John Harrold & Anson Abraham

ubiquity.tools

ubiquity

# Useful Links

## Files and more information

**Project page**
ubiquity.tools

**Forum/Mailing list**
help.ubiquity.tools

**Analysis template**
template.ubiquity.tools

**Presentation PDF (this file)**
presentation.ubiquity.tools

**Handout PDF**

handout.ubiquity.tools

**Workshop Files**
workshop.ubiquity.tools

# Using These Slides

These slides are put together to act as both a tutorial and descriptive documentation. An index of the more important components can be found at the link (IDX) at the bottom of each slide. Each slide also contains a link to the main Table of Contents (TOC ) which can be used to jump between the major parts.

Directory names, `system.txt` snippets and examples, MATLAB commands and etc. are written using `this font`

Files for the examples and exercises discussed here are found in their respective directories. These will be listed at the bottom of slides when relevant.

A **state** is used to refer to a variable that is defined by a set of differential equations. These are sometimes referred to as compartments, but this can be a little misleading since many different states can exist in the same physiological space.

# Using These Slides
## Alpha and Beta

This documentation attepts to provide access to as many features as possible, while acknowleding that development is a continuous process. Certain features may not be complete or lack extensive testing. To identify these features, alpha and beta flags are placed at the bottom left of the relevant slides:

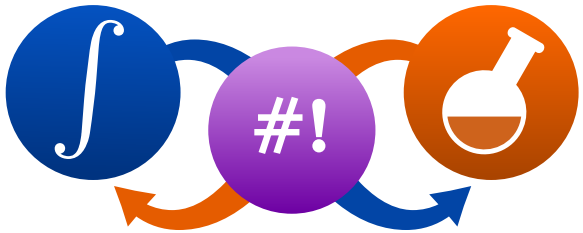

This is a feature that is close to completion and it may not even require much more to be considered "finished". It is provied for two reasons: First it might be useful to users in the current state. Second to find users who may be able to provide input and roudn off the edges.


This is a new feature that is complete and in working order to the best of our knowledge, but because it is new results should be scrutinized accordingly.

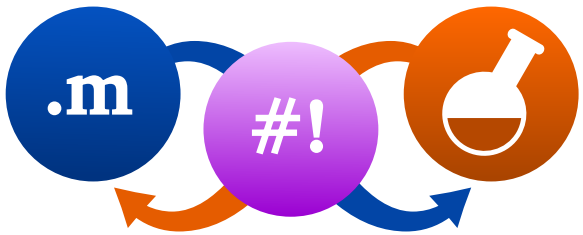
If you want to help progress the features marked alpha, or if you have any problem with beta or any other features, feel free to post on the forum: help.ubiquity.tools

# Overview
## Ubiquity model development tools

**Ubiquity Language**




**MATLAB Workflow**

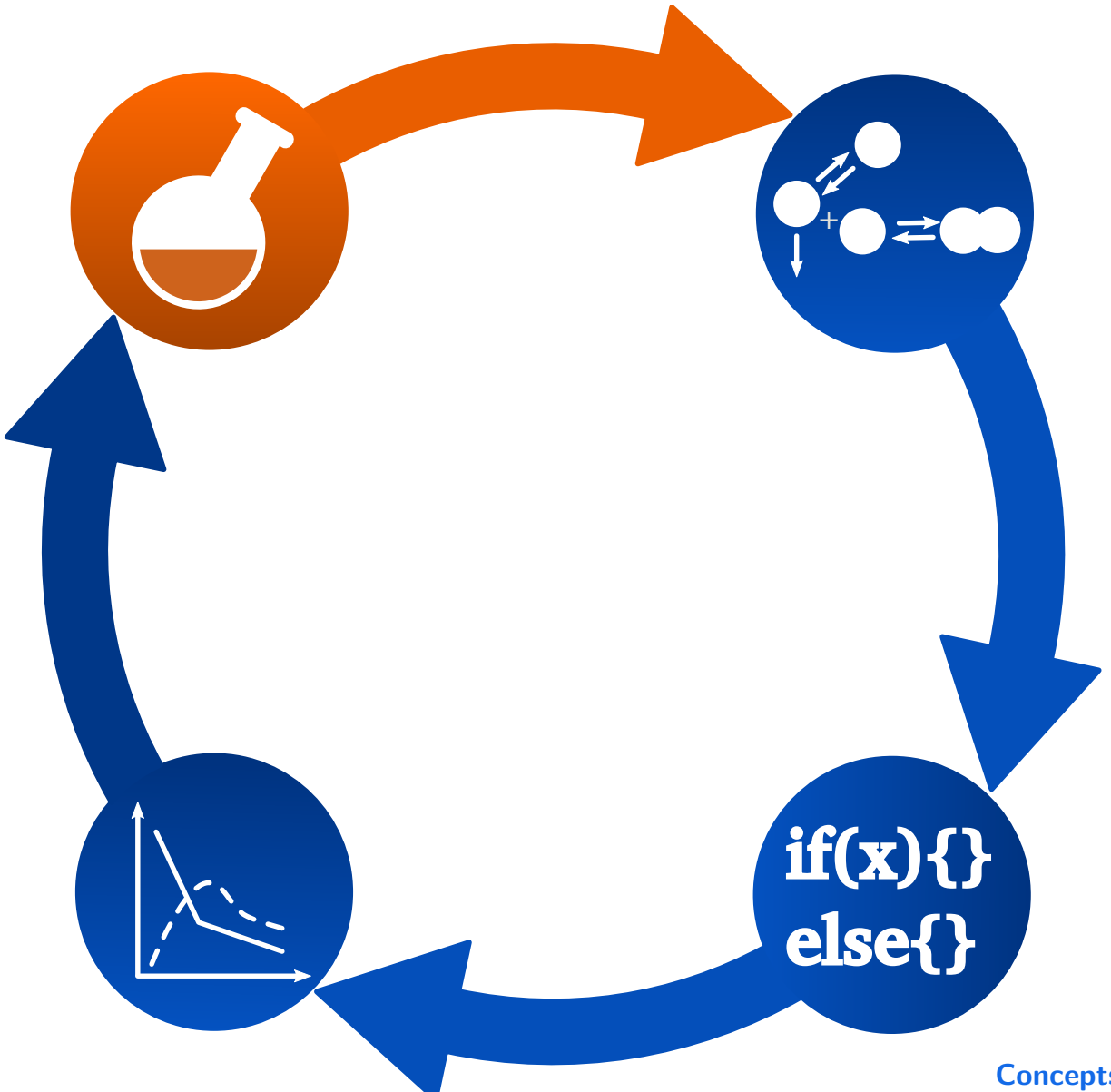


# Table of Contents: Ubiquity Language I

# Table of Contents: Ubiquity Language II

# What is the role of a Modeller?

## Interactions with experimentalists and other scientists



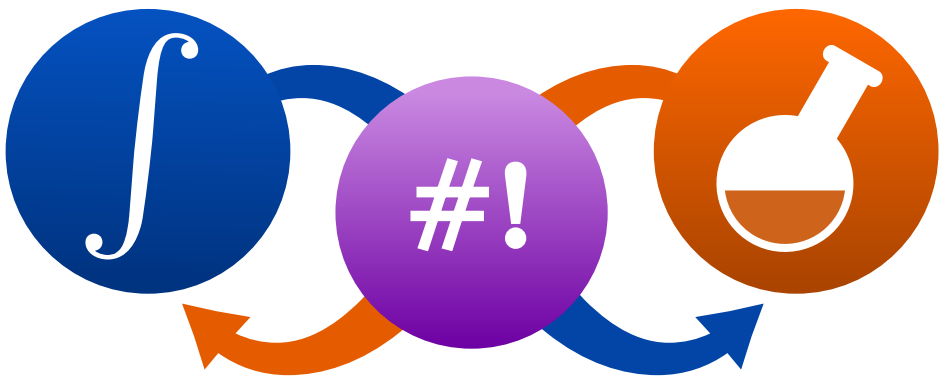
# Improving Efficiency & Satisfaction

When it comes to modeling, what slows you down? What limits your ability to accomplish your objectives?

Tedium — Beyond just typing out ODEs

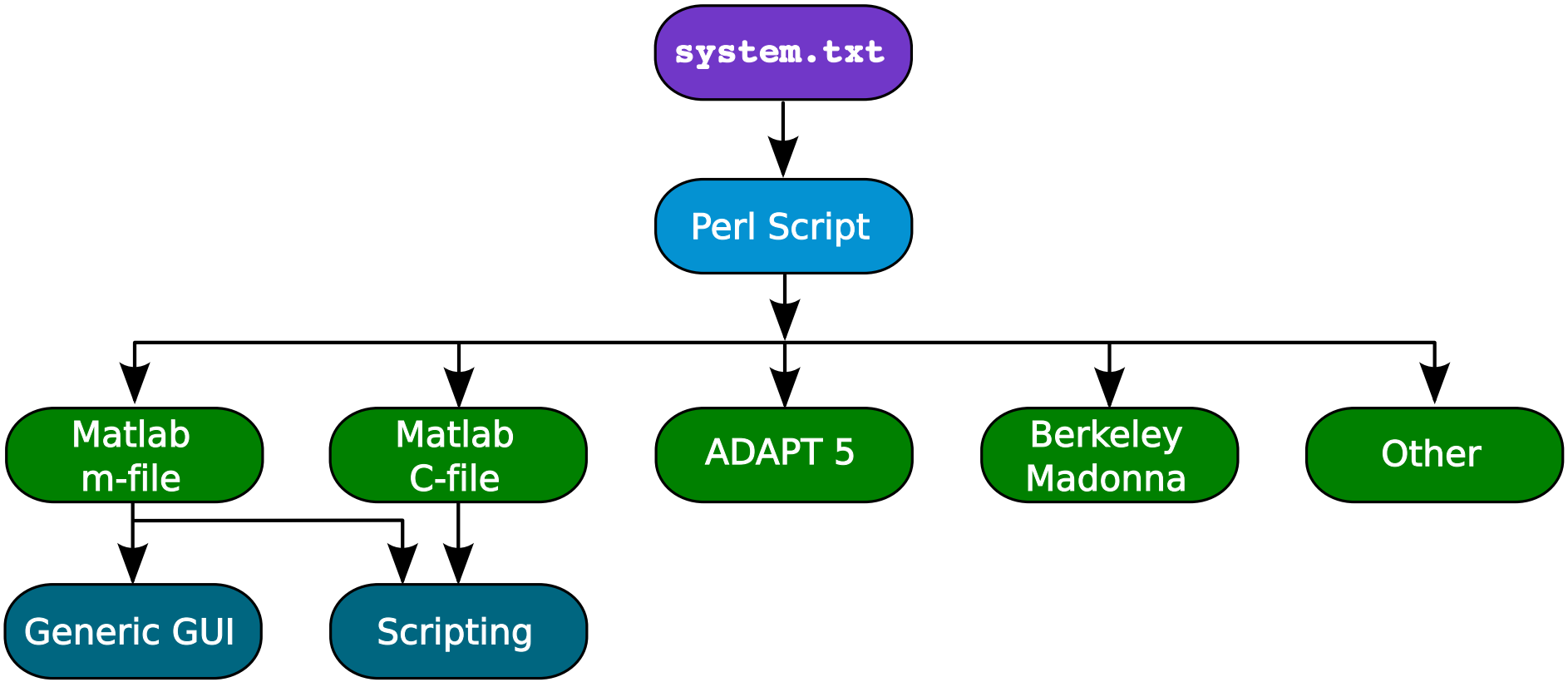
Transferability — Sharing my work with other modelers

Accessibility — Engaging and involving other scientists



# Methodology

**It's not where you're going but how you get there.**



# Quicklook

# Quicklook: Important Files

## template.ubiquity.tools

system_help.txt — Reference file intended to provide documentation for the different aspects of this framework. If you want to know how to do something, this is where you should look.

build_system.pl — Perl script used to convert a system.txt file into the different targets used.

system_template.txt — Contains commented examples of all the different system descriptors used here. Copy this file to system.txt to start a new system from scratch.

system_help_reserved_words.txt — This file is generated after the system has been built. It contains a list of reserved words that should be avoided in terms of parameter and state names. The build script should warn you if you are using any of these words.

# Quicklook: Parameters

**Delimiters:** `<P>`, `<As>`, & `<Ad>`

## System parameters

```
#   name  value  lower  upper  units  editable  grouping
#                bound  bound
<P> CL    1.0    eps    inf    L/hr   yes       System
<P> Vp    1.0    eps    inf    1/hr   yes       System
```

## Secondary Parameters (Static)

If we wanted to calculate the rate of elimination (`kel`) and half-life (`thalf`) we would do it in the following way (order is important).

```
<As> kel = CL/Vp
<As> thalf = SIMINT_LOGN[2.0]/kel
```

Static secondary parameters can be defined in terms of system or previously defined static secondary parameters.

## Secondary Parameters (Dynamic)

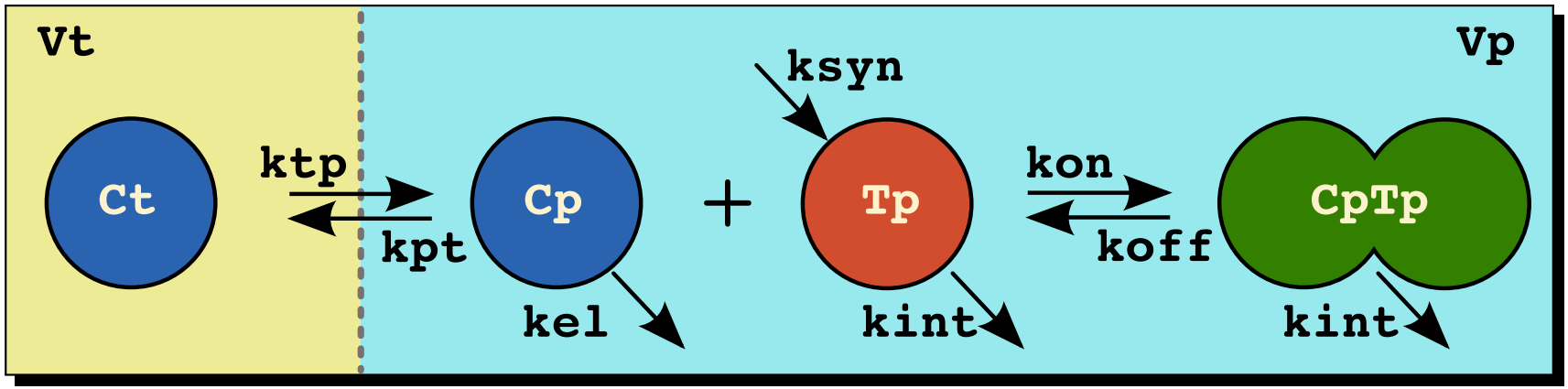
If you are modeling amount `Ap` and want to define concentration, you can create a dynamic secondary parameter.

```
<Ad> Cp  = Ap/Vp
```

Dynamic secondary parameters can be defined in terms of system parameters, static secondary parameters, or previously defined dynamic secondary parameters.

