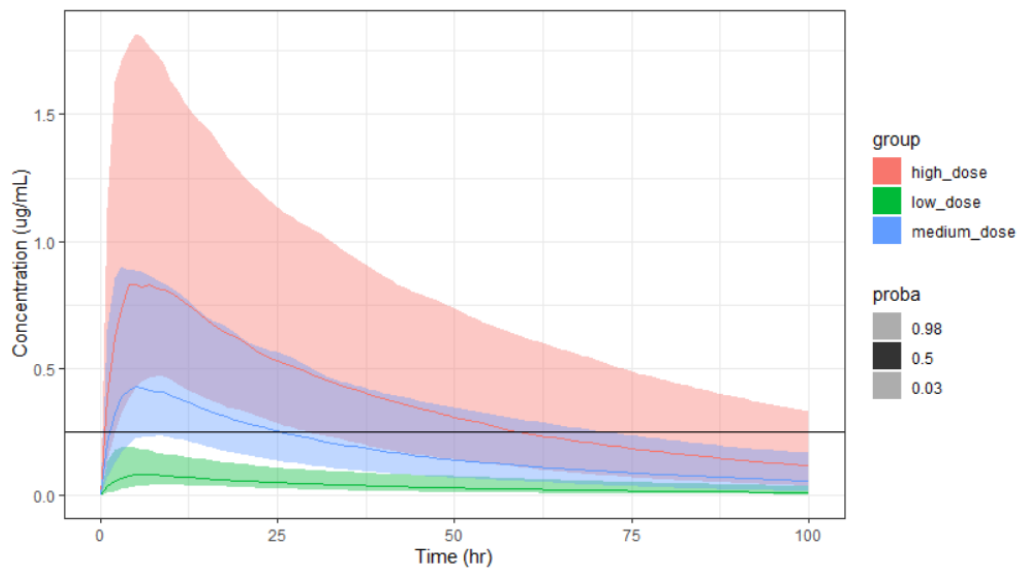
With `number=2` and `level=95`, we can select to display only the median and the 95% prediction interval.

```
project.file <- "~/../lixoft/simulx/simulx2020R1/demos/5.simulatic  
prctilemlx(project=project.file, outputVariableName = "Cc", number
```



To display the prediction intervals on top of each other, use `facet=F`. As a ggplot object is returned, additional ggplot functions can be used to overlay additional features or modify the legend, etc.

```
prctilemlx(project=project.file, outputVariableName = "Cc", facet=
```



See also <http://simulx.webpopix.org/mlxr/prctilemlx/> for more examples.

writeData(): write simulations to MonolixSuite data set format

Description

Format outputs of simulx simulations and write datasets in monolix and pkanalix project format.

Usage

```
writeData(
  project = NULL,
```

```

filename = "simulated_dataset.csv",
sep = ",",
ext = "csv",
nbdigits = 5
)

```

Arguments

<code>project</code>	<p>(<i>string</i>) a simulx project with extension <code>.smlx</code>. If no project is specified, the function will run on the project that is already loaded.</p>
<code>filename</code>	<p>(<i>string</i>) (<i>optional</i>) file path to dataset. (default "<code>simulated_dataset.csv</code>") In case of multiple replicates, the function creates one dataset per replicate with name <code>\$filename_repi</code> If filename contains an extension, it must be "<code>csv</code>" or "<code>txt</code>". If it does not, extension is defined by <code>ext</code> argument.</p>
<code>sep</code>	<p>(<i>string</i>) (<i>optional</i>) Separator used to write dataset file. (default "<code>,</code>") It must be one of "<code>\t</code>", "<code>,</code>", "<code>;</code>", "<code>;</code>"</p>
<code>ext</code>	<p>(<i>bool</i>) (<i>optional</i>) Extension used to write dataset file. (default "<code>csv</code>") It must be one of "<code>csv</code>", "<code>txt</code>" To defined only if filename with no extension</p>
<code>nbdigits</code>	<p>(<i>integer</i>) (<i>optional</i>) number of decimal digits in output file. (default = 5)</p>

Details

The generated data set has the following headers: `id`, `occ` (if occasions have been used), `time`, `amt`, `adm`, `rate`, `y` (observations), `ytype` (observation id), `evid` (if a washout was defined), and the covariates names.

WARNING: `writeData` function is not implemented for simulx project with regressors in MonolixSuite version 2020R1. When a `loq` is defined in the `simulx()` function, it is not taken into account when writing the data set.

Value

a dataframe if one single simulation, a list of dataframes if multiple replicates.

Examples

The Simulx-GUI demo simulates a PK/PD data set. The saved data set has the standard format to be loaded in Monolix or Simulx.

```
project.file <- "~/../lixoft/simulx/simulx2020R1/demos/1.overview,  
writeData(project=project.file, filename = "simdata.csv")
```

id	time	y	ytype	amt	age	wt	sex
1	0	.	.	100	50	66.7	1
1	0	105.2612	2	.	50	66.7	1
1	24	30.83899	2	.	50	66.7	1
1	24	9.49323	1	.	50	66.7	1
1	36	21.64449	2	.	50	66.7	1
1	36	6.36576	1	.	50	66.7	1
1	48	13.97912	2	.	50	66.7	1

6.2. RsSimulx simulx() function

simulx(): simulation of mixed effects models and longitudinal data

Description

Compute predictions and sample data from `Mlxtran` and `R` models

Usage

```
simulx(  
  model = NULL,  
  parameter = NULL,  
  covariate = NULL,  
  output = NULL,  
  treatment = NULL,  
  regressor = NULL,  
  occasion = NULL,  
  varlevel = NULL,  
  group = NULL,  
  project = NULL,  
  nrep = 1,  
  npop = NULL,  
  fim = NULL,  
  saveSmlxProject = NULL,  
  result.file = NULL,  
  addlines = NULL,  
  settings = NULL  
)
```

Arguments

model	<p>(<i>string</i>) a Mlxtran model used for the simulation. It can be a text file or an output of the inLine function.</p>
parameter	<p>One of</p> <ul style="list-style-type: none">• a <i>vector</i> of parameters with their names and values,• a <i>dataframe</i> with parameters defined for each id or each pop• a <i>string</i>, path to a data frame (csv or txt file)• a <i>string</i> corresponding to the parameter elements automatically generated by monolix (only when simulation is based on a Monolix project). One of the following mlx parameter elements: "mlx_Pop", "mlx_PopUncertainSA", "mlx_PopUncertainLin", "mlx_PopIndiv", "mlx_PopIndivCov", "mlx_CondMean", "mlx_EBEs", "mlx_CondDistSample"
covariate	<p>One of</p> <ul style="list-style-type: none">• a <i>vector</i> of covariates with their names and values.• a <i>dataframe</i> with covariates defined for each id• a <i>string</i>, path to a data frame (csv or txt file)• a <i>string</i> corresponding to the covariate elements automatically generated by monolix (only when simulation is based on a Monolix project). One of the following mlx covariate elements: "mlx_Cov" and "mlx_CovDist"
output	<p>output or list of outputs. An output can be defined by</p> <ul style="list-style-type: none">• a <i>string</i> corresponding to the output elements automatically generated by monolix (only when simulation is based on a Monolix project) – the format is <code>mlx_nameofoutput</code>.• a <i>list</i> with fields<ul style="list-style-type: none">• name: a <i>vector</i> of output names• time:<ul style="list-style-type: none">• a <i>vector</i> of times• a <i>dataframe</i> with columns id, time (columns lloq, uloq, limit are optional)

- a *string*, path to a data frame (csv or txt file)
- `lloq`: lower limit of quantification (when time is a vector of times)
- `uloq`: upper limit of quantification (when time is a vector of times)
- `limit`: lower bound of the censoring interval (when time is a vector of times)

treatment

treatment or list of treatments. A treatment can be defined by

- A *list* with fields
 - `time`: a *vector* of input times,
 - `amount`: a *scalar* or a *vector* of amounts,
 - `rate`: a *scalar* or a *vector* of infusion rates (default=Inf),
 - `tinf`: a *scalar* or a *vector* of infusion times (default=0),
 - `washout`: a *scalar* or a *vector* of boolean (default=F),
 - `adm`: (or `type`) the administration type (default=1),
 - `repeats`: the treatment cycle (optional), a *vector* with fields
 - `cycleDuration`: the duration of a cycle,
 - `NumberOfRepetitions`: the number of cycle repetition
 - `probaMissDose`: the probability to miss each dose (optional).
 - a *dataframe* with treatments defined for each id
 - a *string*, path to a data frame (csv or txt file)
 - a *string* corresponding to the treatment elements automatically generated by monolix (only when simulation is based on a Monolix project) – for example `mlx_Ad1`.

regressor

treatment or list of treatments. A treatment can be defined by

- a *list* with fields
 - `name`: a vector of regressor names,
 - `time`: a vector of times,
 - `value`: a vector of values.