# Quicklook: System Components

**Delimiters:** <ODE:?>, <C>, <S:?> & <=?:?=>




## ODEs

```
<ODE:Ct>   Cp*kpt*Vp/Vt  - Ct*ktp
<ODE:Cp>  -Cp*kpt        + Ct*ktp*Vt/Vp -  kel*Cp    + koff*CpTp - kon*Cp*Tp
<ODE:Tp>                 + ksyn/Vp      - kint*Tp    + koff*CpTp - kon*Cp*Tp
<ODE:CpTp>                              - kint*CpTp  - koff*CpTp + kon*Cp*Tp
```

## Processes

```
# Compartments
Ct; Vt; ktp <C> Cp; Vp; kpt
# Equilibrium
Cp + Tp <=kon:koff=> CpTp
```

```
# Target Turnover:
ksyn/Vp <S:Tp>   kint*Tp
        <S:Cp>    kel*Cp
        <S:CpTp> kint*CpTp
```

examples/system-tmdd.txt

# Quicklook: System Inputs

**Delimiters: <B:?> & <R:?>**

Information about system inputs is (dosing units, default values, etc) are specified <B:?>, for bolus inputs and <R:?> for infusion rates.

## Bolus Inputs

A row must be specified for default bolus times and one for each state to receive a bolus. Values are specified in the units listed and the scale is a mathematical expression that converts dosing in the specified units into the system units.

```
#  type       state   values      scale   units
<B:times>;             [0 ];        24;   days
<B:events>;   Cp;      [10];     70/V1;   mpk
```

## Infusion Rates

Rates are sepecified in pairs (times and levels) that are grouped together with a rate name (myrate). This name is used in the ODEs. Levels represent the rate of infusion which are held constant until the next time. The scale is used to scale from the specified units into the system units.

```
#  name      time/levels  values scale  units
<R:myrate>;  times;       [0 30]; 1/60;  min
<R:myrate>;  levels;      [1  0];   60;  mg/min
```

# Quicklook: Building the System

## Stand Alone

The only requirement is that you have perl installed (no additional modules). To build the system just run the script `build_system.pl`, and look in `transient/` for the outputs.

## In MATLAB

MATLAB has a built in perl interpreter, and you can build the system using `build_system.m` (notice the `.m` extension). After building the system, you can then run the model either through the GUI or as a script:

Use `model_gui` to run the model interactively and `build_exe.m` to create a stand alone executable.

Copy the file `transient/auto_simulation_driver.m` to the main directory in the template. This is an example with the components you might want to change in order to run the model from a script.

# system.txt
# Basic Components

# System Parameters

<P>

System parameters are specified using the <P> identifier. This is followed by seven different fields separated by spaces. Because spaces are used to separate the fields, **the fields may not contain spaces**. The clearance parameter would be defined in the following way:

```
#    name   value   lb   ub   units editable grouping
<P>  CL     1.0     eps  inf  1/hr  yes      System
```

The comments are used used to indicate what each field represents. The first field is the parameter name followed by a nominal value. The bounds of the parameters are then specified. In this case the lower bound `eps` represents the smallest nonzero number and `inf` is positive infinity. Basically saying the parameter has to have a positive value. To be boundless the lower bound would be `-inf`.

The next three fields are used in the GUI. The `units` are obvious. The editable field can be either `yes` to indicate that it is displayed in the GUI and the user can change it, or `no` and the variable will be hidden and fixed at the nominal value. The grouping (`System` above) is used to group parameters in the GUI.

# Secondary Parameters

## <As> & <Ad>

While we can write everything in terms of system parameters, it's convenient to break things down into intermediate calculations or secondary parameters. There are two different types of secondary parameters used here:

**Static Secondary Param.**
Parameter does not change during a simulaiton. These are specified using the <As> delimiter and can be written in terms of system parameters or previously defined secondary parameters.

```
<As>  kel = CL/Vp
```

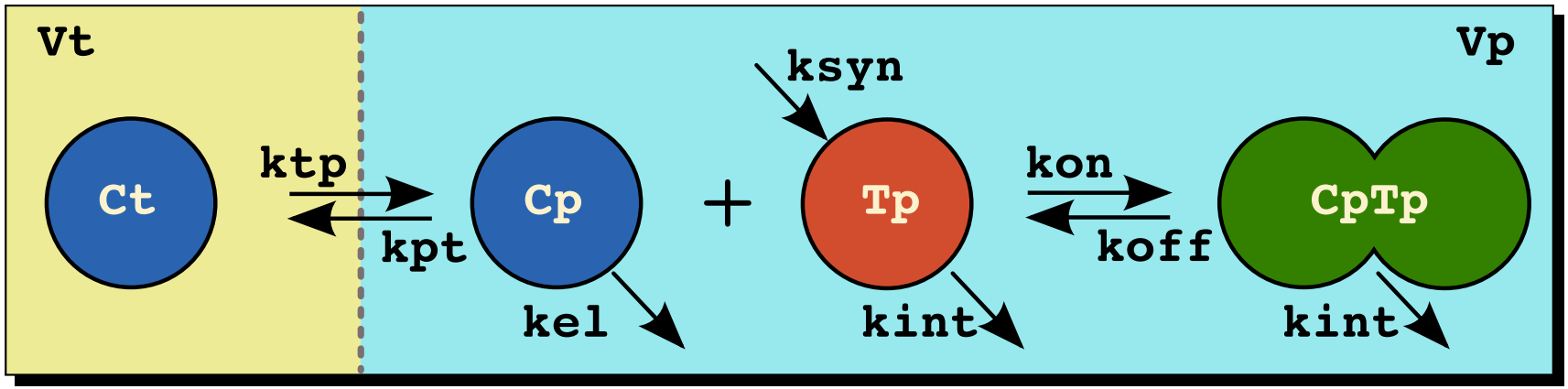
**Dynamic Secondary Param.**
Parameter can change during a simulation. This typically means it is defined in terms of a state or another dynamic secondary parameter. The <Ad> descriptor is used:

```
<Ad>  Cp = Ap/Vp
```

Static secondary parameters can be used to define initial conditions (**More**), and dynamic secondary parameters are used to create piecewise-continuous variables (**More**). For both, the **order** of definition is important.

# Differntial Equations

<ODE:?>




With the TMDD system shown above, there are four different states: The free drug in the tissue (Ct) and in the plasma (Cp) space; the free target (Tp) and the drug/target complex (CpTp). The most straight-forward way to describe this system is to write out the differential equations using the <ODE:?> descriptor:

```
<ODE:Ct>   Cp*kpt*Vp/Vt - Ct*ktp
<ODE:Cp> -Cp*kpt        + Ct*ktp*Vt/Vp    + koff*CpTp - kon*Cp*Tp
<ODE:Tp>      + ksyn/Vp       - kint*Tp   + koff*CpTp - kon*Cp*Tp
<ODE:CpTp>                    - kint*CpTp - koff*CpTp + kon*Cp*Tp
<ODE:Cp>                      -  kel*Cp
```

notice the multiple entries for Cp. These components are simply **added** together internally.

**Is this the simplest way to represent this system?**

# Process-Centric Alternative

## Beyond writing out ODEs

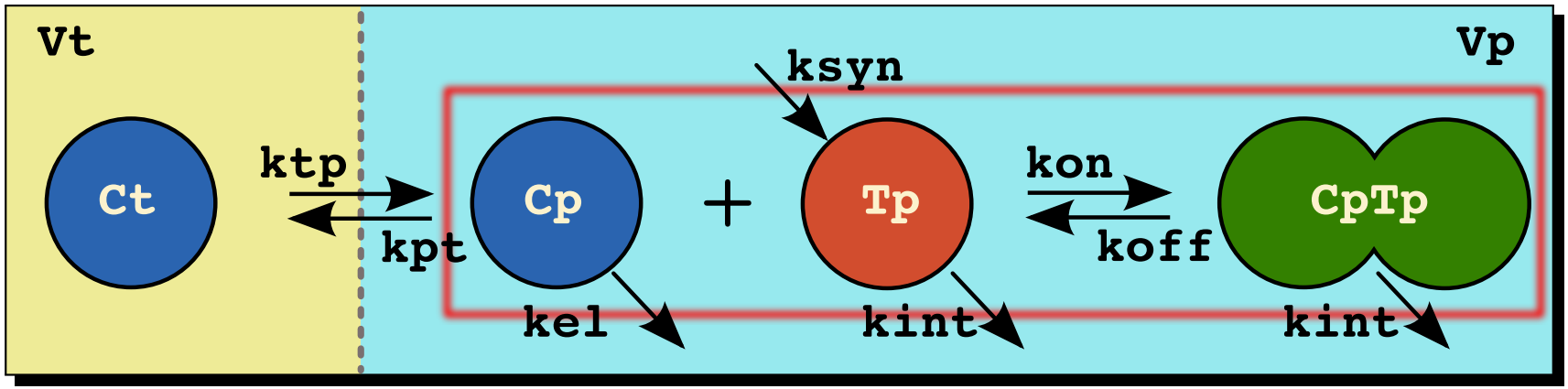
Consider systems where multiple species are in equilibrium together–trimers and quadramers forming–writing out ODEs can become tedious and error-prone. It may be desirable to represent the system instead as a combination of the underlying processes that are occurring. This framework provides the following additional methods for creating a system:

- Chemical Reactions — notations for describing chemical reactions as well as equilibria
- Turnover Processes — representing the rates of production and consumption of a species
- Inter-Compartmental Movement — convenient methods for describing movement of species between physiological compartments

**These components can be used together along with the ODE descriptor to construct the full system**

# Processes: Chemical Reactions

<=?:?=> or =?=>




Chemical reactions can be specified using two different notations. Either as an equilibrium reaction or forward reactions. The two are equilvalent and can be used to represent the system above:

**Equilibrium**

```
Cp + Tp <=kon:koff=> CpTp
```

**Reaction Rates**

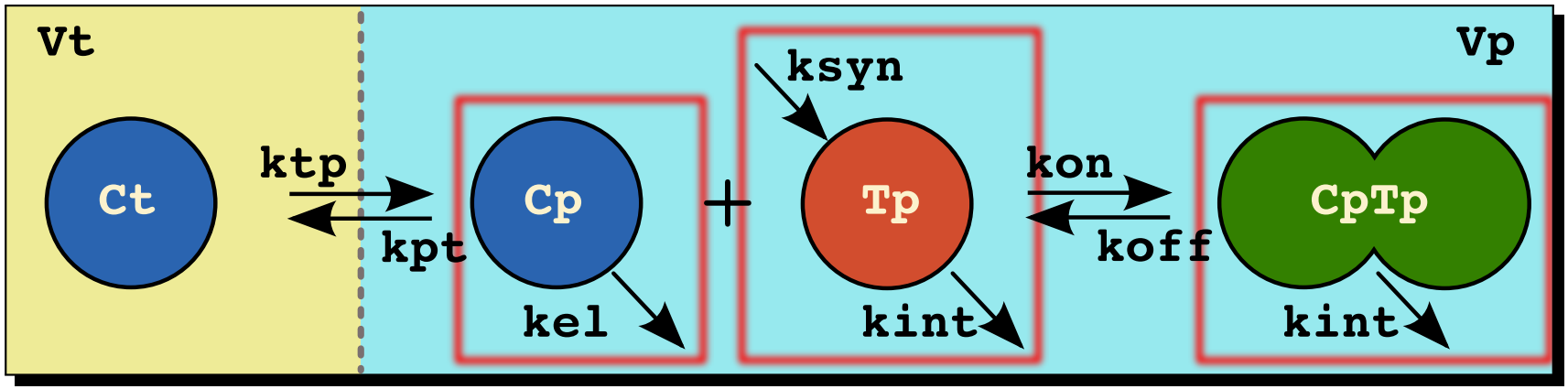
```
Cp + Tp =kon=>  CpTp
CpTp    =koff=> Cp + Tp
```

A couple things to note here. The on and off rates must have already been defined as **system** or **secondary parameters**. Also `Cp`, `Tp` and `CpTp` must be **states** and cannot be parameters.

examples/system-tmdd.txt

# Processes: Turnover

<S:?>




The reaction rates can be used to understand conversion of one species into another. The production and elimination of a species can be specified using the <S:?> notation. The following are both equivalent:

**Turnvoer**

```
ksyn/Vp <S:Tp>   kint*Tp
        <S:Cp>    kel*Cp
        <S:CpTp> kint*CpTp
```

**Turnover & ODEs**

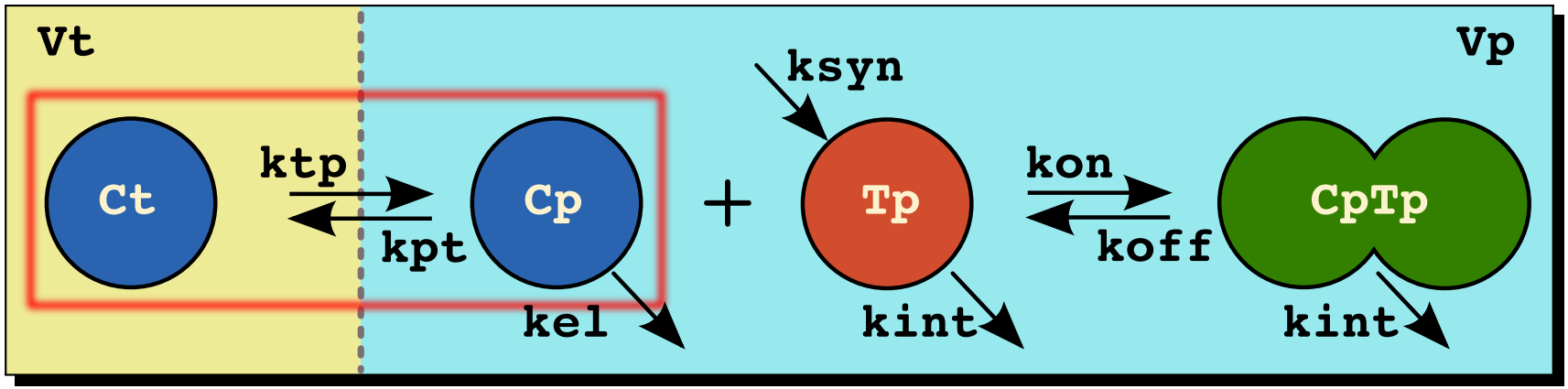
```
ksyn/Vp <S:Tp>  kint*Tp
<ODE:Cp>      - kel*Cp
<ODE:CpTp>    - kint*CpTp
```

On the left side of the <S:Tp> rates of production of Tp are listed. And the rates of elimination or consumption are listed on the right side. Multiple rates can be listed and these are separated by semicolons (;).

examples/system-tmdd.txt

# Processes: Inter-Compartment Transport

<C>




Movement between compartments contains three pieces of information for each compartment: a **state**, the **volume** of the compartment, and the **rate** of loss from the compartment. These are specified for this system using the following syntax:

```
Ct; Vt; ktp <C> Cp; Vp; kpt
```

Note that `Ct` and `Cp` are **states** and the volumes (`Vt` & `Vp`) as well as the rates (`ktp` & `kpt`) are **parameters**. This statement is expanded internally to account for the movement between compartments. If the states were amounts in stead of concentrations (e.g. `At` & `Ap`) then the volumes would just be `1.0`:

```
At; 1.0; ktp <C> Ap; 1.0; kpt
```

examples/system-tmdd.txt

# Other State Information

## Initial Conditions: <I>

It is not necessary to declare a state. This is done automatically when one of the process descriptors (<ODE:?>, <=?:?=>, =?=>, <C>, or <S:?>) are used. These different components are used to create the ODEs which describe the system.