	<ul style="list-style-type: none"> • a <i>dataframe</i> with columns id, time, and regressor values • a <i>string</i>, path to a data frame (csv or txt file)
occasion	<p>An occasion can be defined by</p> <ul style="list-style-type: none"> • A <i>list</i> with fields <ul style="list-style-type: none"> • name: name of the variable which defines the occasions, • time: a <i>vector</i> of times (beginnings of occasions) <ul style="list-style-type: none"> • A <i>dataframe</i> with columns id, time, occasion • a <i>string</i>, path to a data frame (csv or txt file) • "none", to delete an occasion structure
varlevel	deprecated, use occasion instead.
group	<p>a <i>list</i>, or a list of lists, with fields:</p> <ul style="list-style-type: none"> • size: size of the group (default=1), • parameter: if different parameters per group are defined, • covariate: if different covariates per group are defined, • output: if different outputs per group are defined, • treatment: if different treatments per group are defined, • regressor: if different regression variables per group are defined. <p>"level" field is not supported anymore in RsSimulx.</p>
project	the name of a Monolix project
nrep	Samples with or without uncertainty depending which element is given as "parameter".
npop	deprecated, set parameter = "mlx_popUncertainSA" or "mlx_popUncertainLin" instead.
fim	deprecated, set parameter = "mlx_popUncertainSA" or "mlx_popUncertainLin" instead.
saveSmlxProject	If specified, smlx project will be save in th path location (by default smlx project is not saved)

<code>result.file</code>	deprecated
<code>addlines</code>	<p>a <i>list</i> with fields:</p> <ul style="list-style-type: none"> • <code>formula</code>: string, or vector of strings, to be inserted . <p>“section”, “block” field are not supported anymore in RsSimulx. You only need to specify a formula. The additional lines will be added in a new section EQUATION.</p>
<code>settings</code>	<p>a <i>list</i> of optional settings</p> <ul style="list-style-type: none"> • <code>seed</code>: initialization of the random number generator (integer) (by default a random seed will be generated) • <code>id.out</code>: add (TRUE) / remove (FALSE) columns id and group when only one element (N = 1 or group = 1) (default=FALSE) • <code>kw.max</code>: deprecated. • <code>replacement</code>: deprecated, use <code>samplingMethod</code> instead • <code>samplingMethod</code>: str, Sampling method used for the simulation. One of “keepOrder”, “withReplacement”, “withoutReplacement” (default “keepOrder”) • <code>out.trt</code>: TRUE/FALSE (default = TRUE) output of simulx includes treatment • <code>sharedIds</code>: Vector of Elements that share ids. Available types are “covariate”, “output”, “treatment”, “regressor”, “population”, “individual” (default c()) • <code>sameIndividualsAmongGroups</code>: boolean, if True same individuals will be simulated among all groups (default False) • <code>exportData</code>: boolean, if True and if a path to save the smlx project (<code>saveSmlxProject</code>) is specified, export the simulated dataset (smlx project directory/Simulation/simulatedData.txt) (default False)

Details

simulx takes advantage of the modularity of hierarchical models for simulating different components of a model: models for population parameters, individual covariates, individual parameters and longitudinal data.

Furthermore, `simulx` allows to draw different types of longitudinal data, including continuous, count, categorical, and time-to-event data.

The models are encoded using either the model coding language `Mlxtran`. `Mlxtran` models are automatically converted into C++ codes, compiled on the fly and linked to R using the `RJSONIO` package. That allows one to implement very easily complex models and to take advantage of the numerical solvers used by the C++ `mlxLibrary`.

Value

A list of data frames. Each data frame is an output of `Simulx`

Examples

Example 1: simulation from scratch

This example shows how to create a Simulation from scratch and plot the results.

```
myModel <- inlineModel("
  [LONGITUDINAL]
  input = {A, k, c, a}
  EQUATION:
  t0 = 0
  f_0 = A
  ddt_f = -k*f/(c+f)
  DEFINITION:
  y = {distribution=normal, prediction=f, sd=a}
  [INDIVIDUAL]
  input = {k_pop, omega}
  DEFINITION:
  k = {distribution=lognormal, prediction=k_pop, sd=omega}
")
```

```
f <- list(name='f', time=seq(0, 30, by=0.1))
y <- list(name='y', time=seq(0, 30, by=2))
parameter <- c(A=100, k_pop=6, omega=0.3, c=10, a=2)
```

```
res <- simulx(model = myModel,
  parameter = parameter,
  occasion = data.frame(time=c(0, 0), occ=c(1, 2)),
  output = list(f,y),
  group = list(size=4),
  saveSmlxProject = "./project.smlx")
```

```
res <- simulx(model = myModel,
  parameter = parameter,
```

```

occasion = data.frame(time = c(0, 0, 0, 0),
occ1 = c(1, 1, 2, 2),
occ2 = c(1, 2, 3, 4)),
output = list(f,y),
group = list(size=4))

res <- simulx(model = myModel,
parameter = parameter,
output = list(f,y),
group = list(size=4))

plot(ggplotmlx() + geom_line(data=res$f, aes(x=time, y=f, colour=:
geom_point(data=res$y, aes(x=time, y=y, colour=id)))
print(res$parameter)

```

Example 2: simulation from a Monolix project

This example shows how to simulate from a Monolix project (using a Monolix demo): first with all the simulation elements exported from Monolix (Simulx resimulates the dataset from Monolix using the population parameter estimated in the Monolix project), then with a new design for the treatment (4 doses are given to all individuals), output (Cc is recorded instead of CONC, so the residual error is not included in the simulations) and group size (20 simulated individuals).

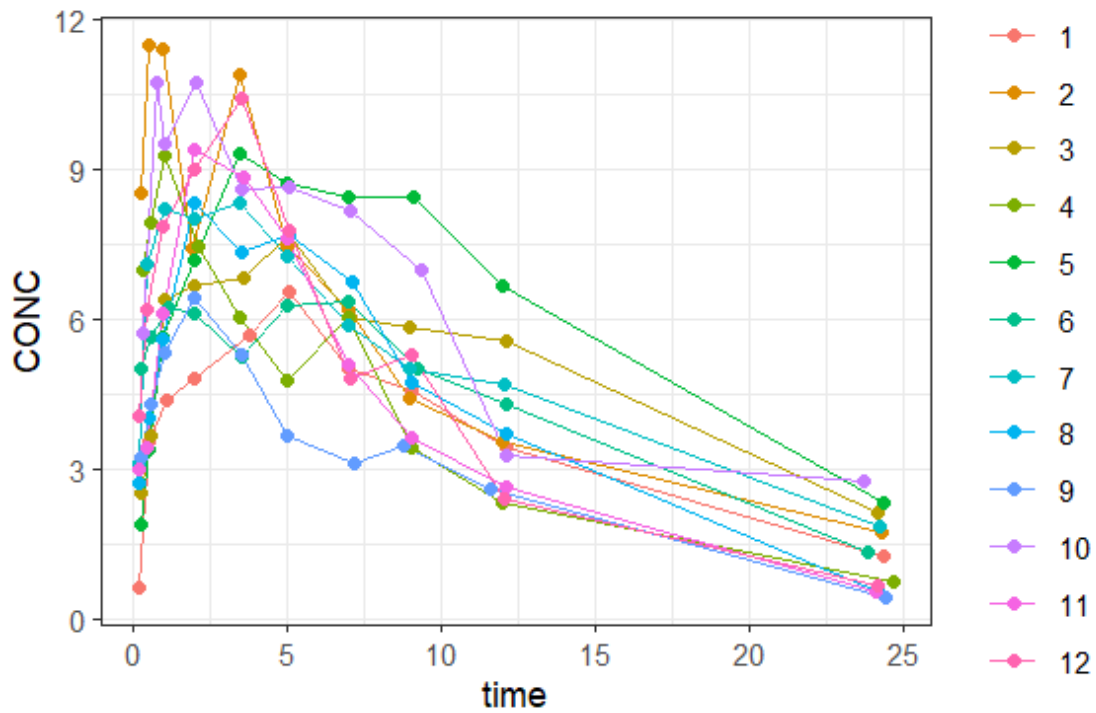
```

# First use the Monolix connectors to get the path to the demos ar
library(lixoftConnectors)
initializeLixoftConnectors(software = "monolix")
demoProject <- paste0(getDemoPath(), "/1.creating_and_using_models,
loadProject(demoProject)
runPopulationParameterEstimation()

# Then use RsSimulx for the simulations
library(RsSimulx)

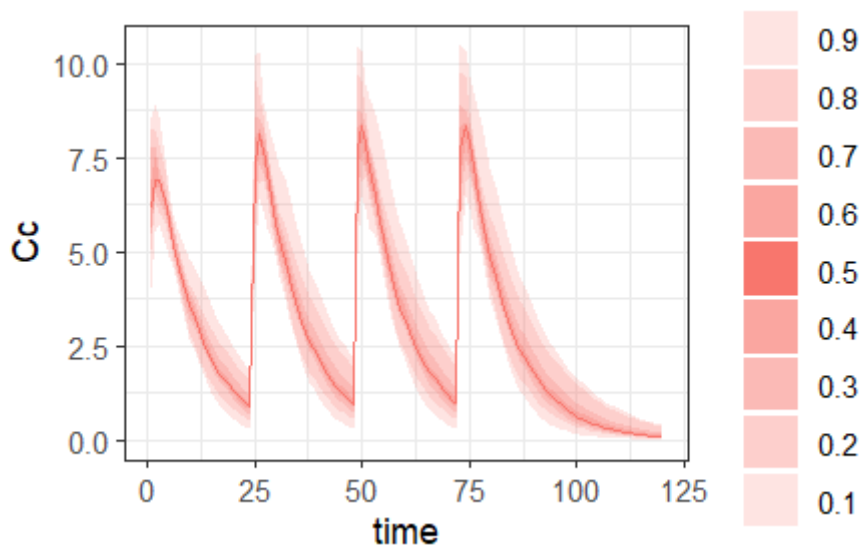
res <- simulx(project = demoProject)
plot(ggplotmlx() + geom_line(data=res$CONC, aes(x=time, y=CONC, c
geom_point(data=res$CONC, aes(x=time, y=CONC, colour=id)))