Any system of ODEs needs to have initial conditions defined. By **default**, all states will have **initial conditions = zero**. It's possible to specify non-zero initial conditions using the <I> descriptor. General format is:

```
<I> state_name = state_IC
```

**The initial conditions can be a number, parameter, static secondary parameter or an expression with any of these.**

For the TMDD system shown on the previous slide, the initial condition of the state Tp may be defined in terms of the synthesis and internalization rate in the following way:

```
<P> ksyn  1.0  eps  inf  nmoles/hr  yes System
<P> kint  1.0  eps  inf       1/hr  yes System
<P> Vp    1.0  eps  inf       L     yes System
<As> Tp_IC = ksyn/Vp/kint
<I> Tp = Tp_IC
```

# Bolus Dosing

<B:?>

The <B:?> descriptor is used to define bolus events. These act as default values in the GUI and as placeholders for scripts; both can be altered by the user. Dosing informaiton is broken down into a list of `times` that bolus events can occur and a list of `events` for each compartment that will receive a bolus dose.

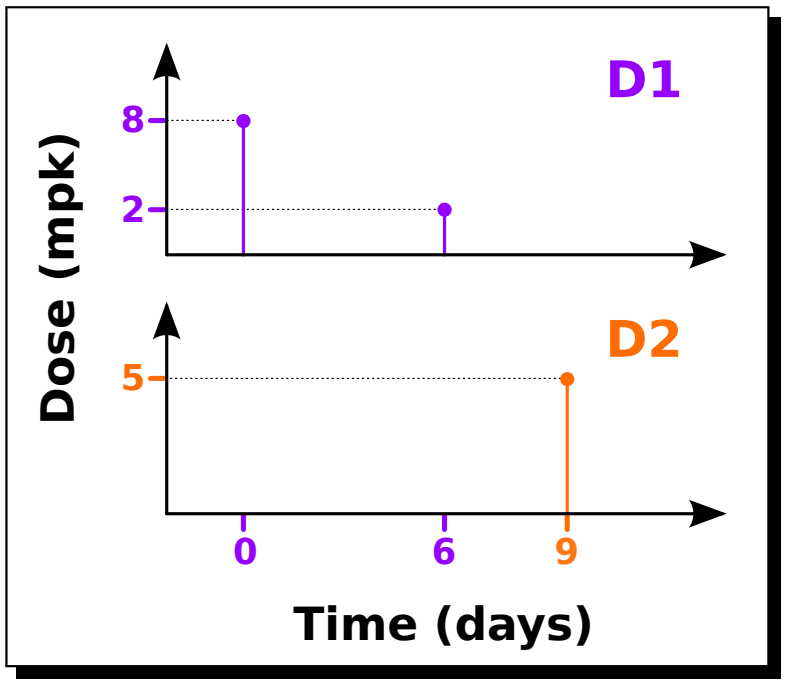
Each of these has a `scale` that is used to convert the bolus dosing information from proscribed units (mg once daily) into the units in which the system is coded (nM and hours). So if dosing is done on days 0, 1, 2... but the simulation time is hours, then the scale for the dosing times is 24.

The events contain the magnitude of the bolus at a given time. If you have multiple states receiving a bolus, the times must include all times in which a bolus may be applied to the system. If a state does not receive a bolus on a particular time, it's magnitude at that time is 0.

# Bolus Dosing (Contd.)

## A complete example

In this example we want to dose two different drugs into two differnt states. Drug 1 (**D1**) will be dosed into `Cp1` and drug 2 (**D2**) into `Cp2`. Dosing will be in mg/kg but concentrations are in mg/ml. The dosing time is in days, but the simulation time units are hours. We will be dosing **D1** at **8** & **2** mpk on days **0** & **6**. **D2** will be dosed at **5** mpk on day **9**.




```
#  type          state          values          scale       units
<B:times>;                     [0 6 9];          24;        days
<B:events>;      Cp1;          [8 2 0];        70/V1;       mpk
<B:events>;      Cp2;          [0 0 5];        70/V2;       mpk
```

Assume `V1` and `V2` are the comparmental volumes for **D1** and **D2** in ml, and the subject body weight is 70 kg.

# Infusion Rates

<R:?>

Rates of infusion are defined using the <R:?> descriptor. Like bolus values, infusion rates have two components. There is a component that specifies switching times. And each switching time has a correspoinding rate of infusion. This infusion rate will be held constant until the next time. Also like the bolus specification there is a **scale** associated with both infusion times and the levels that converts the proscriptive units into the units of the simulation. Consider the following example:

```
#  name      time/levels  values  scale    units
<R:myrate>;  times;       [0 30]; 1/60;    min
<R:myrate>;  levels;      [1  0];   60;    mg/min
```

These two entries create the infusion rate called myrate. This can be used in any of your system specifications (e.g., <ODE:Cp> myrate/Vp). The first row specifies the times when the rate is changed (0 and 30 minutes). If the system is coded in terms of hours, then the scale of 1/60 must be used. The levels indicate a rate of 1 mg/min that is switched off at 30 minutes. This has to be converted to mg/hr using the scale of 60.

# Covariates

## <CV:?> & <CVTYPE:?>

For simulation purposes covariates (normally found in a data set) need to be defined. Covariates can be be either constant or change with time. The times must be the same scale as the system. The following defines the value for the covariate RACE:

```
<CV:RACE>; times;  [ 0];     weeks
<CV:RACE>; values; [ 1];     race
```

Covariates can also change with time. In this case consider the subject weight WGT. It begins at 70 and measurements are made at several time points.

```
<CV:WGT>; times;  [ 0 10 20 30 60];     weeks
<CV:WGT>; values; [70 65 60 58 56];     kg
```

Next we can alter how the simulations will interpret these values. By setting the type of covariate. By default the weight will be linearly interpolated (type = 'linear'), however we can hold the weight constant until the next measurement is encountered by declaring the type as 'step'

```
<CVTYPE:WGT> step
```

**Note the time units must be the same as the simulation time.**

# Covariates and Parameter Sets

## <CVSET:?:?> & <PSET:?:?>?

When a covariate has been defined, it will be associted with the default parameter set and all other sets that have benen created. If you have created a parameter set using the <PSET> notation, you can overwrite the value of a covariate for that parameter set using the <CVSET> notation.

For example, if the model was parameterized for male and female subjects we can define two parameter sets. Using the definintion of the WGT covariate to be associated with males (default) parameter set. We can now create a new parameter set (female):

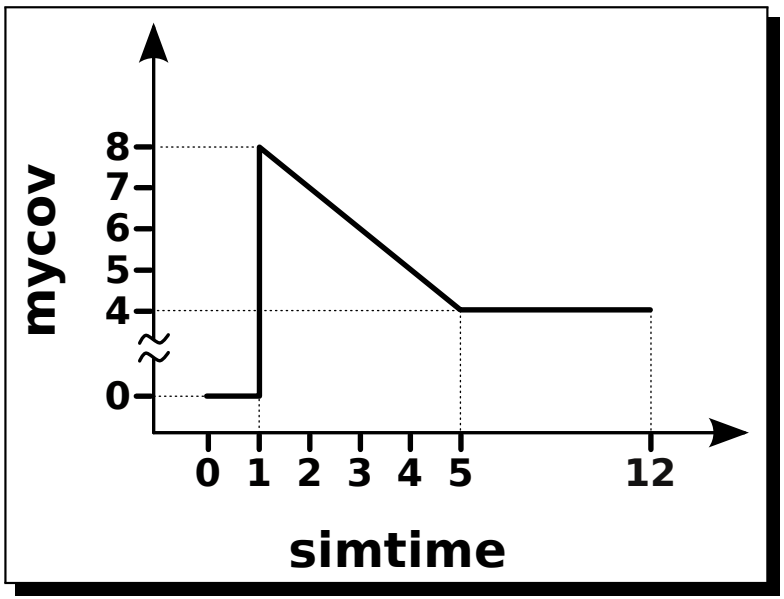
```
<PSET:default>  Male
<PSET:female>   Female
```

And the default values for the covariate can be changed for the set 'female':

```
<CVSET:female:WGT>; times;   [ 0 10 30 50]
<CVSET:female:WGT>; values;  [60 55 52 50]
```

# Complex Covariate Profiles

It is possible to create time varying inputs using the <CV:?> notation. For example the the input profile to the right can be constructed using the code below.




```
<CV:mycov>   ;  times; [0 .999 1 2 3 4 5 12 12.001];  hours
<CV:mycov>   ; values; [0 0    8 7 6 5 4 4  0     ];  -
<CVTYPE:mycov> linear
```

# System Outputs

<O>

Outputs are defined here in terms of states, parameters, secondary parameters, and input rates listed above. The format used is:

```
<O> name = expression
```

For example:

```
<O> A_obs      = A
<O> Coverage   = A/(KD + A)
```

Outputs that begin with QC will not be displayed in the GUI. This is intended to make them available at the scripting level for quality control (QC) purposes.

# Operators & Functions

SIMINT_

Most of the standard operators behave as expected (+, -, *, & /) because most languages use these consistently. There are however certain operators and functions that differ between languages. For example, consider the power function ($a^b$). In FORTRAN this would be a**b, in MATLAB it is a^b, and in C it is pow(a,b).

Now given the objectives here (write once and create multiple formats), this can be quite challenging. The solution used here is to convert language specific functions and operators into generic functions. So the power operator would be:

```
SIMINT_POWER[a][b]
```

This would then be converted to the appropriate output format depending on the output target.

# Operators & Functions

| Operator/Function | Example | Format |
|---|---|---|
| power | $a^b$ | SIMINT_POWER[a][b] |
| exponential | $e^a$ | SIMINT_EXP[a] |
| log base 10 | $\log(a)$ | SIMINT_LOG10[a] |
| log base e | $\ln(a)$ | SIMINT_LOGN[a] |
| less than | $a < b$ | SIMINT_LT[a][b] |
| less than or equal | $a \leq b$ | SIMINT_LE[a][b] |
| greater than | $a > b$ | SIMINT_GT[a][b] |
| greater than or equal | $a \geq b$ | SIMINT_GE[a][b] |
| equal | $a == b$ | SIMINT_EQ[a][b] |
| and | $a$ *and* $b$ | SIMINT_AND[a][b] |
| or | $a$ *or* $b$ | SIMINT_OR[a][b] |

# Timescales

<TS:?>

Each system has default units in which it is constructed, and should be indicated in the comments of the model. It can be useful (for generating figures for example) to show simulations in different time scales. Now this can be achieved by multiplying the time outputs by the correct scaling factor. However this requires the end user to (1) remember the original timescale and (2) correctly scale that value.

Now while this is not particularly challenging from a mathematical perspective, it introduces a chance for error. It is possible, instead, to specify time scale information using the <TS:?> descriptor. If the system is coded in hours, the following will define timescales for the default (hours), days, weeks and months:

```
<TS:hours>  1.0
<TS:days>   1.0/24.0
<TS:weeks>  1.0/24.0/7.0
<TS:months> 1.0/24.0/7.0/4.0
```

Here ? represents the name of the timescale and the numeric expression is the scale that converts the system time (hours) into the specified timescale.

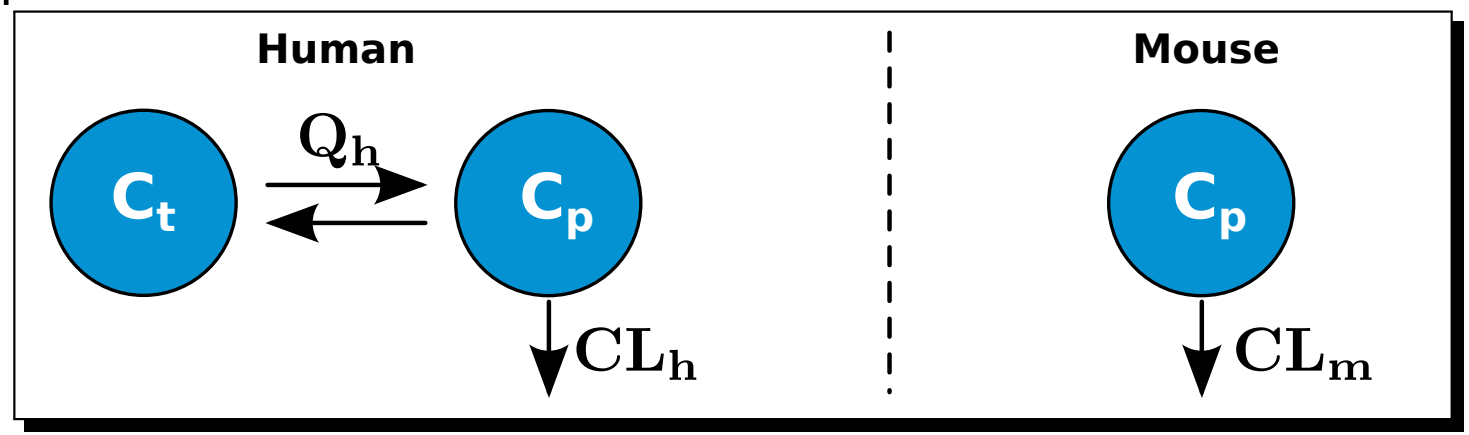
These are used both in the **GUI** and at the command line in MATLAB and **R**

# Extended Features

# Parameter Sets

## Describing multiple species, diseased/non-diseased, etc. <PSET:?:?>?

Often a model will be developed to incorporate different situations or scenarios. For example, a model may be used to describe both healthy and diseased individuals. When these differences are simply parametric in nature, it can be cumbersome to code a model multiple times (once for each parameterization). This framework provides a mechanism for including multiple parameterizations withing the same input file. Consider the system below where we want to describe antibody disposition. For humans this is described by a two compartment model, but for mice a single compartment is needed.



# Default Parameter Set

<PSET:default> default name

First we create a set of parameters describing the human scenario. These are the mean parameters taken from the literature [DM]:

```
<P> Weight   70.0    eps     inf      kg      yes    System # Organism weight
<P> CL       0.0129  eps     inf      L/hr    yes    System # Systemic Clearance
<P> Q        0.0329  eps     inf      L/hr    yes    System # Inter-compartmental
     clearance
<P> Vp       3.1     eps     inf      L       yes    System # Vol. central compartment
<P> Vt       2.8     eps     inf      L       yes    System # Vol. peripheral
     compartment
```

When a parameter is created using the <P> descriptor it is part of the default parameter set. This is the short name for a parameter set. A longer more verbose name can be given as well, and this is what will be seen in the GUI. The human parameter set can be labeled using the PSET descriptor in the following way:

```
<PSET:default> mAb in Human
```

Where default is the parameter name, and "mAb in Human"is the value shown to the user in the GUI.

[DM] Dirks & Meibohm Clin. PK 2010 Oct 1;49(10):633-59

# Alternate Parameter Sets

<PSET:?:?>?

Next, to add the parameterization for mice we simply create a new set in the following way:

<PSET:mouse>  mAb in Mouse

This alone would create a new parameter set with a short name mouse, and is an exact copy of the default parameter set. To identify the parametric differences between the mouse and human we use PSET in the following way:

```
<PSET:mouse:Weight>   0.020   # 20 gram mouse
<PSET:mouse:CL>       7.71e-6
<PSET:mouse:Q>        0.0
<PSET:mouse:Vp>       1.6e-3
<PSET:mouse:Vt>       1       # arbitrary
```

Consider the clearance parameter entry where we want the murine half-life of an antibody [VR]:

<PSET:mouse:CL>  7.71e-6

We use the set name (mouse) and the parameter name (CL) and then we overwrite the default with the specified value 7.71e-6.

*[VR] Vieira & Rajewsky Eur J Immunol. 1988 Feb;18(2):313-6*

# Parameter Sets and Their Use

## Implications for analysis

### Target Formats

Once the system is built, it will create input files for many different pieces of software. For target, an appropriate input fill will be created for each parameter set. For ADAPT a single FORTRAN file is generated, but multiple `prm` files are created. In the previous example, two Berkeley Madonna files will be created, and they are distinguished by the short name for the parameter set:

`target_berkeley_madonna-default.txt`

`target_berkeley_madonna-mouse.txt`

### MATLAB GUI & Scripting

In the GUI, a pulldown menu above the parameters will allow the user to select the active parameter set. The template also contains scripts and functions for a MATLAB workflow. This is described more fully in that section (**More**).

# Repetitive & Combinatorial Systems

## Defining the system with mathematical sets

### Consider the following systems:

**PBPK:** Most of the organs in these systems are mathematically identical, with only variations in the parameters. However coding each of these organs or modifying an existing system (say to incorporate the presences of a target in each organ) can become tedious.

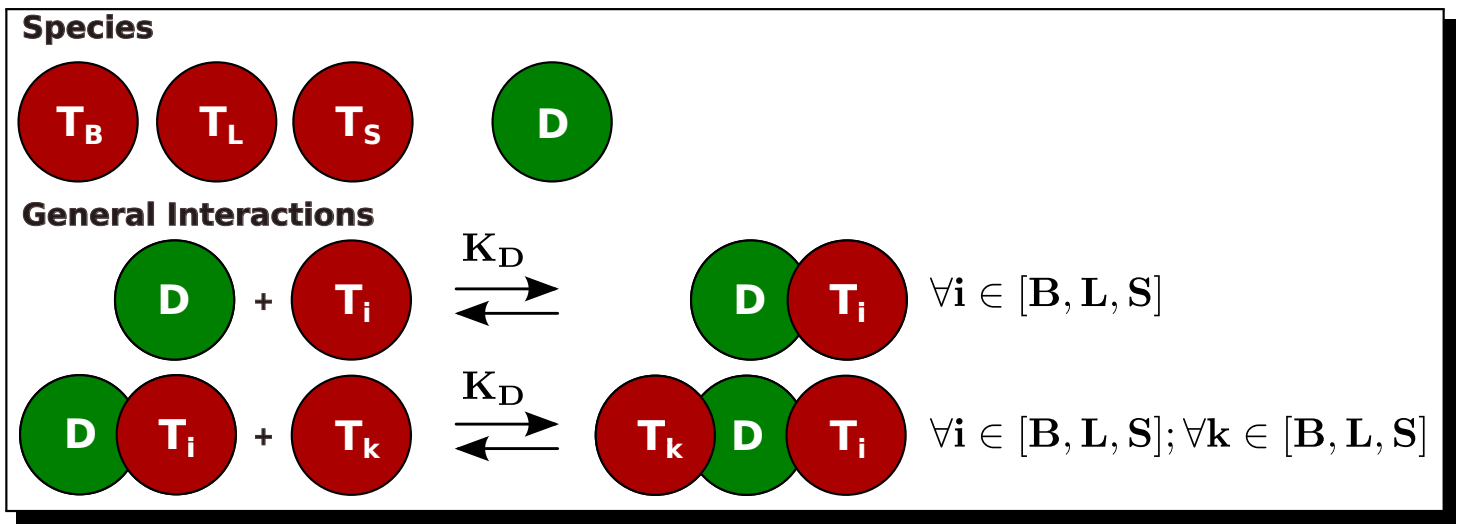
**Anti-drug antibody generation:** If we consider ADAs generated in response to therapeutic proteins, the response will consist of a distribution of ADAs in terms of their concentration and a separate distribution in terms of their affinity. Modeling this maturation process and the interactions between the ADAs, the therapeutic protein, and drug targets becomes unmanageable quite quickly.

The question is: How can we make difficult problems easy and intractable problems possible? The **solution** implemented here allows the system components to be defined in terms of **mathematical sets**.

# Assay Binding Model

<SET:??>?

Consider the interactions occurring in an assay designed to detect drug (D) present in serum. In this assay a biotinylated target (TB) is used to pull down the drug and a labeled target (TL) is the signaling molecules used. The assay will provide a signal when a complex containing both TB and TL are present (TB:D:TL or TL:D:TB). Samples can contain target as well (TS) which can interfere with the assay. To model this assay, the following interactions should be considered:



# Creating Sets & Defining Initial Conditions

## Basic components of mathematical sets

Several options are available to construct the system in the previous slide. The ODEs could simply be typed out for every possible combination. It's also possible to use the equilibrium `<=kon:koff=>` for all the interactions as well. However, there is another option that will handle the enumeration more easily. First we define the two mathematical sets `TSi` and `TSk`:

```
<SET:TSi>  TL; TB; TS
<SET:TSk>  TL; TB; TS
```

With these defined we can then use the curly brace notation (`{ }`) with **any** of the descriptors used to construct a system.

For example, the initial conditions for each of the target states are defined as parameters (`T0_TL`, `T0_TS`, `T0_TB`) in the model. These have to be identified as initial conditions using the `<I>` notation, and can be done with a single statement:

This line:

```
<I> {TSi} = T0_{TSi}
```

$\Rightarrow$

Expands to:

```
<I> TL = T0_TL
<I> TB = T0_TB
<I> TS = T0_TS
```

examples/system-sets.txt

# Defining Species Interactions

## Combinatorial enumeration

Similar to the initial condition, the equlibrium between the monomeric drug and the different targets can be defined using a single statement:

```
D        + {TSi}    <=kon:koff=>        D_{TSi}
```

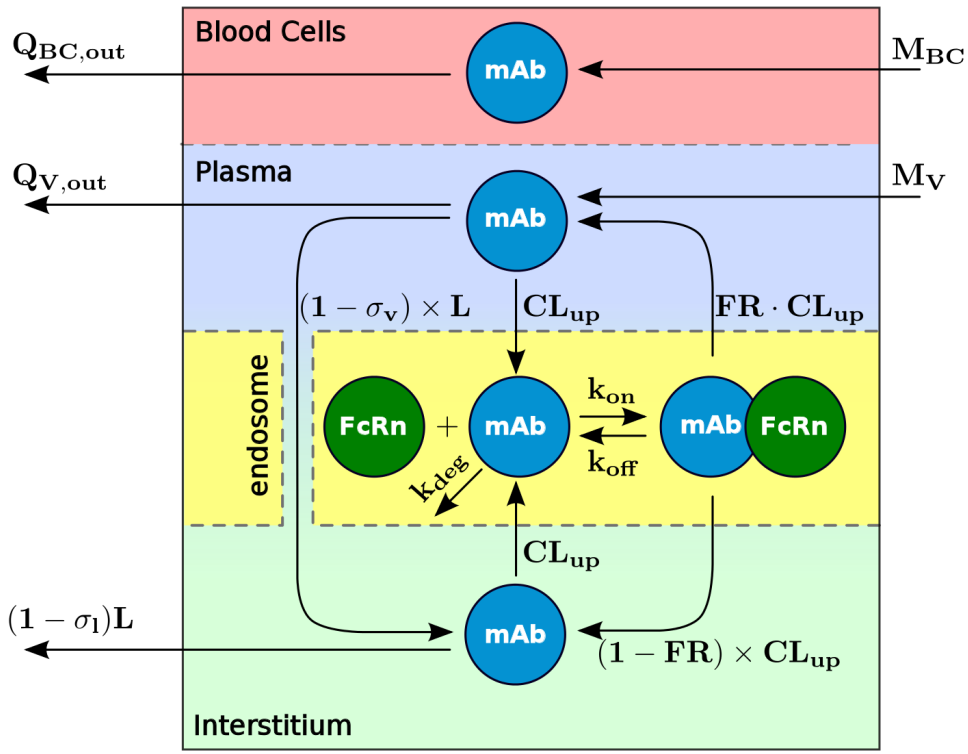
That uses only one of the sets (`TSi`), and will be expanded for each element in that set. To account for the formation of complexes that contain a drug molecule and two different target molecules, the following statement is used:

```
D_{TSi} + {TSk}    <=kon:koff=> {TSk}_D_{TSi}
```

This statement contains two different sets (`TSi` and `TSk`). When multiple sets are encountered, every possible combination is evaluated.

# PBPK Abstraction

## PBPK models using mathematical sets




### Flow Abstraction

Volumetric Flow calculations:

From Lung — Organs upstream of current organ

$$\begin{aligned} Q_{BC,tot} &= Q_{BC} + \sum_{US} Q_{BC,US} \\ Q_{V,tot} &= Q_V + \sum_{US} Q_{V,US} - L_{US} \\ Q_{BC,out} &= Q_{BC,tot} \\ Q_{V,out} &= Q_{V,tot} - L \end{aligned}$$

Mass Flow calculations:

*For all organs except the lung:*

$$\begin{aligned} M_V &= C_{V,LUNG} Q_V + \sum_{US} C_{V,US}(Q_{V,US} - L_{US}) \\ M_{BC} &= C_{BC,LUNG} Q_{BC} + \sum_{US} C_{BC,US}(Q_{BC,US}) \end{aligned}$$

*For the lung*

$$\begin{aligned} M_V &= (Q_V) C_{PLASMA} \\ M_{BC} &= (Q_{BC}) C_{BC} \end{aligned}$$

In order to take advantage of the set notation, we must first alter slightly how organs are modeled. Consider the liver which has drug entering directly from the lung as well as those organs upstream of the liver (e.g. the gut). Traditionally each of these flows would be defined separately as inputs into the liver. Instead we define the total mass flow in the vascular space ($M_V$) for each organ as a dynamic secondary parameter (see the figure). When this is done, then the ODEs (See [SB]) describing each organ are structurally the same.

*[SB] Shah & Betts (2012)*

# PBPK Implementation

This abstraction from the previous slide has been implemented for the system described by [SB], and can be seen by looking at the system-pbpk.txt file. The set ORG was created with all of the organs listed. This enabled all of the organ ODES to be written using the portion of the system.txt file listed below (separated into the vascular, endosomal, and interstitial spaces).

Relevant portion of system.txt file

```
# defining the organ set
<SET:ORG>  HEART;  LUNG;  MUSCLE;  SKIN; ADIPOSE; BONE; BRAIN; KIDNEY; LIVER; SM_INT; LG_INT; PANCREAS; THYMUS; SPLEEN; OTHER

#  Plasma (Vascular Space)
#                Vascular     Vascular Mass            Loss to                                  Pinocytosis             Pinocytosis
#                Mass In      Leaving                  Interstitium                             Uptake                  Return
<ODE:C_V_{ORG}> (M_VI_{ORG} - Q_VOUT_{ORG}*C_V_{ORG} - (1.0-SIGMA_V_{ORG})*L_{ORG}*C_V_{ORG} - CL_UP_{ORG}*C_V_{ORG} + CL_UP_{ORG}*FR*C_E_B_{ORG})/V_V_{ORG}
#  Blood Cells (Vascular Space)
<ODE:C_BC_{ORG}> MT_BC_{ORG}/V_BC_{ORG}

#
# Endosomaal space
#                   Endosomal Uptake                                  FcRn Binding                          FcRn                      Degradation
# Free Drug                                                           Association                           Disassociation
<ODE:C_E_UB_{ORG}>  (C_V_{ORG} + C_I_{ORG})*CL_UP_{ORG}/V_E_{ORG} - K_on_FCRN*C_E_UB_{ORG}*FCRN_{ORG}  + K_off_FCRN*C_E_B_{ORG} - K_deg*C_E_UB_{ORG}
# Free FCRN
<ODE:FCRN_{ORG}>                C_E_B_{ORG}*CL_UP_{ORG}/V_E_{ORG} - K_on_FCRN*C_E_UB_{ORG}*FCRN_{ORG}  + K_off_FCRN*C_E_B_{ORG}
# FCRN Bound
<ODE:C_E_B_{ORG}>             - C_E_B_{ORG}*CL_UP_{ORG}/V_E_{ORG} + K_on_FCRN*C_E_UB_{ORG}*FCRN_{ORG}  - K_off_FCRN*C_E_B_{ORG}

#
# Interstitial Space
#
<ODE:C_I_{ORG}>   (1.0-SIGMA_V_{ORG})*L_{ORG}*C_V_{ORG}/V_I_{ORG}
<ODE:C_I_{ORG}> - (1.0-SIGMA_I_{ORG})*L_{ORG}*C_I_{ORG}/V_I_{ORG}
<ODE:C_I_{ORG}> - CL_UP_{ORG}*C_I_{ORG}/V_I_{ORG}
<ODE:C_I_{ORG}> + CL_UP_{ORG}*(1.0-FR)*C_E_B_{ORG}/V_I_{ORG}
```

examples/system-pbpk.txt

# Simulation Flow Control

## Controlling model structure and parameterization on the fly

It may be necessary to alter aspects of the system depending on the situation. This may be simple or complex, and there are two different ways to control this:

Structural: Consider a model with a soluble target. It may be desirable to code the model where the target **may** or **may not** transport into the peripherial tissues. This can be accomplished by creating a switching parameter that can be either one or zero that is multiplied by the transport rates. Then the model can be parameterized (**More**) for two different parameter sets: one where the control parameter is 1 (transport enabled) and another where it is 0 (transport disabeled).