```



```
res <- simulx(project = demoProject,
              treatment = list(time=seq(0, by=24,length=4), amount
                                group = list(size=20),
                                output = list(name="Cc", time=1:120))
```

```
prctilemlx(res$Cc)
```



6.3. RsSimulx for backward compatibility of mlxR scripts

RsSimulx contains a function `simulx()` which works in the same way as the `simulx()` function of the `mlxR` package. **To use your scripts initially developed with `mlxR` with MonolixSuite 2020, 2021 and 2023, you need to replace `library(mlxR)` by `library(RsSimulx)` and run the rest of your script as before.**

Removed features

A few functions and features which were available in the `mlxR` package are not available in the `RsSimulx` package anymore, due to technical constraints. They are listed below.

- the following functions have been removed:
 - `pkmodel()`
 - `monolix2simulx()`
 - `readDataMlx()`
- the following arguments have been removed from the `simulx()` function
 - in `"treatment="`, the **"target"** field is not accepted anymore. Use `"adm"` instead.
 - in `"group="`, the **"level"** field is not accepted anymore, unless it is equal to `"individual"`. With `RsSimulx`, the variability is always at the individual level.
- The following outputs have been removed:
 - **originalID** data frame (but `originalID` is now part of result tables returned by `getSimulationResults()` in tables of individual parameters and outputs next to the ID column, and as a separate table `ID_mapping` if external elements have been used)

Modifications requiring small code adaptations

- the individual parameters and covariates are now outputted by default in the `'parameter'` argument. It is not necessary to request them explicitly.
- the `writeDataMlx()` function has been replaced by the function `exportsimulatedData()` which is called without R object as input. When called just after a `simulx()` call, it will use the `simulx` project in memory to write the data set. Note that the `'log'` will not be taken into account.
- overlapping occasions are now supported but when the occasion element has an `id` column, all elements must follow this occasions structure. See the [occasion definition](#) in `Simulx-GUI` for more details.
- for time-to-event outputs, it is now required to give a vector of two times (start and end of the observation period) instead of only the start time:

```
out <- list(name="Event", time=c(0, 400))
```
- Negative amount with positive infusion rates are not possible anymore. Use the `reset` or `empty` macros instead.
- it is not possible to define distributions in the `[COVARIATE]` or `[POPULATION]` block

anymore. Instead, sample the covariates or population parameters in the R script and provide them as a data frame.