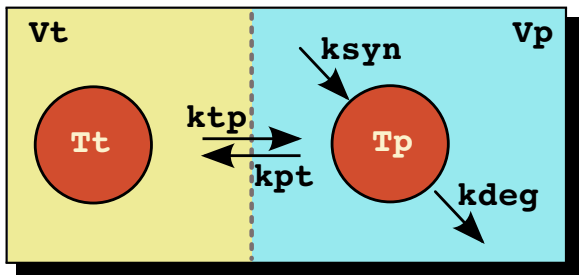
Time and Variable Dependent: We have a one compartment model with linear elimination that we want to switch to a slower rate after a certain time or when the concentration of the drug drops to a certain level (whichever comes first).

The following slides will illustrate each method.

# Target Presence in Peripherial Tissues

## Structural control through parameterization

In this example it is assumed that the target concentrations in both volumes will be specified, and that the nominal rate of transport from the plasma to the tissue `kpt_nom` will be specified. The nominal reverse transport can be calculated from mass balances:




```
<As> ktp_nom = kpt_nom*Tp_IC*Vp/(Tt_IC*Vt)
```

The parameter `TISSUE` is used to control the flows. In the default parameter set, the value is `1`. And in the `t_off` parameter set, it has a value of `0`.

```
<PSET:default>          Tt On
<PSET:t_off>            Tt Off
<PSET:t_off:TISSUE>     0
```

The values for the transport rates used in the ODEs are then the following secondary parameters:

```
<As> ktp = ktp_nom*TISSUE
<As> kpt = kpt_nom*TISSUE
```

See system file indicated below for more details

**It is important to construct quality control scripts to verify that the system is behaving as expected**

examples/system-target.txt

# Piecewise-Continuous Parameters

<IF:?:?>

In this example we specify fast (kelf) and slow (kels) rates of elimination. We want to have the fast rate be **active** when the drug concentration is above Cth and the time is below Tf. The system parameters would look like:

```
<P> kelf   1.0    eps   inf     1/time  yes     System
<P> kels   0.01   eps   inf     1/time  yes     System
<P> Cth    10     eps   inf     conc    yes     System
<P> Tf     10     eps   inf     time    yes     System
```

Now we need to define the rate of elimination such that the constraints above are followed. First we define kel as a dynamic secondary parameter with a value of 0.0. Then we define the different conditions and relevant values:

```
<Ad> kel  = 0.0
<IF:kel:COND> SIMINT_AND[SIMINT_LT[SIMINT_TIME][Tf]][SIMINT_GT[Cp][Cth]]; kelf
<IF:kel:ELSE> kels
```

examples/system-pwc.txt

# Piecewise-Continuous Parameters (Contd.)

## The details

To specify a conditional assignment use the statement:

`<IF:name:COND> boolean; value`

Here `name` is the name of the secondary parameter be defined and `COND` indicates that we have a boolean condition that needs to be satisfied. The condition (`boolean`) can be and, or, greater than, etc. relationships. The functions used to describe these relationships are specified in `system_help.txt`. The parameter will be assigned to have the `value` when this boolean relationship is true. These conditions can be a funciton of different elements of the system depending on whether or not `name` refers to a static or dynamic secondary parameter:

`<As>` function of system parameters, previously defined static secondary parmaeters and covariates that do not change for a given subject.

`<Ad>` function of system parameters, static secondary parmaeters, states, previously defined dynamic secondary parameters and covariates.

It is important to include a default `ELSE` condition:

`<IF:name:ELSE> value`

# Defining Data Files

<DATA:?:?>

<DATA:FILE:CSV> data/mydata.csv

This is used to define the data file to be used. This should be in a CSV format and optionally the first row can contain column names.

<DATA:HEADER:AUTOMATIC>

If this option is used, then the build script will attempt to read the first row of the specified CSV file. These will then be used to define the names of the columns.

<DATA:HEADER:MANUAL> col1; col2; col3;

Alternatively it's possible to specify the names of the columns manually. This is done by using the MANUAL option and then specifying the names delimited by semicolons.

# Variance Parameters

<VP>

Variance parameters are specified using the same format as system parameters (<P>) :

```
#     name   value   lower_bound   upper_bound   units   editable   grouping
<VP>  CL     1.0     eps           inf           1/hr    yes        Variance
```

The difference being that the <VP> descriptor is used and that the grouping is set to Variance.

# Inter-Individual Variability

## <IIV:?> ? & <IIVCOR:?> ?

To define an IIV term named ETACL with a variance of 0.15 use the following descriptor

```
<IIV:ETACL> 0.15
```

---

The specification to the right can be used to associate this IIV term with the parameter CL and specify that it has a lognormal distribution (LN). Alternatively a normal (N) distribution can be used.

```
<IIV:ETACL:LN> CL
```

---

Next we specify the IIV term ETAV with a variance of 0.1. This IIV term also has a lognormal distribution and is applied to the parameter V.

```
<IIV:ETAV>    0.10
<IIV:ETAV:LN> V
```

---

Now we can define the covariance (off-diagional elements) between CL and V to be 0.01 by using <IIVCOR:?:?>

```
<IIVCOR:ETAV:ETACL> 0.01
```

---

The order isn't important and the IIV terms can be reversed

```
<IIVCOR:ETACL:ETAV> 0.01
```

# Inter-Individual Variability and Parameter Sets

## <IIVSET:?> ? & <IIVCORSET:?> ?

By default all parameter sets will have inter individual variability specified using the IIV <IIV> and <IIVCOR> descriptors. To associate a specific set of IIVs to a parameter set use the <IIVSET> and <IIVCORSET> desciriptors. These set descriptors operate differently than the parameter set descriptors (<PSET>) in that the entire variance covariacne matrix needs to be specified.

If the parameterset MYPSET has been defined then the following could be used to define the IIV for the parameters Q and CL

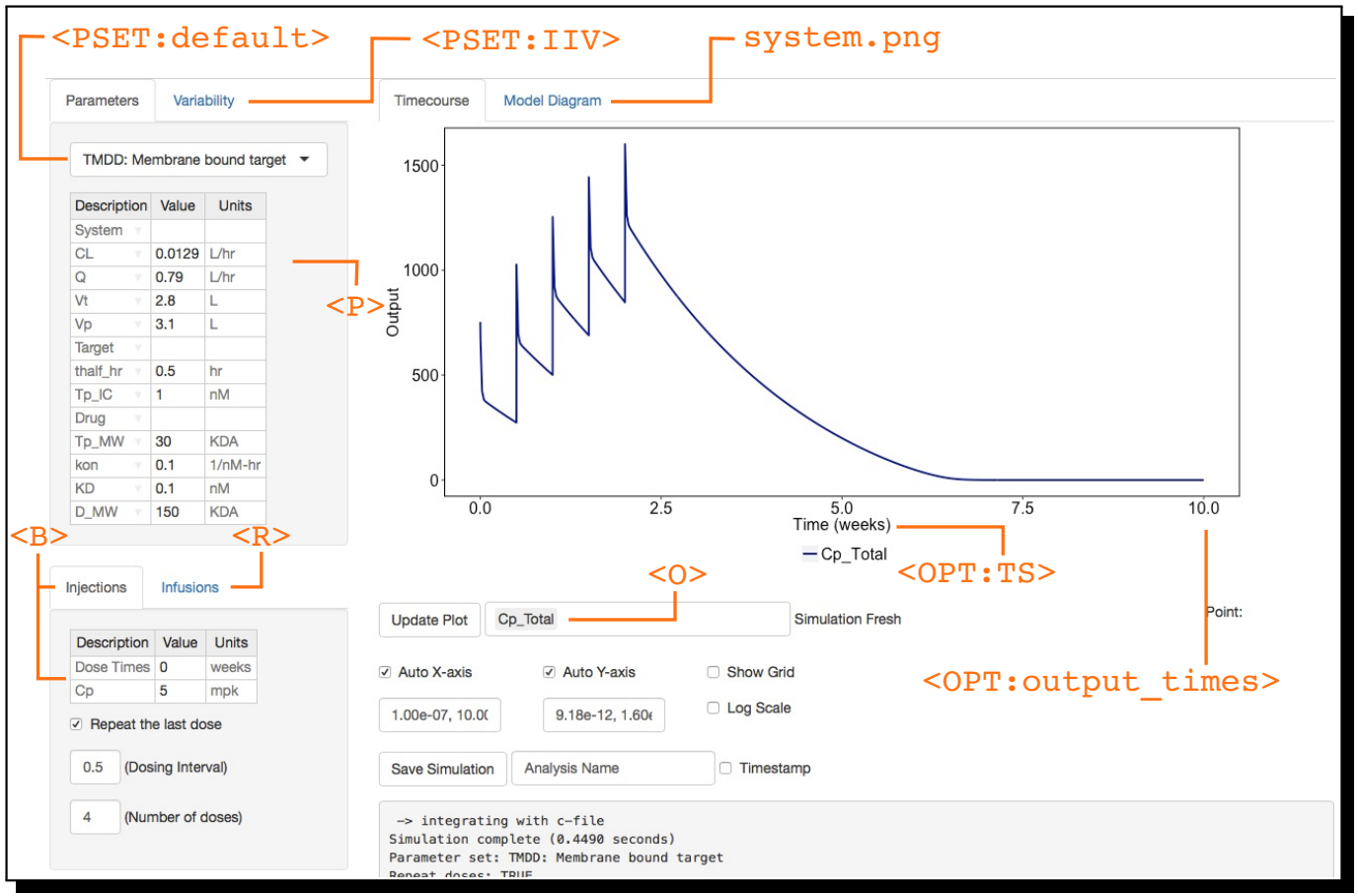
```
<IIVSET:MYPSET:ETAQ>              0.05
<IIVSET:MYPSET:ETAQ:LN>           Q

<IIVSET:MYPSET:ETACL>             0.25
<IIVSET:MYPSET:ETACL:LN>          CL

<IIVCORSET:MYPSET:ETAQ:ETACL>   .01
```

# General Interface Elements

## Linking elements between system.txt and the GUI




Model elements: ubiquity_app.R, <B:?>, <O>, <PSET:?:?>?, & <TS:?>. For more details on the Shiny interface (customization, usage, deployment etc) see **Shiny app section** of the R workflow section portion of the workshop.

# Software Targets

# Building the System

To build a system file you simply need to run the Perl script included in the template (`build_system.pl`). If run with no arguments it will read in the file `system.txt` an compile files for the different software targets. If you want to build a different system file (`mysys.txt`) , you simply make that the first argument to `build_system.pl`

```
perl build_system.pl mysys.txt
```

After the system has been built, files will be created in `transient` to support the different software targets.

# MATLAB Workflow

Several files related to analysis in MATLAB are generated, and are listed below. For a detailed description of these files and their use, see the part on the MATLAB Workflow.

Template scripts:

- auto_simulation_driver.m
- auto_analysis_estimation.m

Support files

- auto_fetch_system_information.m
- auto_map_simulation_output.m
- auto_initial_sizes.h
- auto_common_block.h
- auto_odes.h
- auto_odes.m
- auto_outputs.h
- auto_remap_odes.h
- auto_sim.m

# R Workflow

Several files related to analysis in R are generated, and are listed below. For a detailed description of these files and their use, see the part on the R Workflow.

Template scripts:

- auto_simulation_driver.r
- auto_analysis_estimation.r

Support files

- auto_rcomponents.r
- r_ode_model.c

# Adapt

Currently the system is being generate as a Fortran file (`target_adapt_5.for`) For each parameter set (`PSN`) a separate parameter file is also generated: `target_adapt_5-PSN.prm`. This should be able to be used for naïve-pooled analysis and individual simulations. Currently, the population components need to be added:

- IIV terms
- Covariate hooks

# Berkeley-Madonna

For each parameter set (PSN) a target file will be generated:
target_berkeley_madonna-PSN.txt
Placeholders are created for inputs such as infusion rates and covariates. These and bolus inputs will need to be specified by the end user.

# mrgsolve

For each parameter set (PSN) a target file will be generated:
target_mrgsolve-PSN.cpp

# NONMEM

## <NONMEM:INPUT:DROP:?> & <NONMEM:INPUT:RENAME:?>?

For each parameter set (PSN) a target file will be generated:

target_nonmem-PSN.txt

This software target is incomplete and is intended to provide a starting point for a NONMEM analysis. If there is a NONMEM user that would like to work to develop this target please reach out to the Ubiquity developers here: help.ubiquity.tools

There are certain NONMEM specific considerations. In the system.txt file you can specify a dataset using the <DATA:?:?> delimiter. When the system is built that dataset will be read in and if there are headers it will construct the $DATA component of the control stream. If you want to drop the column NTIME and rename the column WT to WEIGHT you can use the following:

```
<NONMEM:INPUT:DROP:NTIME>
<NONMEM:INPUT:RENAME:WT> WEIGHT
```

# NONMEM

## <INDEX:?:?> ? & <AMTIFY>?;?;?

It can be necessary to link state and output numbers in the dataset to the order of the states (differential equations) and outputs within the control stream. To force the state Ap to to be the first ODE and Cp to be the second output you can use the following:

```
<INDEX:STATE:Ap>    1
<INDEX:OUTPUT:Cp>   2
```

If in your `system.txt` file you are modeling in terms of concentration but you want to convert that to amounts within the control stream you can use the `<AMTIFY>` notation. If you defined the state `Cp` but want it to be `Ap` within the control stream and these are related by `Cp = Ap/Vp` and `Vp` is a parameter the following would be used:

```
<AMTIFY> Cp; Ap; Ap/Vp
```

# Table of Contents: Matlab Workflow I

## Model ID and simulation

# Table of Contents: Matlab Workflow II

## Model ID and simulation

# Implementation Details

# Toolboxes

The following toolboxes are needed:

- Simulink
- Optimization Toolbox
- Statistics and Machine Learning Toolbox

# Template

## The starting point for an analysis

To begin an analysis, create a copy of the `template` directory. This should contain all of the files that will be needed to create a system, build the different targets, and run the system within MATLAB.

The following slides will describe the MATLAB workflow for compiling and running simulations. File and directory names are specified **relative** to the root of the template directory.

- `examples` — example system files
- `library` — commonly used functions for running simulations, reading data files, and presenting output
- `transient` — directory created when a system is built that contains the generated files (`auto*` and `target*`)

# Custom Functions

## provision_workspace

In the template there is a directory named `library`, this contains the following two directories:

- `matlab_general` These are general functions that are used to facilitate analysis and output generation. In this directory there is a file called `notes.txt` which contains a listing of the functions, a brief description of their use, and an indication of the file was obtained from the MATLAB file exchange.
- `id_simulation` These are functions used specifically for running simulations and parameter estiamtion. Some of these are intended to be used directly while others are intermediate functions (functions called by other functions) and not intended to be run by the end user directly.

The script in the root of the template directory named `provision_workspace.m` is used to add these directories to the MATLAB path. **This script must be run near the beginning of any analysis.**

# Building the System

## build_system translating system.txt into target formats

In the template there is an examples directory. Copy the file examples/system-tmdd.txt to system.txt. And run the script build_system.m.

If no errors are encountered something like the text to the right should be displayed. The last message (Simulink found) indicates that you have a license for Simulink and the C file was compiled. This will allow you to switch between the slower m-file and the faster C-file.

```
>> build_system
-----------------------------------

Building the system to
generate the model targets
   -> Matlab: C/Simulink
   -> Matlab: m-file
-----------------------------------

Simulink found
mex-ing the model file
Building with 'Xcode with Clang'.
MEX completed successfully.
-----------------------------------
```

If any errors are encountered while building the system (such as parameters and states that share the same name), they will be displayed here.

build_system.m

# Running Simulations

# Example System

The TMDD system in the examples directory is being used as a starting point to discuss how simulations are run (`examples/system-tmdd.txt`). Modifications made to this system, specifically simulations with variability, will be highlighted.

We will plot the output of the total drug (`Cp_total`) relative to the affinity (`KD`) as well as the target coverage (`Coverage`). This will be done in response to dosing 10 mpk every two weeks for a total of four doses.

So in the `simulation_example` directory run `build_system` to compile the system.

# Running Simulations

## auto_simulation_driver & model_gui

**After building the system** there are two options for running the simulation. First, and probably the easiest, is to run the simulation through the graphical interface. This is done by running the following command:

```
» model_gui
```

Building the system creates several files in the `transient` directory. The file `auto_simulation_driver.m` provides all of the components needed to run the simulation at the scripting level. But **you should not edit this file directly** because it will be over written each time the system is built. Instead copy the file from the transient directory to the main directory (e.g., `analysis.m`).

# Simulation Template

As examples, two scripts are provided which build off of the simulation template (`auto_simulation_driver.m`)

`analysis_single.m` Simulates the typical response to dosing. These are deterministic simulations which provide a single profile.

`analysis_multiple.m` In this case the response of multiple subjects with interindivudal variability is simulated with the same dosing, covariates, etc.

Each of these files begins basically the same way. These common elements are described first and then the components that are different are then addressed.

# General Simulation Components

## Common elements

First the workspace is cleared and the system is rebuilt. This ensures that any changes in the `system.txt` file are included in the simulations:

```
clear; close all;
build_system;
```

Next the paths are added to make the functions described above available:

```
provision_workspace;
```

The function `auto_fetch_system_information` returns the system configuration variable `cfg`. This is a data structure that stores information about the system and simulations to be performed.

```
cfg = auto_fetch_system_information();
```

# General Simulation Components

## Common elements (Contd.)

Next the default parameter set is selected and the corresponding vector of parameters is retrieved:

```
cfg = system_select_set(cfg, 'default');

parameters =  system_fetch_parameters(cfg)
```

Because many default values for dosing may be specified in the `system.txt` file, these are first zeroed out. Then the value for bolus dosing to the state `Cp` is specified:

```
cfg = system_zero_inputs(cfg);
cfg = system_set_bolus(cfg, 'Cp', ...  %
                       [ 0  2  4  6], ... % weeks
                       [10 10 10 10]);    % mpk
```

# General Simulation Components

## Common elements

The `cfg` variable contains default options for running simulations. These can be overwritten using `system_set_option`. Here we are specifying the output times:

```
cfg = system_set_option(cfg, 'simulation',   ...
                             'output_times', ...
                            linspace(0,48*24, 1000)');
```

Next we specify that we want to use the compiled c-file with `simulink` to speed up integration:

```
cfg = system_set_option(cfg, 'simulation',     ...
                             'integrate_with', ...
                             'simulink');
```

Lastly the ode solver `ode23s` is specified.

```
cfg = system_set_option(cfg, 'simulation', ...
                             'solver', ...
                             'ode23s');
```

# Simulation Output

## run_simulation_ubiquity

```
som = run_simulation_ubiquity(parameters, cfg);
```

When a simulation is run, a data structure is created (named `som` here). This data structure **maps** simulation information into different fields. There are five main fields

`raw` The raw simulation output that MATLAB provides (fields: `t` (simulation times), `x` (state variables), and `y` (outputs))

`times` This contains at least one field `sim_times` (the default simulation time values), and additional fields for each time scale specified using `<TS:?>`

`states` This structure contains a field for each state in the system as well

`outputs` Similarly each output specified using `<O>` will also have a field

`meta` Composed of two fields (`parameters` & `secondary_parameters`) each with fields of named values for those parameters (system or secondary) that were used for this simulation run

`analysis_single.m`

# Simulation Output (Contd.)

## Some examples using the TMDD system

The remainder of the generated analysis script will simply plot each specified output. This portion can be modified depending on the specific analysis being run. Here are some examples that may prove to be useful:

### Simulation Output in Scripts:

The following will allow you to see the fractional coverage (**an output**) over **weeks**:

```
plot(som.times.weeks, som.outputs.Coverage)
```

If you would rather see how the free target (**a state**) changes over **days**:

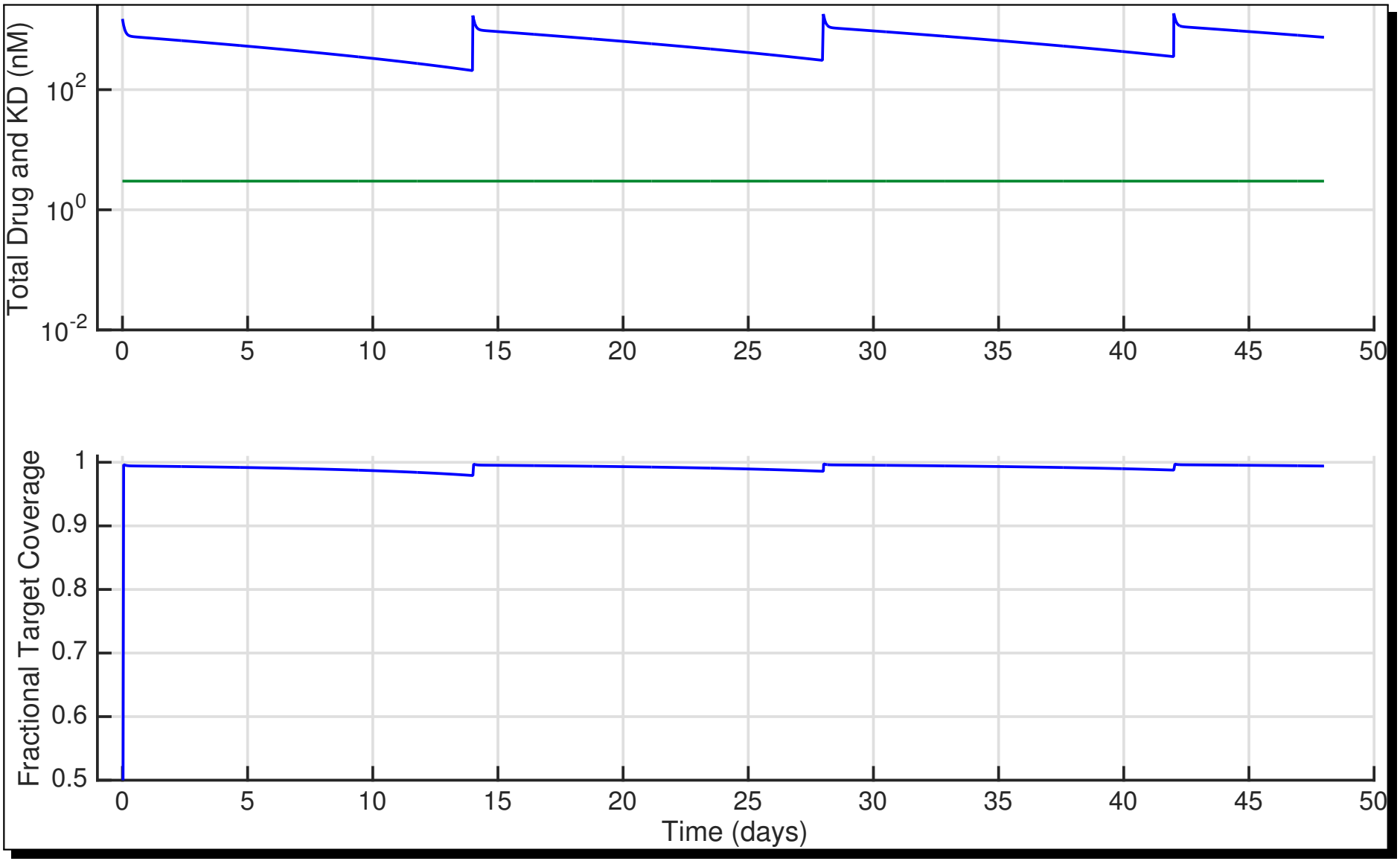
```
plot(som.times.days, som.states.Tp)
```

If you're curious, and would like to confirm that the correct initial condition was used in the simulation:

```
disp(som.meta.parameters.Tp_IC)
```

# PK, Coverage and KD Typical Value

## Total drug and range of KDs (top), coverage (bottom)



# Simulation Output (Contd.)

## Validating system at steady state: `check_steady_state`

Sometimes we may want to validate portions of the system to make sure that changes made to certain aspects have not inadvertently introduced any errors. One check, is to make sure the system is running at steady state in the absence of dosing. If we fix the dosing to zero (`system_zero_inputs`), we can run the script and create the `som` variable. This can then be checked by:

```
check_steady_state(som)
```

What this does is look at all of the states in `som` to see if there are any significant changes (those not attributable to numerical errors). If no problems are found you should see the following message:

```
|-> No steady state offsets found
```

This indicates that the system is running at steady state. Otherwise, you may see something like the message to the right. This can be used to debug your system and identify the problems with steady state calculations.

```
#-> Possible steady state offset
#-> range       |              | state
#-> (max-min)   | max(abs(s)) | state
#->-----------------------------------
#-> 2.510e-04   | 1.000e+01   | Tp
#->-----------------------------------
#-> Deviations from steady state found
#-> See above for details
#-> Machine Precision is 2.220e-16
```

`library/id_simulation/check_steady_state.m`

# Simulating Multiple Subjects

## Specifying variability

The TMDD example is modified by assuming that the variance of CL, Vt and Tp_IC is 0.05 and the covariance of CL and Vt is 0.01. We also assume the affinity has a relatively high variance of 0.5. To do this we would need to add the following to the examples/system-tmdd.txt file:

```
<IIV:ETACL>              0.05
<IIV:ETACL:LN>           CL
<IIV:ETAVt>              0.05
<IIV:ETAVt:LN>           Vt
<IIV:ETATp_IC>           0.05
<IIV:ETATp_IC:LN>        Tp_IC
<IIVCOR:ETAVt:ETACL>     0.01
<IIV:ETAKD>              0.5
<IIV:ETAKD:LN>           KD
```

See <IIV:?> ? & <IIVCOR:?> ? for more information.

# Simulating Multiple Subjects

## Setting optoins and running the simulation

The simulation inputs and options are defined the same as the would be for simulations of a single subject. What can be altered are the stochastic options. Here we specify that we want to simulate 100 subjects:

```
cfg  = system_set_option(cfg, 'stochastic', 'nsub', 100);
```

You can also specify the desired confidence interval and the random seed to be used.

Then using the typical parameter values and the inputs specified those 100 subjects will be simulated:

```
pred = simulate_subjects(parameters, cfg);
```

The variable `pred` is a data structure, similar to `som` for individual simulations, which contains the output of the stochastic simulations.

# Multiple Subject Output

## Parameters, outputs & states

### Parametric Information

The subject level parameter information is stored in two locations:

`pred.subjects.all` Full parameter vector (one per column) for each subject

`pred.subjects.name` A field for each parameter with interindividual variability. Each field contains a vector of parameter values that were used.

### Time-Course Information

The other fields contain the time-course for each subject:

`pred.times` A field for every timescale containing the sample times from the simulation.

`pred.states` and `pred.outputs` There is a field for each state or output which contains a profile for each subject (one per column) and each row corresponds to the sampling times in `pred.times`

# Multiple Subject Output
## Statistical Information

### Time-Course Information (Contd.)

`pred.states_stats` and `pred.outputs_stats`

There is a field for each state or output which contains the following fields:

- `lb_ci`: lower bound of the confidence interval for that named value
- `ub_ci`: upper bound of the confidence interval for that named value
- `mean`: mean of the prediction for that named value
- `median`: median of the prediction for that named value

These are all vectors corresponding to the sampling times in `prediction.times`.

# Multiple Subject Output
## atch variables

### Fields to generate shaded regions

`pred.times_patch`, `pred.states_patch` and `pred.outputs_patch` These contain vectors to be used with the patch command to generate shaded regions. A field for each specified timescale is provided in `times_patch` and a corresponding field for each state and output is found in `states_patch` and `outputs_patch`.