- in the model, to properly identify covariates, individual parameters and population parameters, they must be declared in the corresponding block. The [model structure](#) is detailed on the [mlxtran](#) page.

Reproducibility

Running the same script with RsSimulx or mlxR will lead to different simulated values because the seed is read in a different way by the two packages. When running the same script several times with RsSimulx, the simulated values will be the same.

6.4. mlxR documentation



R package mlxR is deprecated. It is compatible with MonolixSuite versions up to 2019R2. To use the `simulx` function with newer versions of MonolixSuite, you can use [RsSimulx](#) package.

The legacy documentation for the mlxR package can be downloaded from [here](#). The documentation contains user guides for the package functions, as well as multiple case studies and R scripts.

7. Case studies

The case studies below show how to use Simulx for simulations.

Setting up a simulation from scratch

You can start a Simulx project by importing a Monolix run or you can define every element needed for the simulation from scratch. In this webinar, we explain how to setup a simulation by defining the model, parameters, treatments and outputs. A special focus will be given on the model writing in mlxtran language, with examples of translations from Nonmem and literature models.

The slides explain how to define each part of the mlxtran model and show screenshots of how to define the elements for the simulation in the GUI. The key points and results are also on the slides. The webinar can be reproduced step-by-step using the provided Simulx projects.

[Download slides](#) | [Download Simulx projects](#)

From Monolix to Simulx: how to explore new scenarios

When you have built a model, you can use it to compare new dosing regimens for the next trial, calculate the expected power of a study and select the most successful strategy? Learn effortless exploration of new scenarios with Simulx GUI – intuitive, flexible, and powerful application to simulate and compare countless strategies – and switch your focus to analysis and decision-making.

This video explains using Simulx with Monolix projects, it shows step-by-step how to simulate different groups, define target outcomes and assess the uncertainty and presents features that bring the best insight from simulations.

[Download slides](#) | [Download projects](#)

Optimizing Sample Size of a Phase III Trial

What if we could reduce the size of a phase III study from 400 to 80 subjects and make it twice as short? Thanks to model-informed clinical trial design, it's possible! Watch an inspiring story based on real-world case study data and learn about Simulx at the same time. In an hour, we will take you from the modeling of phase II studies in Monolix to the estimation of a phase III study power in Simulx to finally optimizing sample size and study duration. Example scripts to script Simulx from R as done in this webinar can be found [here](#).

[Download slides](#)

8. Frequently asked questions

- [Download, Installation, Run and Display issues](#)
- [General questions](#)

- Model
- Definition tab
- Exploration tab
- Simulation tab
- Results and Outputs
- Plots
- LixoftConnectors
- Share your Feedback

GENERAL

- **How can I reuse my previous mlxR scripts (which were using the simulx() R function)?**

Simulx has evolved a lot between the 2019 and 2020 versions, and its API also. mlxR (and its simulx() function) is not compatible with Simulx versions starting from 2020.

However Simulx 2020 is available via a very convenient and intuitive [GUI](#), and via a [complete R API](#), which allows to do from R any step that exists in the GUI.

To make the transition smoother, we also provide an additional [R package RsSimulx](#) with a simulx() wrapper function using the R API, very similar to the simulx() function of mlxR. All deprecated arguments coming from the mlxR syntax will generate a warning with RsSimulx versions starting from 2.0.0.

- **It is possible to automatically convert a Nonmem model to a Simulx model?**

No, but the conversion is relatively easy. For the [structural model](#), you can replace the Nonmem ADVAN routines 1-3 by the [pkmodel macro](#), and the [syntax for ODEs](#) is easy to adapt. Parameter distributions and covariate effects are defined in the [\[INDIVIDUAL\] section](#).

You can also check our case studies (video and slides) "[Setting up a simulation from scratch](#)" which gives examples of translations from Nonmem to Simulx.

- **How can I share a Simulx project?** A Simulx project is composed of a .smlx file, possibly external files you have used to define elements (model file, covariate table, etc) and a result folder. All these must be shared, keeping the relative path between them. In the Simulx project settings (**Settings > Project settings > "Save the user files in the results folder"**), you can request to save the external files (model and other input files) in the result folder. This way, you only need to share the .smlx file and the result folder.
- **How fast is Simulx?** Simulx is very fast because it performs the calculations using a C++ engine (the same as Monolix).
- **Is it possible to script my simulations?** Yes. All actions done in the GUI can also be done via R functions. [Have a look at the lixoftConnectors](#).