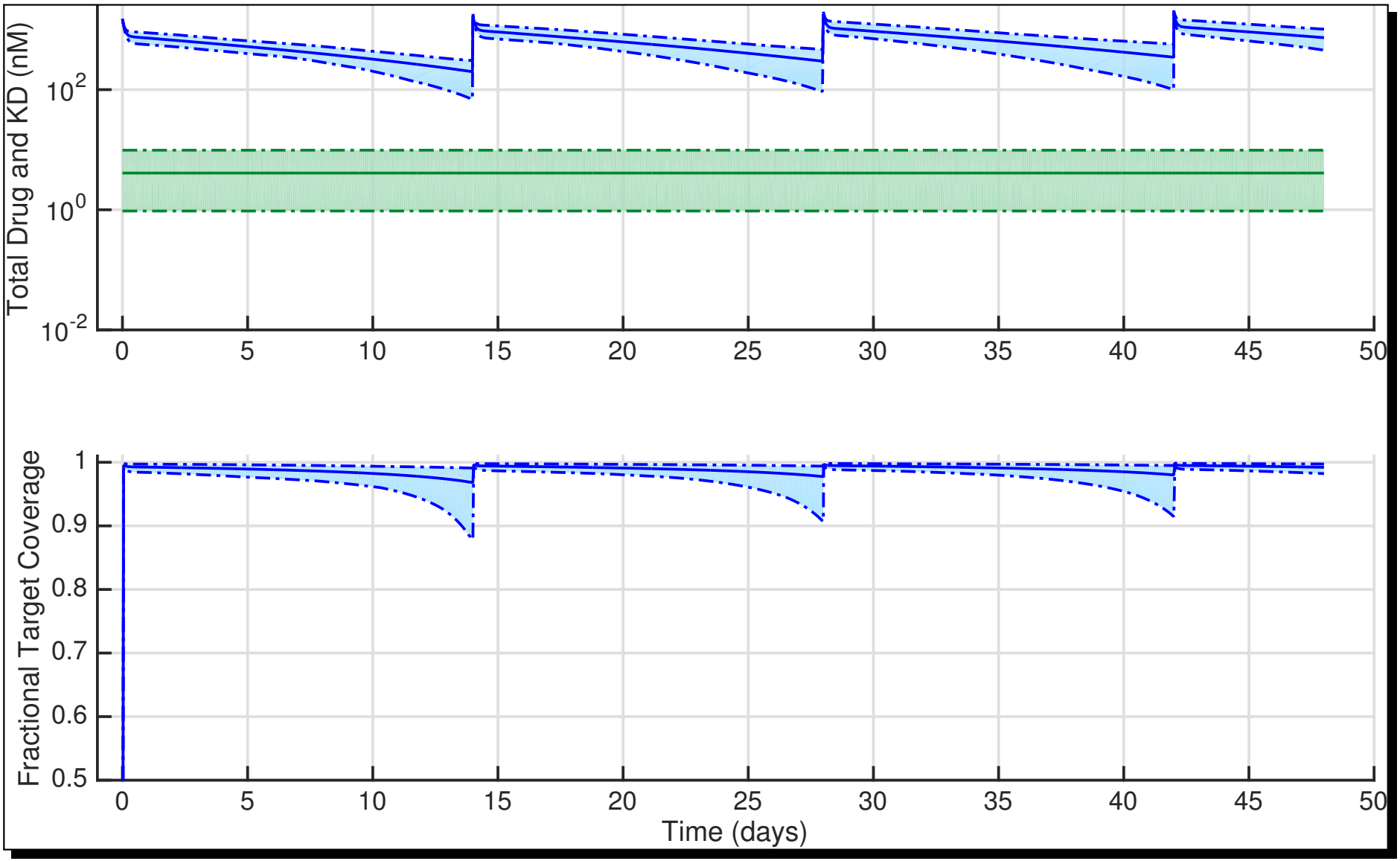
For example to plot the median response with a shaded confidence interval for the output `Cp_Total` over the timescale of `weeks`, the following could be used:

```
patch(pred.times_patch.weeks, ...
      pred.outputs_patch.Cp_Total.ci, 'y');

plot(pred.times.weeks, ...
     pred.outputs_stats.Cp_Total.mean, 'b-');
```

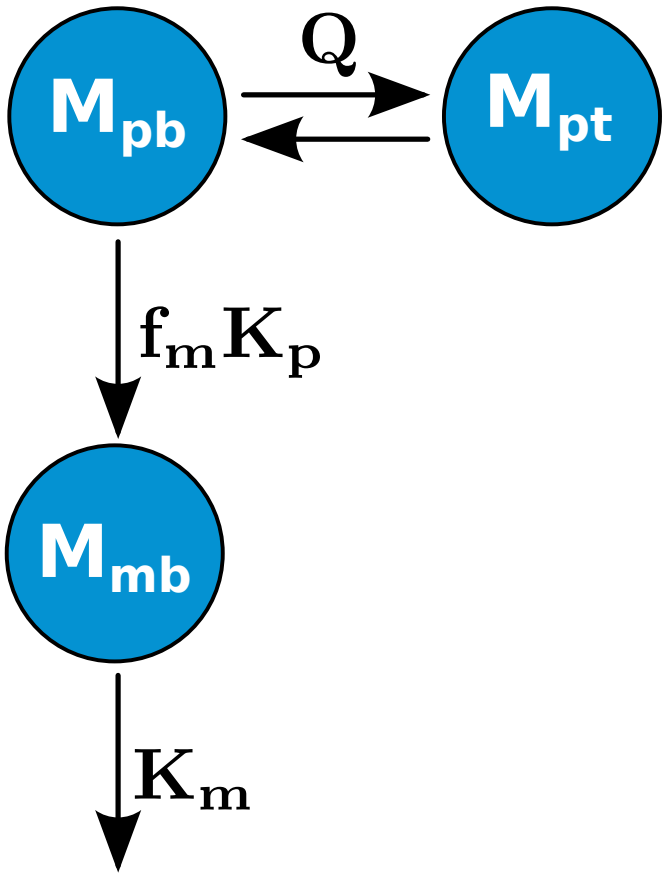
# PK, Coverage and KD with Varability

## Total drug and range of KDs (top), coverage (bottom)



# Parameter Estimation

# Example: Parent-Metabolite PK Model




$$\frac{dM_{pb}}{dt} = -\left(K_p + \frac{Q}{V_p}\right) M_{pb} + \frac{Q}{V_t} M_{pt}$$

$$\frac{dM_{pt}}{dt} = \frac{Q}{V_p} M_{pb} - \frac{Q}{V_t} M_{pt}$$

$$\frac{dM_{mb}}{dt} = K_p M_{pb} - K_m M_{mb}$$

$$C_{pb} = \frac{M_{pb}}{V_p}, \qquad C_{mb} = \frac{M_{mb}}{V_m / f_m}$$

# Representation in system.txt Format

```
<P> Vp     10.0    1e-5    100   L          yes        System
<P> Vt     10.0    1e-5    100   L          yes        System
<P> Vm     30.0    1e-5    100   L          yes        System
<P> CLp     1.0    1e-5    100   L/hr       yes        System
<P> CLm     1.0    1e-5    100   L/hr       yes        System
<P> Q       0.3    1e-5    100   L/hr       yes        System

<PSET:default> Original Estimates

<VP> slope_parent      0.1   1e-9     10   --   no Variance
<VP> slope_metabolite  0.1   1e-9     10   --   no Variance

<B:times>;             [  0  ];       1;          hours
<B:events>;    Mpb;    [  0  ];      70;          mpk

<ODE:Mpb> -(CLp/Vp + Q/Vp)*Mpb + Q/Vt*Mpt
<ODE:Mpt> Q/Vp*Mpb - Q/Vt*Mpt
<ODE:Mmb> CLp/Vp*Mpb  - CLm/Vm*Mmb

<O> Cpblood      = Mpb/Vp
<O> Cmblood      = Mmb/Vm

<TS:hours> 1
<TS:days>  1/24

<IIV:ETAVp>     0.08
<IIV:ETAVp:LN> Vp
<IIV:ETAVt>     0.08
<IIV:ETAVt:LN> Vt
<IIV:ETACLp>    0.08
<IIV:ETACLp:LN> CLp
```

estimation_example/system.txt

# Overall Workflow

auto_analysis_estimation

Analysis Process

- Create system configuration variable (cfg), identify the parameter set and parameters to be estimated.
- Specify the simulation and estimation options
- Load relevant datasets
- Define cohorts to be included in the estimation
- Perform the parameter estimation
- Simulate the response of the cohorts to the estimation results
- Plot the cohort data and simulated predictions

analysis_parent.m Analyze parent data at the 10 and 30 mg doses. With the variance equal to model prediction squared.
analysis_parent_metabolite.m Analyze only parent and metabolite data at the 10 and 30 mg doses. Using maximum likelihood with a proportional error model.

# Preamble

```
clear; close all;
flowctl = 'estimate';
analysis_name = 'parent_d1030';
```

The top part of the analysis scripts are used to clear out the workspace, and create variables to control the flow of the script `flowctl` and identify the analysis with a short name `analysis_name`. Normally when beginning an estimate `flowctl` can be set to `'plot guess'`. This will plot the model predictions over the data for each cohort for the initial guess to visualize how well the initial guess predicts the data. The `analysis_name`* will be prepended to different outputs of the estimation process and stored in `output/`. Next the system is built and the paths are setup. The variable `mc` contains some predefined colors that can be used in plotting.

```
build_system
provision_workspace;
mc = fetch_color_codes;
```

* Names must begin with a letter and can contain letters, numbers and underscores

# System Information

The system information is stored in the ubiquity model object `cfg`. The parameters that are going to be estimated are listed in `pnames` (the others will be fixed). This is passed to `system_select_set`when the default parameter set is selected.

```
% Pulling out the system information
cfg = auto_fetch_system_information;
% Selecting the default parameter set:
pnames = {'Vp'
          'Vt'
          'CLp'
          'Q'};
cfg  = system_select_set(cfg, 'default', pnames);
```

Next different options (solver, estimation, etc) are set using `system_set_option`.

```
cfg = system_set_option(cfg, 'simulation', 'integrate_with', 'simulink');
cfg = system_set_option(cfg, 'solver',     'solver',         'ode23s');
```

# Loading Data

Datasets can be loaded at the scripting level using the function `system_load_dataset`. These can be loaded from excel sheets, delimited files (csv or tab), or data frames using the following syntax:

```
cfg = system_load_dataset(cfg, 'DSNAME'  'data.xls', 'sheetname');
cfg = system_load_dataset(cfg, 'DSNAME'  'data.csv');
cfg = system_load_dataset(cfg, 'DSNAME'  'data.tab');
```

Where `DSNAME` is the name* of the dataset to be used internally. Multiple datasets can be loaded as long as they are given different names. Datasets should be in a NONMEM-ish format with the **first row containing the column header names**. Also, only observation records are needed. Alternatively a dataset, if it is a **CSV file**, can be specified in the `system.txt` file using the `<DATA:?:?>`. A data file loaded this way will have the name `default`.

* Names must begin with a letter and can contain letters, numbers and underscores

# Datafile Format

Parent (PT) and metabolite (MT) tiemcourse and BLQ are mg/L. The times (TIME) are in hours and the DOSE is in mg/kg. Values below the BLQ are listed as -1.

| TIME | ID | PT | MT | BLQ | DOSE |
|---|---|---|---|---|---|
| 2.0833e-02 | 1.0000e+00 | 8.6226e+00 | -1.0000e+00 | 1.0000e+00 | 1.0000e+00 |
| 4.1667e-02 | 1.0000e+00 | 8.1785e+00 | -1.0000e+00 | 1.0000e+00 | 1.0000e+00 |
| 7.0000e+00 | 1.0000e+00 | 2.3188e+00 | -1.0000e+00 | 1.0000e+00 | 1.0000e+00 |
| 1.4000e+01 | 1.0000e+00 | 1.9491e+00 | 1.0260e+00 | 1.0000e+00 | 1.0000e+00 |
| 1.4000e+01 | 2.0000e+00 | 1.3818e+00 | 1.1974e+00 | 1.0000e+00 | 1.0000e+00 |
| 2.1000e+01 | 2.0000e+00 | -1.0000e+00 | 1.0475e+00 | 1.0000e+00 | 1.0000e+00 |
| 2.8000e+01 | 2.0000e+00 | -1.0000e+00 | 1.1375e+00 | 1.0000e+00 | 1.0000e+00 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 7.0000e+01 | 6.0000e+01 | 3.3954e+00 | 1.1927e+01 | 1.0000e+00 | 3.0000e+01 |
| 8.4000e+01 | 6.0000e+01 | 1.6391e+00 | 5.5330e+00 | 1.0000e+00 | 3.0000e+01 |
| 9.8000e+01 | 6.0000e+01 | 1.7888e+00 | 4.6481e+00 | 1.0000e+00 | 3.0000e+01 |

To load this dataset and name it pmdata the following is used:

```
cfg = system_load_dataset(cfg, 'pmdata' , 'data.csv');
```

Data: estimation_example/data.csv

# Defining Cohorts

The following removes any cohorts that have been defined:

```
cfg = system_clear_cohorts(cfg);
```

Next we create the data structure `cohort`. Each cohort has a name* (eg `dose_10`). The dataset (`pmdata`) containing the information for this cohort is identified. Next it is necessary to define a filter (ie cohort filter `cf`) which identifies the subset of the dataset (`DOSE = 10`) that pertains to this cohort:

```
clear cohort;
cohort.name                                    = 'dose_10';
cohort.cf.DOSE                                 = [10];
cohort.dataset                                 = 'pmdata';
```

Next we define the dosing for this cohort. It is only necessary to define those inputs that are non-zero.

```
cohort.inputs.bolus.Mpb.AMT                    = [10];
cohort.inputs.bolus.Mpb.TIME                   = [0];
```

* Names must begin with a letter and can contain letters, numbers and underscores

# Defining Cohorts (Contd.)

Next we need to match the outputs in the model to the outputs in the dataset. Under `cohort.outputs`there is a field for each output. Here **output name** of the serum pk of the parent output is `Parent`. The times and observations in the dataset are found in the `'TIME'` column and the `'PT'` column (missing data specified by -1). These are mapped to the model outputs (which MUST have the same units) `'hours'` and `'Cpblood'`:

```
cohort.outputs.parent.obs.time          = 'TIME';
cohort.outputs.parent.obs.value         = 'PT';
cohort.outputs.parent.obs.missing       = -1;
cohort.outputs.parent.model.time        = 'hours';
cohort.outputs.parent.model.value       = 'Cpblood';
```

**Note: Output names should be consistent between cohorts so they will be grouped together when plotting results.**

# Defining Cohorts (Contd.)

You must also specify the variance model to use for this cohort/output combination. This is a string that can be `'1'` for least squares estimation. Or a mathematical formula using any of the system parameters and PRED, TIME, and OBS for the model prediction, sample time and observation respectively. For example `'PRED^2'` or `'SLOPE*PRED^2'` would be valid if `SLOPE` were a system or variance parameter.

In this example a least squares estimation is being performed:

```
cohort.outputs.parent.model.variance          = '1';
```

Lastly this cohort must be added to `cfg`:

```
cfg = system_define_cohort(cfg, cohort);
```

This is a brief overview, see the help for `system_define_cohort` more details on how to describe a cohort.

# Parameter Estimation

Parameter estimation is performed using the following function:

```
pest =system_estimate_parameters(cfg, flowctl, analysis_name);
```

The parameter `flowctl` can take either of the following values:

| | |
|---|---|
| `'plot previous estimate'` | `'estimate'` |
| `'previous estimate as guess'` | `'plot guess'` |

The results are stored in `output` using the value of `analysis_name` (`'parent_d1030'` here) creating the following files:

- `parent_d1030-monitor_estimation_progress.jpg` (`.pdf` and `.png`) - This is an image of the estimation status figure that is updated during the estimation.
- `parent_d1030-parameters_all.csv`* CSV file of all the parameters (names, guesses, estimates, etc) parameter
- `parent_d1030-parameters_est.csv`* Same as the previous csv file except it only contains the estimated parameters.
- `parent_d1030-report.txt` Estimation report with the estimates, variance/covariance matrix, AIC, etc.

* these fields are only created if the solution statistics were successfully calculated.

# Parameter Estimation

The estimates are stored in `pest` and this vector can be used to simulate the system for all of the cohorts:

```
erp = system_simulate_estimation_results(pest, cfg);
```

The variable `erp` contains the simulated results at the parameter estimates and the data for each cohort. This can then be plotted:

```
system_plot_cohorts(erp, plot_opts, cfg);
```

The variable `plot_opts` is used to control how predictions and data are overlaid. The general format for a plot option for a given output (`OUTPUT`) is:

```
plot_opts.outputs.OUTPUTt.option        = value
```

For example, to set the axis scale to logarithmic the following would be used:

```
plot_opts.outputs.OUTPUT.yscale         = 'log'
```

The help for `system_plot_cohorts` details the different options that can be set.

# Accessing Estimation Results

The following simulates the system for each cohort.

```
erp = system_simulate_estimation_results(pest, cfg)
```

The variable `erp` contains the simulated information in both a structured format and as a flat file. To access details for a given cohort `COHORT` and output `OUTPUT`, the following structure would be used:

```
erp.cohorts.COHORT.OUTPUT
```

This has two fields. The first `od` has the following fields:

- `var` - variance
- `time_smooth` - smooth time vector
- `pred_smooth` - smooth prediction vector
- `time` - observation times
- `obs` - observations
- `pred` - predictions at `time`

The output from `run_simulation_ubiquity` for the `COHORT`/`OUTPUT` combination can be accessed using the `som` field:

```
erp.cohorts.COHORT.OUTPUT.som
```

# Accessing Estimation Results

The other field of `erp` is `erp.flat`, this is a cell array with the following headers:

- `time` - time in the units of the dataset
- `obs` - observation (`type = 'record'`), -1 (`type='smooth'`)
- `pred` - model prediction
- `var` - variance (`type = 'record'`), -1 (`type='smooth'`)
- `res` - residual error (`type = 'record'`), -1 (`type='smooth'`)
- `wres` - weighted residual error (`type = 'record'`), -1 (`type='smooth'`)
- `cohort` - cohort name
- `output` - output name
- `type` - entry type (`record` or `smooth`)

To save to a csv file use:`cell2csv(erp.flat, 'myfile.csv')`

# Speeding up Estimation

Parameter estimation involves running multiple simulations at each estimation step. The speed of the estimation is dependent on how fast simulations run and the number of simulations that need to be run. In the context of these scripts there are two factors under your control:

**Number of cohorts:** Each cohort represents an extra set of simulations that needs to be performed. To decrease the number of simulations that needs to be performed, group your data so that you minimize the number of cohorts.

**Number of sample points:** Simulation speed decreases as the number of sample times increases. Set the simualtion output times to sample sparsely before the estimation. Then before running the simulations to generate figures (just before before `system_simulate_estimation_results`)redefine the simulation output times with more frequent samples.

# Controlling Scripts with `cfg`

# System Information

## cfg

The following command:

```
cfg = auto_fetch_system_information()
```

Creates a data structure called `cfg`, and this variable contains all of the information about the system specified in the `system.txt` file. This includes parameter values, parameterizations (parameter sets), covariates, default dosing, etc. It also contains the details which control a simulation (individual and stochastic).

At the scripting level it is necessary to alter these default values. The next few slides discuss how to control simulations by using `system` functions.

# Parameter Information

Selecting the default parameter set

```
cfg = system_select_set(cfg, 'default')
```

Retrieving a vector of parameters for the current parameter set:

```
parameters = system_fetch_parameters(cfg)
```

To overwrite parameter PNAME with VALUE use the following:

```
parameters = system_set_parameter(cfg, ...
                                  parameters, ...
                                  'PNAME', ...
                                  VALUE)
```

Covariates may also change with the selected parameter set. These can also be overwritten. To change the covariate COV, at times TV to values CV use the following:

```
cfg = system_set_covariate(cfg, 'COV', TV, CV)
```

# Model Inputs

By default the model inputs will have the values specified in the `system.txt` file. When attempting to develop a placebo response or construct a specific set of input profiles, it may be convenient to **zero** out the values of the inputs (bolus and infusion rates). This can be accomplished in the following manner:

```
cfg = system_zero_inputs(cfg)
```

With all of the inputs zeroed out, input profiles can be built. To set the bolus dosing information for a state `SNAME` at times `TV` to values `BV` use the following:

```
cfg = system_set_bolus(cfg, 'SNAME', TV, BV)
```

To set the infusion rate information for a rate `RNAME` at times `TV` to rates `RV` use the following:

```
cfg = system_set_rate(cfg, 'RNAME', TV, RV)
```

# Variability

The variability in the system is specified using <IIV:?> ? & <IIVCOR:?> ?.
The variance and covariance can be specified using the following command:

```
cfg = system_set_iiv(cfg, IIV1, IIV2, VALUE)
```

If you want to set the covariance of ETACL and ETAVc to 0.03 the following would be used:

```
cfg = system_set_iiv(cfg,      ...
                     'ETACL',  ...
                     'ETAVc',  ...
                     0.03)
```

# Options

## system_set_option

Different options are defined using the following function:

```
system_set_option(cfg, 'GROUP', ...
                       'OPTION', ...
                        VALUE) ...
```

To use this you need to specify a GROUP, an OPTION and a VALUE. For example to specify the simulation output times should be from 0 to 24, the following would be used:

```
cfg = system_set_option(cfg, 'simulation',    ...
                             'output_times',  ...
                             linspace(0,24,100))
```

In the subsequent slides, the different groupings are described with the options enumerated and the default values.

# Simulation Options

`group = 'simulation'`

This grouping controls general aspects of the simulation and has the following options:

`'include_important_output_times'`- Automatically add bolus, infusion rate switching times, etc: `'yes'`(default), `'no'`.

`'integrate_with'`- Specify if the ODE solver should use the the matlab script (`'m-file'`, default) or compiled C through Simulink (`'simulink'`)

`'output_times'`- Vector of times to evaulate the simulation (default `linspace(0,100)`).

`'solver'`- Selects the ODE solver: `'ode23s'` (default), `'ode15s'`, `'ode45'`, etc.; see the documentation for MATLAB ODE solvers for an exhaustive list.

# ODE Solver

`group = 'solver'`

Depending on the solver, different options can be set. The documentation for MATLAB ODE solvers lists the different solvers. For a full list of options, see the documentation for ODESET (`help odeset`). Some common options to consider are:

- `'AbsTol'` - Relative error tolerance
- `'RelTol'` - Absolute error tolerance
- `'MaxStep'` - Maximum integration step size

To select the `ode23s` solver and set the maximum step size to `0.01`, the following would be used:

```
cfg=system_set_option(cfg, 'simulation',
                           'solver',  'ode23s')
cfg=system_set_option(cfg, 'solver',
                           'MaxStep',  0.01)
```

# Stochastic Simulations

group = 'stochastic'

When running stochastic simulations (inter-individual variability applied to system parameters) it can be useful to specify the following:

'ci' - Confidence interval (default 95)

'nsub' - Number of subjects (default 100)

'seed' - Seed for the random numebr generator (default 8675309)

'ponly' - Only generate the subject parameters but do not run the simulations (default FALSE)

'outputs' - A character array of the predicted outputs to include (default all outputs defined by <O>)

'states' - A character array of the predicted states to include(default all states

# Stochastic Simulation Examples

group = 'stochastic'

If you wanted to generate 1000 subjects but only wanted the parameters, you would use the following:

```
cfg = system_set_option(cfg, 'stochastic',  ...
                             'nsub',    1000)
cfg = system_set_option(cfg, 'stochastic',  ...
                             'ponly',    true)
```

If you wanted to exclude states and only include the output Cp_nM, you would do the following:

```
cfg = system_set_option(cfg,         'stochastic',  ...
                        'states',    {})
cfg = system_set_option(cfg,         'stochastic',  ...
                        'outputs',   {'Cp_nM')
```

# Estimation

group = 'estimation'

To perform an estimation the MATLAB function fminsearch is used. This is controlled by using the optimset function. These values are set using the option optimization_options. For example to dispaly the iterations, and set the maximum number of iterations, function evaluations, and the tolerance the following would be used:

```
cfg = system_set_option(cfg, 'estimation', ...
            'optimization_options', ...
             optimset('Display',   'iter', ...
                      'TolFun',     1e-3,  ...
                      'MaxIter',    3000,  ...
                      'MaxFunEval', 3000));
```

See the documentation for OPTIMSET (help optimset)

# Estimation (Contd)

group = 'estimation'

The option 'effort' can be set to a value of 1 and a single parameter estimation will be performed. This will monitor and display the estiamation process. The monitor has the following options:

'monitor_exit_when_stable' - 'yes' or 'no' (default)

'monitor_iteration_history' - This is a positive integer (default 100)

'monitor_slope_tolerance' - Positive number (default 0.001)

When 'monitor_exit_when_stable' is 'yes' the optimizer will finish up when the parameters and objective function have 'stabilized'. This is accomplished by fitting a line to these values over a specified iteration history and comparing the largest slope to a tolerance.

When the 'effort' is greater than 1, the monitoring will be turned off and the estimation routine to try "harder" to find a good parameter estimate by performing a simulated annealing routine to avoid local minima.

# Estimation (Contd)

group = 'estimation'

The figure generated during a single estimation is controlled by the monitor_status_function option. This can be customized by the user. For example it can be used to write the parameters at each optimization iteration to a file. To do this simply copy the default estimation status function (estimation_status.m) into the template root to say mystatus.m. Modify that copy, and tell the estimation routine to use that function:

```
cfg = system_set_option(cfg, 'estimation', ...
                             'monitor_status_function', ...
                             'mystatus');
```

If you wish to disable the status funciton (prevent the generation of the figure) just set the monitory_status_funciton to an empty string:

```
cfg = system_set_option(cfg, 'estimation', ...
                             'monitor_status_function', '');
```

# Estimation (Contd)

## Setting the initial guess for parameters

When performing a parameter estimation, the initial guess will be the value specified in the `system.txt` file for the currently selected parameter set. The following command can be used after the parameter set has been selected to specify the value (`VALUE`) of the parameter `PNAME` and optionally the lower (`lb`) and upper (`ub`) bounds:

```
cfg = system_set_guess(cfg, 'PNAME', VALUE, lb, ub);
```

To set the initial guess for the parameter Vc to a value of 3, the following would be used:

```
cfg = system_set_guess(cfg, 'Vc', 3);
```

To specify the guess and overwrite the upper bound on Vc and set it to 5

```
cfg = system_set_guess(cfg, 'Vc', 3, [], 5);
```

# Loading Datasets

```
cfg = system_load_dataset(cfg, 'dsname', 'data_file' , 'data_sheet');
```

`dsname` - Short name given to the data set. This hsould begin with a letter and can contain any combination of letters, numbers and _. No spaces and no dashes can be used.

`data_file` - This is the name of the file and it can have the following extensions: xls, csv, and tab. This file should be NONMEM-ish with the column names in the first row.

`data_sheet` - When the file type is xls, then the excel sheet can be specified as an option.

# Defining Cohorts

The following clears any cohorts currently defined in the system:

```
cfg = system_clear_cohorts(cfg);
```

Subsequently, cohorts can be defined using the following command:

```
cfg = system_define_cohort(cfg, cohort);
```

Where `cohort` is a data structure describing the cohort being added. This contains information about the dataset to be used, how the dataset should be filtered to return only data for that cohort, the outputs being used, inputs (dosing, infusions, etc) for this cohort, etc.

The following slides will describe the fields in cohort and their format. For detailed information and examples see the help for `system_define_cohort`.

# Defining Cohorts (Contd.)

## General options

`cohort.name`- This is a required field that contains a string that must begin with a letter and can contain letters, numbers and _.

`cohort.dataset`- Name of the data set defined using `system_load_dataset`.

`cohort.cp`- This is a data structure with fields that are parameter names and values that these parameters should be set for this specific cohort.

`cohort.cf`- This is a filter that is applied to the dataset to return the subset that applies to the current cohort. This is a data structure where the fields are the column names of the dataset. The values represent the values in the record that must be true to be included. If multiple fields are provided they will be 'anded' together to extract this subset.

# Defining Cohorts (Contd.)

## Inputs

`cohort.inputs`- There is a field here for each type of input (bolus, infusion_rates, and covariates). Each of these inputs has the same format. A field for the input name and under that input name there is an `TIME` and `AMT` fields. These are vectors of times and amounts with the units described in the `system.txt` file. **It is only necessary to specify the bolus and infusion rates that are nonzero for this cohort and the covariate that is different from the parameter set for this cohort.**

Bolus

```
cohort.inputs.bolus.STATE.TIME          = [];
cohort.inputs.bolus.STATE.AMT           = [];
```

Infusion Rates

```
cohort.inputs.infusion_rates.RNAME.TIME = [];
cohort.inputs.infusion_rates.RNAME.AMT  = [];
```

Covariates

```
cohort.inputs.covariates.CNAME.TIME     = [];
cohort.inputs.covariates.CNAME.AMT      = [];
```

# Defining Cohorts (Contd.)

## Outputs

`cohort.outputs`- There is a field for each output here. This field, like the name for the cohort, should begin with a letter and can contain only letters, numbers and _. For each output there is an `obs` field that describes the columns in the dataset to be used that contain both the independent variable (`time`) and the dependent variable (`output`). If there is missing data, an optional field of `missing` can be specified and the value used to indicate a missing observation can be specified (e.g. -1).

There is also a field called `model` which contains the timescale from the model which matches the `time` in the dataset. It also contains a `value` field that maps the model output to the observations in the dataset.

`cohort.options`- This field contains optional information used to customize how this cohort is simulated and what is returned.

`cohort.output_times`- By default, the simulation output times will be used. If it is necessary to have finer control over this, a different vector of times can be specified.

# System Information

When modfying the system, it's possible to loose track of the changes that have been made. To see the current state of the system, the function `system_view` can be used:

```
system_view(cfg)
```

By default this will show **all** of the information about the system. It's also possible to specify certain components using the second argument (`'parameters'`, `'bolus'`, `'infusion_rates'`, `'covariates'`, etc).