MODEL

- **Do I have to define correlations between all parameters (i.e full omega matrix)?** No. The omega matrix need to be block diagonal only. [See the correlation definition here](#).

DEFINITION tab

- **Is it possible to include uncertainty of the population parameters?** Yes. If you have obtained sets of population parameters via bootstrap, you can define the [population parameter element via an external file](#) with several lines, each line

corresponding to a different set of population parameters. You can also define the population parameters using the type "distribution" which will allow to sample the population distribution from an uncertainty distribution. You can for instance choose normal distributions and give the estimated standard errors in the "sd" column. However this does not include correlations between estimates. To include those, you need to sample to population parameters from their uncertainty distribution outside Simulx and provide them as external file. We will provide soon a R package Rssimulx to help for this.

Note that the a different of population parameters will be used for each replicate.

- **Is it possible to sample covariates from a truncated normal distribution?** Not yet. For the moment you can use a logit distribution if you need bounds, or sample the covariate elsewhere and provide an external file.
- **Can I flexibly define a treatment depending on covariates (such as WT or BSA)?** Yes, this is possible if you have the covariate appearing in your model. Select "scale amount by a covariate" in the treatment definition.

EXPLORATION tab

no question yet

SIMULATION tab

- **Can we perform trial simulations accounting for drop-outs?** You can define a joint model with PD and a time-to-event model representing the drop-outs. Simulx will simulate the time of drop-out, but not remove observations happening after the drop-out. This will request to post-process the results outside Simulx.
- **Can I calculate the time of Cmax?** This can be done in the **outcome definition**. Select the output corresponding to the concentration, then "max per id" and "time of max".

RESULTS and OUTPUTS

- **How can I get the simulated values?** The population parameters, individual parameters, doses, covariates and regressors are saved as .txt files in the Simulation folder of the result folder. By default the simulated values are not exported because they can be very large in size. However it is possible to choose this option in **Settings > Preferences > Export simulation files**. Outcomes and endpoints are saved in the Endpoints folder of the results.

You can also get the results using the [lixoftConnectors R package](#).

- **What is the best way to post-process the Simulx results in R?** You can use the R package [lixoftConnectors](#) to interact with your Simulx run. In particular, the function [getSimulationResults\(\)](#) allows you to recover all results. These results can then be post-processed using typical R code.

PLOTS

- **Is it possible to display a 90% prediction interval?** Yes, by default the different shades of blue in the “Output distribution” plot represent different percentiles. If in the “Display” settings in the right panel you choose `band=2` and `level=90`, then you will see a 90% prediction interval.

LIXOFTCONNECTORS

- **setGroupElement() worked in 2021R1 but gives an error in the 2021R2 version.** The error message has been added in the R2 version when potentially incompatible elements are set together, eg. parameters as EBEs and covariates. In this case you need to set each element separately, as the error message suggests.
- **getSimulationResults() does not return results** It can happen if the results are too large (too many data points). In the 2021R2 version there are additional two arguments, `ID` and `REP`, to request a subset of results.

SHARE YOUR FEEDBACK

- **I would like to give a suggestion for the future versions of Simulx.** [Share your feedback!](#)
- **I need more details in the documentation for a specific feature.** [Share your feedback!](#)


8.1. List of known bugs

The list of known bugs in every version of MonolixSuite can be found in the release notes of the next version.

- [Release Notes for MonolixSuite2024R1](#)
- [Release Notes for MonolixSuite2023R1](#)
- [Release Notes for MonolixSuite2021R2](#)
- [Release Notes for MonolixSuite2021R1](#)
- [Release Notes for MonolixSuite2020R1](#)

We list below the known bugs in Simulx 2024R1 (they will be fixed in the next version).
The 2024R1 version of MonolixSuite can be downloaded [here](#).

Help Guide Powered by Documentor

[New Project](#) [Definition](#) [Exploration](#) [Simulation](#) [R-functions](#)
[Case studies](#) [FAQ](#) [Single page](#) [PDF documentation](#) 



A web site of the network [Lixoft.com](#) | Follow us on [LinkedIn](#)